Dr. Mars Caroline Wibowo, ST, M.Mm.Tech.

YAYASAN PRIMA AGUS TEKNIK

MEMBUAT VIDEO GAME dengan 3D Unity

Membuat Video Game Dengan 3D Unity

Penulis :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

ISBN : 9 786235 734606

Editor :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Penyunting :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji syukur pada Tuhan Yang Maha Esa bahwa buku yang berjudul "**Membuat Video Game dengan 3D Unity**" ini dapat diselesaikan dengan baik. Pada zaman yang serba menggunakan teknologi sekrang ini banyak orang yang menyukai video game baik itu dimainkan di dalam computer dan laptop ataupun yang bisa dimainkan melalui perangkat tablet dan smartphone. Pernahkah anda mempunyai keinginan untuk membuat sebuah game sendiri akan tetapi anda berpikir bahwa proses pembuatan game sendiri itu sangat rumit dan membutuhkan perangkat yang mumpuni agar proses rendering game sempurna. Akan tetapi sebenarnya membuat game sangat mudah dan bahkan lebih menyenangkan daripada saat kita memainkannya karena selama proses pembuatan game kita mengatur sendiri ide dan kreativitas ke dalam jalan cerita permainan. Banyak sekali alat bantu untuk membuat game yang biasa disebut game engine guna memudahkan pembuatan game dan dapat kita temukan dengan mudah di internet.

Salah satu game engine yang paling sering digunakan oleh para developer game yaitu bernama Unity. Game engine ini dikembangkan oleh Unity Technologies dimana diluncurkan pertama kali pada tahun 2005. Unity dapat menciptakan sebuah program interaktif baik 2D maupun 3D. yang terpenting, Unity memungkinkan anda untuk menggabungkan asset dan pengkodean skrip C# untuk membuat game. Dalam proses pembuatan game dengan game engine Unity, anda akan menggunakan Unity's Asset Store untuk mengunduh resource art dan code tambahan.

Di dalam buku ini akan dijelaskan cara membuat dan mengembangkan video game dengan 3D Unity yang akan dijelaskan mulai dari pengenalan setup unity sampai mempublikasikan dan mempromosikan game yang telah dibuat. Jika anda membaca dengan teliti serta memahaminya anda akan siap untuk mebuat game asli anda sendiri, baik sebagai pengembang indie, kontributor tim kecil, atau bahkan sebagai karyawan pada umumnya di perusahaan video game.

Semarang, Mei 2022
Penulis

Dr. Mars Caroline Wibowo, ST, M.Mm.Tech.

DAFTAR ISI

BAB 1
PENGENALAN DAN SET UP UNITY

1.1 MENGAPA UNITY BEGITU HEBAT?

Unity adalah game engine berkualitas profesional yang digunakan untuk membuat video game yang menargetkan berbagai platform. Itu adalah jawaban yang cukup lugas untuk pertanyaan langsung "Apa itu Unity?" Namun, apa sebenarnya arti jawaban itu, dan mengapa Unity begitu hebat?

Unity sebagai Game Engine

Pada dasarnya, *Unity* adalah *Game engine* yang telah berkembang menjadi IDE/*Rapid Development Tool*. Sedangkan *Game Engine* adalah sejumlah besar kode yang menangani dan membuat game memiliki fungsi, termasuk fisika untuk rendering, pencahayaan, fungsi kamera dasar, dan banyak lagi lainnya. *Unreal Engine* adalah contoh lain dari game engine CryENGINE 3. Sedangkan masih ada banyak lainnya termasuk Torsi, Lumberyard, Ogre3D, Blender, JavaFX, dan lain sejenisnya. Jika Anda memprogram game sepenuhnya dari awal tanpa menggunakan game engine yang sudah ada sebelumnya, Anda perlu mengkodekan setiap detail masing-masing, yang berarti banyak developeran bahkan sebelum Anda mulai menambahkan level. Kembali pada masa ZX Spectrum dan Amstrad, game engine bisa jauh lebih sederhana. Sebagian besar sprite terbuat dari sekitar 50 piksel. Kompleksitas permainan saat ini akan membuat satu orang tidak mungkin melakukannya sendiri jika bukan karena adanya solusi seperti Unity.

Unity juga cross-compatible, artinya dapat bertindak seperti jembatan antara kode Anda dan perangkat apa pun yang Anda targetkan. Kompilasi game Anda mengompresi semua aset dan mengubahnya menjadi format file yang tepat untuk ditambahkan ke masing-masing platform distribusi. Singkatnya, Unity menangani semua hal di balik layar untuk Anda dan memungkinkan Anda mengembangkan game yang hebat tanpa khawatir tentang menciptakan kembali roda atau tentang bagaimana cahaya harus membiaskan melalui berbagai bahan. Seolah-olah hukum alam semesta telah dibuat, dan yang harus Anda lakukan hanyalah memenuhinya. Unity kemudian menangani bagian akhir yang diperlukan untuk mengubah dunia Anda menjadi game nyata yang siap untuk didistribusikan.

Saat ini, sebagian besar developer—bahkan studio besar—menggunakan IDE yang sudah jadi seperti Unity atau Unreal. Kadang-kadang, sebuah game akan menggunakan mesin yang dibuat khusus (seperti permainan "ritme kekerasan" Thumper), tetapi ini biasanya memiliki mekanisme permainan yang unik yang menjamin pembuatan mesin khusus, dan mereka biasanya menghabiskan waktu lama dalam developeran. Karena Unity membuat hidup jauh lebih mudah tanpa memperkenalkan batasan utama, tidak ada alasan untuk tidak menggunakannya (atau opsi serupa seperti Unreal). "Melakukannya sendiri" hanya membuat tantangan menjadi jauh lebih sulit, tanpa manfaat nyata.

Unity sebagai IDE

Hal yang membuat Unity menjadi sangat berarti bagi developer yakni, Unity merupakan game engine sekaligus game maker dengan antarmuka yang ramah pengguna yang memungkinkan elemen dapat didrag dan dijatuhkan di sekitar layar dengan mudah. Untuk menggunakan bahasa yang lebih teknis, Unity bukanlah game maker melainkan IDE, yang merupakan akronim untuk *Integrated Development Environment*/Lingkungan Developeran Terpadu, yang pada dasarnya adalah seperangkat alat komprehensif yang digunakan untuk developeran dan diatur dengan antarmuka sederhana yang memungkinkan Anda melihat dan memodifikasi berbagai aspek berbeda dari satu program. Jika Anda ingin

membuat aplikasi Android tanpa Unity, Anda perlu menggunakan IDE lain—kemungkinan besar Android Studio, yang memungkinkan Anda melihat kode, folder aset, informasi debug, preview grafis, dan sebagainya. Dalam kasus Unity, Anda melihat tampilan scene Anda (biasanya level), hierarki semua elemen dalam scene tersebut (disebut GameObjects), detail untuk item mana pun yang Anda pilih untuk fokus, folder aset Anda, dan banyak lagi.

1.2 UNITY VS. UNREAL 4 (DAN LAINNYA)

Saya mengatakan tidak ada alasan untuk tidak menggunakan Unity, tetapi itu tidak sepenuhnya benar. Ada satu alasan bagus yang mungki akan Anda pilih untuk tidak menggunakan Unity. Mungkin perbandingan terdekat dapat ditarik dengan Unreal 4 yang memiliki banyak fitur serupa. Keduanya berfungsi penuh dengan sangat sedikit batasan, dan inimenjadikannya dua IDE paling populer untuk studio indie. Jadi mana platform yang lebih baik dari keduanya? Mengapa memilih Unity? Seperti biasa, jawabannya bergantung pada jenis game apa yang Anda rencanakan untuk dikembangkan.

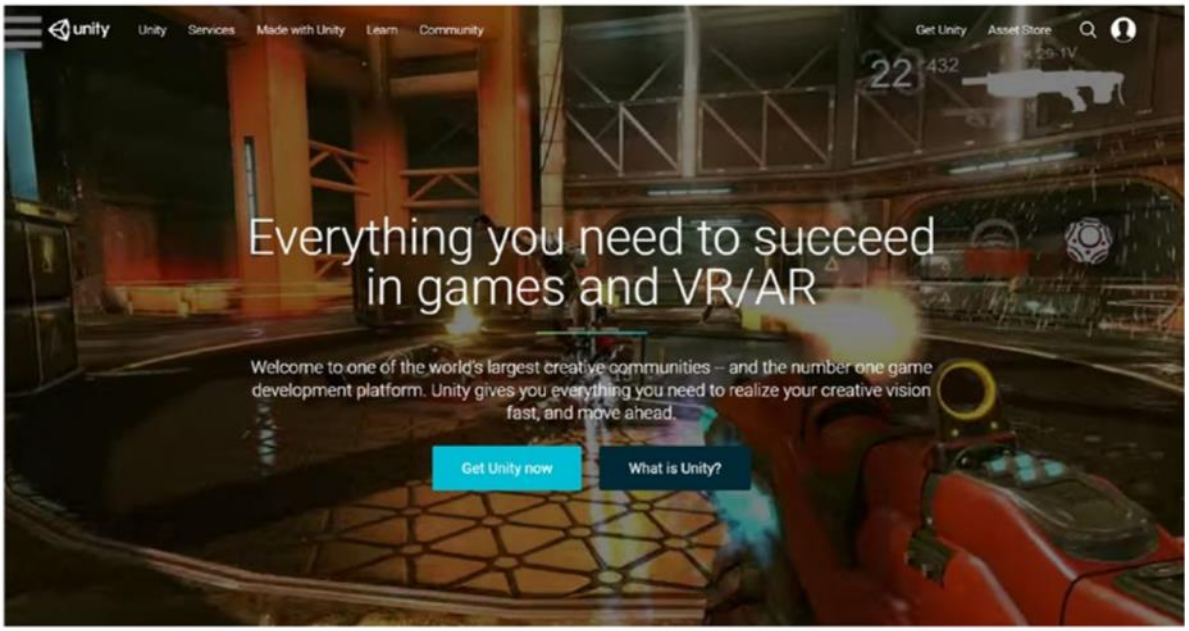
Meskipun tidak banyak di dalamnya — dan seringkali hanya karena preferensi personal — Unity bisa dibilang memiliki dukungan bawaan yang lebih baik untuk developeran game seluler dan developeran game. Unity adalah Game Engine paling populer di ponsel. Ini juga memastikan bahwa ada komunitas besar di luar sana untuk memberikan dukungan kepada pembuat konten, serta persediaan aset khusus yang hampir tak terbatas di Asset Store, yang dapat mempercepat developeran secara drastis.

Gambar 1.1 logo Unreal kiri dan Unity Kanan

Banyak orang juga menyukai *flow* Unity yang membuat Anda membangun dapat game menggunakan sistem entitas sederhana (GameObjects) dan komponen (skrip). Ini tentu masalah pendapat, tetapi cukup untuk mengatakan bahwa sebagian besar dari Unity sangat intuitif dan mudah digunakan sebagai pemula. Disisi lain, Unreal memiliki kurva belajar yang lebih curam dan tidak terorganisir dengan baik. Tetapi Unreal 4 memiliki kemampuan grafis yang lebih baik untuk mengembangkan game dengan tampilan triple-A. Termasuk juga open source, yang secara efektif berarti Anda dapat mengakses *source code* dan membuat perubahan pada mesin itu sendiri.

1.3 ORIGIN OF UNITY

Unity dikembangkan oleh sebuah perusahaan bernama Unity Technologies SF, yang didirikan pada tahun 2004 oleh David Helgason, Nicholas Francis, dan Joachim Ante di Kopenhagen, Denmark. Gambar dibawah ini menunjukkan situs web resmi pada saat penulisan.



Gambar 1.2 halaman utama Unity hari ini

Sebelumnya, ketiga developer tersebut menyebut diri mereka Over the Edge Entertainment dan telah mengerjakan sebuah game untuk Mac bernama *GooBall* yang memiliki gameplay yang mirip dengan *Super Monkey Ball*. Meskipun permainan gagal membuat percikan, tim menyadari bahwa mesin tersebut mungkin memiliki nilai bagi developer lain. Karena itu, mereka kemudian mengumumkan Unity 3D untuk OS X di pameran dagang WWDC 2006. Sejak itu, Unity mengalami banyak iterasi dan developeran hingga sekarang jauh lebih komprehensif dalam hal platform didukung dan fitur yang dicakup. Versi 1.1 datang dengan dukungan game maker untuk Microsoft Windows dan browser, bersama dengan dukungan untuk plugin C/C++. Versi 2.0 menambahkan dukungan untuk Microsoft DirectX, dan pada tahun 2008 Unity iPhone dirilis. Versi 3.0, dirilis pada 2010, ini merupakan langkah besar lainnya, karena tim ingin menjalankan program di Windows, yang mengharuskannya untuk dibangun kembali dari awal. Jadi versi 3.0 menggabungkan Windows, iPhone, serta dukungan untuk Wii dan berbagai platform lain yang sebelumnya hanya didukung oleh editor mandiri yang terpisah. Sekarang nama Unity akhirnya masuk akal, dan ini juga mendukung untuk Android.

Unity versi 4.3 melihat pembaruan penting lainnya: penyertaan dukungan 2D out-of-the-box dengan Unity2D. Sampai titik ini, developer pada dasarnya harus "meretas" IDE untuk mendukung 2D dengan menggunakan sudut kamera tetap dan menambahkan tekstur ke bidang datar untuk membuat latar belakang. Sekarang, pembuat konten dapat dengan lebih cepat dan mudah membuat game 2D yang sesungguhnya menggunakan sprite dan metode konvensional lainnya. Menurut Unity Technologies, Unity 5.0-lah yang akan menjadi rilis terbesar dan terpenting, dengan kinerja yang lebih baik di seluruh papan dan pembaruan besar untuk sistem animasi, mixer audio, shader, dan banyak lagi. Dengan demikian, banyak orang menyebut Unity sebagai Unity5 sekarang. Versi terbaru Unity pada saat penulisan ini adalah 5.5.0. Ini memiliki sejumlah perbaikan untuk Android khususnya yang harus meningkatkan kinerja.

Bagaimana jika Anda Memiliki Versi Unity yang Lebih Baru?

Penting untuk dicatat bahwa Unity terus mengembangkan platformnya dan menambahkan fitur dan peningkatan baru. Jadi, pada saat Anda membaca buku ini, Anda mungkin menemukan bahwa beberapa elemen akan berbeda dari yang dijelaskan di sini.

Mungkin Anda membaca dari masa depan yang jauh dan Anda menggunakan Unity 200. Namun, kemungkinan besar, perubahan apa pun yang Anda temui mungkin akan kecil. Tentu saja, developer middleware seperti Unity Technologies bekerja keras untuk menghindari kerusakan kode dengan pembaruan di masa mendatang, dan itu berarti sebagian besar fungsi dasar masih tetap berfungsi dengan baik. Namun dalam beberapa kasus yang jarang terjadi, sebaris kode mungkin disorot di Unity dan dideskripsikan sebagai tidak digunakan lagi. Itu berarti itu didukung tetapi tidak disarankan. Jika Anda memperhatikan ini, pencarian Google cepat akan membantu Anda menemukan cara baru yang benar untuk menangani fungsi tersebut.

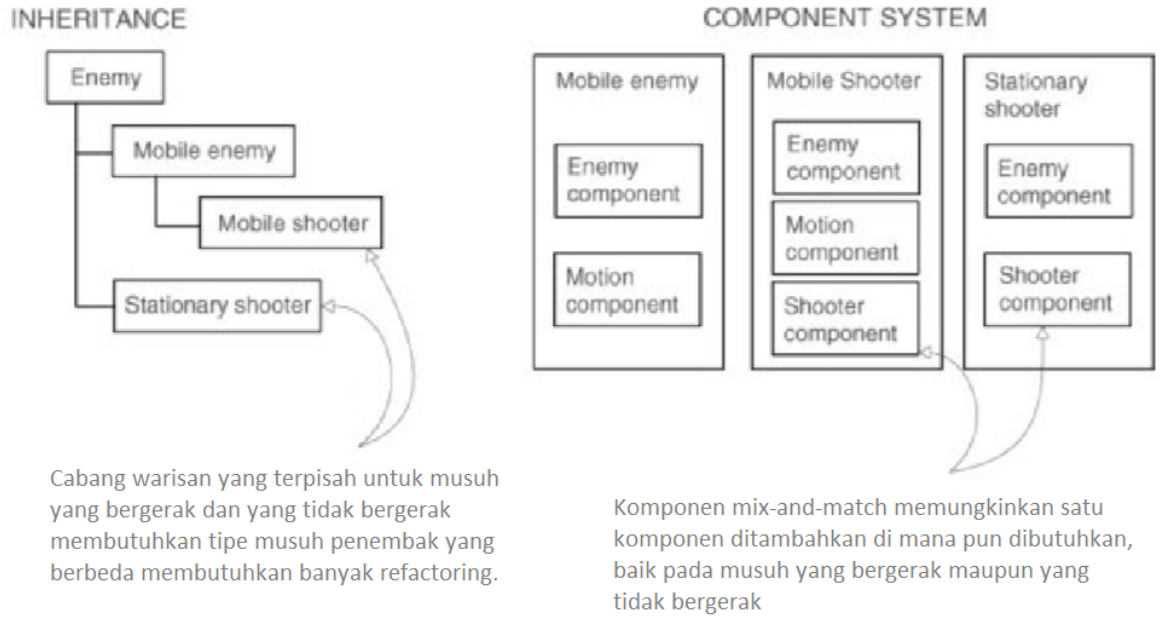
1.4 KEKUATAN DAN KEUNGGULAN UNITY

Game engine menyediakan banyak fitur yang berguna di banyak permainan yang berbeda, jadi permainan yang diimplementasikan menggunakan mesin itu mendapatkan semua fitur tersebut sambil menambahkan aset seni khusus dan kode permainan khusus untuk permainan itu. Unity memiliki simulasi fisika, peta normal, oklusi ambien ruang layar (*screen space ambient occlusion/*SSAO), bayangan dinamis...dan masih banyak lagi. Banyak game engine membanggakan fitur-fitur seperti itu, tetapi Unity memiliki dua keunggulan utama dibandingkan alat developeran game mutakhir lainnya yang serupa: alur kerja visual yang sangat produktif, dan dukungan lintas platform tingkat tinggi.

Alur kerja visual adalah desain yang cukup unik, berbeda dari kebanyakan lingkungan developeran game lainnya. Sementara alat developeran game lainnya sering kali merupakan campuran rumit dari bagian-bagian berbeda yang harus dipermasalahkan, atau mungkin perpustakaan pemrograman yang mengharuskan Anda untuk menyiapkan lingkungan developeran terintegrasi (*integrated development environment*/IDE), rantai pembangunan, dan yang lainnya, alur kerja developeran di Unity adalah berlabuh oleh editor visual yang canggih. Editor digunakan untuk menata scene dalam game Anda dan untuk menyatukan aset seni dan kode menjadi objek interaktif. Keunggulan editor ini adalah memungkinkan game berkualitas profesional dibuat dengan cepat dan efisien, memberikan alat developer untuk menjadi sangat produktif sambil tetap menggunakan daftar ekstensif teknologi terbaru dalam video game.

Catatan, Sebagian besar alat developeran game lain yang memiliki editor visual sentral juga dibebani dengan dukungan skrip yang terbatas dan tidak fleksibel, tetapi Unity tidak mengalami kerugian itu. Meskipun semua yang dibuat untuk Unity pada akhirnya melewati editor visual, antarmuka inti ini melibatkan banyak proyek penautan ke kode khusus yang berjalan di game engine Unity. Itu tidak berbeda dengan menautkan di kelas dalam pengaturan proyek untuk IDE seperti Visual Studio atau Eclipse. Pemrogram berpengalaman tidak boleh mengabaikan lingkungan developeran ini, salah mengartikannya sebagai game maker klik-bersama dengan kemampuan pemrograman terbatas!

Editor sangat membantu untuk melakukan iterasi cepat, mengasah permainan melalui siklus pembuatan prototipe dan pengujian. Anda dapat menyesuaikan objek di editor dan memindahkan berbagai hal bahkan saat game sedang berjalan. Plus, Unity memungkinkan Anda untuk menyesuaikan editor itu sendiri dengan menulis skrip yang menambahkan fitur dan menu baru ke antarmuka.



Gambar 1.3 Warisan vs. komponen

Selain keunggulan produktivitas editor yang signifikan, kekuatan utama lainnya dari perangkat Unity adalah dukungan lintas platform tingkat tinggi. Unity tidak hanya multiplatform dalam hal target penerapan (Anda dapat menerapkan ke PC, web, seluler, atau konsol), tetapi multiplatform dalam hal alat developeran (Anda dapat mengembangkan game di Windows atau Mac OS). Sifat platformagnostik ini sebagian besar karena Unity dimulai sebagai software khusus Mac dan kemudian di-porting ke Windows. Versi pertama diluncurkan pada tahun 2005, tetapi sekarang Unity mencapai versi utama kelima (dengan banyak pembaruan kecil yang sering dirilis). Awalnya, Unity hanya mendukung Mac untuk developeran dan penerapan, tetapi dalam beberapa bulan Unity telah diperbarui untuk bekerja di Windows juga. Versi yang berurutan secara bertahap menambahkan lebih banyak platform penyebaran, seperti pemutar web lintas platform pada tahun 2006, iPhone pada tahun 2008, Android pada tahun 2010, dan bahkan konsol game seperti Xbox dan PlayStation. Baru-baru ini mereka telah menambahkan penerapan ke WebGL, kerangka kerja baru untuk grafik 3D di browser web. Beberapa game engine mendukung target penerapan sebanyak Unity, dan tidak ada yang membuat penerapan ke banyak platform menjadi begitu sederhana. Sementara itu, selain kekuatan utama ini, manfaat ketiga dan lebih halus datang dari sistem komponen modular yang digunakan untuk membuat objek game. Dalam sistem komponen, "komponen" adalah paket fungsionalitas yang dipadupadankan, dan objek dibangun sebagai kumpulan komponen, bukan sebagai hierarki kelas yang ketat. Dengan kata lain, sistem komponen adalah pendekatan yang berbeda (dan biasanya lebih fleksibel) untuk melakukan pemrograman berorientasi objek, di mana objek game dibangun melalui komposisi daripada pewarisan.

Dalam sistem komponen, objek ada pada hierarki datar dan objek yang berbeda memiliki koleksi komponen yang berbeda, daripada struktur pewarisan di mana objek yang berbeda berada di cabang pohon yang sama sekali berbeda. Pengaturan ini memfasilitasi pembuatan prototipe cepat, karena Anda dapat dengan cepat mencampur dan mencocokkan komponen yang berbeda daripada harus memfaktorkan ulang rantai pewarisan saat objek berubah.

Meskipun Anda dapat menulis kode untuk mengimplementasikan sistem komponen khusus jika tidak ada, Unity sudah memiliki sistem komponen yang kuat, dan sistem ini bahkan

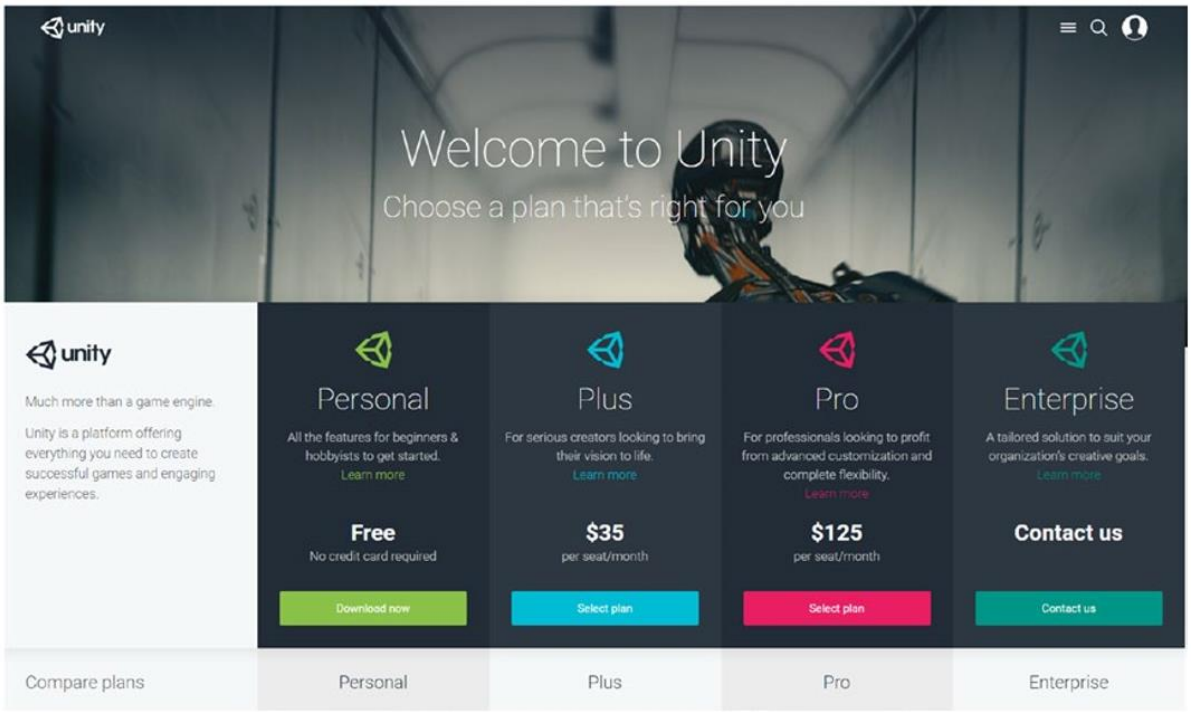
terintegrasi secara mulus dengan editor visual. Daripada hanya dapat memanipulasi komponen dalam kode, Anda dapat melampirkan dan melepaskan komponen dalam editor visual. Sementara itu, Anda tidak terbatas hanya membangun objek melalui komposisi; Anda masih memiliki opsi untuk menggunakan pewarisan dalam kode Anda, termasuk semua pola desain praktik terbaik yang muncul berdasarkan pewarisan.

Unity memiliki banyak keunggulan yang menjadikannya pilihan tepat untuk mengembangkan game dan saya sangat merekomendasikannya, tetapi saya akan lalai jika tidak menyebutkan kelemahannya. Secara khusus, kombinasi editor visual dan pengkodean yang canggih, meskipun sangat efektif dengan sistem komponen Unity, tidak biasa dan dapat menimbulkan kesulitan. Dalam scene yang kompleks, Anda dapat kehilangan jejak objek mana dalam scene yang memiliki komponen tertentu yang terpasang. Unity memang menyediakan fungsionalitas pencarian untuk menemukan skrip terlampir, tetapi pencarian itu bisa lebih kuat; terkadang Anda masih menghadapi situasi di mana Anda perlu memeriksa semua yang ada di scene secara manual untuk menemukan hubungan skrip. Ini tidak sering terjadi, tetapi ketika itu terjadi, itu bisa membosankan.

Kerugian lain yang dapat mengejutkan dan membuat frustrasi bagi programmer berpengalaman adalah Unity tidak mendukung penautan di pustaka kode eksternal. Banyak perpustakaan yang tersedia harus disalin secara manual ke setiap proyek di mana mereka akan digunakan, sebagai lawan dari referensi satu lokasi pusat bersama. Kurangnya lokasi pusat untuk perpustakaan dapat membuat canggung untuk berbagi fungsionalitas antara beberapa proyek. Kerugian ini dapat diatasi melalui penggunaan sistem kontrol versi yang cerdas, tetapi Unity tidak mendukung fungsi ini di luar kotak. Kelemahan ketiga berkaitan dengan bekerja dengan Prefab. Prefab adalah konsep khusus untuk Unity dan dijelaskan dalam bab 3; untuk saat ini, yang perlu Anda ketahui adalah bahwa prefab adalah pendekatan yang fleksibel untuk mendefinisikan objek interaktif secara visual. Konsep prefab sangat kuat dan unik untuk Unity (dan ya, itu terkait dengan sistem komponen Unity), tetapi bisa sangat aneh untuk mengedit prefab. Mengingat prefab adalah bagian yang sangat berguna dan sentral dari bekerja dengan Unity, saya berharap versi mendatang meningkatkan alur kerja untuk mengedit prefab.

Lisensi

Seolah-olah Unity belum cukup mengagumkan, bagian terbaik dari semua ini adadalah "Unity sepenuhnya gratis untuk digunakan". Di masa depan, saat ambisi Anda tumbuh, Anda mungkin membutuhkan fitur tambahan atau penghasilan di atas ambang IDR 50juta, tetapi kebanyakan pemula akan baik-baik saja kecuali permainan mereka berkembang pesat. Pada dasarnya, ada beberapa jenis akun berbeda yang dapat Anda buat, seperti yang ditunjukkan pada gambar dibawah ini, dengan masing-masing harga berbeda dan memiliki batasan berbeda.



Gambar 1.4 Bagi kebanyakan orang, paket personal gratis sudah lebih dari cukup

1.5 PERSONAL

Akun gratis yang dapat Anda gunakan adalah akun Personal. Akun ini tidak dikenakan biaya apa pun, dan Unity bahkan tidak mengambil royalti. Satu-satunya batasan adalah Anda tidak diperbolehkan menghasilkan lebih dari IDR1000K per tahun dengan lisensi ini. Tidak apa-apa, karena Anda dapat dengan mudah beralih ke salah satu perjanjian lisensi lainnya setelah Anda mulai menyerahkan sejumlah besar. Ingatlah bahwa batasan IDR 1000K ini juga berlaku untuk uang yang terkumpul, artinya jika Anda mengumpulkan uang di Kickstarter dan jumlahnya melebihi IDR1000K, maka itu diperhitungkan. Aturan ini juga berlaku untuk keuntungan yang berasal dari sumber lain, termasuk iklan pay per click atau pembelian dalam aplikasi. Ada juga beberapa fitur dan batasan yang hilang dengan akun personal. Misalnya, pembuat konten yang menggunakan akun Personal diharuskan untuk menampilkan layar splash Unity saat game boot (menunjukkan kepada pengguna bahwa Anda membuat game di Unity), dan Anda tidak akan memiliki akses ke developer analytics secara real time atau Cloud Build. Untuk game online multigame, akun Personal hanya mengizinkan 20 player sekaligus.

Unity Plus

Unity Plus saat ini berharga IDR525K per bulan dan menghilangkan splash screen sambil meningkatkan batas pendapatan hingga IDR3200K per tahun fiskal. Ini juga menambahkan dukungan dan fitur tambahan yang mungkin berguna untuk developer yang lebih besar, seperti dukungan untuk 50 pengguna bersamaan dan kit aset yang didiskon.

Unity Pro

Unity Pro menaikkan harga hingga IDR1.875Kper bulan dan sepenuhnya menghapus batas pendapatan, yang berarti Anda bisa menjadi sangat kaya tanpa membayar apa pun jika diinginkan. Ini juga menyediakan sejumlah layanan pro, termasuk dukungan 200 player bersamaan, lebih banyak analisis dan kinerja, dukungan untuk tim besar, dan banyak lagi.

Unity Enterprise

Akhirnya, keanggotaan Enterprise memungkinkan Anda dapat memilih fitur untuk membuat platform developeran yang dirancang khusus untuk kebutuhan independen Anda. Ini adalah opsi paling premium, dan harganya sebenarnya tidak tercantum tapi menunjukkan

bahwa harganya sangat mahal. Singkatnya, itu tidak akan menjadi sesuatu yang perlu dikhawatirkan oleh kebanyakan orang yang membaca buku ini untuk sementara waktu. Faktanya, untuk sebagian besar orang, akun Personal dasar akan lebih dari cukup dan harus menyediakan semua fitur dan fleksibilitas yang Anda butuhkan.
Perhatikan, harga yang saya sebutkan diatas merupakan harga pada saat penulisan, dan tentu saja dapat berubah. Saya sangat merekomendasikan Anda untuk menghemat uang Anda dengan cara langganan setiap tahun karena akan jauh lebih hemat jika dibanding dengan langganan/pembelian bulanan.

1.6 MENGUNDUH UNITY DAN KOMPONEN YANG DIPERLUKAN

Mari kita mulai dengan hal-hal teknis. Anda tentu saja perlu mengatur Unity dan menjalankannya di komputer Anda. Ini cukup mudah untuk sebagian besar, tetapi ingatlah bahwa Anda juga memerlukan beberapa software tambahan juga. Secara khusus, Anda harus mengunduh dan menginstal yang berikut:

- Unity
- Android SDK (bersama dengan Android Studio 2.3)
- Java JDK
- Visual Studio

Android SDK adalah Android Software Development Kit. Ini adalah seperangkat alat software yang disediakan oleh Google yang dapat bertindak seperti jembatan dalam mengakses fungsionalitas hardware Anda. Dengan kata lain, ini menyediakan kode sumber yang dibutuhkan Unity untuk membuat game Anda kompatibel di platform Android. Ini juga berisi beberapa alat lain yang mungkin berguna bagi Anda di masa mendatang—termasuk emulator yang memungkinkan Anda menguji aplikasi Android langsung di desktop. Anda juga perlu memilih untuk menginstal Dukungan Build Android, tetapi Anda melakukannya melalui Unity Installer, jadi tidak perlu mengunduhnya secara terpisah. Java JDK adalah kit developeran lain, kali ini untuk Java. Inilah yang dibutuhkan komputer Anda untuk mendukung developeran Java,

Gambar 1.5 logo Java

Visual Studio adalah apa yang akan Anda gunakan untuk menangani pemrograman aktual di Unity. Saat Anda mulai menulis skrip, Anda akan mengeditnya di jendela Visual Studio terpisah—namun kita bisa mengkhawatirkannya nanti. Perhatikan bahwa Anda sebenarnya tidak memerlukan Visual Studio, dan Anda dapat menggunakan opsi alternatif seperti MonoDevelop. Tetapi Visual Studio tentu saja merupakan pilihan yang lebih disukai dari keduanya dan akan membuat hidup Anda sedikit lebih mudah.

Mengunduh Unity

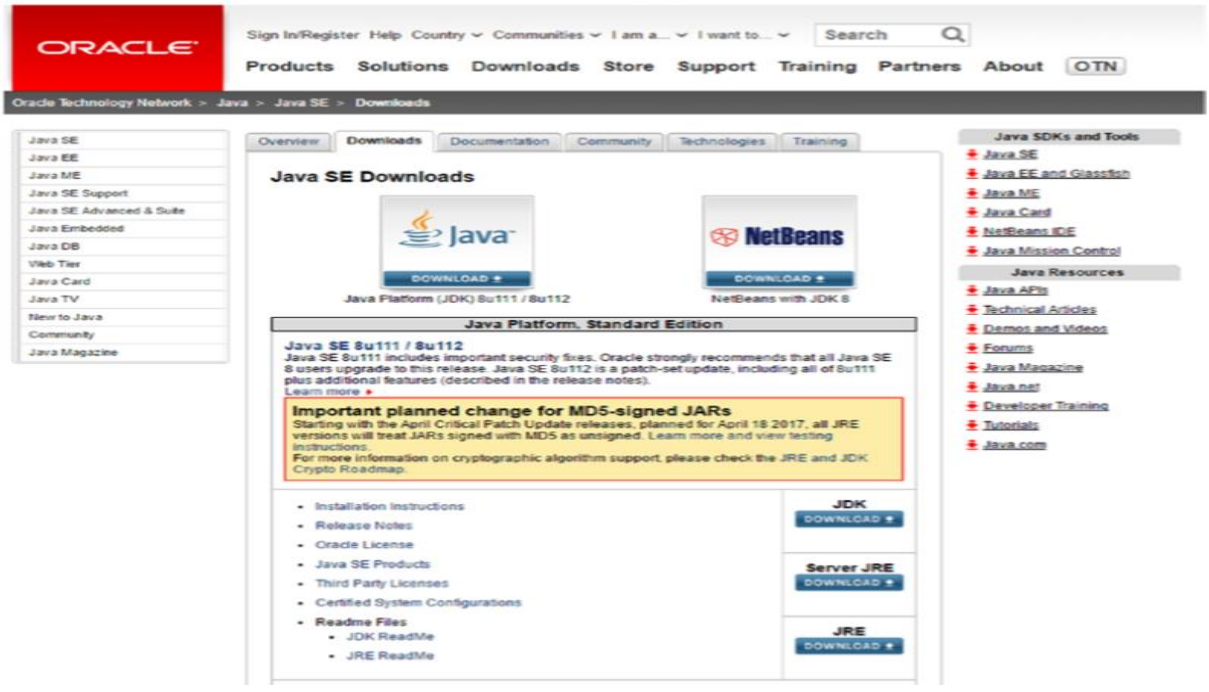
Untuk mulai mengunduh Unity, pertama-tama buka Unity3D.com (https://unity3d.com) dan klik Get Unity Now. Anda kemudian dapat memilih paket yang Anda inginkan (Personal, dalam banyak kasus), dan di sana Anda cukup mengeklik Download Now, lalu Download Installer di halaman berikutnya. Setelah Anda melakukannya, unduhan akan dimulai.

Gambar 1.6 Disinillah Anda akan menemukan Unity

Anda tidak perlu mengunduh Visual Studio secara terpisah, karena Anda dapat melakukannya melalui Unity Downloader. Itu menghemat sedikit waktu dan masalah.

Mengunduh Java JDK

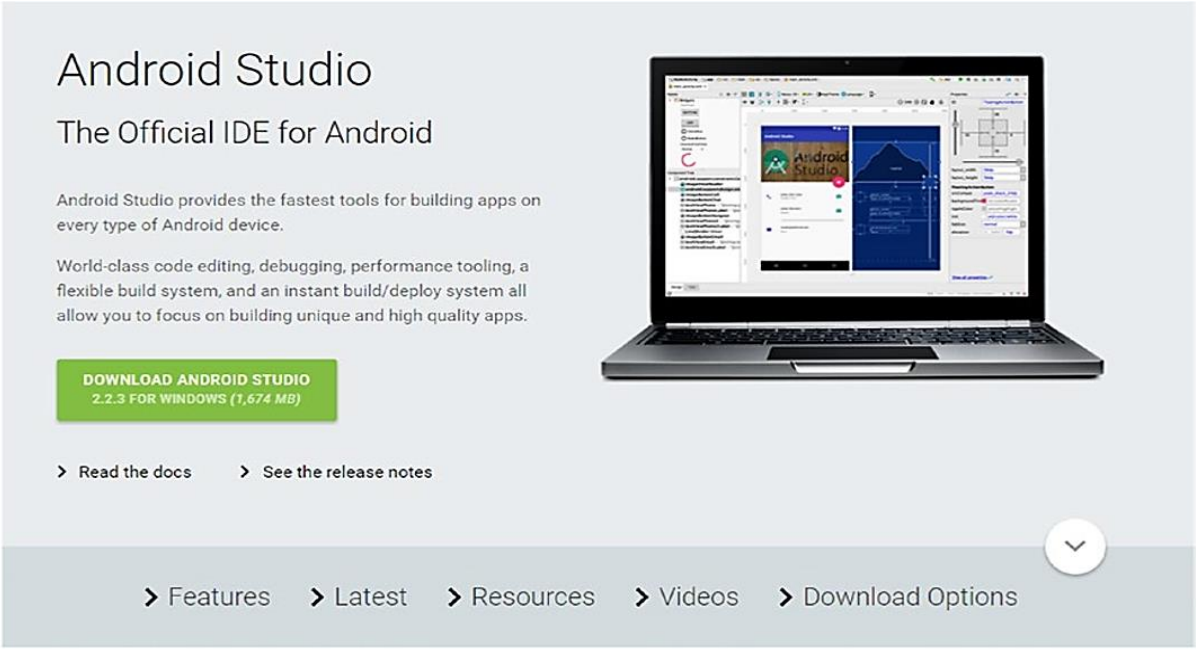
Jika Anda merasa sedikit kewalahan saat ini, jangan khawatir. Setelah Anda menginstal semua bagian ini, mereka akan beroperasi sendiri di latar belakang dan Anda dapat melupakan semuanya. Ini adalah prosedur satu kali (kecuali Anda mengatur Unity di komputer baru) yang tidak perlu Anda khawatirkan lagi. Java JDK adalah yang memungkinkan komputer Anda dan oleh karena itu Unity memahami dan menafsirkan kode Java. Buka www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html dan unduh Java SE Development Kit. Pastikan Anda memilih versi x64 jika komputer Anda mendukungnya.



Gambar 1.7 halaman unduh Java JDK

Mengunduh Android SDK

Untuk mengunduh Android SDK, buka https://developer.android.com/studio/index.html. Klik tombol Unduh Android Studio, terima syarat dan ketentuan, dan biarkan unduhan dimulai. Jika Anda menjalankan komputer dengan hard drive yang relatif kecil, klik Opsi Unduhan. Android Studio adalah IDE Android standar yang digunakan untuk membuat aplikasi biasa, dan Anda sebenarnya tidak memerlukan ini untuk menggunakan Unity. Untuk menyimpan sendiri file besar yang tidak perlu, Anda cukup mengunduh alat baris perintah dan kemudian menggunakan SDKManager yang disertakan untuk mengunduh sisa SDK. Anda akan menemukan petunjuk tentang cara melakukannya di situs web Android Studio.

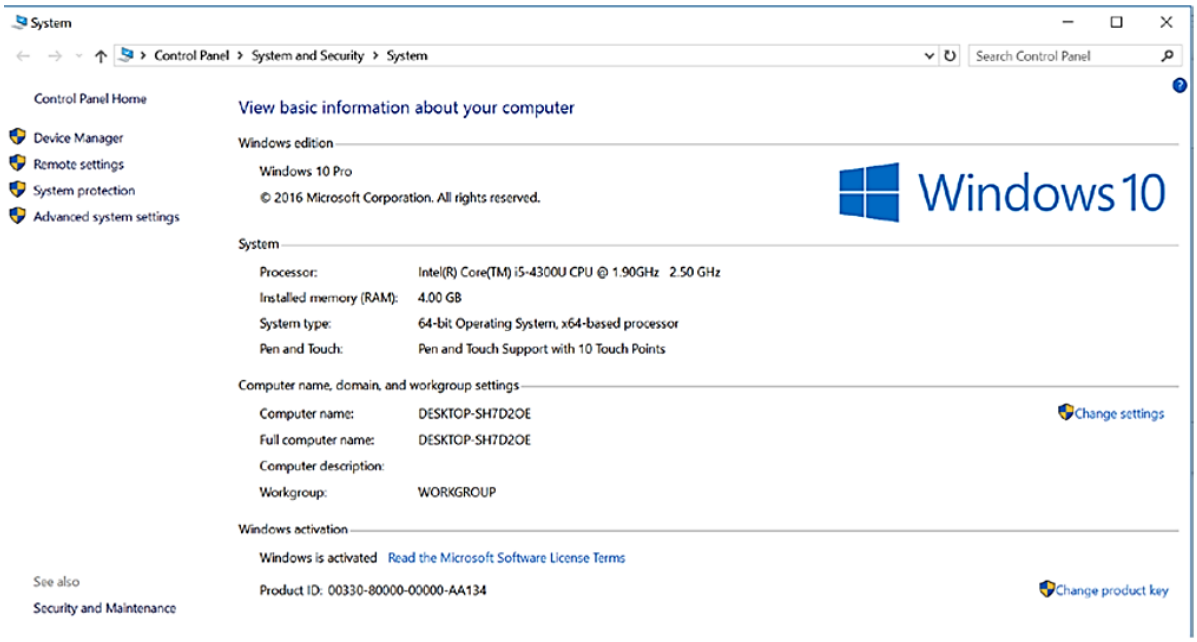


Gambar 1.8 Paling mudah untuk mengunduh Android Studio dan Android SDK secara bersamaan.

Ini lebih rumit, dan Android Studio tentu saja merupakan hal yang berguna untuk dimiliki, jadi jika Anda mampu membeli ruang, saya sarankan hanya menginstal seluruh paket setelah Anda memiliki Java.

Menginstal 3D Unity

Setelah semuanya diunduh, mari emulai instalasi. Mulailah dengan menemukan file tempat Anda menyimpannya, lalu klik dua kali satu per satu secara bergantian. Tidak masalah yang mana yang Anda mulai, dan semua langkah yang diperlukan dijelaskan di bagian ini.



Gambar 1.9 Surface Pro 3 ini adalah 64-bit

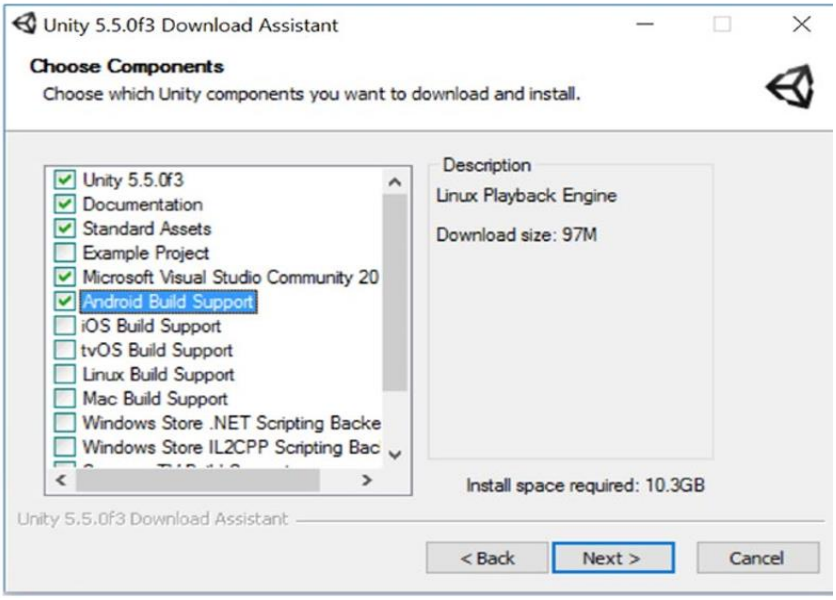
Unity

Ketika Anda mengklik dua kali Unity Installer dan menerima persyaratan perjanjian lisensi, Anda kemudian harus memilih versi Unity yang akan diunduh: 64-bit atau 32-bit. Versi terbaik akan tergantung pada versi Windows Anda karena tidak semua komputer mendukung 64-bit. Untuk memeriksa apakah Anda memiliki prosesor x64- atau x32-bit, buka This PC, klik kanan, lalu buka Properties. Jika komputer Anda 64-bit, itu akan membaca: "Sistem Operasi 64-bit, prosesor berbasis x64."

Jika perangkat Anda mendukung 64-bit, maka pilih yang 64-bit karena ini akan mengaktifkan dukungan konsol dan fitur lainnya. Layar berikutnya yang akan Anda tampilkan adalah layar Download Assistant. Ini memungkinkan Anda memilih komponen mana yang ingin Anda instal bersama dengan Unity dan ini menunjukkan kepada Anda berapa banyak ruang yang Anda butuhkan di komputer Anda. Di sebelah kiri, banyak opsi berbeda yang dicentang, dan akan ada beberapa kotak yang tidak dicentang. Secara default, Anda akan menemukan yang berikut ini dipilih:

- Unity 5.5 0f3
- Dokumentasi
- Aset Standar
- Komunitas Microsoft Visual Studio 2015

Biarkan semua setting apa adanya, karena semuanya penting. Unity dan Documentation sudah cukup jelas, sedangkan Microsoft Visual Studio Community 2015 memungkinkan Anda membuat dan mengedit skrip di game Anda (dibahas sebelumnya). Opsi Aset Standar tidak wajib tetapi akan sangat berguna—ini adalah banyak pilihan skrip, sprite, model 3D, tekstur, dan banyak lagi yang sudah dibuat sebelumnya yang bebas Anda gunakan dalam gim Anda sendiri. Jika Anda memiliki ruang, maka menambahkan ini adalah ide yang sangat bagus. Jika Anda merasa ingin mengembangkan untuk salah satu platform lain yang terdaftar (dan Anda memiliki ruang), lanjutkan dan centang opsi itu juga.

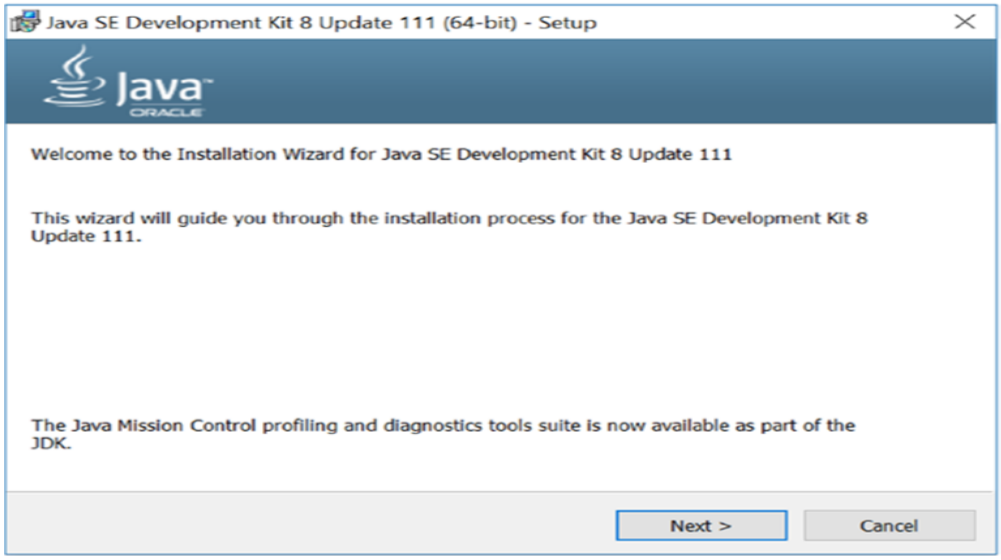


Gambar 1.10 Centang kotak ini untuk kelancaran nanti

Namun, ada satu lagi yang tidak dipilih secara default, yang ingin Anda pastikan Anda dapatkan: Dukungan Android Build. Inilah yang akan memastikan bahwa Anda dapat membuat file APK untuk diunggah ke Play Store dan akan diperlukan saat Anda menguji dan menyelesaikan proyek Anda. Pastikan itu dicentang. Anda juga akan melihat Dukungan iOS Build di sini, serta Dukungan Mac Build, opsi Windows Store, dan banyak lagi. Kabar baiknya adalah Anda dapat kembali dan menambahkan ini nanti jika perlu. Anda akan melihat berapa banyak ruang yang diperlukan (sekitar 10,3 GB pada saat penulisan), dan jika Anda memilikinya di hard drive Anda, Anda dapat melanjutkan dan mengklik Berikutnya lagi. Sekarang pilih di mana Anda ingin Unity diinstal di komputer Anda. Ini sepenuhnya masalah preferensi, tetapi buat catatan mental tentang itu. Terima beberapa persyaratan perjanjian lagi, tekan Next lagi, lalu tunggu Unity untuk menginstal. Pergi dan buatkan diri Anda secangkir kopi, karena ini cenderung memakan waktu cukup lama.

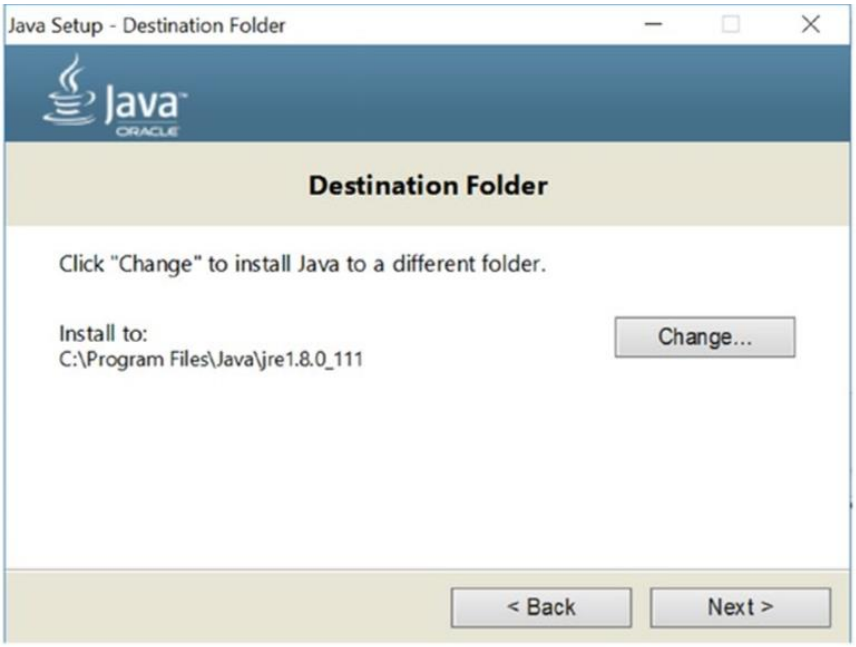
Java JDK

Menginstal Java JDK sangat sederhana. Cukup klik dua kali file dan klik Berikutnya dua kali, dan selesai



Gambar 1.11 Installer JDK

Setelah beberapa bilah kemajuan diisi, Anda akan diberikan opsi untuk memilih folder tujuan. Pada saat penulisan, defaultnya adalah C:\Program Files\Java\jre1.8.0_111 (Gambar 2-13). Tidak apa-apa untuk membiarkannya seperti ini, tetapi Anda mungkin ingin membuat catatan untuk nanti. Klik Berikutnya lagi, penginstal akan selesai, dan Anda akan siap untuk mengambil langkah berikutnya: menginstal Android SDK.



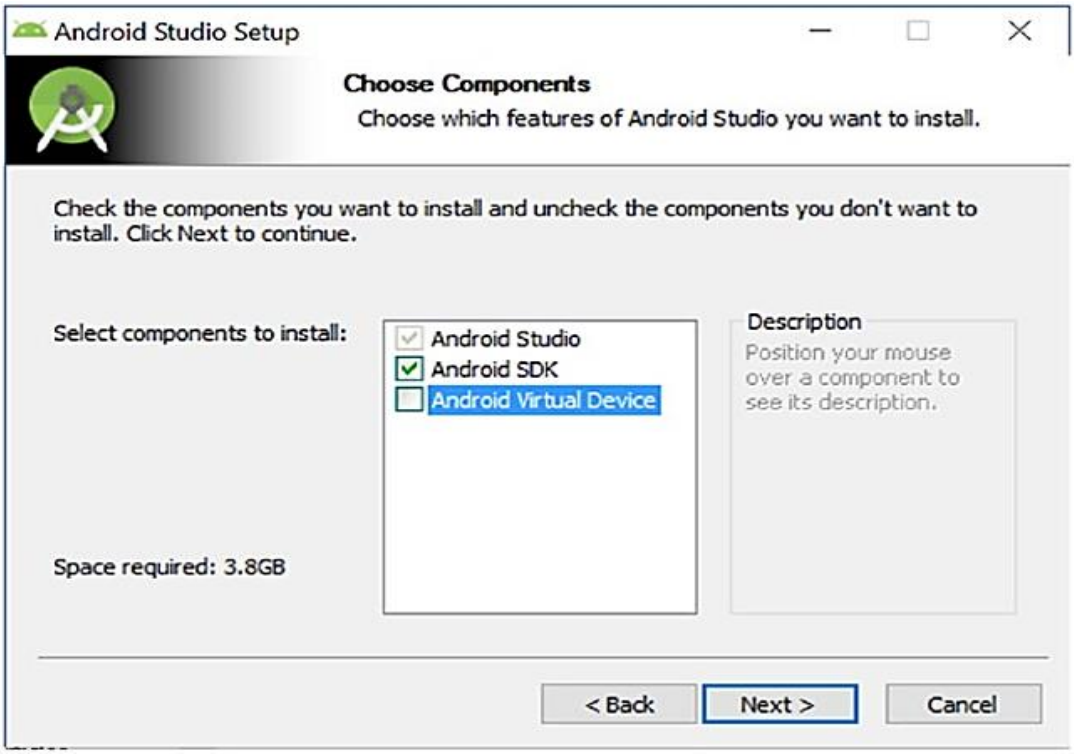
Gambar 1.12 Setel folder tujuan Anda

Android SDK

Terakhir, Anda perlu menginstal Android SDK. Untuk melakukannya, Anda menginstal Android-Studio-Bundle. Klik dua kali .exe yang Anda unduh sebelumnya, lalu klik Berikutnya di layar selamat datang (Gambar 2-14) untuk membuka rangkaian opsi pertama.

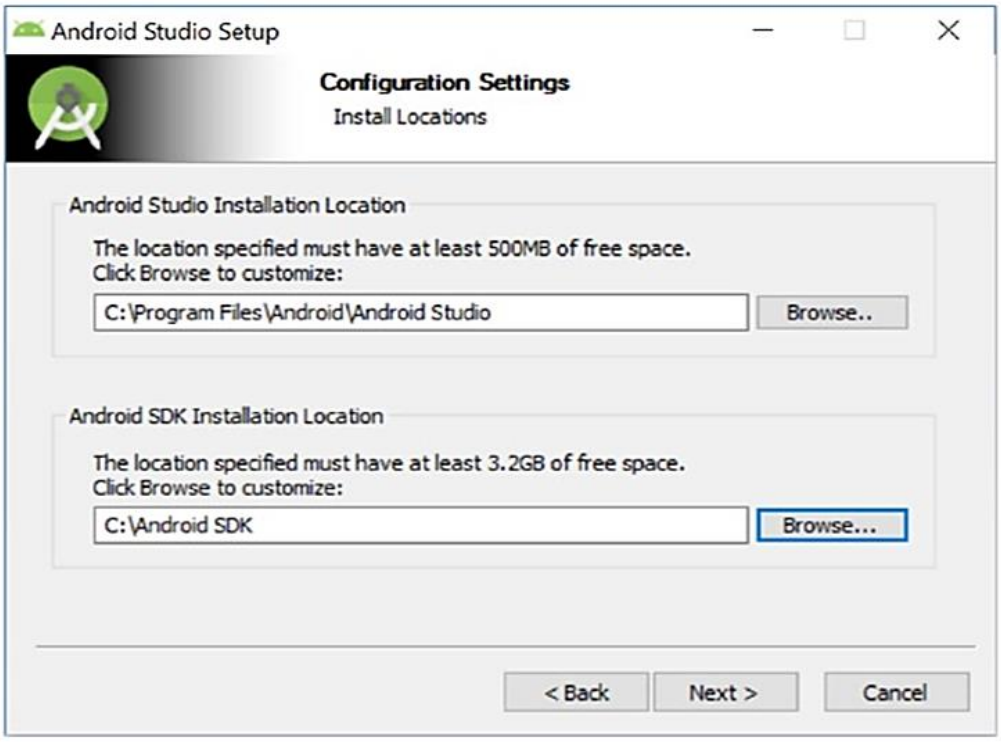
Gambar 1.13 Pemasang Android Studio

Di sini Anda akan memilih apa yang ingin Anda instal. Yang mengganggu, Anda tidak dapat membatalkan pilihan Android Studio (karena Google) tetapi Anda dapat memutuskan apakah Anda menginginkan Android SDK dan Perangkat Virtual.



Gambar 1.14 Putuskan apakah Anda memerlukan AVD

SDK adalah bit utama yang kita butuhkan, jadi pastikan untuk membiarkannya dicentang. Sementara itu, Perangkat Virtual Android adalah emulator yang dapat Anda gunakan untuk menjalankan aplikasi. Kecuali jika Anda membuat game puzzle yang sangat sederhana atau Anda memiliki rig game yang sangat kuat (di sini kita berbicara tentang dua GTX1080 dengan CPU 4 GHz), Anda mungkin tidak akan dapat menggunakan sebanyak ini untuk menguji Anda sepenuhnya. permainan yang direalisasikan. Anda dapat menguji langsung di perangkat Android Anda juga, jadi Anda mungkin ingin menghapus centang yang satu ini jika Anda ingin menghemat data 1 GB. Yang mengatakan, ini dapat berguna untuk tujuan pengujian lainnya (bereksperimen dengan ukuran layar, misalnya), jadi ini adalah panggilan Anda. Apa pun caranya, klik Berikutnya, lalu setujui syarat dan ketentuan. Pada layar berikutnya, Anda dapat memilih lokasi untuk menginstal SDK dan Android Studio. Jika yang terakhir tumpul (secara default, mungkin: C:\Users\rushd\AppData\Local\Android\sdk), maka cari tempat yang lebih sederhana untuk menginstalnya yang memiliki ruang kosong 3,2 GB+. Catat di mana ini, karena Anda akan membutuhkan jalurnya nanti. Saya telah memilih C:\ AndroidSDK, mengingat jalur tidak diizinkan untuk menyertakan spasi (mengganggu).



Gambar 1.15 Terakhir, atur folder tujuan Anda

Klik Next dan lagi pada layar berikutnya, dan kemudian instalasi akan dimulai.

1.7 HARDWARE DAN ALUR KERJA

Sementara semua hal ini diunduh, mari kita ambil jeda singkat ini untuk mempertimbangkan hardware terbaik untuk membuat game Anda dan mencapai penyiapan terbaik. Kabar baiknya adalah, PC Anda tidak perlu terlalu kuat untuk menangani Unity, tetapi Anda akan menginginkan sesuatu yang setidaknya cukup modern. Saya personal menjalankan Unity pada Dominator Pro GT72 6RE serta Surface Pro. Dominator Pro adalah laptop gaming VR-ready yang sangat baru dengan GPU GTX1070 dan tidak mengalami kesulitan apa pun dengan Unity. Surface Pro 3 jelas berada di ujung bawah dalam hal spesifikasi ideal untuk developeran Android dengan Unity. Meskipun saya tidak pernah tidak dapat melakukan apa pun, sistem menjadi agak panas saat saya mengkode, dan beberapa game 3D dapat melihat penurunan kecepatan bingkai.

Berdasarkan ini, saya akan mengatakan bahwa model Surface Pro 3 (i5) mewakili hampir semua spesifikasi minimum yang ingin Anda gunakan. Itu berarti:

- Prosesor 1,9 GHz (peningkatan turbo hingga 2,6 GHz)
- Grafis terintegrasi
- RAM 4GB

Namun, situs web Unity merekomendasikan yang lebih tinggi, dan menyarankan agar developer memiliki setidaknya beberapa bentuk kartu grafis khusus atau DX11 dengan kemampuan tingkat fitur 9.3. Lebih banyak RAM juga akan lebih disukai, terutama jika Anda ingin menggunakan software lain seperti Photoshop (atau alternatif gratis seperti Fusion, DaVinci, atau GIMP) untuk membuat gambar besar dan multitasking antar alat. Anda juga akan menginginkan spesifikasi yang lebih tinggi jika Anda berencana untuk menguji game Anda dengan emulator, meskipun Anda dapat berjuang dengan ini bahkan saat menggunakan mesin yang sangat mumpuni.

GPU adalah prosesor grafis yang digunakan untuk membuat scene 3D lebih cepat. Ini mungkin bukan masalah besar jika Anda mengembangkan sebagian besar dalam 2D (yang sebagian besar akan dikhususkan untuk buku ini) tetapi tentu saja tidak ada salahnya untuk memberi diri Anda pilihan. Demikian juga, akan berguna untuk bagian pemodelan 3D dengan Blender. Selain GPU, tambahan lain yang berguna adalah memiliki hard drive HDD dan SSD. SSD adalah solid state disk—alternatif untuk hard disk drive—dan sangat cepat tetapi umumnya agak lebih kecil daripada HDD yang lebih lama dan lebih lambat. Ini standar untuk mendapatkan SSD sekitar 128 GB atau 256 GB. Karena Unity, Android SDK, dan Android Studio semuanya memakan banyak ruang, mungkin berguna untuk menyimpan setidaknya beberapa file ini di HDD sambil menyimpan file OS dan game Anda di SSD untuk kecepatan. Sekali lagi, ini adalah preferensi, bukan persyaratan. Singkatnya, dimungkinkan untuk puas dengan PC kelas menengah tetapi tentu saja akan lebih baik menggunakan rig game dari beberapa jenis. Terlepas dari hal lain, memiliki rig yang mampu bermain game akan berguna untuk tujuan penelitian.

Membuat Battlestation Anda (Pengaturan Kerja)

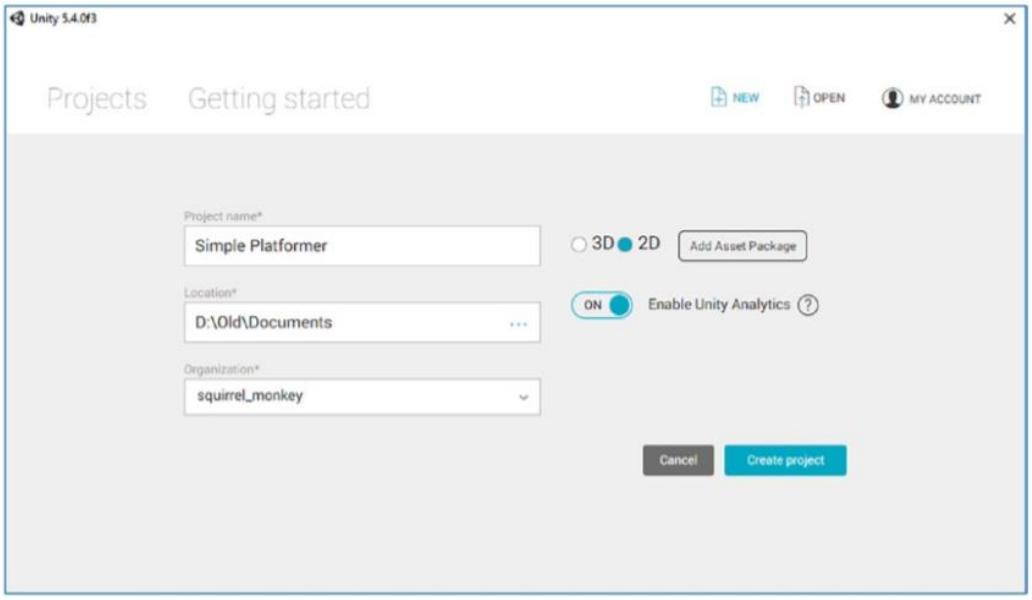
Penting juga untuk memikirkan pengaturan dan lingkungan kerja Anda yang sebenarnya karena ini dapat membuat perbedaan besar pada kenyamanan Anda selama developeran dan dapat menyelamatkan Anda dari sakit kepala di kemudian hari. Salah satu keuntungan besar, misalnya, adalah memiliki monitor besar dan bahkan mungkin layar ultrawide 21:9. Unity memiliki banyak jendela dan panel yang berbeda seperti yang akan kita lihat di Bab 3, dan merupakan keuntungan besar untuk dapat melihat semua ini secara bersamaan sehingga Anda dapat melakukan tugas dengan lebih mudah. Layar besar akan memberikan dorongan besar untuk multitasking Anda, membantu Anda menghindari ketegangan mata, dan meningkatkan imersi (menghilangkan gangguan).

Jika Anda lebih suka bekerja di laptop, pastikan laptop dengan layar lebih besar (18,3 inci atau lebih besar) dan memiliki tenaga kuda yang cukup untuk mengatasinya. Ada sesuatu yang bisa dikatakan untuk bekerja di kafe sambil minum kopi (menurut saya itu meningkatkan produktivitas), tetapi untuk itu Anda idealnya menginginkan laptop baru seperti Razer, Asus, Dell, HP, Lenovo, atau Toshiba. Surface Book atau MacBook juga akan melakukan pekerjaan dengan baik. Tentu saja, petunjuk penginstalan kami ditujukan untuk pengguna Windows, jadi jika Anda menggunakan Mac, Anda harus melalui proses yang sedikit berbeda untuk memulai dan menjalankannya. Anda tentu menginginkan keyboard yang nyaman, dan mouse yang presisi untuk mengetik, menyeret, dan melepas. Keyboard berkabel lebih disukai untuk meningkatkan daya tanggap saat menguji game di komputer. Jika Anda menghabiskan uang untuk membuat pengaturan terbaik, sesuatu seperti keyboard Corsair Gaming akan menyenangkan tidak hanya untuk mengetik tetapi juga untuk menguji dan memainkan game. Jika tidak, pastikan Anda memiliki kursi yang nyaman, idealnya ruang meja besar untuk menyebar dan mengatur catatan dan sketsa, dan ruangan yang sebebas mungkin dari gangguan. Jika Anda mengatur semua itu, Anda akan baik-baik saja untuk memulai dan Anda akan menemukan bahwa pekerjaan Anda berjalan semulus dan semenyenangkan mungkin. Jika Anda berencana melakukan banyak developeran, berinvestasi di ruang kerja yang baik adalah cara yang baik untuk membelanjakan uang Anda dan akan menghasilkan produk akhir yang lebih baik yang diselesaikan lebih cepat.

1.8 MEMULAI INSTALASI PROYEK PERTAMA ANDA

Pada titik ini, Anda harus memiliki Unity dan semua komponen penting yang Anda perlukan untuk menginstalnya di komputer Anda dan siap digunakan. Dan semoga Anda bersemangat untuk terjun dan memulai. Dalam hal ini, mari luncurkan Unity untuk pertama

kalinya dan lakukan sedikit penyiapan terakhir. Pertama, Anda harus masuk ke akun yang Anda buat di situs web. Jika Anda belum melakukannya, klik tautan (Buat Satu) dan Anda akan dibawa ke halaman yang relevan di mana Anda dapat mengaturnya. Anda kemudian dapat masuk menggunakan ID Unity baru Anda dan Anda harus mengonfirmasi bahwa perusahaan Anda menghasilkan kurang dari ambang batas jika Anda ingin tetap menggunakan akun gratis. Setelah Anda melewati layar itu, Anda akan disambut oleh jendela yang memungkinkan Anda memilih dari proyek yang ada atau memulai yang baru. Anda mungkin tidak memiliki apa-apa di sini untuk saat ini, mengingat ini adalah pertama kalinya Anda menggunakannya. Jadi, klik Baru dan pilih nama dan lokasi untuk proyek Anda (Gambar 2-19). Anda juga perlu memastikan bahwa Anda memilih 2D sehingga game akan secara otomatis mendukung format 2D.



Gambar 1.16 Memulai proyek pertama Anda

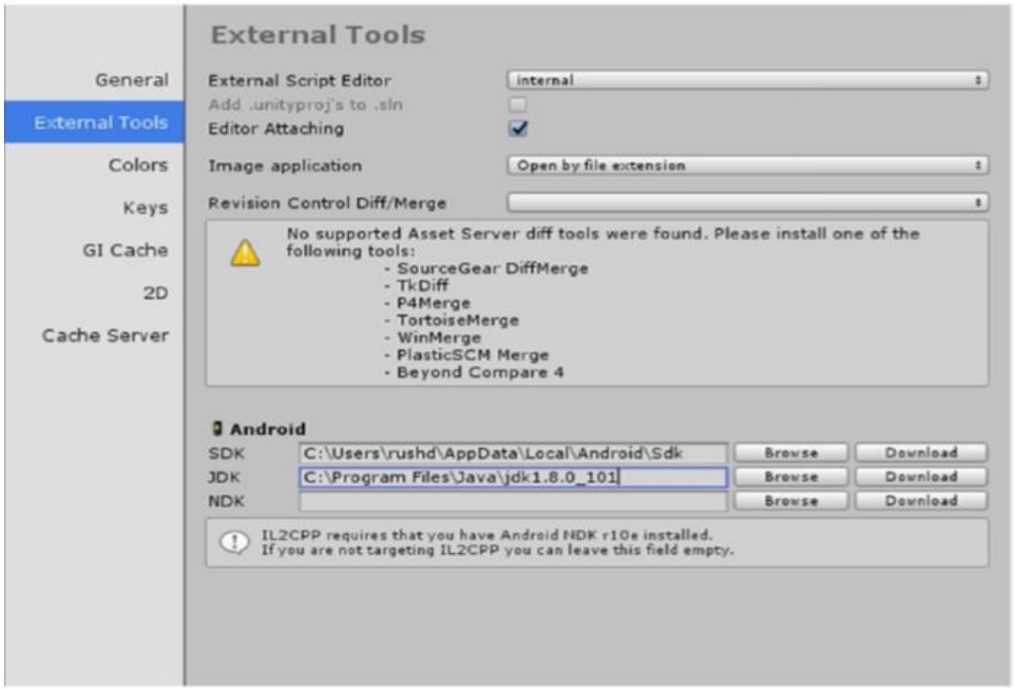
Tidak masalah apa yang Anda sebut game saat ini—ini hanya nama foldernya. Anda dapat mengubah nama APK menjadi apa pun yang Anda suka nanti. Untuk saat ini, sebut saja Simple Platformer. Masukkan itu, centang 2D, lalu klik Buat Proyek. Tunggu sebentar dan Anda akan disambut oleh proyek Unity yang agak kosong. Ada banyak jendela, dan mungkin terlihat sedikit membingungkan pada saat ini, tetapi jangan khawatir, kita akan membahas apa yang dilakukan semuanya di bab berikutnya.

Setting Paths

Mari kita lakukan setup terakhir dengan memberi tahu Unity di mana Android SDK berada di sistem kita. Buka menu atas dan temukan Edit Preferences External Tools. Anda akan menemukan ruang untuk memasukkan lokasi Android SDK (folder utama) dan Java JDK (file). Jika Anda membuat catatan tentang jalur ini sebelumnya, salin dan tempel ke sini. Jika tidak, tekan Browse dan arahkan ke lokasi yang tepat. Jika folder Anda terletak di AppData, mungkin tersembunyi, jadi beri tahu File Explorer untuk Menampilkan Item Tersembunyi (di bawah tab Tampilan) lalu lacak secara manual. Itu ada di suatu tempat—teruslah mencari.

Anda sebenarnya tidak perlu khawatir tentang hal ini sampai Anda datang untuk menguji aplikasi Anda untuk pertama kalinya, tetapi ada baiknya untuk menyiapkan semuanya sebelum kita mulai. Jika mau, Anda dapat melewati tahap ini untuk saat ini, tetapi pastikan untuk kembali ke sana saat peluncuran pertama game Anda. Setelah itu selesai, Unity sekarang sudah siap dan siap digunakan. Saatnya untuk bersemangat karena di Bab 3 kita akan

mulai benar-benar menggunakannya, mengatur beberapa GameObjects, dan bahkan memperkenalkan fisika yang sangat mendasar.



Gambar 1.17 Menyiapkan jalur Anda

1.9 CONTOH GAME YANG DIBUAT DENGAN UNITY

Anda pernah mendengar tentang pro dan kontra Unity, tetapi Anda mungkin masih perlu diyakinkan bahwa alat developeran di Unity dapat memberikan hasil terbaik. Kunjungi galeri Unity di http://unity3d.com/showcase/gallery untuk melihat daftar ratusan game dan simulasi yang terus diperbarui yang dikembangkan menggunakan Unity. Bagian ini hanya membahas beberapa game yang menampilkan sejumlah genre dan platform penerapan.

Desktop (Windows, Mac, Linux)

Karena editor berjalan pada platform yang sama, penyebaran ke Windows atau Mac sering kali merupakan platform target yang paling mudah. Berikut adalah beberapa contoh game desktop dalam genre yang berbeda:

Guns of Icarus Online, sniper pertama yang dikembangkan oleh Muse Games

Gambar 1.18 Tampilan ingame guns of Icarus online

Gone Home, petualangan eksplorasi yang dikembangkan oleh The Fullbright Company

Gambar 1.19 Tampilan ingame gone home

Seluler (iOS, Android)

Unity juga dapat menyebarkan game ke platform seluler seperti iOS (iPhone dan iPad) dan Android (ponsel dan tablet). Berikut adalah beberapa contoh game seluler dalam genre yang berbeda:

Dead Trigger, Sniper pertama yang dikembangkan oleh Madfinger Games

Gambar 1.20 Tampilan ingame dead trigger

Bad Piggies, sebuah game puzzle fisika yang dikembangkan oleh Rovio

Gambar 1.21 Tampilan ingame bad piggies

Tyrant Unleashed, permainan kartu koleksi yang dikembangkan oleh Synapse Games

Gambar 1.22 Tampilan ingame tyrant unleashed

Konsol (PlayStation, Xbox, Wii)

Unity bahkan dapat di-deploy ke konsol game, meskipun developernya harus mendapatkan lisensi dari Sony, Microsoft, atau Nintendo. Karena persyaratan ini dan penyebaran lintas platform yang mudah dari Unity, game konsol juga sering tersedia di komputer desktop. Berikut adalah beberapa contoh game konsol dalam genre yang berbeda:
Assault Android Cactus, penembak arcade yang dikembangkan oleh Witch Beam

Gambar 1.23 Tampilan ingame assault android cactus

Golf Club, simulasi olahraga yang dikembangkan oleh HB Studios

Gambar 1.24 Tampilan ingame golf club

Seperti yang dapat Anda lihat dari contoh-contoh ini, kekuatan Unity pasti dapat ditranslate ke dalam game berkualitas komersial. Tetapi bahkan dengan keunggulan signifikan Unity dibandingkan alat developeran game lainnya, pendatang baru mungkin memiliki kesalahpahaman tentang keterlibatan pemrograman dalam proses developeran. Unity sering digambarkan hanya sebagai daftar fitur tanpa pemrograman yang diperlukan, yang merupakan pandangan menyesatkan yang tidak akan mengajari orang apa yang perlu mereka ketahui untuk menghasilkan judul komersial. Meskipun benar bahwa Anda dapat mengklik bersama prototipe yang cukup rumit menggunakan komponen yang sudah ada sebelumnya bahkan tanpa melibatkan programmer (yang merupakan prestasi yang cukup besar), pemrograman yang ketat diperlukan untuk bergerak melampaui prototipe yang menarik ke game yang dipoles untuk dirilis.

1.10 CARA MENGGUNAKAN UNITY

Sampai sini, saya anggap Anda sudah menginstal Unity, luncurkan Unity untuk mulai menjelajahi antarmukanya. Anda mungkin ingin melihat contoh, jadi buka proyek contoh yang disertakan; instalasi baru akan membuka proyek contoh secara otomatis, tetapi Anda juga

dapat memilih File > Buka Proyek untuk membukanya secara manual. Contoh proyek dipasang di direktori pengguna bersama, seperti C:\Users\Public\Documents\Unity Projects\ di Windows, atau Users/Shared/Unity/ di Mac OS. Anda mungkin juga perlu membuka contoh scene, jadi klik dua kali file scene Mobil yang ditemukan dengan membuka SampleScene/Scenes/ di browser file di bagian bawah editor.

Gambar 1.25 Bagian dari antarmuka di Unity

Antarmuka di Unity dibagi menjadi beberapa bagian: tab Scene, tab Game, Toolbar, tab Hierarchy, Inspector, tab Project, dan tab Console. Setiap bagian memiliki tujuan yang berbeda tetapi semuanya sangat penting untuk siklus hidup pembuatan game:

- Anda dapat menelusuri semua file di tab Proyek.
- Anda dapat menempatkan objek dalam scene 3D yang sedang dilihat menggunakan tab Scene.
- Toolbar memiliki kontrol untuk bekerja dengan scene.
- Anda dapat menarik dan melepaskan hubungan objek di tab Hierarki.
- Inspector mencantumkan informasi tentang objek yang dipilih, termasuk kode yang ditautkan.
- Anda dapat menguji bermain dalam tampilan Game sambil menonton keluaran kesalahan di tab Konsol.

Ini hanyalah tata letak default di Unity; semua tampilan yang berbeda ada di tab dan dapat dipindahkan atau diubah ukurannya, berlabuh di tempat yang berbeda di layar. Nanti Anda dapat bermain-main dengan menyesuaikan tata letak, tetapi untuk saat ini tata letak default adalah cara terbaik untuk memahami apa yang dilakukan semua tampilan.

Scene view, Game view, dan Toolbar

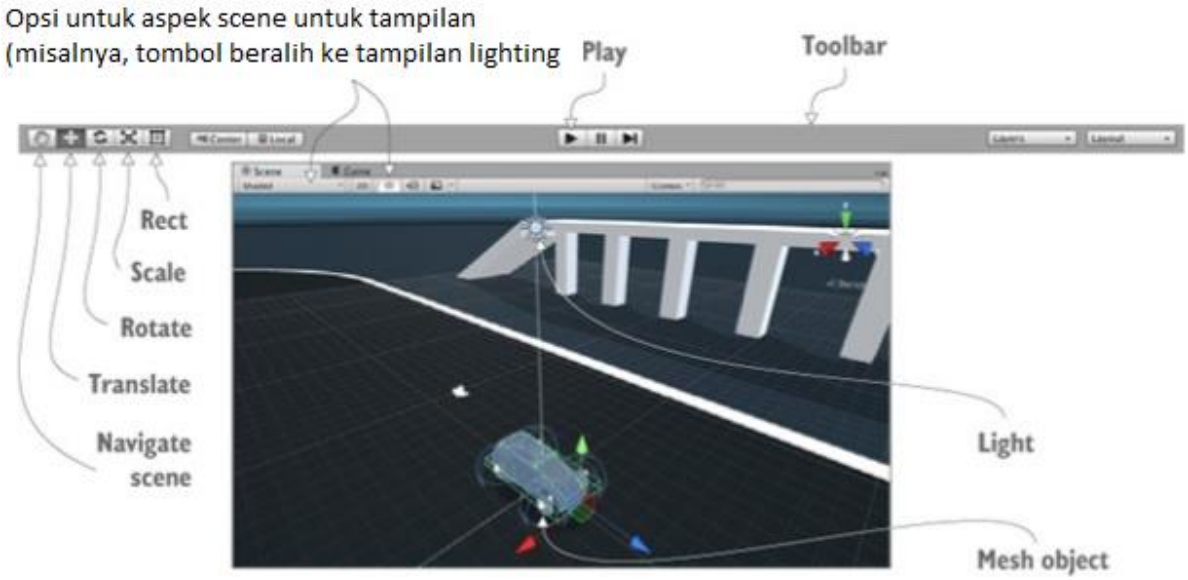
Bagian antarmuka yang paling menonjol adalah tampilan Scene di tengah. Di sinilah Anda dapat melihat seperti apa dunia game dan memindahkan objek. Objek mesh dalam scene muncul sebagai objek mesh (didefinisikan dalam sekejap). Anda juga dapat melihat sejumlah objek lain dalam scene, yang diwakili oleh berbagai ikon dan garis berwarna: kamera, lampu, sumber audio, daerah tumbukan, dan sebagainya. Perhatikan bahwa tampilan yang

Anda lihat di sini tidak sama dengan tampilan di game yang sedang berjalan—Anda dapat melihat sekeliling scene sesuka hati tanpa dibatasi oleh tampilan game.

Definisi -Objek mesh adalah objek visual dalam ruang 3D. Visual dalam 3D dibangun dari banyak garis dan bentuk yang terhubung; maka kata mesh. Tampilan Game bukanlah bagian terpisah dari layar melainkan tab lain yang terletak tepat di sebelah Scene (cari tab di kiri atas tampilan). Beberapa tempat di antarmuka memiliki banyak tab seperti ini; jika Anda mengklik tab yang berbeda, tampilan akan digantikan oleh tab baru yang aktif. Saat game sedang berjalan, yang Anda lihat di tampilan ini adalah gamenya. Tidak perlu berpindah tab secara manual setiap kali Anda menjalankan game, karena tampilan secara otomatis beralih ke Game saat game dimulai.

Tip, Saat game sedang berjalan, Anda dapat beralih kembali ke tampilan Scene, memungkinkan Anda untuk memeriksa objek di scene yang sedang berjalan. Kemampuan ini sangat berguna untuk melihat apa yang terjadi saat game sedang berjalan dan merupakan alat debugging yang berguna yang tidak tersedia di sebagian besar game engine.

Berbicara tentang menjalankan game, itu sesederhana menekan tombol Play tepat di atas tampilan Scene. Seluruh bagian atas antarmuka disebut sebagai Toolbar, dan Play terletak tepat di tengah. Gambar 1.10 memisahkan antarmuka editor lengkap untuk menampilkan hanya Toolbar di bagian atas, serta tab Scene/Game tepat di bawahnya.



Gambar 1.26 Screenshoot editor dipangkas untuk menampilkan Toolbar, Scene, dan Game

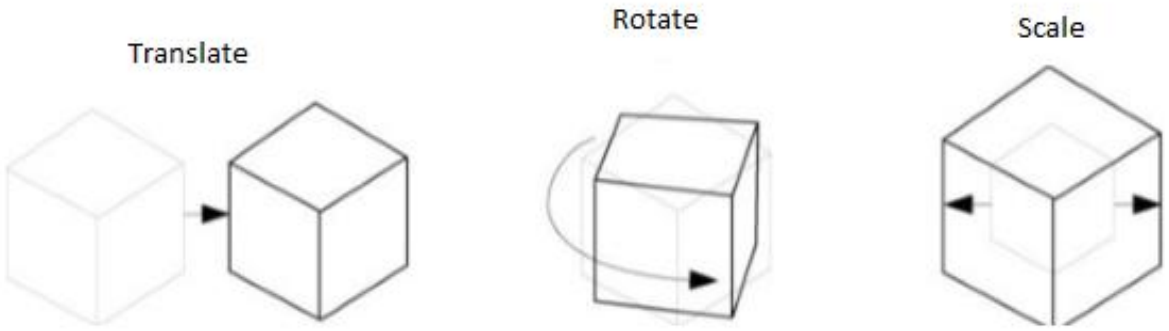
Di sisi kiri Toolbar terdapat tombol untuk navigasi scene dan mengubah objek—cara melihat sekeliling scene dan cara memindahkan objek. Saya sarankan Anda meluangkan waktu untuk berlatih melihat-lihat scene dan objek bergerak, karena ini adalah dua aktivitas terpenting yang akan Anda lakukan di editor visual Unity (mereka sangat penting sehingga mereka mendapatkan bagiannya sendiri setelah yang ini). Sisi kanan Toolbar adalah tempat Anda akan menemukan menu tarik-turun untuk tata letak dan layer. Seperti yang disebutkan sebelumnya, tata letak antarmuka Unity fleksibel, sehingga menu Tata Letak memungkinkan Anda beralih antar tata letak. Adapun menu Layers, itu adalah fungsionalitas lanjutan yang dapat Anda abaikan untuk saat ini (layer akan disebutkan di bab selanjutnya).

Menggunakan mouse dan keyboard

Navigasi scene terutama dilakukan menggunakan mouse, bersama dengan beberapa modifier yang digunakan untuk memodifikasi apa yang dilakukan mouse. Tiga manuver

navigasi utama adalah Move, Orbit, dan Zoom. Gerakan mouse tertentu untuk masing-masing dijelaskan dalam lampiran A di akhir buku ini, karena gerakan tersebut bervariasi tergantung pada mouse yang Anda gunakan. Pada dasarnya, tiga gerakan berbeda melibatkan klik-dan-drag sambil menahan beberapa kombinasi Alt (atau Option di Mac) dan Ctrl. Luangkan beberapa menit untuk bergerak di sekitar scene untuk memahami apa yang dilakukan Move, Orbit, dan Zoom.

Transformasi objek juga dilakukan melalui tiga manuver utama, dan tiga gerakan navigasi scene analog dengan tiga transformasi: Translate, Putar, dan Scale



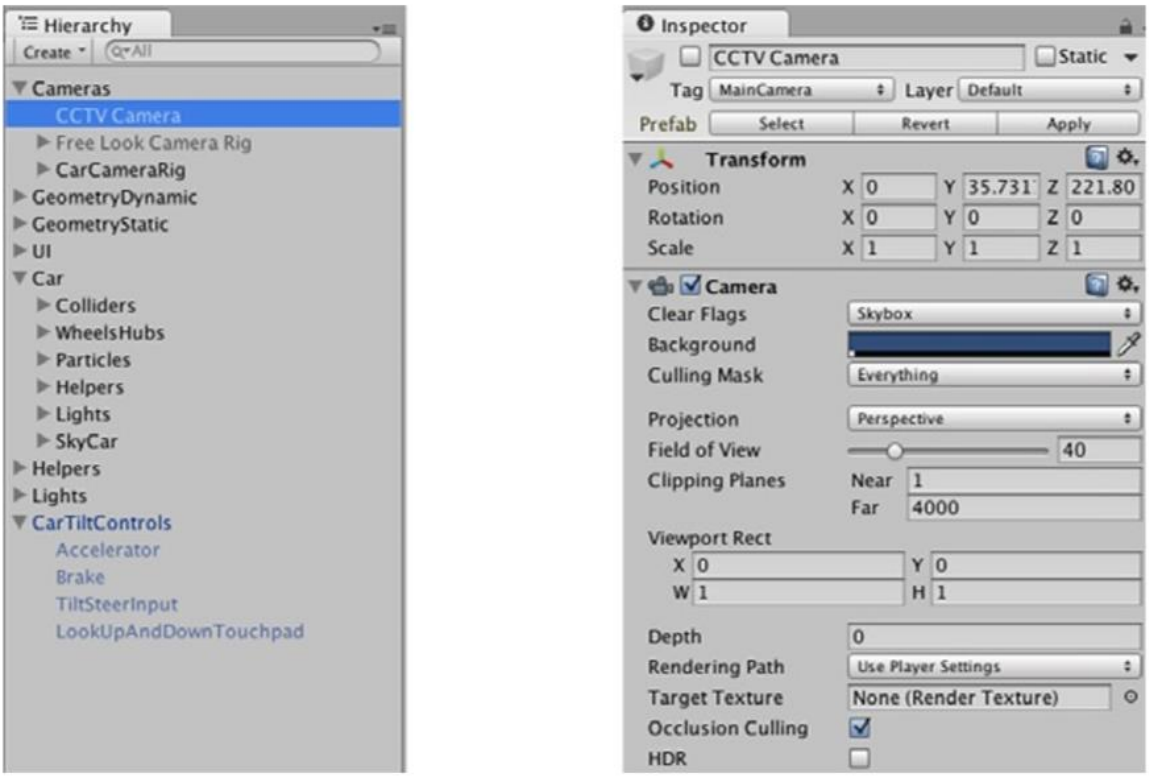
Gambar 1.27 Menerapkan tiga transformasi: Translate, Putar, dan Scale. (Garis yang lebih terang adalah keadaan objek sebelumnya sebelum diubah.)

Saat Anda memilih objek dalam scene, Anda kemudian dapat memindahkannya (istilah teknis yang akurat secara matematis adalah translate), memutar objek, atau scaling seberapa besar objek tersebut. Berkaitan kembali dengan navigasi scene, Move adalah saat Anda Translate kamera, Orbit adalah saat Anda Memutar kamera, dan Zoom adalah saat Anda Scaling kamera. Selain tombol pada Toolbar, Anda dapat beralih di antara fungsi-fungsi ini dengan menekan W, E, atau R pada keyboard. Saat Anda mengaktifkan transformasi, Anda akan melihat sekumpulan panah atau lingkaran berkode warna muncul di atas objek dalam scene; ini adalah alat Transform, dan Anda dapat mengeklik dan menyeret alat ini untuk menerapkan transformasi.

Ada juga alat keempat di sebelah tombol transformasi. Disebut alat Rect, ini dirancang untuk digunakan dengan grafik 2D. Alat yang satu ini menggabungkan gerakan, rotasi, dan penscalean. Operasi ini harus menjadi alat terpisah dalam 3D tetapi digabungkan dalam 2D karena ada satu dimensi yang tidak perlu dikhawatirkan. Unity memiliki sejumlah pintasan keyboard lain untuk mempercepat berbagai tugas. Lihat lampiran A untuk mempelajarinya. Dan dengan itu, ke bagian antarmuka yang tersisa!

Tab Hierarchy dan Inspector

Melihat sisi layar, Anda akan melihat tab Hierarchy di sebelah kiri dan Inspector di sebelah kanan. Hierarchy adalah tampilan daftar dengan nama setiap objek dalam scene yang terdaftar, dengan nama-nama bersarang bersama sesuai dengan hubungan hierarki mereka dalam scene. Pada dasarnya, ini adalah cara memilih objek berdasarkan nama daripada memburunya dan mengkliknya di dalam Scene. Hierarchy linkage mengelompokkan objek bersama-sama, mengelompokkannya secara visual seperti folder dan memungkinkan Anda untuk memindahkan seluruh grup bersama-sama.



Gambar 1.28 Screenshoot editor dipangkas untuk menampilkan tab Hierarchy dan Inspector

Inspector menunjukkan informasi tentang objek yang dipilih saat ini. Pilih objek dan Inspector kemudian diisi dengan informasi tentang objek itu. Informasi yang ditampilkan cukup banyak daftar komponen, dan Anda bahkan dapat melampirkan atau menghapus komponen dari objek. Semua objek game memiliki setidaknya satu komponen, Transform, jadi setidaknya Anda akan selalu melihat informasi tentang pemosisian dan rotasi di Inspector. Sering kali objek akan memiliki beberapa komponen yang tercantum di sini, termasuk skrip yang dilampirkan ke objek itu.

Tab Proyek dan Konsol

Di bagian bawah layar Anda akan melihat Proyek dan Konsol. Seperti Scene dan View, ini bukan dua bagian layar yang terpisah, melainkan tab yang dapat Anda alihkan. Proyek menunjukkan semua aset (seni, kode, dan sebagainya) dalam proyek. Secara khusus, di sisi kiri tampilan adalah daftar direktori dalam proyek; saat Anda memilih direktori, sisi kanan tampilan menunjukkan file individual di direktori itu. Daftar direktori di Project mirip dengan tampilan daftar di Hierarchy, tetapi sementara Hierarchy menunjukkan objek dalam scene, Project menampilkan file yang tidak terdapat dalam scene tertentu (termasuk file scene—saat Anda menyimpan scene, itu muncul di Proyek!).

Gambar 1.29 Screenshoot editor dipangkas untuk menampilkan tab Proyek dan Konsol

Tampilan proyek mencerminkan direktori Aset pada disk, tetapi Anda biasanya tidak boleh memindahkan atau menghapus file secara langsung dengan membuka folder Aset. Jika Anda melakukan hal-hal itu dalam tampilan Proyek, Unity akan tetap sinkron dengan folder itu. Konsol adalah tempat pesan dari kode muncul. Beberapa dari pesan ini akan menjadi keluaran debug yang Anda tempatkan dengan sengaja, tetapi Unity juga mengeluarkan pesan kesalahan jika menemui masalah dalam skrip yang Anda tulis.

Bangun dan jalankan dengan pemrograman Unity

Sekarang mari kita lihat bagaimana proses kerja pemrograman di Unity. Meskipun aset seni dapat ditata dalam editor visual, Anda perlu menulis kode untuk mengontrolnya dan membuat game menjadi interaktif. Unity mendukung beberapa bahasa pemrograman, khususnya JavaScript dan C#. Ada pro dan kontra untuk kedua pilihan tersebut, tetapi Anda akan menggunakan C# di seluruh buku ini.

Mengapa memilih C# daripada JavaScript?

Semua daftar kode dalam buku ini menggunakan C# karena memiliki sejumlah keunggulan dibandingkan JavaScript dan lebih sedikit kerugian, terutama untuk developer profesional (tentu saja bahasa yang saya gunakan di tempat kerja).

Salah satu manfaatnya adalah C# diketik dengan kuat, sedangkan JavaScript tidak. Sekarang, ada banyak argumen di antara programmer berpengalaman tentang apakah pengetikan dinamis adalah pendekatan yang lebih baik untuk, katakanlah, developeran web, tetapi pemrograman untuk platform game tertentu (seperti iOS) sering kali mendapat manfaat dari atau bahkan memerlukan pengetikan statis. Unity bahkan telah menambahkan direktif #pragma strict untuk memaksa pengetikan statis dalam JavaScript. Meskipun secara teknis ini berhasil, ini melanggar salah satu prinsip dasar tentang bagaimana JavaScript beroperasi, dan jika Anda akan melakukannya, maka Anda lebih baik menggunakan bahasa yang secara intrinsik diketik dengan kuat.

Ini hanyalah salah satu contoh bagaimana JavaScript dalam Unity tidak sama dengan JavaScript di tempat lain. JavaScript di Unity memang mirip dengan JavaScript di browser web, tetapi ada banyak perbedaan dalam cara kerja bahasa di setiap konteks. Banyak developer menyebut bahasa di Unity sebagai UnityScript, nama yang menunjukkan kesamaan tetapi terpisah dari JavaScript. Keadaan “serupa tetapi berbeda” ini dapat menimbulkan masalah bagi programmer, baik dalam hal mendatangkan pengetahuan tentang JavaScript dari luar Unity, maupun dalam hal menerapkan pengetahuan pemrograman yang diperoleh dengan bekerja di Unity.

Mari kita telusuri contoh penulisan dan menjalankan beberapa kode. Luncurkan Unity dan buat new project; pilih File > New Project untuk membuka jendela New Project. Ketikkan nama untuk proyek, lalu pilih di mana Anda ingin menyimpannya. Sadarilah bahwa proyek Unity hanyalah sebuah direktori yang penuh dengan berbagai aset dan file pengaturan, jadi simpan proyek di mana saja di komputer Anda. Klik Buat Proyek dan kemudian Unity akan menghilang sebentar saat menyiapkan direktori proyek.

Peringatan, proyek Unity mengingat versi Unity yang mana mereka dibuat dan akan mengeluarkan peringatan jika Anda mencoba membukanya dalam versi yang berbeda. Terkadang tidak masalah (misalnya, abaikan saja peringatan jika muncul saat membuka unduhan sampel buku ini), tetapi terkadang Anda ingin membuat cadangan proyek Anda sebelum membukanya. Ketika Unity muncul kembali, Anda akan melihat proyek kosong. Selanjutnya, mari kita bahas bagaimana program Anda dijalankan di Unity.

Bagaimana kode berjalan di Unity: komponen skrip

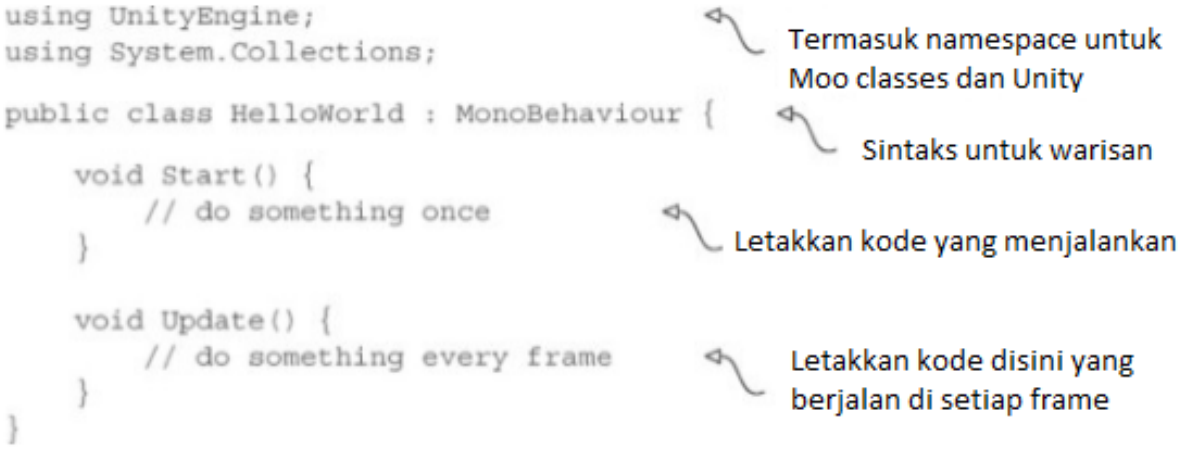
Semua eksekusi kode di Unity dimulai dari file kode yang ditautkan ke objek di scene. Pada akhirnya itu semua adalah bagian dari sistem komponen yang dijelaskan sebelumnya;

objek game dibangun sebagai kumpulan komponen, dan koleksi itu dapat menyertakan skrip untuk dieksekusi.

Catatan, Unity mengacu pada file kode sebagai skrip, menggunakan definisi "skrip" yang paling umum ditemui dengan JavaScript yang berjalan di browser: kode dieksekusi di dalam game engine Unity, versus kode yang dikompilasi yang berjalan sebagai executable-nya sendiri. Tapi jangan bingung karena banyak orang mendefinisikan kata tersebut secara berbeda; misalnya, "skrip" sering merujuk pada program utilitas mandiri yang pendek. Skrip di Unity lebih mirip dengan kelas OOP individual, dan skrip yang dilampirkan ke objek dalam scene adalah instance objek.

Seperti yang mungkin sudah Anda duga dari deskripsi ini, di Unity, skrip adalah komponen—tidak semua skrip, ingatlah, hanya skrip yang mewarisi dari MonoBehaviour, kelas dasar untuk komponen skrip. MonoBehaviour mendefinisikan dasar yang tidak terlihat tentang bagaimana komponen dilampirkan ke objek game, dan (seperti yang ditunjukkan dalam daftar 1.1) mewarisi darinya menyediakan beberapa metode yang dijalankan secara otomatis yang dapat Anda timpa. Metode tersebut termasuk Start(), yang dipanggil sekali ketika objek menjadi aktif (yang umumnya segera setelah level dengan objek tersebut dimuat), dan Update(), yang dipanggil setiap frame. Jadi kode Anda dijalankan ketika Anda memasukkannya ke dalam metode yang telah ditentukan ini.

Sebuah frame adalah satu siklus dari kode permainan perulangan. Hampir semua video game (tidak hanya di Unity, tetapi video game secara umum) dibangun di sekitar game loop inti, di mana kode dijalankan dalam siklus saat game sedang berjalan. Setiap siklus termasuk menggambar layar; maka nama bingkai (seperti rangkaian bingkai foto film).



Gambar 1.30 Templat kode untuk komponen skrip dasar

Inilah isi file saat Anda membuat skrip C# baru: kode boilerplate minimal yang mendefinisikan komponen Unity yang valid. Unity memiliki templat skrip yang tersimpan di dalam aplikasi, dan ketika Anda membuat skrip baru, skrip itu menyalin templat itu dan mengganti nama kelas agar sesuai dengan nama file (yang dalam kasus saya adalah HelloWorld.cs). Ada juga shell kosong untuk Start() dan Update() karena itu adalah dua tempat paling umum untuk memanggil kode kustom Anda (walaupun saya cenderung sedikit menyesuaikan spasi di sekitar fungsi-fungsi tersebut, karena template tidak sesuai dengan cara saya seperti spasi putih dan saya rewel tentang itu).

Untuk membuat skrip, pilih C# Script dari menu Buat yang Anda akses baik di bawah menu Aset (perhatikan bahwa Aset dan GameObjects keduanya memiliki daftar untuk Buat tetapi menunya berbeda) atau dengan mengklik kanan di tampilan Proyek. Ketikkan nama untuk skrip baru, seperti HelloWorld. Seperti yang akan dijelaskan nanti di bab ini (lihat

gambar 1.31), Anda akan mengklik-dan-menarik file skrip ini ke objek dalam scene. Klik dua kali skrip dan itu akan secara otomatis dibuka di program lain yang disebut MonoDevelop, dibahas selanjutnya.

Skrip

Seperti yang disebutkan sebelumnya di jam ini, menggunakan skrip adalah cara untuk mendefinisikan perilaku. Skrip dilampirkan ke objek di Unity seperti komponen lain dan memberi mereka interaktivitas. Secara umum ada tiga langkah yang terlibat dalam bekerja dengan skrip di Unity:

1. Buat skrip.
2. Lampirkan skrip ke satu atau lebih objek game.
3. Jika skrip memerlukannya, isi properti apa pun dengan nilai atau objek game lainnya. Sisa pelajaran ini membahas langkah-langkah ini.

Gambar 1.31 Tampilan Inspector dari sebuah skrip.

Membuat Skrip

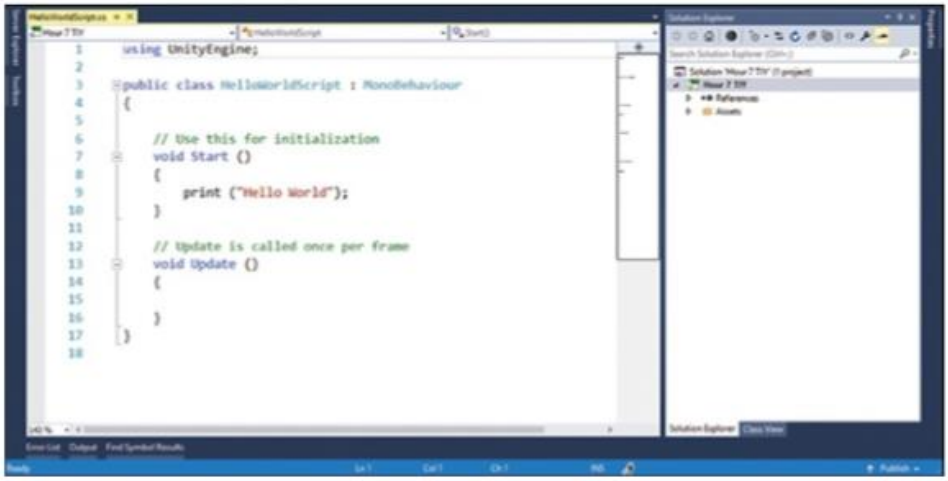
Sebelum membuat skrip, yang terbaik adalah membuat folder Skrip di bawah folder Aset dalam tampilan Proyek. Setelah Anda memiliki folder untuk menampung semua skrip Anda, cukup klik kanan folder tersebut dan pilih Create > C# Script. Kemudian beri nama skrip Anda sebelum melanjutkan.

Bahasa skrip

Unity memungkinkan Anda untuk menulis skrip dalam C# atau JavaScript. Buku ini menggunakan bahasa C# untuk semua skrip karena sedikit lebih fleksibel dan kuat di Unity. Perlu dicatat bahwa JavaScript sedang dihapus, dan opsi untuk membuat file JavaScript baru

bahkan tidak muncul di editor lagi. Di beberapa titik di masa depan, dukungan untuk bahasa tersebut akan dihentikan sepenuhnya.

Setelah skrip dibuat, Anda dapat melihat dan memodifikasinya. Mengklik skrip dalam tampilan Proyek memungkinkan Anda melihat isi skrip dalam tampilan Inspector. Mengklik dua kali skrip dalam tampilan Proyek membuka editor default Anda, tempat Anda dapat menambahkan kode ke skrip. Dengan asumsi bahwa Anda telah menginstal komponen default dan tidak mengubah apa pun, mengklik dua kali file akan membuka lingkungan developeran Visual Studio



Gambar 1.32 Software Visual Studio dengan jendela editor yang ditampilkan

Nama Skrip

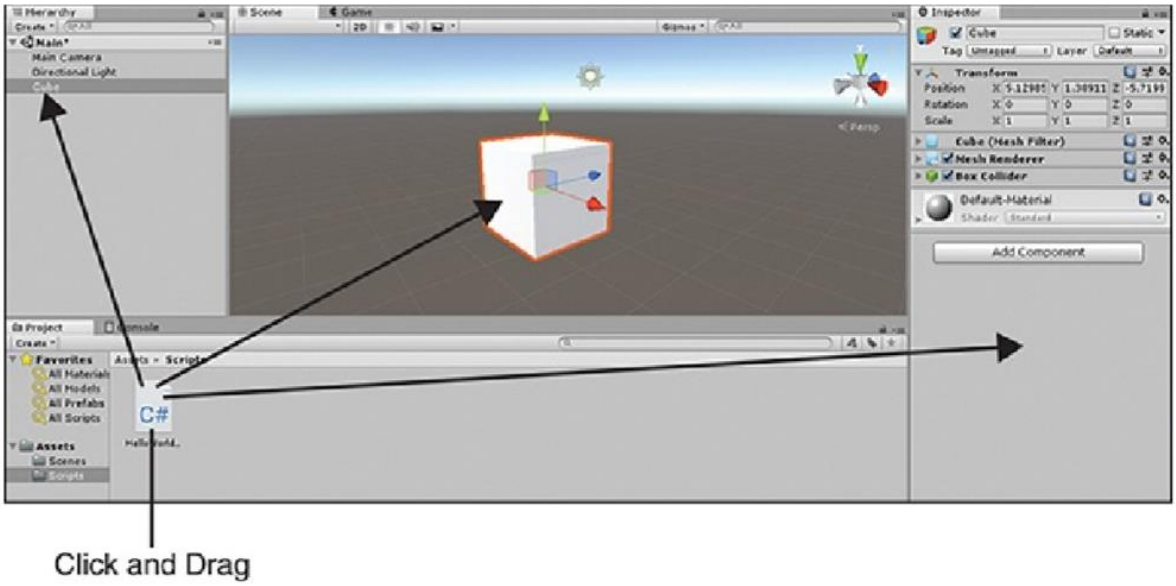
Anda baru saja membuat skrip bernama HelloWordScript; nama file skrip yang sebenarnya penting. Pada Unity dan C#, nama file harus sesuai dengan nama class yang ada di dalamnya. Kelas dibahas nanti dalam jam ini, tetapi untuk saat ini, cukuplah untuk mengatakan bahwa jika Anda memiliki skrip yang berisi kelas bernama MyAwesomeClass, file yang berisi itu akan diberi nama MyAwesomeClass.cs. Perlu juga dicatat bahwa kelas, dan karena itu nama file skrip, tidak boleh berisi spasi.

Lebih lanjut tentang IDE

Visual Studio adalah software yang kuat dan kompleks yang dibundel dengan Unity. Editor seperti ini dikenal sebagai IDE (lingkungan developeran terintegrasi), dan mereka membantu Anda menulis kode untuk game. Karena IDE sebenarnya bukan bagian dari Unity, buku ini tidak membahasnya secara mendalam. Satu-satunya bagian dari Visual Studio yang perlu Anda ketahui sekarang adalah jendela editor. Jika ada hal lain yang perlu Anda ketahui tentang IDE, itu tercakup dalam jam saat dibutuhkan. (Catatan: Sebelum Unity 2018.1, IDE bernama MonoDevelop juga dikemas dengan Unity. Anda masih dapat memperoleh dan menggunakan software ini satu per satu, tetapi MonoDevelop tidak lagi dikirimkan bersama mesinnya.)

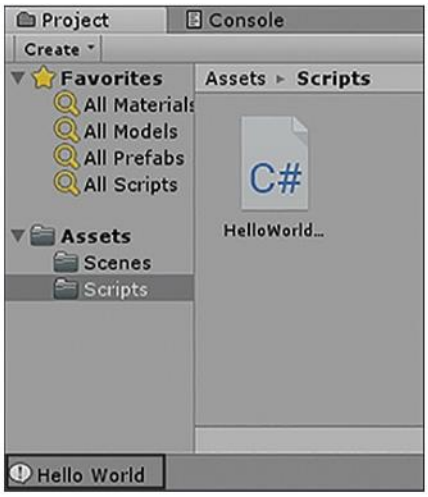
Melampirkan Script

Untuk melampirkan skrip ke objek game, cukup klik skrip di tampilan Project dan drag ke objek. Anda dapat menyeret skrip ke objek dalam tampilan Hierarki, tampilan Scene, atau tampilan Inspector (dengan asumsi bahwa objek dipilih). Setelah dilampirkan ke objek, skrip menjadi komponen objek itu dan terlihat di tampilan Inspector.



Gambar 1.33 Mengklik dan menyeret skrip ke objek yang diinginkan.

Untuk melihat ini beraksi, lampirkan skrip HelloWorldScript yang Anda buat sebelumnya ke Kamera Utama. Anda sekarang akan melihat komponen bernama Hello World Script (Script) di tampilan Inspector. Jika Anda menjalankan scene, Anda melihat Hello World muncul di bagian bawah editor, di bawah tampilan Proyek



Gambar 1.34 Kata-kata Hello World keluar saat menjalankan scene.

Anatomi Skrip Dasar

Di bagian sebelumnya, Anda memodifikasi skrip untuk menampilkan beberapa teks ke layar, tetapi konten skrip tidak dijelaskan. Di bagian ini, Anda akan melihat template default yang diterapkan ke setiap skrip C# baru. (Perhatikan bahwa skrip yang ditulis dalam JavaScript memiliki komponen yang sama meskipun terlihat sedikit berbeda.) Listing 7.1 berisi kode lengkap yang dibuat untuk Anda oleh Unity saat Anda membuat skrip baru bernama HelloWorldScript.

Daftar kode default

```
using UnityEngine; using System.Collections;
public class HelloWorldScript : MonoBehaviour
{
```

```
    // Use this for initialization
    void Start() {
    }
    // Update is called once per frame void Update() {
    }
}
```

Kode ini dapat dipecah menjadi tiga bagian: bagian using, bagian deklarasi kelas, dan isi kelas.

Bagian Penggunaan

Bagian pertama skrip mencantumkan pustaka yang akan digunakan skrip. Ini terlihat seperti ini:

```
Using UnityEngine;
Using System.Collections;
```

Secara umum, Anda tidak terlalu sering mengubah bagian ini dan sebaiknya biarkan saja untuk sementara waktu. Baris-baris ini biasanya ditambahkan untuk Anda saat Anda membuat skrip di Unity. Pustaka System.Collections bersifat opsional dan sering kali dihilangkan jika skrip tidak menggunakan fungsionalitas apa pun darinya.

Bagian Deklarasi Kelas

Bagian selanjutnya dari skrip disebut deklarasi kelas. Setiap skrip berisi kelas yang dinamai menurut skrip. Ini terlihat seperti ini:

```
Public class HelloWorldScript : MonoBehaviour { }
```

Semua kode di antara kurung buka { dan kurung tutup } adalah bagian dari kelas ini dan karena itu merupakan bagian dari skrip. Semua kode Anda harus berada di antara tanda kurung ini. Seperti pada bagian using, Anda jarang mengubah bagian deklarasi kelas dan sebaiknya biarkan saja untuk saat ini.

Isi Kelas

Bagian di antara tanda kurung buka dan tutup kelas dianggap “di dalam” kelas. Semua kode Anda ada di sini. Secara default, skrip berisi dua metode di dalam kelas, Mulai dan Perbarui:

```
// Use this for initialization
void Start () {
}
// Update is called once per frame
void Update () {
}
```

Metode dibahas secara lebih rinci di Jam 8, "Scripting, Bagian 2." Untuk saat ini, ketahuilah bahwa kode apa pun di dalam metode Mulai berjalan saat scene pertama kali dimulai. Kode apa pun di dalam metode Pembaruan berjalan secepat mungkin—bahkan ratusan kali per detik.

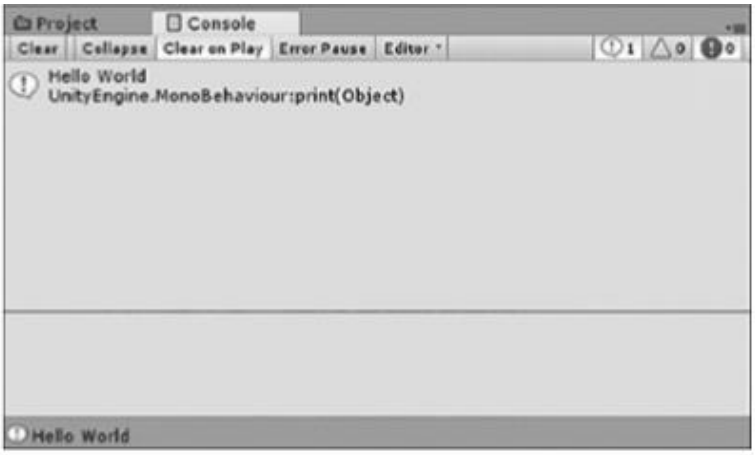
Komentar

Bahasa pemrograman memungkinkan pembuat kode untuk meninggalkan pesan bagi mereka yang membaca kode nanti. Pesan-pesan ini disebut komentar. Kata apa pun yang mengikuti dua garis miring (//) "dikomentari". Ini berarti bahwa komputer akan melewatinya dan tidak mencoba membacanya sebagai kode. Anda dapat melihat contoh komentar di “Creating a Script” di awal jam ini.

Konsol

Ada jendela lain di editor Unity yang belum disebutkan sampai sekarang: Konsol. Pada dasarnya, Console adalah jendela yang berisi output teks dari game Anda. Seringkali, ketika ada kesalahan atau keluaran dari skrip, pesan ditulis ke Konsol. Gambar 7.5 menunjukkan

Konsol. Jika jendela Konsol tidak terlihat, Anda dapat mengaksesnya dengan memilih Jendela > Konsol.



Gambar 1.35 Jendela Konsol

Menggunakan Metode Bawaan

Sekarang Anda siap untuk mencoba metode bawaan Mulai dan Perbarui dan lihat cara kerjanya. Skrip PentingFunctions yang telah selesai tersedia di aset buku untuk Jam 7. Cobalah untuk menyelesaikan latihan berikut ini sendiri, tetapi jika Anda buntu, lihat aset buku:

1. Buat proyek atau scene baru. Tambahkan skrip ke proyek bernama PentingFunctions. Klik dua kali skrip untuk membukanya di editor kode Anda.
2. Di dalam skrip, tambahkan baris kode berikut ke metode Mulai:
   print ("Start runs before an object Updates");
3. Simpan skrip, dan di Unity, lampirkan ke Kamera Utama. Jalankan scene dan perhatikan pesan yang muncul di jendela Konsol.
4. Kembali ke Visual Studio, tambahkan baris kode berikut ke metode Update:
   print ("This is called once a frame");
5. Simpan skrip dan mulai dan hentikan scene dengan cepat di Unity. Perhatikan bagaimana, di Konsol, ada satu baris teks dari metode Mulai dan ada banyak baris dari metode Pembaruan.

Variabel

Terkadang Anda ingin menggunakan sedikit data yang sama lebih dari sekali dalam sebuah skrip. Dalam kasus seperti itu, Anda memerlukan placeholder untuk data yang dapat digunakan kembali. Placeholder seperti itu disebut variabel. Tidak seperti matematika tradisional, variabel dalam pemrograman dapat berisi lebih dari sekadar angka. Mereka dapat menyimpan kata-kata, objek kompleks, atau skrip lainnya.

Membuat Variabel

Setiap variabel memiliki nama dan tipe yang diberikan kepada variabel saat dibuat. Anda membuat variabel dengan sintaks berikut:

<variable type> <name>;

Jadi, untuk membuat bilangan bulat bernama num1, ketikkan yang berikut ini:

int num1;

Sintaks

Istilah sintaks mengacu pada aturan bahasa pemrograman. Sintaks menentukan bagaimana segala sesuatu disusun dan ditulis sehingga komputer tahu cara membacanya. Anda mungkin telah memperhatikan bahwa setiap pernyataan, atau perintah, dalam skrip sejauh ini telah diakhiri dengan titik koma. Ini juga merupakan bagian dari sintaks C#.

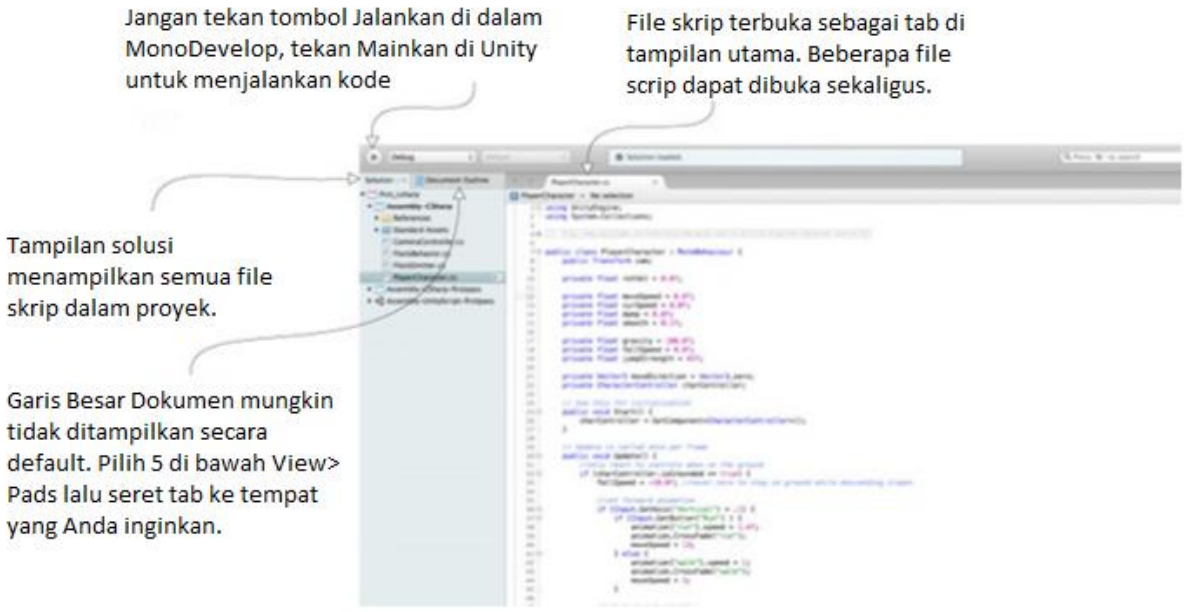
Melupakan titik koma menyebabkan skrip Anda tidak berfungsi. Jika Anda ingin tahu lebih banyak tentang sintaks C#, lihat Panduan C# di https://docs.microsoft.com/en-us/dotnet/csharp/.

Tabel 1.1 Jenis Variable C#

Jenis	Deskripsi
Int	Singkatan dari integer, int menyimpan bilangan bulat positif atau negatif.
Float	float menyimpan data titik-mengambang (seperti 3.4) dan merupakan tipe angka default di Unity. bilangan float di Unity selalu ditulis dengan f setelahnya, seperti 3.4f, 0f, .5f, dan seterusnya.
Double	double juga menyimpan angka floating-point; namun, ini bukan tipe nomor default di Unity. Biasanya dapat menampung angka yang lebih besar daripada float.
Bool	Kependekan dari Boolean, bool menyimpan true atau false (sebenarnya ditulis dalam kode sebagai benar atau salah).
Char	Singkatan dari character, char menyimpan satu huruf, spasi, atau karakter khusus (seperti a, 5, atau !). nilai char ditulis dengan tanda kutip tunggal ('A').
string	Tipe string menampung seluruh kata atau kalimat. Nilai string ditulis dengan tanda kutip ganda ("Hello World").

Menggunakan MonoDevelop, IDE lintas platform

Pemrograman tidak dilakukan dalam Unity persis, melainkan kode ada sebagai file terpisah yang Anda arahkan ke Unity. File skrip dapat dibuat dalam Unity, tetapi Anda masih perlu menggunakan beberapa editor teks atau IDE untuk menulis semua kode di dalam file yang awalnya kosong. Unity dibundel dengan MonoDevelop, IDE lintas platform open source untuk C#. Anda dapat mengunjungi www.monodevelop.com untuk mempelajari lebih lanjut tentang software ini, tetapi versi yang digunakan adalah versi yang dibundel bersama dengan Unity, bukan versi yang diunduh dari situs web mereka, karena beberapa modifikasi dilakukan pada software dasar agar lebih baik. mengintegrasikannya dengan Unity.



Gambar 1.36 Bagian dari antarmuka di MonoDevelop

MonoDevelop mengatur file ke dalam kelompok yang disebut solusi. Unity secara otomatis menghasilkan solusi yang memiliki semua file skrip, jadi Anda biasanya tidak perlu khawatir tentang itu. Karena C# berasal dari produk Microsoft, Anda mungkin bertanya-tanya apakah Anda dapat menggunakan Visual Studio untuk melakukan pemrograman untuk Unity. Jawaban singkatnya adalah ya, Anda bisa. Alat pendukung tersedia dari www.unityvs.com tetapi saya biasanya lebih suka MonoDevelop, sebagian besar karena Visual Studio hanya berjalan di Windows dan menggunakan IDE itu akan mengikat alur kerja Anda ke Windows. Itu tidak selalu merupakan hal yang buruk, dan jika Anda sudah menggunakan Visual Studio untuk melakukan pemrograman maka Anda dapat terus menggunakannya dan tidak memiliki masalah mengikuti buku ini (di luar bab pengantar ini, saya tidak akan berbicara tentang IDE). Mengikat alur kerja Anda ke Windows, bagaimanapun, akan bertentangan dengan salah satu keuntungan terbesar menggunakan Unity, dan hal itu dapat terbukti bermasalah jika Anda perlu bekerja dengan developer berbasis Mac di tim Anda dan/atau jika Anda ingin menyebarkan permainan Anda ke iOS. Meskipun C# berasal dari produk Microsoft dan dengan demikian hanya bekerja pada Windows dengan .NET Framework, C# kini telah menjadi standar bahasa terbuka dan ada kerangka kerja lintas platform yang signifikan: Mono. Unity menggunakan Mono untuk tulang punggung pemrogramannya, dan menggunakan MonoDevelop memungkinkan Anda untuk menjaga seluruh alur kerja developeran lintas platform.

Ingatlah selalu bahwa meskipun kode ditulis dalam MonoDevelop, kode tersebut tidak benar-benar dijalankan di sana. IDE adalah editor teks yang cukup bagus, dan kodenya dijalankan saat Anda menekan Play di dalam Unity.

Lingkup Variabel

Lingkup variabel mengacu pada di mana variabel dapat digunakan. Seperti yang telah Anda lihat di skrip, kelas dan metode menggunakan tanda kurung buka dan tutup untuk menunjukkan apa yang menjadi milik mereka. Daerah antara dua kurung sering disebut sebagai blok. Alasan mengapa ini penting adalah bahwa variabel hanya dapat digunakan di blok tempat variabel tersebut dibuat. Jadi, jika variabel dibuat di dalam metode Mulai skrip, variabel itu tidak tersedia di metode Perbarui karena keduanya adalah dua blok yang berbeda. Mencoba menggunakan variabel yang tidak tersedia akan menghasilkan kesalahan. Jika variabel dibuat di dalam kelas tetapi di luar metode, variabel tersebut akan tersedia untuk kedua metode karena kedua metode berada di blok yang sama dengan variabel (blok kelas). Daftar 7.2 menunjukkan hal ini.

Daftar Demonstrasi Kelas dan Level Blok Lokal

```
// This is in the "class block" and will
// be available everywhere in this class
private int num1;
void Start ()
{
// this is in a "local block" and will
// only be available in the Start method
int num2;
}
```

Publik dan Personal

Ini disebut pengubah akses, dan diperlukan hanya untuk variabel yang dideklarasikan di tingkat kelas. Ada dua pengubah akses yang perlu Anda gunakan: personal dan publik. Banyak yang bisa dikatakan tentang dua pengubah akses, tetapi apa yang benar-benar perlu

Anda ketahui saat ini adalah bagaimana pengaruhnya terhadap variabel. Pada dasarnya, variabel private (variabel dengan kata private sebelumnya) hanya dapat digunakan di dalam file tempat ia dibuat. Skrip lain dan editor tidak dapat melihatnya atau memodifikasinya dengan cara apa pun. Variabel personal dimaksudkan untuk penggunaan internal saja. Variabel publik, sebaliknya, terlihat oleh skrip lain dan bahkan editor Unity. Ini memudahkan Anda untuk mengubah nilai variabel Anda saat itu juga dalam Unity. Jika Anda tidak menandai variabel sebagai publik atau personal, defaultnya adalah personal.

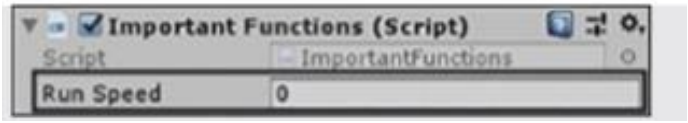
Memodifikasi Variabel Publik di Unity

Ikuti langkah-langkah ini untuk melihat bagaimana variabel publik terlihat di editor Unity:

1. Buat skrip C# baru dan di Visual Studio tambahkan baris berikut di kelas di atas metode Mulai:

int runSpeed publik;

2. Simpan skrip dan kemudian, di Unity, lampirkan ke Kamera Utama.
3. Pilih Kamera Utama dan lihat di tampilan Inspector. Perhatikan skrip yang baru saja Anda lampirkan sebagai komponen. Sekarang perhatikan bahwa komponen memiliki properti baru: Kecepatan Jalankan. Anda dapat memodifikasi properti itu dalam tampilan Inspector, dan perubahan tersebut akan tercermin dalam skrip saat runtime. Gambar 7.6 menunjukkan komponen dengan properti baru. Angka ini mengasumsikan bahwa script yang dibuat bernama PentingFunctions.



Gambar 1.37 Properti Kecepatan Jalankan baru dari komponen skrip.

Operator

Semua data dalam variabel tidak berguna jika Anda tidak memiliki cara untuk mengakses atau memodifikasinya. Operator adalah simbol khusus yang memungkinkan Anda melakukan modifikasi pada data. Mereka umumnya jatuh ke dalam salah satu dari empat kategori: operator aritmatika, operator penugasan, operator kesetaraan, dan operator logika.

Operator Aritmatika

Operator aritmatika melakukan beberapa operasi matematika standar pada variabel. Mereka umumnya hanya digunakan pada variabel angka, meskipun ada beberapa pengecualian.

Tabel 1.2 Operator Aritmetik

Operator	Deskrispi
+	Tambahan. Menambahkan dua angka bersama-sama. Dalam kasus string, tanda + menggabungkan, atau menggabungkan, mereka. Berikut ini adalah contohnya: "Halo" + "Dunia"; // menghasilkan "HelloWorld"
-	Pengurangan. Mengurangi angka di sebelah kiri dengan angka di sebelah kanan.
*	Perkalian. Mengalikan dua angka bersama-sama.
/	Divisi. Membagi bilangan di sebelah kiri dengan bilangan di sebelah kanan.

%	Modulus. Membagi angka di sebelah kiri dengan angka di sebelah kanan tetapi tidak mengembalikan hasilnya. Sebagai gantinya, modulus mengembalikan sisa pembagian. Perhatikan contoh berikut:
	10 % 2; // returns 0
	6 % 5; // returns 1
	24 % 7; // returns 3

Operator aritmatika dapat di-cascade bersama untuk menghasilkan string matematika yang lebih kompleks, seperti dalam contoh ini:

```
x + (5 * (6 − y) / 3);
```

Operator aritmatika bekerja dalam urutan operasi matematika standar. Matematika dilakukan dari kiri ke kanan, dengan apa pun dalam tanda kurung dihitung terlebih dahulu, perkalian dan pembagian dilakukan kedua, dan penambahan dan pengurangan dilakukan ketiga

Operator Penugasan

Operator penugasan persis seperti apa bunyinya: Mereka menetapkan nilai ke variabel. Operator penugasan yang paling menonjol adalah tanda sama dengan, tetapi ada lebih banyak operator penugasan yang menggabungkan beberapa operasi. Semua tugas di C# dari kanan ke kiri. Ini berarti bahwa apa pun yang ada di sisi kanan akan dipindahkan ke kiri. Pertimbangkan contoh-contoh ini:

```
x = 5; // This works. It sets the variable x to 5.
5 = x; // This does not work. You cannot assign a variable to a value (5).
```

Tabel 1.3 operator penugasan.

Operator	Deskripsi
=	Menetapkan nilai di sebelah kanan ke variabel di sebelah kiri.

+ =, − =, *=, /=	Operator penugasan singkatan yang melakukan beberapa operasi aritmatika berdasarkan simbol yang digunakan dan kemudian memberikan hasilnya ke apa pun yang ada di sebelah kiri. Perhatikan contoh berikut:
	x = x + 5; // Menambahkan 5 ke x dan kemudian menetapkannya ke x
	x += 5; // Lakukan hal yang sama seperti di atas, hanya singkatan

++, −	Operator singkatan disebut sebagai operator increment dan decrement. Mereka menambah atau mengurangi angka sebesar 1. Perhatikan contoh berikut:
	x = x + 1; // Menambahkan 1 ke x dan kemudian menetapkannya ke x
	x++; // Lakukan hal yang sama seperti di atas, hanya singkatan

Operator Kesetaraan

Operator kesetaraan membandingkan dua nilai. Hasil dari operator persamaan selalu bernilai benar atau salah. Oleh karena itu, satu-satunya tipe variabel yang dapat menampung hasil dari operator persamaan adalah Boolean. (Ingat bahwa Boolean hanya dapat berisi benar atau salah.) Tabel 7.4 menjelaskan operator kesetaraan.

Tabel 1.4 Operator Kesetaraan

Operator	Deskripsi

= =	Jangan bingung dengan operator penugasan (=), operator ini mengembalikan nilai true hanya jika kedua nilai sama. Jika tidak, ia mengembalikan false. Pertimbangkan contoh-contoh ini:
	5 == 6; // Returns false
	9 == 9; // Returns true

>, <	Ini adalah operator "lebih besar dari" dan "kurang dari". Pertimbangkan contoh-contoh ini:
	5 > 3; // Returns true
	5 < 3; // Returns false

>=, <=	Ini mirip dengan "lebih besar dari" dan "kurang dari" kecuali bahwa mereka adalah operator "lebih besar dari atau sama dengan" dan "kurang dari atau sama dengan". Pertimbangkan contoh-contoh ini:
	3 >= 3; // Returns true
	5 <= 9; // Returns true

!=	Ini adalah operator "tidak sama", dan mengembalikan nilai true jika kedua nilai tidak sama. Jika tidak, ia mengembalikan false. Pertimbangkan contoh-contoh ini:
	5 != 6; // Returns true
	9 != 9; // Returns false

Latihan Tambahan

Dalam aset buku untuk Jam 7 adalah skrip yang disebut EqualityAndOperations.cs. Pastikan untuk memeriksanya untuk beberapa latihan tambahan dengan berbagai operator.

Operator Logika

Operator logika memungkinkan Anda untuk menggabungkan dua atau lebih nilai Boolean (benar atau salah) menjadi satu nilai Boolean. Mereka berguna untuk menentukan kondisi yang kompleks.

Tabel 1.5 Operator Logika

Operator	Deskripsi

	Dikenal sebagai operator AND, ini membandingkan dua nilai Boolean dan menentukan apakah keduanya benar. Jika salah satu, atau keduanya, dari nilai salah, operator ini mengembalikan false. Pertimbangkan contoh-contoh ini:
	true && false; // Returns false
	false && true; // Returns false
	false && false; // Returns false
	true && true; /Returns true

	Dikenal sebagai operator OR, ini membandingkan dua nilai Boolean dan menentukan apakah salah satunya benar. Jika salah satu atau kedua nilai benar, operator ini mengembalikan nilai true. Pertimbangkan contoh-contoh ini:
	true || false; // Returns true
	false || true; // Returns true
	false || false; // Returns false
	true || true; // Returns true

	Dikenal sebagai operator NOT, ini mengembalikan kebalikan dari nilai Boolean. Pertimbangkan contoh-contoh ini:
	!true; // Returns false
	!false; // Returns true

Bersyarat

Sebagian besar kekuatan komputer terletak pada kemampuannya untuk membuat keputusan yang belum sempurna. Akar dari kekuatan ini terletak pada Boolean benar dan salah. Anda dapat menggunakan nilai Boolean ini untuk membangun kondisional dan mengarahkan program pada jalur yang unik. Saat Anda membangun aliran logika Anda melalui kode, ingatlah bahwa mesin hanya dapat membuat satu keputusan sederhana pada satu waktu. Namun, gabungkan cukup banyak keputusan itu, dan Anda dapat membangun interaksi yang kompleks.

Pernyataan If

Dasar dari conditional adalah pernyataan if, yang terstruktur seperti ini:

```
If ( <some Boolean condition> )
{
// do something
}
```

Struktur if dapat dibaca sebagai "jika ini benar, lakukan ini." Jadi, jika Anda ingin menampilkan "Hello World" ke Console jika nilai x lebih besar dari 5, Anda dapat menulis sebagai berikut:

```
if (x > 5)
{
print("Hello World");
}
```

Ingat bahwa isi dari kondisi pernyataan if harus bernilai benar atau salah. Menempatkan angka, kata, atau apa pun di sana tidak akan berfungsi:

```
if("Hello" == "Hello") // Correct
if (x + y) // Incorrect
```

Terakhir, kode apa pun yang ingin Anda jalankan jika kondisi bernilai true harus berada di dalam tanda kurung buka dan tutup yang mengikuti pernyataan if.

Pernyataan If/Else

Pernyataan if bagus untuk kode kondisional, tetapi bagaimana jika Anda ingin membagi program Anda ke dua jalur yang berbeda? Pernyataan if / else memungkinkan Anda untuk melakukan itu. If / else adalah premis dasar yang sama dengan pernyataan if, kecuali dapat dibaca lebih banyak seperti “jika ini benar lakukan ini; jika tidak (lain), lakukan hal lain ini.” Pernyataan if/else ditulis seperti ini:

```
if ( <some Boolean condition> )
{
// Do something
}
else
{
// Do something else
}
```

Misalnya, jika Anda ingin mencetak “X lebih besar dari Y” ke Konsol jika variabel x lebih besar dari variabel y, atau Anda ingin mencetak “Y lebih besar dari X” jika x tidak lebih besar dari y, Anda bisa menulis berikut ini:

```
if (x > y)
{
print("X is greater than Y");
}
else
{
print("Y is greater than X");
}
```

Pernyataan if/else if

Terkadang Anda ingin kode Anda menyimpang ke salah satu dari banyak jalur. Anda mungkin ingin pengguna dapat memilih dari pilihan opsi (seperti menu, misalnya). Sebuah if / else if terstruktur dengan cara yang sama seperti dua struktur sebelumnya, kecuali bahwa ia memiliki beberapa kondisi:

```
if( <some Boolean condition> )
{
// Do something
}
else if ( <some other Boolean condition> )
{
// Do something else
}
else {
// The else is optional in the IF ELSE IF statement
/ Do something else
}
```

Misalnya, jika Anda ingin menampilkan nilai huruf seseorang ke Konsol berdasarkan persentasenya, Anda dapat menulis yang berikut:

```
if (grade >= 90) {
print("You got an A");
} else if (grade >= 80) {
print("You got a B");
} else if(grade >= 70) {
print("You got a C");
} else if (grade >= 60) {
print("You got a D");
} else {
print("You got an F");
}
```

While Loop/Perulangan while

Perulangan while adalah bentuk paling dasar dari iterasi. Ini mengikuti struktur yang mirip dengan pernyataan if:

```
While ( <some Boolean condition> )
{
// do something
}
```

Satu-satunya perbedaan adalah bahwa pernyataan if menjalankan kode yang dikandungnya hanya sekali, sedangkan perulangan menjalankan kode yang dikandungnya berulang-ulang sampai kondisinya menjadi salah. Oleh karena itu, jika Anda ingin menjumlahkan semua angka antara 1 dan 100 dan kemudian menampilkannya ke Konsol, Anda dapat menulis sesuatu seperti ini:

```
intsum = 0; int count = 1;
while (count <= 100) { sum += count; count++; }
print(sum);
```

Seperti yang Anda lihat, nilai count dimulai dari 1 dan bertambah 1 setiap iterasi —atau eksekusi loop—sampai sama dengan 101. Ketika count sama dengan 101, nilainya tidak kurang dari atau sama dengan 100, sehingga loop keluar . Menghilangkan baris count++ menghasilkan loop yang berjalan tanpa batas—jadi pastikan itu ada. Selama setiap iterasi dari loop, nilai count ditambahkan ke variabel sum. Saat loop keluar, jumlah ditulis ke Console. Singkatnya, while loop menjalankan kode yang dikandungnya berulang-ulang selama kondisinya benar. Ketika kondisinya menjadi salah, ia berhenti mengulang.

for loop

for loop mengikuti ide yang sama dengan perulangan while, kecuali strukturnya sedikit berbeda. Seperti yang Anda lihat dalam kode untuk loop sementara, Anda harus membuat variabel hitungan, Anda harus menguji variabel (sebagai kondisi), dan Anda harus meningkatkan variabel semua pada tiga baris terpisah. For loop memadatkan sintaks itu menjadi satu baris. Ini terlihat seperti ini:

```
for (<create a counter>; <Boolean conditional>; <increment the counter >)
{
// Do something
}
```

Loop for memiliki tiga kompartemen khusus untuk mengontrol loop. Perhatikan titik koma, bukan koma, di antara bagian di header loop for. Kompartemen pertama membuat variabel untuk digunakan sebagai penghitung. (Nama umum untuk penghitung adalah i, kependekan dari iterator.) Kompartemen kedua adalah pernyataan kondisional dari loop. Kompartemen ketiga menangani peningkatan atau penurunan penghitung. Contoh perulangan while sebelumnya dapat ditulis ulang menggunakan perulangan for. Ini akan terlihat seperti ini:

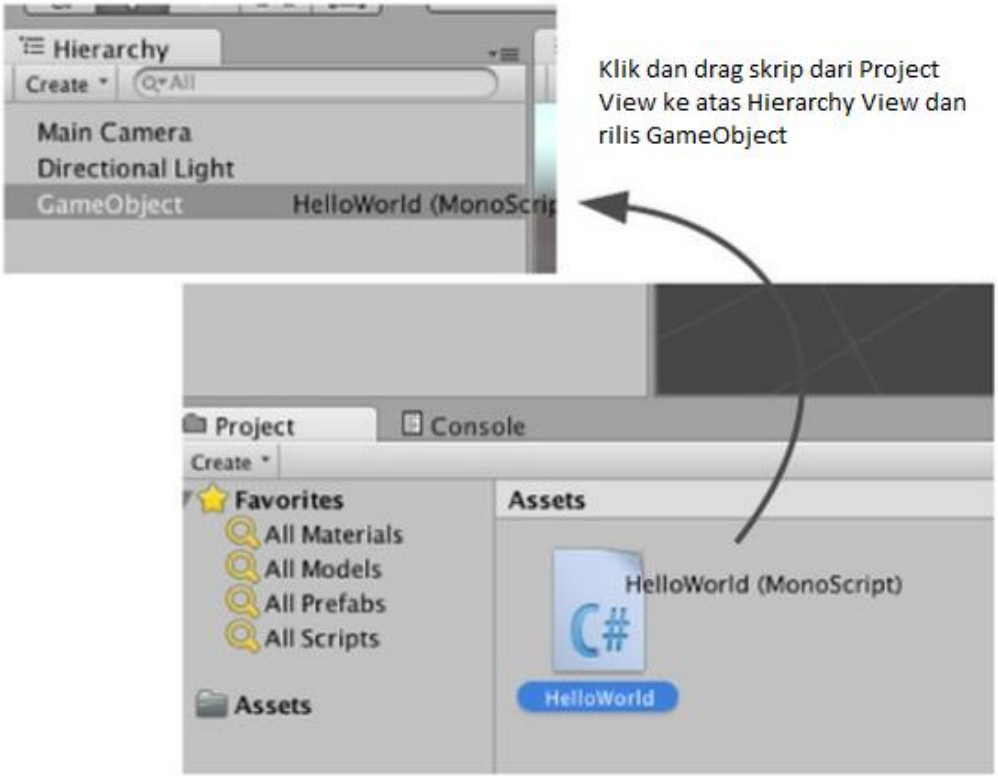
```
intsum = 0;
for (int count = 1; count <= 100; count++)
{
sum += count;
}
print(sum);
```

Seperti yang Anda lihat, bagian berbeda dari loop menjadi padat dan memakan lebih sedikit ruang. Anda dapat melihat bahwa for loop sangat bagus dalam hal-hal seperti menghitung.

Mencetak ke konsol: Halo Dunia!

Baiklah, Anda sudah memiliki skrip kosong di proyek, tetapi Anda juga memerlukan objek dalam scene untuk melampirkan skrip. Ingat gambar 1.1 yang menggambarkan bagaimana sistem komponen bekerja; skrip adalah komponen, sehingga perlu ditetapkan sebagai salah satu komponen pada suatu objek.

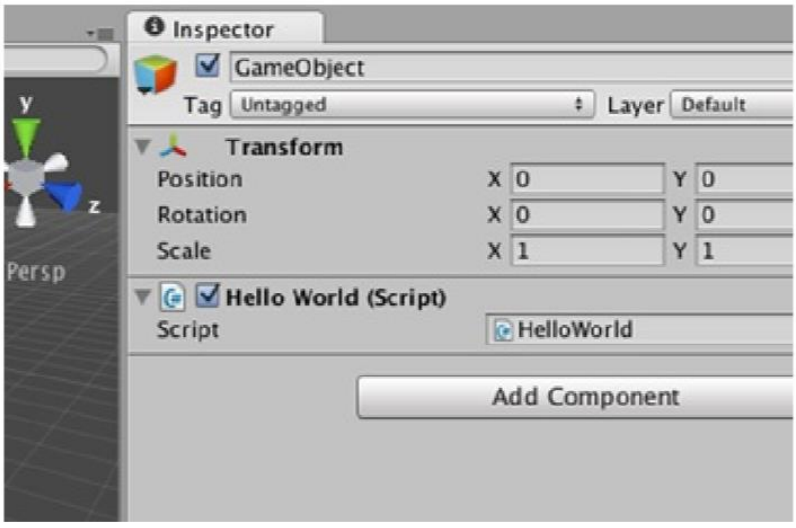
Pilih GameObject > Create Empty, dan GameObject kosong akan muncul di daftar Hierarchy. Sekarang drag skrip dari tampilan Proyek ke tampilan Hierarki dan letakkan di GameObject yang kosong. Seperti yang ditunjukkan pada gambar 1.15, Unity akan menyorot tempat yang valid untuk meletakkan skrip, dan meletakkannya di GameObject akan melampirkan skrip ke objek itu. Untuk memverifikasi bahwa skrip dilampirkan ke objek, pilih objek dan lihat tampilan Inspector. Anda akan melihat dua komponen terdaftar: komponen Transform yang merupakan komponen posisi/rotasi/scale dasar yang dimiliki semua objek dan yang tidak dapat dihapus, dan di bawahnya, skrip Anda.



Gambar 1.38 Cara menautkan skrip ke GameObject

Akhirnya tindakan menyeret objek dari satu tempat dan menjatuhkannya ke objek lain akan terasa rutin. Banyak tautan berbeda di Unity dibuat dengan menyeret sesuatu di atas satu sama lain, tidak hanya melampirkan skrip ke objek.

Saat skrip ditautkan ke objek, dengan skrip muncul sebagai komponen di Inspector. Sekarang skrip akan dijalankan saat Anda memutar scene, meskipun belum ada yang terjadi karena Anda belum menulis kode apa pun. Ayo lakukan selanjutnya!



Gambar 1.39 Skrip tertaut ditampilkan di Inspector

Buka skrip di MonoDevelop, tempat klasik untuk memulai ketika mempelajari lingkungan pemrograman baru adalah dengan mencetak teks "Hello World!" jadi tambahkan baris ini di dalam metode Start(), seperti yang ditunjukkan pada daftar berikut.

Gambar 1.40 Menambahkan pesan konsol

Apa yang dilakukan oleh perintah Debug.Log() adalah mencetak pesan ke tampilan Konsol di Unity. Sementara garis tersebut masuk ke dalam metode Start() karena, seperti yang telah dijelaskan sebelumnya, metode tersebut dipanggil segera setelah objek menjadi aktif. Dengan kata lain, Start() akan dipanggil sekali segera setelah Anda menekan Play di editor. Setelah Anda menambahkan perintah log ke skrip Anda (pastikan untuk menyimpan skrip), tekan Mainkan di Unity dan alihkan ke tampilan Konsol. Anda akan melihat pesan "Halo Dunia!" muncul. Selamat, Anda telah menulis skrip Unity pertama Anda! Dalam bab-bab selanjutnya, kodenya akan lebih rumit, tentu saja, tetapi ini adalah langkah pertama yang penting.

1.11 METODE

Metode, sering disebut fungsi, adalah modul kode yang dapat dipanggil dan digunakan secara independen satu sama lain. Setiap metode umumnya mewakili satu tugas atau tujuan, dan seringkali banyak metode dapat bekerja sama untuk mencapai tujuan yang kompleks. Pertimbangkan dua metode yang telah Anda lihat sejauh ini: Mulai dan Perbarui. Masing-masing mewakili satu tujuan tunggal dan ringkas. Metode Mulai berisi semua kode yang dijalankan untuk objek saat scene pertama kali dimulai. Metode Pembaruan berisi kode yang dijalankan setiap bingkai pembaruan scene (yang berbeda dari bingkai render, atau "bingkai per detik").

Metode Singkatan

Anda telah melihat sejauh ini bahwa setiap kali metode Mulai disebutkan, metode kata telah mengikutinya. Ini bisa menjadi rumit untuk selalu harus menentukan bahwa kata yang digunakan adalah metode. Anda tidak dapat menulis hanya Mulai, karena orang tidak akan tahu apakah yang Anda maksud adalah kata, variabel, atau metode. Cara yang lebih singkat untuk menangani ini adalah dengan menggunakan tanda kurung dengan kata. Jadi, metode Start dapat ditulis ulang hanya sebagai Start(). Jika Anda pernah melihat sesuatu yang ditulis seperti SomeWords(), Anda langsung tahu bahwa penulis sedang berbicara tentang metode bernama SomeWords.

Anatomi Metode

Sebelum bekerja dengan metode, Anda harus melihat bagian-bagian berbeda yang menyusunnya. Berikut ini adalah format umum dari sebuah metode:

<return type> <name> (<parameter list>)
{
<Inside the method's block>
}

Nama Metode

Setiap metode harus memiliki nama yang unik (semacam…lihat bagian “Tanda Tangan Metode” di bawah). Meskipun aturan yang mengatur nama diri ditentukan oleh bahasa yang digunakan, pedoman umum yang baik untuk nama metode meliputi yang berikut:

- Buat nama metode deskriptif. Itu harus berupa tindakan atau kata kerja.
- Spasi tidak diperbolehkan. Hindari penggunaan karakter khusus (misalnya, !, @, *, %, $) dalam nama metode.

- Bahasa yang berbeda memungkinkan karakter yang berbeda. Dengan tidak menggunakan apa pun, Anda menghindari risiko masalah.

Nama metode penting karena memungkinkan Anda mengidentifikasi dan menggunakannya.

Tipe Pengembalian

Setiap metode memiliki kemampuan untuk mengembalikan variabel kembali ke kode apa pun yang disebut. Tipe dari variabel ini disebut tipe kembalian. Jika suatu metode mengembalikan bilangan bulat (bilangan bulat), tipe pengembaliannya adalah int. Demikian juga, jika suatu metode mengembalikan nilai true atau false, tipe pengembaliannya adalah bool. Jika suatu metode tidak mengembalikan nilai apa pun, metode itu masih memiliki tipe pengembalian. Dalam hal itu, tipe pengembalian adalah batal (tidak berarti apa-apa). Metode apa pun yang mengembalikan nilai akan melakukannya dengan kata kunci return.

Daftar Parameter

Sama seperti metode dapat meneruskan variabel kembali ke kode apa pun yang disebut, kode panggilan dapat meneruskan variabel. Variabel ini disebut parameter. Variabel yang dikirim ke metode diidentifikasi di bagian daftar parameter metode. Sebagai contoh, sebuah metode bernama Attack yang mengambil bilangan bulat yang disebut musuh ID akan terlihat seperti ini:

```
void    Attack(int enemyID) {
}
```

Seperti yang Anda lihat, saat menentukan parameter, Anda harus memberikan tipe dan nama variabel. Beberapa parameter dipisahkan dengan koma.

Tanda Tangan Metode

Kombinasi dari tipe pengembalian metode, nama, dan daftar parameter sering disebut sebagai tanda tangan metode. Sebelumnya pada jam ini saya menyebutkan bahwa suatu metode harus memiliki nama yang unik, tetapi itu tidak sepenuhnya benar. Yang benar adalah bahwa suatu metode harus memiliki tanda tangan yang unik. Karena itu, pertimbangkan dua metode ini:

```
void    MyMethod()
{}
void    MyMethod(int number)
{}
```

Meskipun kedua metode ini memiliki nama yang sama, mereka memiliki daftar parameter yang berbeda dan karenanya berbeda. Praktek memiliki metode yang berbeda dengan nama yang sama juga bisa disebut kelebihan metode.

Blok Metode

Blok metode adalah tempat kode metode berada. Setiap kali metode digunakan, kode di dalam blok metode dieksekusi.

Mengidentifikasi Bagian Metode

Luangkan waktu sejenak untuk meninjau bagian-bagian berbeda dari suatu metode dan kemudian pertimbangkan metode berikut: Klik di sini untuk melihat gambar kode

```
int TakeDamage(int damageAmount)
{
int health = 100;
return health - damageAmount;
}
```

Dapatkah Anda mengidentifikasi potongan-potongan berikut?

1. Apa nama metodenya?
2. Jenis variabel apa yang dikembalikan oleh metode?

3. Apa parameter metodenya? Ada berapa banyak?
4. Kode apa yang ada di blok metode?

Metode sebagai Pabrik

Konsep metode dapat membingungkan bagi seseorang yang baru mengenal pemrograman. Seringkali, kesalahan dibuat mengenai parameter metode dan metode apa yang dikembalikan. Cara yang baik untuk menjaganya tetap lurus adalah dengan memikirkan metode sebagai pabrik. Pabrik menerima bahan mentah yang mereka gunakan untuk membuat produk. Metode bekerja dengan cara yang sama. Parameternya adalah bahan yang Anda masukkan ke "pabrik", dan pengembaliannya adalah produk akhir dari pabrik itu. Pikirkan saja metode yang tidak mengambil parameter sebagai pabrik yang tidak membutuhkan barang mentah. Demikian juga, pikirkan metode yang tidak mengembalikan apa pun sebagai pabrik yang tidak menghasilkan produk akhir. Dengan membayangkan metode sebagai pabrik kecil, Anda dapat bekerja untuk menjaga aliran logika tetap lurus di kepala Anda.

Metode Penulisan

Ketika Anda memahami komponen suatu metode, menulisnya menjadi mudah. Sebelum Anda mulai menulis metode sendiri, luangkan waktu sejenak untuk menjawab tiga pertanyaan utama:

- Tugas spesifik apa yang akan dicapai metode tersebut?
- Apakah metode memerlukan data luar untuk mencapai tugasnya?
- Apakah metode ini perlu mengembalikan data apa pun?

Menjawab pertanyaan-pertanyaan ini akan membantu Anda menentukan nama metode, parameter, dan data yang dikembalikan. Perhatikan contoh ini: Seorang player dipukul dengan bola api. Anda perlu menulis metode untuk mensimulasikan ini dengan menghapus 5 poin kesehatan. Anda tahu apa tugas spesifik dari metode ini. Anda juga tahu bahwa tugas tidak memerlukan data apa pun (karena Anda tahu ini membutuhkan 5 poin) dan mungkin harus mengembalikan nilai kesehatan yang baru. Anda bisa menulis metode seperti ini:

```
int TakeDamageFromFireball()
{ int playerHealth = 100;
return playerHealth - 5;
}
```

Seperti yang Anda lihat dalam metode ini, kesehatan player adalah 100, dan 5 dikurangi darinya. Hasilnya (yaitu 95) dilewatkan kembali. Jelas, ini bisa ditingkatkan. Sebagai permulaan, bagaimana jika Anda ingin bola api menghasilkan damage lebih dari 5 poin? Anda kemudian perlu tahu persis berapa banyak kerusakan yang seharusnya dilakukan bola api pada waktu tertentu. Anda akan membutuhkan variabel, atau dalam hal ini parameter. Metode baru Anda dapat ditulis sebagai berikut:

```
int TakeDamageFromFireball(int damage)
{
int playerHealth = 100;
return playerHealth - damage;
}
```

Sekarang Anda dapat melihat bahwa kerusakan dibaca dari metode dan diterapkan pada kesehatan. Tempat lain di mana ini dapat ditingkatkan adalah dengan kesehatan itu sendiri. Saat ini, player tidak akan pernah kalah karena kesehatan mereka akan selalu disegarkan kembali ke 100 sebelum kerusakan dikurangi. Akan lebih baik untuk menyimpan kesehatan

player di tempat lain sehingga nilainya tetap. Anda kemudian bisa membacanya dan menghapus kerusakan dengan tepat. Metode Anda kemudian dapat terlihat seperti ini:

```
int TakeDamageFromFireball(int damage, int playerHealth)
{
return playerHealth - damage;
}
```

Dengan memeriksa kebutuhan Anda, Anda dapat membangun metode yang lebih baik dan lebih kuat untuk game Anda.

Penyederhanaan

Dalam contoh sebelumnya, metode yang dihasilkan hanya melakukan pengurangan dasar. Ini terlalu disederhanakan demi instruksi. Dalam lingkungan yang lebih realistis, ada banyak cara untuk menangani tugas ini. Kesehatan player dapat disimpan dalam variabel milik skrip. Dalam hal ini, itu tidak perlu dibaca. Kemungkinan lain adalah menggunakan algoritma kompleks dalam metode TakeDamageFromFireball untuk mengurangi kerusakan yang masuk dengan beberapa nilai armor, kemampuan menghindar player, atau perisai magis. Jika contoh di sini tampak konyol, ingatlah bahwa itu dimaksudkan untuk menunjukkan berbagai elemen topik.

Menggunakan Metode

Setelah metode ditulis, yang tersisa hanyalah menggunakannya. Menggunakan metode sering disebut sebagai memanggil atau memanggil metode. Untuk memanggil metode, Anda hanya perlu menulis nama metode diikuti dengan tanda kurung dan parameter apa pun. Jadi, jika Anda mencoba menggunakan metode bernama SomeMethod, Anda akan menulis yang berikut:

```
SomeMethod();
```

Jika SomeMethod() memerlukan parameter integer, Anda menyebutnya seperti ini:

```
// Method call with a value of 5
SomeMethod(5);
// Method call passing in a variable int x = 5;
SomeMethod(x);
// do not write "int x" here.
```

Perhatikan bahwa saat Anda memanggil metode, Anda tidak perlu menyediakan tipe variabel dengan variabel yang Anda lewati. Jika SomeMethod() mengembalikan nilai, Anda ingin menangkapnya dalam variabel. Kodenya bisa terlihat seperti ini (dengan asumsi tipe pengembalian Boolean; pada kenyataannya, bisa apa saja):

```
bool    result = SomeMethod();
```

Menggunakan sintaks dasar ini adalah semua yang ada untuk memanggil metode.

Memanggil Metode

Mari kita bekerja lebih jauh dengan metode TakeDamageFromFireball yang dijelaskan di bagian sebelumnya. Latihan ini menunjukkan bagaimana memanggil berbagai bentuk metode. (Anda dapat menemukan solusi untuk latihan ini sebagai FireBallScript di aset buku untuk Jam 8.) Ikuti langkah-langkah berikut:

1. Buat proyek atau scene baru. Buat skrip C# bernama FireBallScript dan masukkan tiga metode TakeDamageFromFireball yang dijelaskan sebelumnya. Ini harus masuk ke dalam definisi kelas, pada tingkat indentasi yang sama dengan metode Start() dan Update(), tetapi di luar kedua metode tersebut.

2. Pada metode Start, panggil metode TakeDamageFromFireball() pertama dengan mengetikkan perintah berikut:

```
int x = TakeDamageFromFireball();
print("Player health: " + x);
```

3. Lampirkan skrip ke Kamera Utama dan jalankan scenenya. Perhatikan output di Console. Sekarang panggil metode TakeDamageFromFireball() kedua di Start() dengan mengetik berikut ini (letakkan di bawah bit pertama kode yang Anda ketik; tidak perlu menghapusnya):

```
int y = TakeDamageFromFireball(25);
print("Player health: " + y);
```

4. Sekali lagi, jalankan scene dan catat output di Console. Terakhir, panggil metode TakeDamageFromFireball() terakhir di Start() dengan mengetik berikut ini:

```
int z = TakeDamageFromFireball(30, 50);
print("Player health: " + z);
```

5. Jalankan scene dan catat hasil akhirnya. Perhatikan bagaimana ketiga metode berperilaku sedikit berbeda satu sama lain. Perhatikan juga bahwa Anda memanggil masing-masing secara khusus, dan versi yang benar dari metode TakeDamageFromFireball() digunakan berdasarkan parameter yang Anda masukkan.

Bantuan Menemukan Kesalahan

Jika Anda mendapatkan kesalahan saat mencoba menjalankan skrip, perhatikan nomor baris dan nomor karakter yang dilaporkan di Konsol, di akhir pesan kesalahan. Selanjutnya, Anda dapat "membangun" kode Anda di dalam Visual Studio dengan menggunakan Ctrl+Shift+B (Command+Shift+B pada Mac). Saat Anda melakukan ini, Visual Studio memeriksa kode Anda dan menunjukkan kesalahan apa pun dalam konteks, menunjukkan dengan tepat di mana garis bermasalah itu. Cobalah.

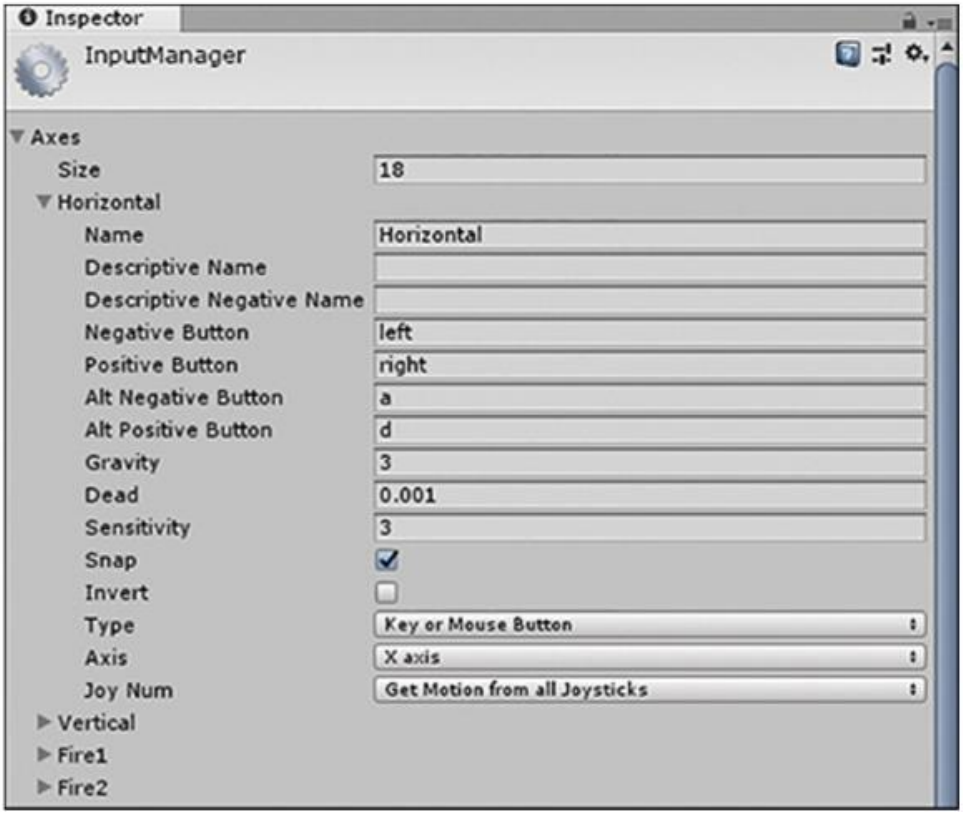
1.12 INPUT

Tanpa input player, video game hanya akan menjadi video. Input player bisa datang dalam berbagai variasi. Input dapat berupa fisik—misalnya, gamepad, joystick, keyboard, dan mouse. Ada controller kapasitif seperti layar sentuh yang relatif baru di perangkat seluler modern. Ada juga perangkat gerak seperti Wii Remote, PlayStation Move, dan Microsoft Kinect. Rarer adalah input audio yang menggunakan mikrofon dan suara player untuk mengontrol game. Di bagian ini, Anda akan mempelajari semua tentang menulis kode untuk memungkinkan player berinteraksi dengan game Anda menggunakan perangkat fisik.

Dasar Input

Dengan Unity (seperti kebanyakan game engine lainnya), Anda dapat mendeteksi penekanan tombol tertentu dalam kode untuk membuatnya interaktif. Namun, hal itu membuat sulit untuk mengizinkan player memetakan kembali kontrol sesuai keinginan mereka, jadi sebaiknya hindari melakukan hal itu. Untungnya, Unity memiliki sistem sederhana untuk memetakan kontrol secara umum. Dengan Unity, Anda mencari sumbu tertentu untuk mengetahui apakah seorang player menginginkan tindakan tertentu. Kemudian, ketika player menjalankan permainan, ia dapat memilih untuk membuat kontrol yang berbeda berarti sumbu yang berbeda. Anda dapat melihat, mengedit, dan menambahkan sumbu yang berbeda dengan menggunakan Manajer Input. Untuk mengakses

Manajer Input, klik Edit > Pengaturan Proyek > Input. Di Manajer Input, Anda dapat melihat berbagai sumbu yang terkait dengan tindakan input yang berbeda. Secara default, ada 18 sumbu input, tetapi Anda dapat menambahkan sumbu sendiri jika diinginkan. Gambar 8.1 menunjukkan Manajer Input default dengan sumbu horizontal diperluas.



Gambar 1.41 Input manager

Manajer Input.

Sementara sumbu horizontal tidak secara langsung mengontrol apa pun (Anda akan menulis skrip untuk melakukannya nanti), itu menunjukkan bahwa player bergerak ke samping.

Tabel 1.6 Properti Sumbu

Properti	**Deskripsi**
Name	Nama sumbu. Ini adalah bagaimana Anda mereferensikannya dalam kode.
Descriptive Name/ Descriptive Negative Name	Nama verbose untuk sumbu yang akan muncul ke player dalam konfigurasi game. Negatif adalah nama yang berlawanan. Misalnya: "Ke kiri" dan "Ke kanan" akan menjadi pasangan nama dan nama negatif.
Negative Button/ Positive Button	Tombol yang meneruskan nilai negatif dan positif ke sumbu. Untuk sumbu horizontal, ini adalah panah kiri dan tombol panah kanan.
Alt Negative Button/ Alt Positive Button	Tombol alternatif untuk meneruskan nilai ke sumbu. Untuk sumbu horizontal, ini adalah tombol A dan D.

Gravity	Seberapa cepat sumbu kembali ke 0 setelah tombol tidak lagi ditekan.
Dead	Nilai di bawah mana input apa pun akan diabaikan. Ini membantu mencegah jittering dengan perangkat joystick.
Sensitivity	Seberapa cepat sumbu merespons input.
Snap	Saat dicentang, Snap menyebabkan sumbu langsung menuju ke 0 saat arah yang berlawanan ditekan.
Invert	Memeriksa properti ini akan membalikkan kontrol.
Type	Jenis input. Jenisnya adalah tombol keyboard/mouse, gerakan mouse, dan gerakan joystick.
Axis	Sumbu yang sesuai dari perangkat input. Ini tidak berlaku untuk tombol.
Joy Num	Joystick mana untuk mendapatkan input. Secara default, properti ini mendapat input dari semua joystick.

Skrip Input

Setelah sumbu Anda diatur di Manajer Input, bekerja dengannya dalam kode menjadi sederhana. Untuk mengakses salah satu input player, Anda akan menggunakan objek Input. Lebih khusus lagi, Anda akan menggunakan metode GetAxis dari objek Input. GetAxis() membaca nama sumbu sebagai string dan mengembalikan nilai sumbu tersebut. Jadi, jika Anda ingin mendapatkan nilai sumbu horizontal, ketikkan yang berikut ini:

```
Float hVal = Input.GetAxis("Horizontal");
```

Dalam kasus sumbu horizontal, jika player menekan tombol panah kiri (atau tombol A), GetAxis() mengembalikan angka negatif. Jika player menekan tombol panah kanan (atau tombol D), metode mengembalikan nilai positif.

Membaca di Input Pengguna

Ikuti langkah-langkah ini untuk bekerja dengan sumbu vertikal dan horizontal dan dapatkan ide yang lebih baik tentang cara menggunakan input player:

1. Buat proyek atau scene baru. Tambahkan skrip bernama PlayerInput ke proyek dan lampirkan skrip ke Kamera Utama.
2. Tambahkan kode berikut ke metode Update di skrip PlayerInput:

```
Float hVal = Input.GetAxis("Horizontal"); float vVal =
Input.GetAxis("Vertical"); if(hVal != 0)
print("Horizontal movement selected: " + hVal); if(vVal != 0)
print("Vertical
movement selected: " + vVal);
```

Kode ini harus dalam Pembaruan agar terus membaca input.

3. Simpan skrip dan jalankan scene. Perhatikan apa yang terjadi di Konsol saat Anda menekan tombol panah. Sekarang cobalah kunci W, A, S, dan D. Jika Anda tidak melihat apa pun, klik di jendela Game dengan mouse dan coba lagi.

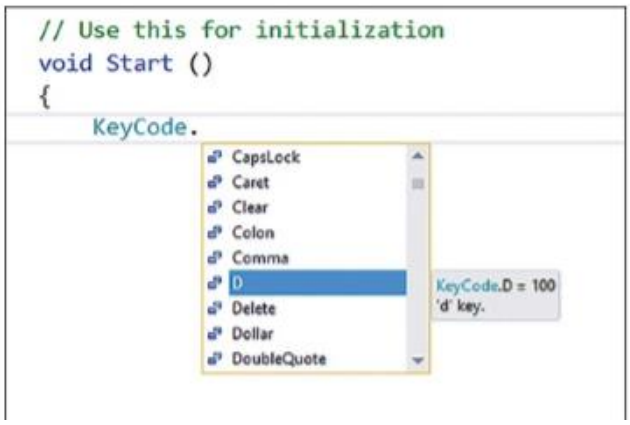
Input Kunci Spesifik

Meskipun Anda biasanya ingin berurusan dengan sumbu umum untuk input, terkadang Anda ingin menentukan apakah tombol tertentu telah ditekan. Untuk melakukannya, Anda akan kembali menggunakan objek Input. Namun kali ini, Anda akan menggunakan metode GetKey, yang membaca kode khusus yang sesuai dengan kunci tertentu. Kemudian mengembalikan true jika kunci saat ini turun atau salah jika kunci saat ini tidak turun. Untuk menentukan apakah tombol K saat ini ditekan, Anda mengetik berikut ini:

bool isKeyDown = Input.GetKey(KeyCode.K);

Menemukan Kode Kunci

Setiap kunci memiliki kode kunci tertentu. Anda dapat menentukan kode kunci dari kunci tertentu yang Anda inginkan dengan membaca dokumentasi Unity. Atau, Anda dapat menggunakan alat bawaan Visual Studio untuk menemukannya. Setiap kali Anda mengerjakan skrip di Visual Studio, Anda selalu dapat mengetikkan nama objek yang diikuti dengan titik. Ketika Anda melakukannya, menu dengan semua opsi yang mungkin muncul. Demikian juga, jika Anda mengetikkan kurung buka setelah mengetikkan nama metode, menu yang sama akan muncul, menunjukkan kepada Anda berbagai opsi. Gambar 8.2 mengilustrasikan penggunaan menu pop-up ini untuk menemukan kode kunci untuk tombol Esc.



Gambar 1.42 Pop up otomatis di Visual Studio

Membaca dalam Penekanan Tombol Tertentu

Ikuti langkah-langkah berikut untuk menulis skrip yang menentukan apakah tombol tertentu ditekan:

1. Buat proyek atau scene baru. Tambahkan ke proyek skrip bernama PlayerInput (atau ubah yang sudah ada) dan lampirkan ke Kamera Utama.
2. Tambahkan kode berikut ke metode Update di skrip PlayerInput:

```
if(Input.GetKey(KeyCode.M))
print("The 'M' key is pressed down");
if(Input.GetKeyDown(KeyCode.O))
print("The 'O' key was pressed");
```

3. Simpan skrip dan jalankan scene. Perhatikan apa yang terjadi saat Anda menekan tombol M versus apa yang terjadi saat Anda menekan tombol O. Secara khusus, perhatikan bahwa tombol M menyebabkan output sepanjang waktu ditahan, sedangkan tombol O hanya mengeluarkan output saat pertama kali ditekan.

"Harus Bangun untuk Turun"

Dalam latihan Coba Sendiri "Membaca dengan Penekanan Tombol Tertentu", Anda memeriksa input tombol dengan dua cara berbeda. Yaitu, Anda menggunakan Input.GetKey() untuk menguji apakah tombol ditekan sama sekali. Anda juga menggunakan Input.GetKeyDown() untuk menguji apakah tombol ditekan selama frame ini. Versi kedua ini hanya terdaftar saat pertama kali tombol ditekan dan mengabaikan Anda yang menahan tombol. Secara umum, Unity mencari tiga jenis peristiwa penekanan tombol: GetKey(), GetKeyDown(), dan GetKeyUp(). Ini juga konsisten di seluruh metode serupa lainnya:

GetButton(), GetButtonDown(), GetButtonUp(), GetMouseButton(), GetMouseButtonDown(), dan seterusnya. Mengetahui kapan harus mencari penekanan tombol pertama versus tombol yang ditahan adalah penting. Apapun jenis input yang Anda butuhkan, ada metode untuk itu.

Input Mouse

Selain menangkap penekanan tombol, Anda ingin menangkap input mouse dari pengguna. Ada dua komponen untuk input mouse: penekanan tombol mouse dan gerakan mouse. Menentukan apakah tombol mouse ditekan sama seperti mendeteksi penekanan tombol, dibahas lebih awal jam ini. Di bagian ini, Anda akan kembali menggunakan objek Input. Kali ini Anda akan menggunakan metode GetMouseButtonDown, yang mengambil bilangan bulat antara 0 dan 2 untuk menentukan tombol mouse mana yang Anda tanyakan. Metode mengembalikan nilai Boolean yang menunjukkan apakah tombol ditekan. Kode untuk menekan tombol mouse terlihat seperti ini:

```
bool isButtonDown;
isButtonDown = Input.GetMouseButtonDown(0);  // left mouse button
isButtonDown = Input.GetMouseButtonDown(1);  // right mouse button
isButtonDown = Input.GetMouseButtonDown(2);  // center mouse button
```

Gerakan mouse hanya sepanjang dua sumbu: x dan y. Untuk mendapatkan gerakan mouse, Anda menggunakan metode GetAxis dari objek input. Anda dapat menggunakan nama Mouse X dan Mouse Y untuk mendapatkan gerakan di sepanjang sumbu x dan sumbu y, masing-masing. Kode untuk dibaca dalam gerakan mouse akan terlihat seperti ini:

```
float values;
value = Input.GetAxis("Mouse X"); // x axis movement
value = Input.GetAxis("Mouse Y"); // y axis movement
```

Tidak seperti penekanan tombol, gerakan mouse diukur dengan jumlah gerakan mouse sejak frame terakhir saja. Pada dasarnya, memegang kunci menyebabkan nilai meningkat hingga maksimal pada -1 atau 1 (tergantung apakah itu positif atau negatif). Gerakan mouse, bagaimanapun, umumnya memiliki angka yang lebih kecil karena diukur dan diatur ulang setiap frame.

Membaca Gerakan Mouse

Dalam latihan ini, Anda akan membaca gerakan mouse dan menampilkan hasilnya ke Konsol:

1. Buat proyek atau scene baru. Tambahkan ke proyek skrip bernama PlayerInput (atau ubah yang sudah ada) dan lampirkan ke Kamera Utama.
2. Tambahkan kode berikut ke metode Update di skrip PlayerInput:

```
float mxVal = Input.GetAxis("Mouse X");
float myVal = Input.GetAxis("Mouse Y");
if(mxVal != 0)
print("Mouse X movement selected: " + mxVal);
if(myVal != 0)
print("Mouse Y movement selected: " + myVal);
```

3. Simpan skrip dan jalankan scene. Baca Konsol untuk melihat output saat Anda menggerakkan mouse.

Mengakses Komponen Lokal

Seperti yang telah Anda lihat berkali-kali dalam tampilan Inspector, objek terdiri dari berbagai komponen. Selalu ada komponen transformasi, dan secara opsional mungkin ada sejumlah komponen lain, seperti Renderer, Light, dan Camera. Skrip juga merupakan komponen, dan bersama-sama komponen ini memberikan perilaku objek game.

Menggunakan GetComponent

Anda dapat berinteraksi dengan komponen saat runtime melalui skrip. Hal pertama yang harus Anda lakukan adalah mendapatkan referensi ke komponen yang ingin Anda kerjakan. Anda harus melakukan ini di Start() dan menyimpan hasilnya dalam sebuah variabel. Dengan cara ini, Anda tidak perlu membuang waktu untuk mengulangi operasi yang relatif lambat ini. Metode GetComponent<Type>() memiliki sintaks yang sedikit berbeda dari yang Anda lihat hingga saat ini, menggunakan chevron untuk menentukan jenis yang Anda cari (misalnya, Light, Camera, nama skrip). GetComponent() mengembalikan komponen pertama dari jenis yang ditentukan yang dilampirkan ke objek game yang sama dengan skrip Anda. Seperti yang disebutkan sebelumnya di jam ini, Anda kemudian harus menetapkan komponen ini ke variabel lokal sehingga Anda dapat mengaksesnya nanti. Inilah cara Anda melakukannya:

Menggunakan GetComponent

Anda dapat berinteraksi dengan komponen saat runtime melalui skrip. Hal pertama yang harus Anda lakukan adalah mendapatkan referensi ke komponen yang ingin Anda kerjakan. Anda harus melakukan ini di Start() dan menyimpan hasilnya dalam sebuah variabel. Dengan cara ini, Anda tidak perlu membuang waktu untuk mengulangi operasi yang relatif lambat ini. Metode GetComponent<Type>() memiliki sintaks yang sedikit berbeda dari yang Anda lihat hingga saat ini, menggunakan chevron untuk menentukan jenis yang Anda cari (misalnya, Light, Camera, nama skrip). GetComponent() mengembalikan komponen pertama dari jenis yang ditentukan yang dilampirkan ke objek game yang sama dengan skrip Anda. Seperti yang disebutkan sebelumnya di jam ini, Anda kemudian harus menetapkan komponen ini ke variabel lokal sehingga Anda dapat mengaksesnya nanti. Inilah cara Anda melakukannya:

```
lightlightComponent; // A variable to store the light component.
Start()
{
lightComponent = GetComponent<Light>();
lightComponent.type = LightType.Directional;
}
```

Setelah Anda memiliki referensi ke suatu komponen, Anda dapat dengan mudah memodifikasi propertinya melalui kode. Anda melakukannya dengan mengetikkan nama variabel yang menyimpan referensi diikuti dengan titik diikuti oleh properti apa pun yang ingin Anda ubah. Pada contoh di atas, Anda mengubah properti tipe komponen cahaya menjadi Directional.

Mengakses Transform

Komponen yang paling sering Anda gunakan adalah komponen transformasi. Dengan mengeditnya, Anda dapat membuat objek bergerak di sekitar layar. Ingatlah bahwa transformasi suatu objek terdiri dari translasi (atau posisinya), rotasinya, dan scalenya. Meskipun Anda dapat memodifikasinya secara langsung, lebih mudah menggunakan

beberapa opsi bawaan yang disebut metode Translate(), metode Rotate(), dan variabel localScale, seperti yang ditunjukkan di sini:

```
// Moves the object along the positive x axis.
// The '0f' means 0 is a float (floating point number). It is the way Unity reads floats transform.Translate(0.05f, 0f, 0f);
// Rotates the object along the z axis transform.Rotate(0f, 0f, 1f);
// Scales the object to double its size in all directions transform.localScale = new Vector3(2f, 2f, 2f);
```

Menemukan Transformasi

Karena setiap objek game memiliki transformasi, tidak perlu melakukan operasi pencarian eksplisit; Anda dapat mengakses transformasi secara langsung seperti di atas. Ini adalah satu-satunya komponen yang bekerja dengan cara ini; sisanya harus diakses menggunakan metode GetComponent.

Karena Translate() dan Rotate() adalah metode, jika kode sebelumnya dimasukkan ke Update(), objek akan terus bergerak sepanjang sumbu x positif sambil diputar di sepanjang sumbu z.

Mengubah Objek

Ikuti langkah-langkah ini untuk melihat kode sebelumnya beraksi dengan menerapkannya ke objek dalam sebuah scene:

1. Buat proyek atau scene baru. Tambahkan kubus ke scene dan posisikan di (0, -1, 0).
2. Buat skrip baru dan beri nama CubeScript. Tempatkan skrip pada kubus. Di Visual Studio, masukkan kode berikut ke metode Pembaruan:
   ```
   transform.Translate(.05f, 0f, 0f);
   transform.Rotate(0f, 0f, 1f);
   transform.localScale = new Vector3(1.5f, 1.5f, 1.5f);
   ```
3. Simpan skrip dan jalankan scene. Anda mungkin perlu pindah ke tampilan Scene untuk melihat gerakan penuh. Perhatikan bahwa efek dari metode Translate() dan Rotate() bersifat kumulatif, dan variabel localScale tidak; localScale tidak terus berkembang.

Mengakses Objek Lain

Sering kali, Anda ingin skrip dapat menemukan dan memanipulasi objek lain dan komponennya. Melakukannya hanyalah masalah menemukan objek yang Anda inginkan dan memanggil komponen yang sesuai. Ada beberapa cara dasar untuk menemukan objek yang tidak lokal ke skrip atau objek yang dilampirkan skrip.

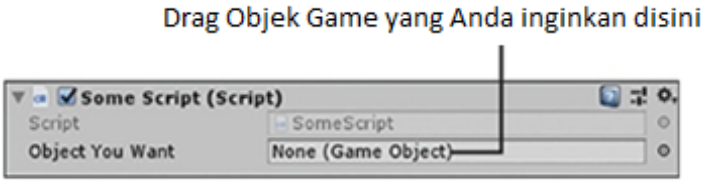
Menemukan Objek Lain

Cara pertama dan termudah untuk menemukan objek lain untuk dikerjakan adalah dengan menggunakan editor. Dengan membuat variabel publik pada level kelas tipe GameObject, Anda cukup menyeret objek yang Anda inginkan ke komponen skrip dalam tampilan Inspector. Kode untuk mengatur ini terlihat seperti ini:

```
// This is here for reference
public class SomeClassScript : MonoBehaviour
{
// This is the game object you want to access
public GameObject objectYouWant;
// This is here for reference void Start()
{ }
```

}

Setelah Anda melampirkan skrip ke objek game, Anda akan melihat properti di Inspector yang disebut Objek yang Anda Inginkan. Cukup drag objek game apa pun yang Anda inginkan ke properti ini untuk mengaksesnya di skrip.



Gambar 1.43 Properti Obyek yang Anda Inginkan baru di Inspector.

Cara lain untuk menemukan objek game adalah dengan menggunakan salah satu metode Find. Sebagai aturan praktis, jika Anda ingin seorang desainer dapat menghubungkan objek, atau jika itu opsional, maka hubungkan melalui Inspector. Jika memutuskan sambungan akan merusak permainan, maka gunakan metode Temukan. Ada tiga cara utama untuk menemukan menggunakan skrip: menurut nama, menurut tag, dan menurut jenis. Salah satu opsi adalah mencari berdasarkan nama objek. Nama objek adalah apa yang disebut di dalam tampilan Hierarki. Jika Anda mencari objek bernama Cube, kodenya akan terlihat seperti ini:

```
// This is here for reference
public class SomeClassScript : MonoBehaviour
{
        // This is the game object you want to access
        private GameObject target;
        // Note this doesn't need to be public if using find.
        // This is here for reference void Start()
        {
        target = GameObject.Find("Cube");
        }
}
```

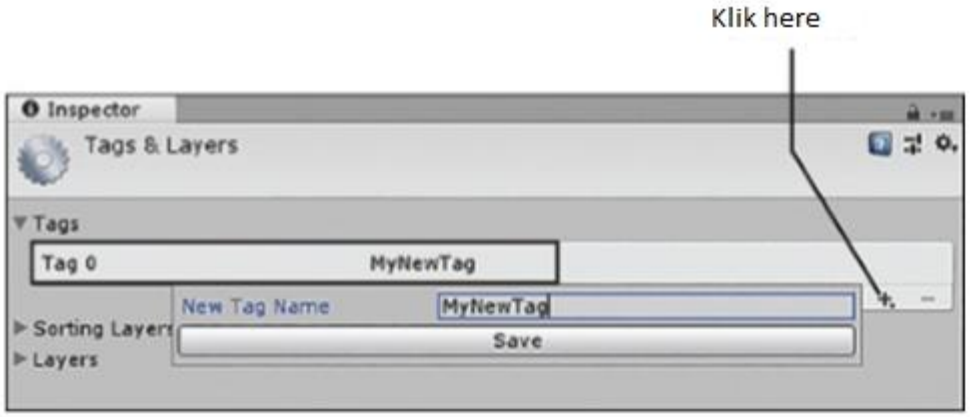
Kekurangan dari metode ini adalah hanya mengembalikan item pertama yang ditemukan dengan nama yang diberikan. Jika Anda memiliki beberapa objek Cube, Anda tidak akan tahu yang mana yang Anda dapatkan.

Menemukan Efisiensi

Ketahuilah bahwa menggunakan metode Find() sangat lambat karena ia mencari setiap objek game dalam scene, secara berurutan, hingga menemukan kecocokan. Dalam scene yang sangat besar, jumlah waktu yang dibutuhkan metode ini dapat menyebabkan penurunan framerate yang nyata dalam game Anda. Dianjurkan untuk tidak pernah menggunakan Find() jika Anda dapat menghindarinya. Yang sedang berkata, ada banyak waktu ketika itu diperlukan. Namun, dalam kasus ini, sangat bermanfaat untuk memanggil metode di Start() dan menyimpan hasil temuan dalam variabel untuk digunakan di masa mendatang guna meminimalkan dampak.

Cara lain untuk menemukan objek adalah dengan tag-nya. Tag suatu objek sangat mirip dengan layernya (dibahas sebelumnya pada jam ini). Satu-satunya perbedaan adalah semantik. Layer digunakan untuk kategori interaksi yang luas, sedangkan tag digunakan untuk

identifikasi dasar. Anda membuat tag menggunakan Pengelola Tag (klik Edit > Setelan Proyek > Tag & Layer). Gambar 8.4 menunjukkan cara menambahkan tag baru ke Tag Manager.



Gambar 1.44 Menambahkan tag baru.

Setelah tag dibuat, cukup terapkan ke objek dengan menggunakan daftar drop-down Tag di tampilan Inspector

Gambar 1.45 Memilih tag.

Setelah tag ditambahkan ke objek, Anda dapat menemukannya dengan menggunakan metode FindWithTag():

```
// This is here for reference public class SomeClassScript : MonoBehaviour
{
    // This is the game object you want to access private GameObject target;
    // This is here for reference void Start()
    {
    target = GameObject.FindWithTag("MyNewTag");
    }
}
```

Ada metode Find() tambahan yang tersedia, tetapi yang dibahas di sini akan bekerja untuk Anda di sebagian besar situasi.

Memodifikasi Komponen Objek

Setelah Anda memiliki referensi ke objek lain, bekerja dengan komponen objek itu hampir sama persis dengan menggunakan komponen lokal. Satu-satunya perbedaan adalah bahwa sekarang, daripada hanya menulis nama komponen, Anda perlu menulis variabel objek dan titik di depannya, seperti:

```
// This accesses the local component, not what you want transform.Translate(0, 0, 0);
// This accesses the target object, what you want targetObject.transform.Translate(0, 0, 0);
```

Mengubah Objek Target Ikuti langkah-langkah ini untuk memodifikasi objek target dengan menggunakan skrip:

1. Buat proyek atau scene baru. Tambahkan kubus ke scene dan posisikan di (0, -1, 0).
2. Buat skrip baru dan beri nama TargetCubeScript. Tempatkan skrip di Kamera Utama. Di Visual Studio, masukkan kode berikut di TargetCubeScript:

```
// This is the game object you want to access private GameObject target;
// This is here for reference void Start()
{
        target = GameObject.Find("Cube");
} void Update()
{
        target.transform.Translate(.05f, 0f, 0f);
        target.transform.Rotate(0f, 0f, 1f);
        target.transform.localScale = new Vector3(1.5f, 1.5f, 1.5f);
}
```

3. Simpan skrip dan jalankan scene. Perhatikan bahwa kubus bergerak meskipun skrip diterapkan ke Kamera Utama.

1.13 PREFAB

Prefab adalah objek kompleks yang telah dibundel sehingga dapat dibuat ulang berulang kali dengan sedikit kerja ekstra. Dalam jam ini, Anda akan mempelajari semua tentang mereka, dimulai dengan apa adanya dan apa yang mereka lakukan. Dari sana, Anda akan belajar cara membuat Prefab di Unity. Anda akan belajar tentang konsep pewarisan. Anda akan menyelesaikan pelajaran dengan mempelajari cara menambahkan Prefab ke scene baik melalui editor dan melalui kode.

Dasar-dasar Prefab

Seperti disebutkan sebelumnya, prefab adalah jenis aset khusus yang menggabungkan objek game. Tidak seperti objek game normal yang hanya ada sebagai bagian dari satu scene, prefab disimpan sebagai aset. Dengan demikian mereka dapat dilihat dalam tampilan Proyek dan digunakan kembali berulang kali di banyak scene. Anda dapat, misalnya, membangun objek kompleks, seperti musuh, mengubahnya menjadi Prefab, dan kemudian menggunakan Prefab itu untuk membangun pasukan. Anda juga dapat membuat salinan Prefab dengan kode. Ini memungkinkan Anda untuk menghasilkan jumlah objek yang hampir tak terbatas saat runtime. Bagian terbaiknya adalah objek game atau koleksi objek game apa pun dapat dimasukkan ke dalam Prefab. Kemungkinannya tidak terbatas!

Latihan Pikiran

Jika Anda kesulitan memahami pentingnya prefab, pertimbangkan ini: Pada jam sebelumnya, Anda membuat game Chaos Ball. Saat membuat game itu, Anda harus membuat

satu bola chaos dan menggandakannya empat kali. Bagaimana jika Anda ingin membuat perubahan pada semua bola kekacauan pada saat yang sama, di mana pun mereka berada di scene atau proyek Anda? Melakukannya bisa jadi sulit—terkadang sangat sulit. Prefab membuatnya sangat mudah, meskipun. Bagaimana jika kamu memiliki game yang menggunakan tipe musuh orc? Sekali lagi, Anda dapat mengatur satu orc dan kemudian menggandakannya berkali-kali, tetapi bagaimana jika Anda ingin menggunakan orc lagi di scene lain? Anda harus benar-benar membuat ulang orc di scene baru. Namun, jika orc itu adalah Prefab, itu akan menjadi bagian dari proyek dan dapat digunakan kembali di sejumlah scene. Prefab adalah aspek penting dari developeran game Unity.

Prefab adalah objek kompleks yang telah dibundel sehingga dapat dibuat ulang berulang kali dengan sedikit kerja ekstra. Dalam jam ini, Anda akan mempelajari semua tentang mereka, dimulai dengan apa adanya dan apa yang mereka lakukan. Dari sana, Anda akan belajar cara membuat Prefab di Unity. Anda akan belajar tentang konsep pewarisan. Anda akan menyelesaikan pelajaran dengan mempelajari cara menambahkan Prefab ke scene baik melalui editor dan melalui kode.

Dasar-dasar Prefab

Seperti disebutkan sebelumnya, prefab adalah jenis aset khusus yang menggabungkan objek game. Tidak seperti objek game normal yang hanya ada sebagai bagian dari satu scene, prefab disimpan sebagai aset. Dengan demikian mereka dapat dilihat dalam tampilan Proyek dan digunakan kembali berulang kali di banyak scene. Anda dapat, misalnya, membangun objek kompleks, seperti musuh, mengubahnya menjadi Prefab, dan kemudian menggunakan Prefab itu untuk membangun pasukan. Anda juga dapat membuat salinan Prefab dengan kode. Ini memungkinkan Anda untuk menghasilkan jumlah objek yang hampir tak terbatas saat runtime. Bagian terbaiknya adalah objek game atau koleksi objek game apa pun dapat dimasukkan ke dalam Prefab. Kemungkinannya tidak terbatas!

Latihan Pikiran

Jika Anda kesulitan memahami pentingnya prefab, pertimbangkan ini: Pada jam sebelumnya, Anda membuat game Chaos Ball. Saat membuat game itu, Anda harus membuat satu bola chaos dan menggandakannya empat kali. Bagaimana jika Anda ingin membuat perubahan pada semua bola kekacauan pada saat yang sama, di mana pun mereka berada di scene atau proyek Anda? Melakukannya bisa jadi sulit—terkadang sangat sulit. Prefab membuatnya sangat mudah, meskipun. Bagaimana jika kamu memiliki game yang menggunakan tipe musuh orc? Sekali lagi, Anda dapat mengatur satu orc dan kemudian menggandakannya berkali-kali, tetapi bagaimana jika Anda ingin menggunakan orc lagi di scene lain? Anda harus benar-benar membuat ulang orc di scene baru. Namun, jika orc itu adalah Prefab, itu akan menjadi bagian dari proyek dan dapat digunakan kembali di sejumlah scene. Prefab adalah aspek penting dari developeran game Unity.

Terminologi Prefab

Anda perlu mengetahui beberapa terminologi tertentu yang terkait dengan bekerja dengan prefab. Jika Anda familiar dengan konsep pemrograman berorientasi objek, Anda mungkin melihat beberapa kesamaan:

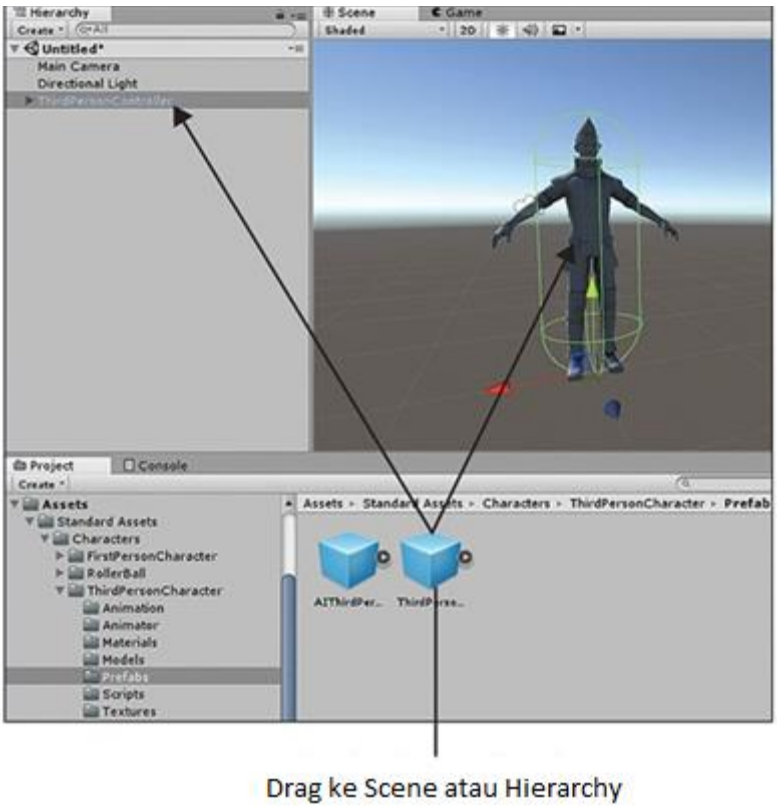
- **Prefab**: Prefab adalah objek dasar. Itu hanya ada dalam tampilan Proyek. Anggap saja sebagai cetak biru.
- **Instance**: Instance adalah objek sebenarnya dari prefab dalam sebuah scene. Jika prefab adalah cetak biru untuk mobil, sebuah instance adalah mobil yang sebenarnya.

Jika sebuah objek dalam tampilan Scene disebut sebagai prefab, itu benar-benar sebuah instance prefab. Contoh frase dari prefab identik dengan objek prefab atau bahkan tiruan dari prefab.

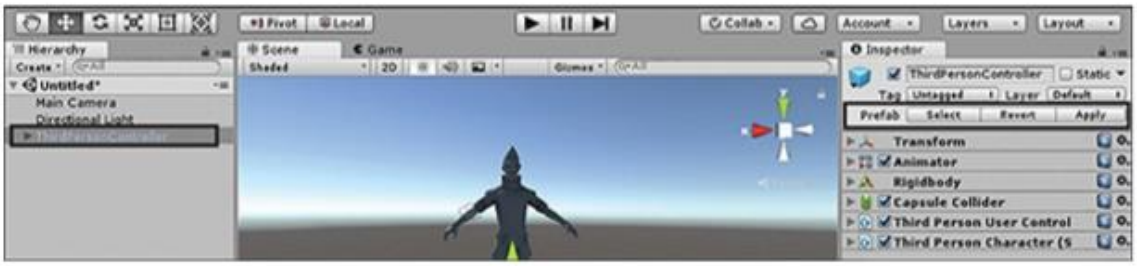
- **Instansiasi**: Instansiasi adalah proses membuat turunan dari prefab. Instantiate adalah kata kerja, digunakan seperti ini: "Saya perlu membuat instance dari prefab ini."
- **Warisan**: Dengan Prefab, warisan tidak sama dengan warisan pemrograman standar. Dalam hal ini, istilah pewarisan mengacu pada sifat di mana semua contoh dari sebuah prefab terkait dengan prefab itu sendiri. Ini dibahas secara lebih rinci nanti dalam jam ini.

Struktur Prefab

Disadari atau tidak, Anda telah bekerja dengan Prefab. Controller karakter Unity adalah Prefab. Untuk membuat instance objek prefab ke dalam scene, Anda hanya perlu mengklik dan menyeretnya ke tempatnya dalam tampilan Scene atau tampilan Hierarchy. Saat melihat tampilan Hierarchy, Anda selalu dapat mengetahui objek mana yang merupakan instance dari prefab karena terlihat berwarna biru. Ini bisa menjadi perbedaan warna yang halus, jadi perhatikan bahwa Anda juga dapat mengetahui bahwa suatu objek adalah Prefab dengan melihat bagian atas Inspector. Sama seperti objek kompleks non-Prefab, instance kompleks dari pabrikan juga memiliki panah yang memungkinkan Anda untuk memperluasnya dan memodifikasi objek di dalamnya.

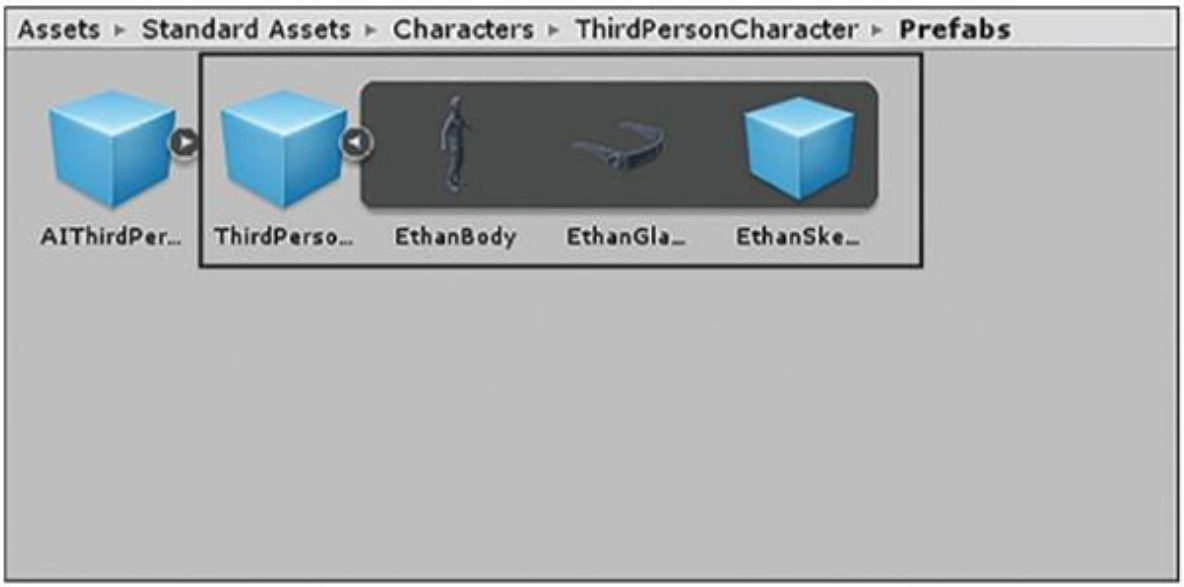


Gambar 1.46 Menambahkan instance prefab ke sebuah scene.



Gambar 1.47 Munculnya instance prefab di Inspector.

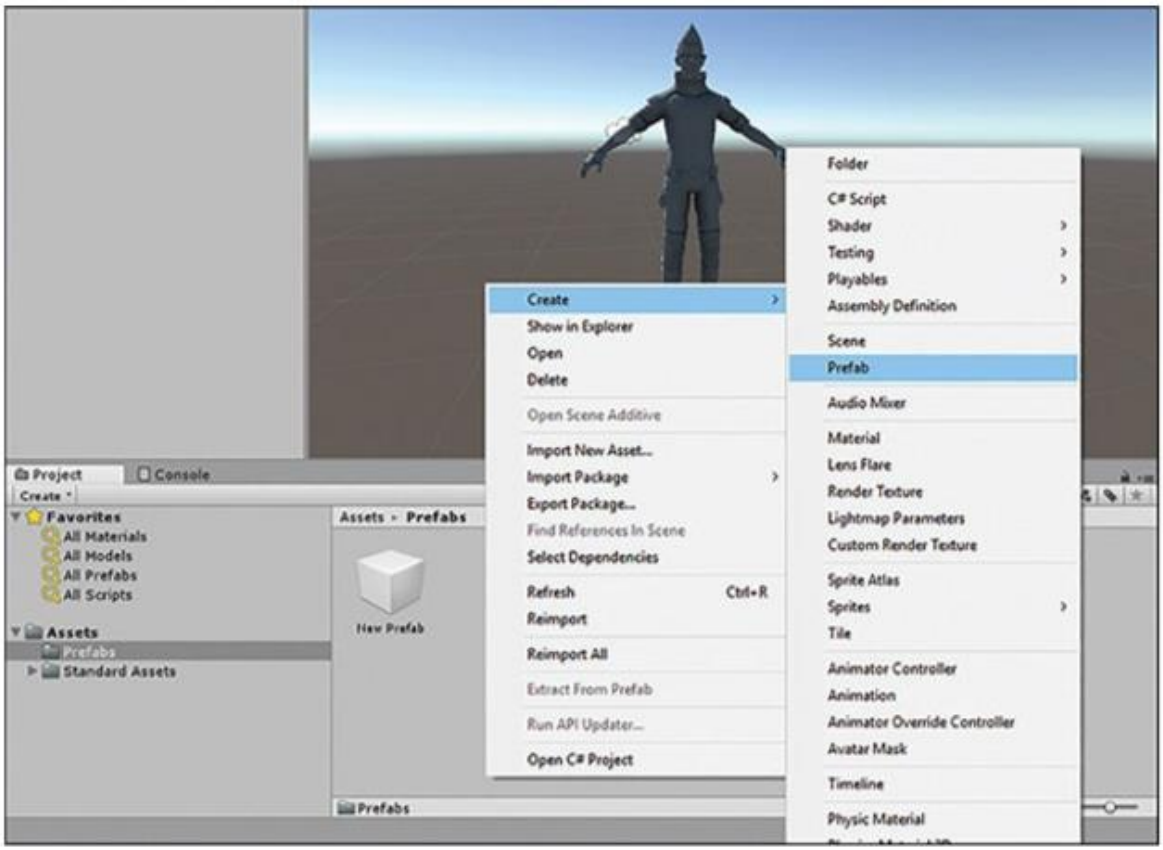
Karena prefab adalah aset milik proyek dan bukan scene tertentu, Anda dapat mengedit prefab dalam tampilan Project atau tampilan Scene. Jika Anda mengedit instance prefab dalam tampilan Scene, Anda harus menyimpan perubahan kembali ke prefab dengan mengklik tombol Apply di kanan atas Inspector. Sama seperti objek game, prefab bisa rumit. Anda dapat mengedit elemen anak dari prefab dengan mengklik panah di sisi kanan prefab. Mengklik panah ini akan memperluas objek yang akan diedit. Mengklik lagi memadatkan Prefab lagi.



Gambar 1.48 Memperluas konten prefab dalam tampilan Project.

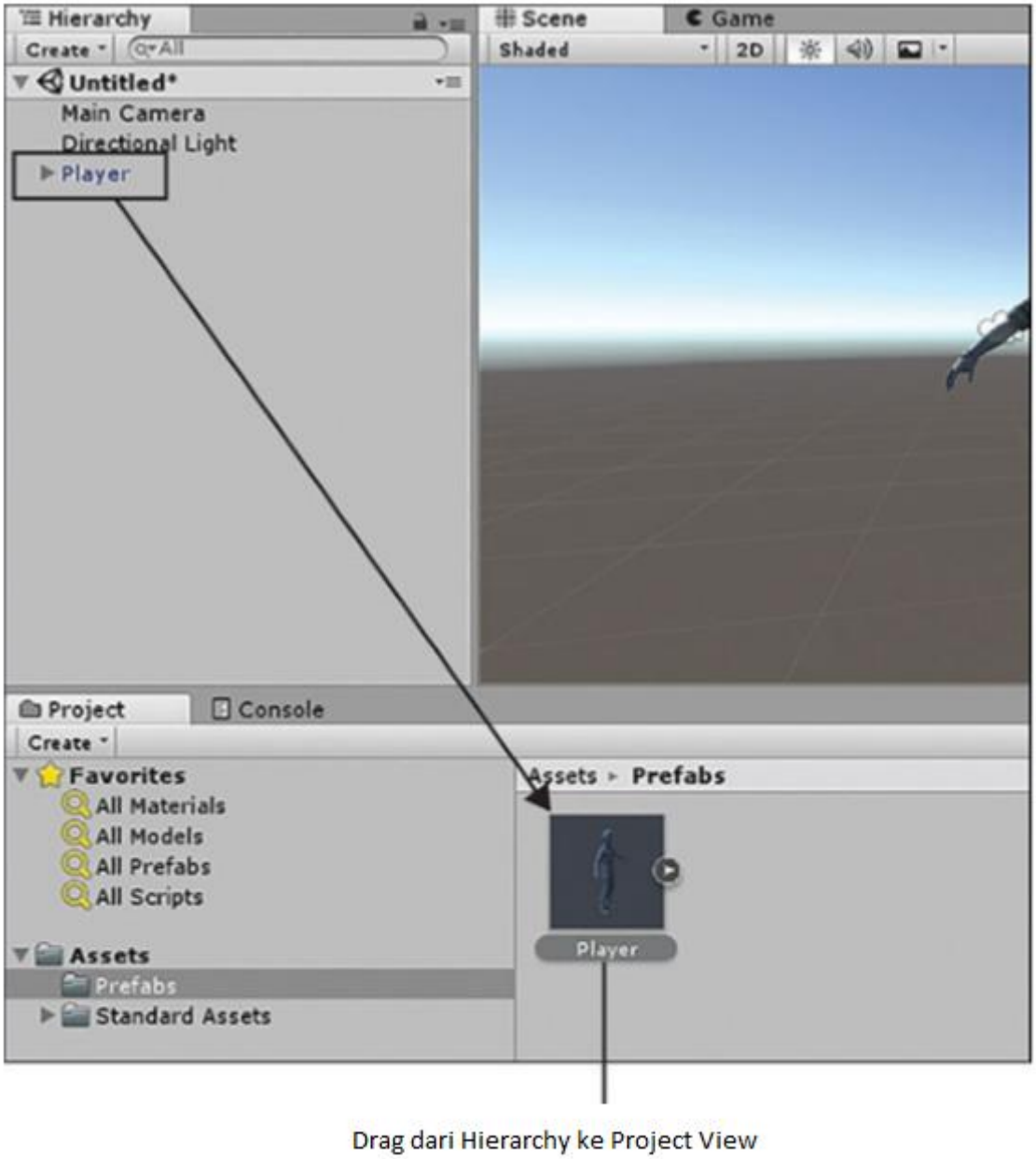
Bekerja dengan Prefab

Menggunakan prefab bawaan Unity itu bagus, tetapi seringkali Anda ingin membuatnya sendiri. Membuat Prefab adalah proses dua langkah. Langkah pertama adalah membuat aset pabrikan. Langkah kedua adalah mengisi aset dengan beberapa konten. Membuat Prefab sangat mudah. Cukup klik kanan pada tampilan Project dan pilih Create > Prefab (lihat Gambar 1.49). Sebuah Prefab baru muncul, dan Anda dapat menamainya apa pun yang Anda inginkan. Karena Prefabnya kosong, itu muncul sebagai kotak putih kosong.



Gambar 1.49 Membuat Prefab baru

Seperti semua aset lainnya, biasanya merupakan ide yang baik untuk memulai dengan membuat folder di bawah Aset dalam tampilan Proyek untuk memuat Prefab Anda. Membuat folder untuk menampung prefab Anda tidak diperlukan, tetapi ini adalah langkah organisasi yang bagus dan harus dilakukan untuk mencegah kebingungan antara prefab dan aset asli (seperti mesh atau sprite). Langkah selanjutnya adalah mengisi Prefab dengan sesuatu. Objek game apa pun bisa masuk ke Prefab. Anda hanya perlu membuat objek sekali dalam tampilan Scene lalu klik dan drag ke aset Prefab. Atau, Anda dapat mempersingkat proses ini menjadi satu langkah dengan menyeret objek game apa pun dari tampilan Hierarki ke tampilan proyek. Melakukannya akan membuat Prefab, menamainya, dan mengisinya—semuanya pada waktu yang sama.



Gambar 1.50 Cara yang lebih cepat untuk membuat Prefab.

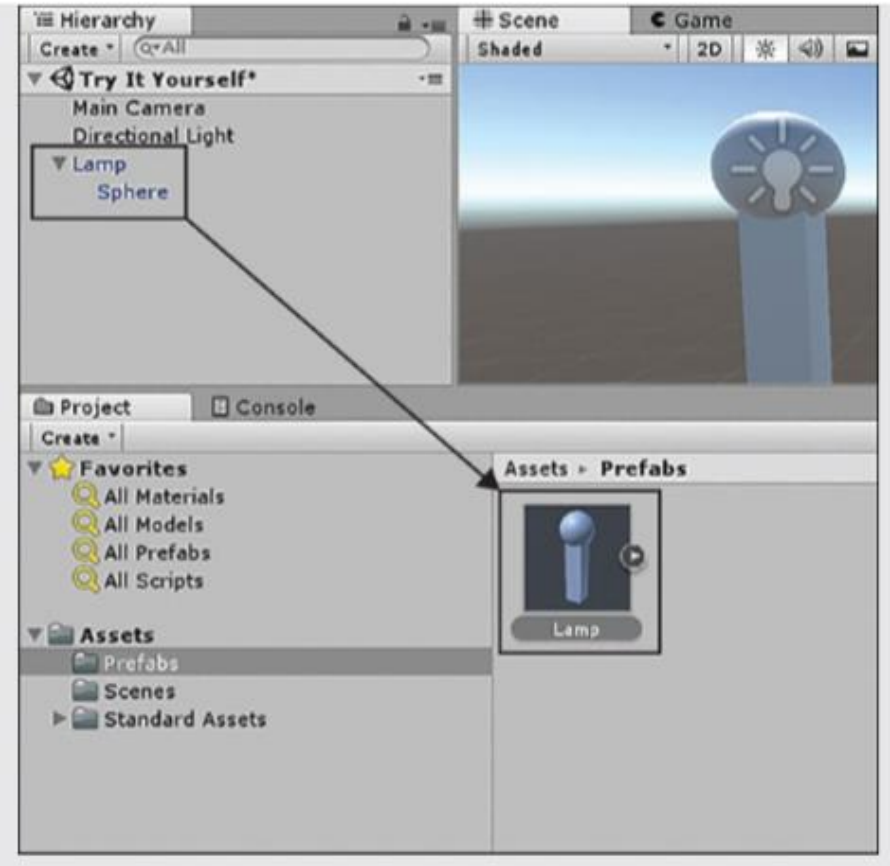
Membuat Prefab

Ikuti langkah-langkah ini untuk membuat prefab dari objek game yang kompleks, yang akan Anda gunakan nanti dalam satu jam ini (jadi jangan hapus!):

1. Buat proyek atau scene baru. Tambahkan kubus dan bola ke TKP. Ganti nama lampu kubus.
2. Posisikan kubus pada (0, 1, 0) dengan scale (0,5, 2, .5). Tambahkan benda tegar ke kubus. Posisikan bola di (0, 2.2, 0). Letakkan komponen titik cahaya pada bola.
3. Klik dan drag bola dalam tampilan Hierarki ke kubus untuk membuat sarang bola di bawah kubus

Gambar 1.51 Bola bersarang di bawah kubus.

4. Buat folder baru di bawah folder Aset dalam tampilan Proyek. Beri nama folder baru Prefabs.
5. Pada tampilan Hierarchy, klik dan drag objek Lamp game (yang berisi bola) ke dalam tampilan Project. Perhatikan bahwa ini menciptakan Prefab yang terlihat seperti lampu. Perhatikan juga bahwa kubus dan bola dalam tampilan Hierarki berubah menjadi biru. Pada titik ini, Anda dapat menghapus kubus dan bola dari scene karena sekarang ada dalam Prefab.



Gambar 1.52 Menambahkan objek ke Prefab

Menambahkan Instance Prefab ke Scene

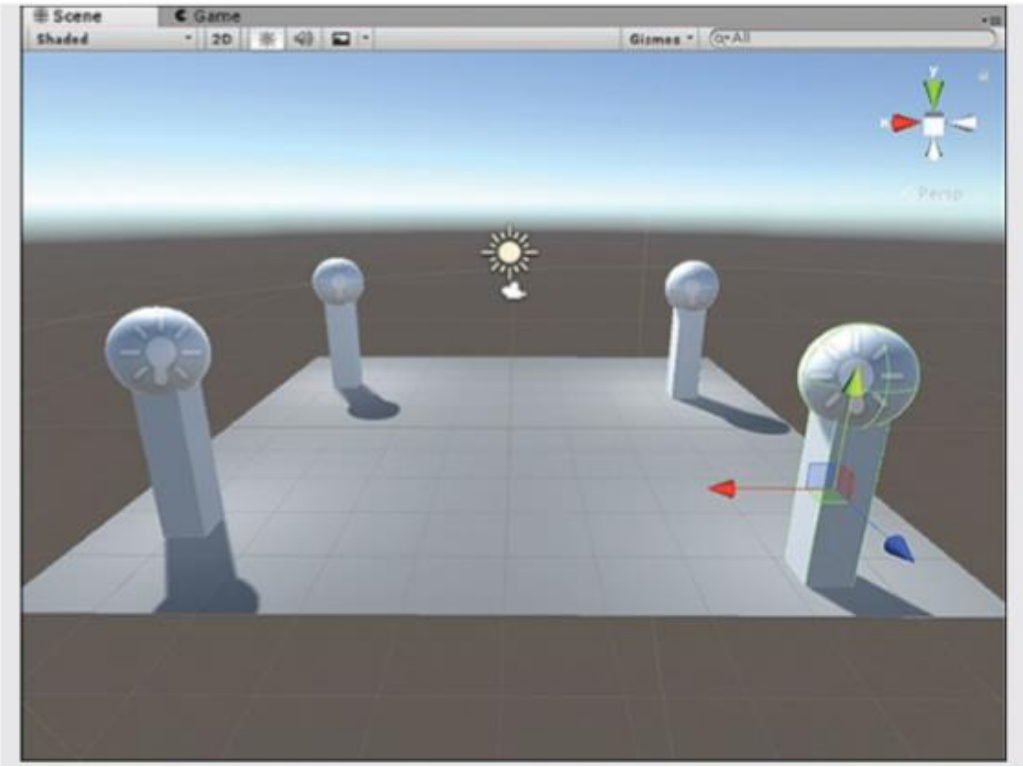
Setelah aset pabrikan dibuat, Anda dapat menambahkannya sebanyak yang Anda inginkan ke sebuah scene atau sejumlah scene dalam sebuah proyek. Untuk menambahkan contoh prefab ke sebuah scene, yang perlu Anda lakukan hanyalah mengklik dan menyeret

prefab dari tampilan Proyek ke tempatnya di tampilan Scene, atau tampilan Hierarki. Jika Anda menyeret ke tampilan Scene, prefab akan dipakai di tempat Anda menyeret. Jika Anda menyeret ke bagian kosong dari tampilan Hierarchy, posisi awalnya adalah apa pun yang diatur dalam Prefab. Jika Anda menyeret ke objek lain dalam tampilan Hierarki, prefab menjadi anak dari objek itu

Membuat Beberapa Instance Prefab

Kali ini, Anda akan menggunakan Prefab itu untuk membuat banyak lampu dalam satu scene. Pastikan untuk menyimpan scene yang dibuat di sini karena Anda akan menggunakannya nanti di jam ini. Ikuti langkah-langkah ini:

1. Buat scene baru dalam proyek yang sama yang digunakan untuk "Membuat Prefab" Coba Sendiri, dan beri nama scene ini Lamps.
2. Buat sebuah pesawat dan pastikan posisinya berada di (0, 0, 0). Ubah nama lantai pesawat. Opsional, berikan lantai bahan abu-abu agar bayangan tampak lebih jelas di atasnya.
3. Drag prefab Lamp Anda ke lantai dalam tampilan Scene. Perhatikan bagaimana lampu melacak ke tumpatan lantai.
4. Drag tiga prefab Lampu lagi ke lantai dan posisikan di dekat sudut



Gambar 1.53 Menempatkan lampu di scene.

Warisan

Ketika istilah warisan digunakan dalam hubungannya dengan prefab, itu mengacu pada link dimana instance dari prefab terhubung ke aset prefab yang sebenarnya. Artinya, jika Anda mengubah aset prefab, semua objek prefab juga otomatis berubah. Ini sangat berguna. Anda akan seperti banyak programmer game lainnya jika Anda menempatkan sejumlah besar objek Prefab ke dalam sebuah scene hanya untuk menyadari bahwa mereka semua membutuhkan perubahan kecil. Tanpa warisan, Anda harus mengubah masing-masing secara

individual. Ada dua cara Anda dapat mengubah aset Prefab. Yang pertama adalah dengan melakukan perubahan pada tampilan Project. Hanya dengan memilih aset pabrikan dalam tampilan Proyek akan menampilkan komponen dan propertinya dalam tampilan Inspector. Jika Anda perlu memodifikasi elemen anak, Anda dapat memperluas prefab (seperti yang dijelaskan sebelumnya jam ini) dan mengubah objek tersebut dengan cara yang sama. Cara lain Anda dapat memodifikasi aset prefab adalah dengan menyeret sebuah instance ke dalam scene. Dari sana, Anda dapat membuat modifikasi besar apa pun yang Anda inginkan. Setelah selesai, cukup klik Terapkan di kanan atas Inspector.

Melanggar Tautan Prefab

Terkadang Anda perlu memutuskan tautan instans prefab ke aset prefab. Anda mungkin ingin melakukan ini jika Anda membutuhkan objek Prefab tetapi Anda tidak ingin objek berubah jika Prefab pernah berubah. Memutus tautan instans ke prefab tidak mengubah instans dengan cara apa pun. Instance masih mempertahankan semua objek, komponen, dan propertinya. Satu-satunya perbedaan adalah bahwa itu bukan lagi turunan dari Prefab dan oleh karena itu tidak lagi terpengaruh oleh warisan. Untuk memutuskan tautan objek ke aset pabrikan, cukup pilih objek dalam tampilan Hierarki. Setelah memilihnya, klik GameObject > Break Prefab Instance. Objek tidak berubah, tetapi namanya berubah dari biru menjadi hitam. Setelah tautan rusak, tautan dapat diterapkan kembali dengan mengklik Kembalikan di tampilan Inspector.

Membuat Instansi Prefab Melalui Kode

Menempatkan objek Prefab ke dalam scene adalah cara yang bagus untuk membangun tingkat yang konsisten dan terencana. Namun, terkadang Anda ingin membuat instance saat runtime. Mungkin Anda ingin musuh muncul kembali, atau Anda ingin mereka ditempatkan secara acak. Mungkin juga Anda membutuhkan begitu banyak instance sehingga menempatkannya dengan tangan tidak lagi memungkinkan. Apa pun alasannya, membuat instance prefab melalui kode adalah solusi yang baik. Ada dua cara untuk membuat instance objek prefab dalam sebuah scene, dan keduanya menggunakan metode Instantiate(). Cara pertama adalah dengan menggunakan Instantiate() seperti ini:

Instantiate(GameObject prefab);

Seperti yang Anda lihat, metode ini hanya membaca variabel GameObject dan membuat salinannya. Lokasi, rotasi, dan scale objek baru sama dengan objek yang dikloning. Cara kedua untuk menggunakan metode Instantiate() adalah seperti ini:

Instantiate(GameObject prefab, Vector3 position, Quaternion rotation);

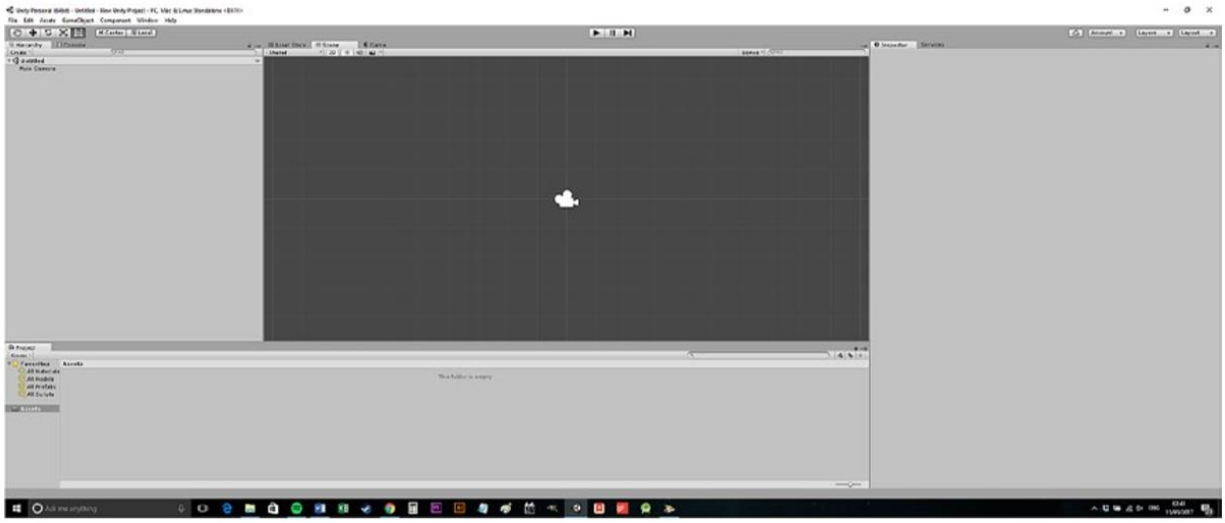
Metode ini membutuhkan tiga parameter. Yang pertama masih objek untuk disalin. Parameter kedua dan ketiga adalah posisi dan rotasi objek baru yang diinginkan. Anda mungkin telah memperhatikan bahwa rotasi disimpan dalam sesuatu yang disebut quaternion. Ketahuilah bahwa inilah cara Unity menyimpan informasi rotasi. (Aplikasi sebenarnya dari quaternion berada di luar cakupan jam ini.) Latihan di akhir jam ini menunjukkan contoh dua metode untuk membuat objek dalam kode.

BAB 2
MENEMUKAN JALAN DAN MEMBANGUN DEMO ANDA DI RUANG 3D

Sebelum Anda membangun apa pun, pertama-tama mari berkenalan dengan berbagai elemen UI, kontrol, dan opsi. Anda akan menghabiskan banyak waktu di sini, jadi sebaiknya Anda membiasakan diri.

2.1 IDE DALAM UNITY

Saat pertama kali melihat Unity (Gambar 3-1), semuanya dapat tampak agak membingungkan dengan banyak jendela, menu, dan opsi yang berbeda. Untungnya, semuanya jauh lebih sederhana daripada yang terlihat begitu Anda mulai. Kami akan membahas untuk apa masing-masing pandangan yang berbeda ini selama bab ini dan pada saat yang sama kami akan dapat menguji aplikasi pertama kami yang sangat sederhana.



Gambar 2.1 Selamat datang di Unity! Anda akan menghabiskan banyak waktu di sini

Scene

Tepat di tengah Unity adalah view yang disebut Scene. Di sinilah banyak pekerjaan Anda akan selesai dan pada dasarnya adalah jendela di mana Anda akan memindahkan berbagai GameObjects Anda dan mengatur semuanya. Ini menunjukkan kepada Anda tampilan layar level/menu yang sedang Anda kerjakan pada waktu tertentu dan memungkinkan Anda memilih dan memposisikan ulang elemen di sekitar layar. Anda dapat memperbesar dan memperkecil dan jika Anda berada dalam mode 3D, Anda juga dapat menggerakkan kamera sekitar 360 derajat.

Asset Store

Di bagian atas jendela Scene terdapat dua tab yang memungkinkan Anda untuk mengubah antara dua fungsi yang berbeda. Tekan tab Asset Store, dan seperti yang Anda duga, tampilan Scene akan diubah untuk tampilan Asset Store. Asset Store adalah tempat Anda menelusuri berbagai aset—skrip, GameObjects, sprite, efek, dan lainnya—untuk disertakan dalam aplikasi Anda sendiri. Aset ini dikembangkan oleh developer Unity lainnya, serta oleh Unity Technologies. Beberapa gratis dan lainnya membutuhkan uang; beberapa adalah tambahan yang sangat sederhana untuk proyek yang sudah ada, sedangkan yang lain praktis merupakan game yang sudah jadi untuk Anda edit sesuka Anda.

Gambar 2.2 Tab Scene, Game, dan Asset Store

Singkatnya, Asset Store membuat hidup Anda jauh lebih mudah dengan memastikan bahwa Anda tidak perlu membuat setiap skrip dan sprite dengan tangan. Yang mengatakan, ini bukan sesuatu yang Anda perlukan untuk sementara waktu, jadi untuk saat ini tetap awasi Scene dan jangan khawatir tentang itu.

Game

Mungkin ada tab ketiga di jendela ini, seperti yang ditunjukkan pada gambar sebelumnya, yang disebut Game. Tampilan ini adalah tempat Anda akan melihat game yang sebenarnya seperti yang akan muncul saat live. Dan saat Anda memainkan game, di sinilah lokasinya (kecuali jika Anda memilih untuk memaksimalkan game saat Anda memainkannya). Jika Game tidak terletak di tempat yang sama, itu akan ada di sekitar sana. Anda memiliki kebebasan untuk mengubah posisi jendela, dan terkadang setelah pembaruan, pengaturan default dapat dipindahkan. Kebanyakan orang harus menemukan bahwa semuanya ada di tempat yang sama dan mereka dapat mengikuti petunjuk ini. Jika tidak, Anda seharusnya dapat menemukan setiap elemen dengan cukup cepat. Anda tidak dapat menarik dan melepaskan elemen ke dalam tampilan Game (seperti yang Anda bisa dalam tampilan Scene) dan Anda juga tidak dapat memilih atau memindahkannya. Karena itu, sebagian besar tampilan Game akan mencerminkan apa yang Anda lihat di tampilan Scene dengan beberapa perbedaan. Pertama, perspektif akan ditetapkan ke kamera dalam game, artinya Anda akan melihat apa yang akan dilihat player saat mereka meluncurkan game. Demikian juga, bila ada beberapa item yang berbagi koordinat X dan Y yang sama, item di atas akan menjadi yang paling dekat dengan kamera di sepanjang sumbu Z, bukan yang dipilih. Jika semua ini terdengar sedikit membingungkan, jangan khawatir—ini akan masuk akal setelah Anda melihat cara kerjanya (dan itu berlaku untuk semua jendela).

Service (Layanan)

Biasanya terletak di sebelah kanan tampilan Scene adalah tab Layanan, yang berbagi jendela dengan Inspector. Ini mencakup hal-hal seperti iklan untuk monetisasi, analitik untuk mempelajari tentang bagaimana player Anda menikmati game Anda, multiplayer, dan sebagainya. Perhatikan bahwa beberapa fitur ini akan hilang atau terbatas jika Anda memiliki versi gratisnya. Untuk saat ini, Anda dapat mengabaikan jendela ini sepenuhnya. Layanan ini terutama akan berlaku untuk proyek yang lebih ambisius dan hanya setelah aplikasi tersebut aktif di Play Store.

Inspector

Berikutnya adalah tab yang sering berbagi jendela dengan Layanan: Inspector. Inspector adalah apa yang akan Anda gunakan untuk melihat dan mengedit detail GameObjects. Jadi saat Anda memilih GameObject seperti sprite dalam tampilan Scene, Anda kemudian dapat menggunakan Inspector untuk melihat hal-hal seperti nama objek, dimensi, skrip apa pun yang mungkin dilampirkan, dan sebagainya.

Gambar 2.3 Inspector dan Service

Anda akan sering menggunakan Inspector, jadi simpan ini di tempat yang dapat Anda lihat. Namun, saat ini, itu akan benar-benar kosong.

Project

Biasanya terletak di sepanjang bagian bawah layar adalah jendela untuk tab Proyek dan Konsol (dan terkadang Game juga ada di sini). Tab Proyek harus terbuka secara default dan merupakan tempat Anda dapat melihat semua file individual yang terkait dengan proyek Anda. Di sepanjang bagian kiri jendela adalah direktori tempat Anda dapat memilih folder, dan di sebelah kanan adalah isi folder itu. Saat ini, proyek Anda hanya memiliki satu folder bernama Aset. Dan di folder itu adalah ... tidak ada. Ini akan menjadi jendela yang berguna saat Anda bekerja karena memungkinkan Anda menemukan sprite yang telah Anda buat dengan software lain dan mengganti nama atau menghapus file yang Anda perlukan untuk gim Anda.

Console

Di sebelah tab Proyek adalah tab Konsol (Gambar 3-4). Di sinilah Anda bisa mendapatkan informasi mengenai status Unity dan aplikasi Anda. Anda akan dapat melihat informasi debug, laporan kerusakan, dan kesalahan, dan ini dapat membantu Anda mengidentifikasi masalah dalam kode Anda atau mencari tahu mengapa game Anda tidak dapat berjalan atau dikompilasi. Ini akan berguna, tetapi kita tidak perlu mengkhawatirkannya untuk sementara waktu, jadi buat Project tetap terlihat di depannya untuk saat ini.

Gambar 2.4 Hierarchy and Console

Hierarchy

Terakhir, salah satu elemen terpenting dalam UI adalah Hierarki, yang hampir selalu ditemukan di sebelah kiri tampilan Scene. Hierarchy menunjukkan daftar semua GameObjects di scene Anda pada waktu tertentu, dan ketika Anda memilih salah satunya, tampilan Scene akan terpusat di atasnya; itu juga akan dibuka di Inspector. Ini memungkinkan Anda dengan cepat menemukan GameObjects tertentu untuk mengedit, dan itu juga satu-satunya cara Anda dapat memilih objek "tak terlihat" seperti checkpoint. Hierarchy juga bisa sangat berguna ketika Anda ingin memilih beberapa objek (mungkin semua koleksi Anda, misalnya) dan memiliki utilitas pencarian yang berguna untuk mengambil item tertentu dengan cepat. Menjaga Hierarchy yang rapi adalah praktik yang baik dan akan membantu Anda bekerja lebih cepat dan efisien di masa mendatang.

Housekeeping

Saya menyarankan Anda membiarkan windows dalam konfigurasi default dalam banyak kasus. Mereka telah diatur seperti itu karena suatu alasan (berfungsi), dan ini akan memudahkan Anda untuk mengikuti petunjuk dalam buku ini. Konsol atau lainnya mungkin tidak berada di tempat yang sama persis, tetapi kami terutama akan menggunakan jendela Scene, Game, Project, dan Inspector untuk saat ini. Jangan khawatir tentang sisanya. Tetapi jika Anda menemukan bahwa UI terasa sempit atau Anda tidak menyukai cara pengaturannya di titik mana pun, Anda dapat mengarahkan penunjuk mouse ke salah satu garis pemisah untuk mengubah ukuran relatif. Anda juga dapat menyeret tab dari satu jendela ke jendela lain, menutup seluruhnya, atau mengembalikannya menggunakan opsi di menu Jendela. Anda

mungkin telah memperhatikan bahwa ada jendela tambahan yang dapat Anda buka, ditemukan di menu Window, termasuk Audio Mixer, Animator, dan Sprite Packer. Beberapa di antaranya akan kami gunakan nanti, tetapi untuk saat ini Anda tidak perlu mengkhawatirkannya; Anda seharusnya dapat melakukan hampir semua hal dengan jendela Hierarki, Scene, Proyek, Game, Asset Store, Konsol, dan Inspector.

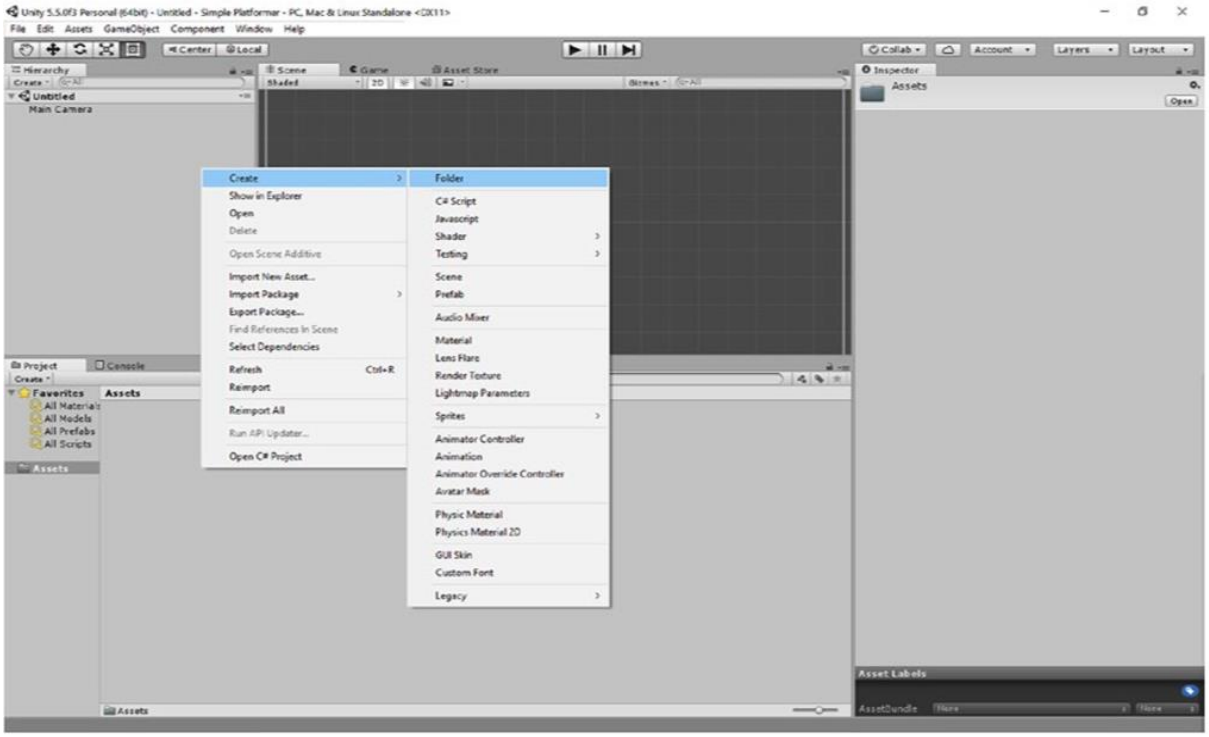
Membuat Jari Kaki Anda Basah dengan Objek dan Scene

Itu cukup teori—waktu untuk menjadi praktis. Untuk benar-benar memahami cara kerja jendela ini dan apa yang perlu Anda lakukan untuk memulai, hal terbaik yang harus dilakukan adalah mulai membangun sesuatu. Setelah Anda melakukannya, Anda akan melihat secara langsung bagaimana semuanya bekerja bersama dan bagaimana Anda akan mengelola alur kerja Anda setelah Anda berkembang.

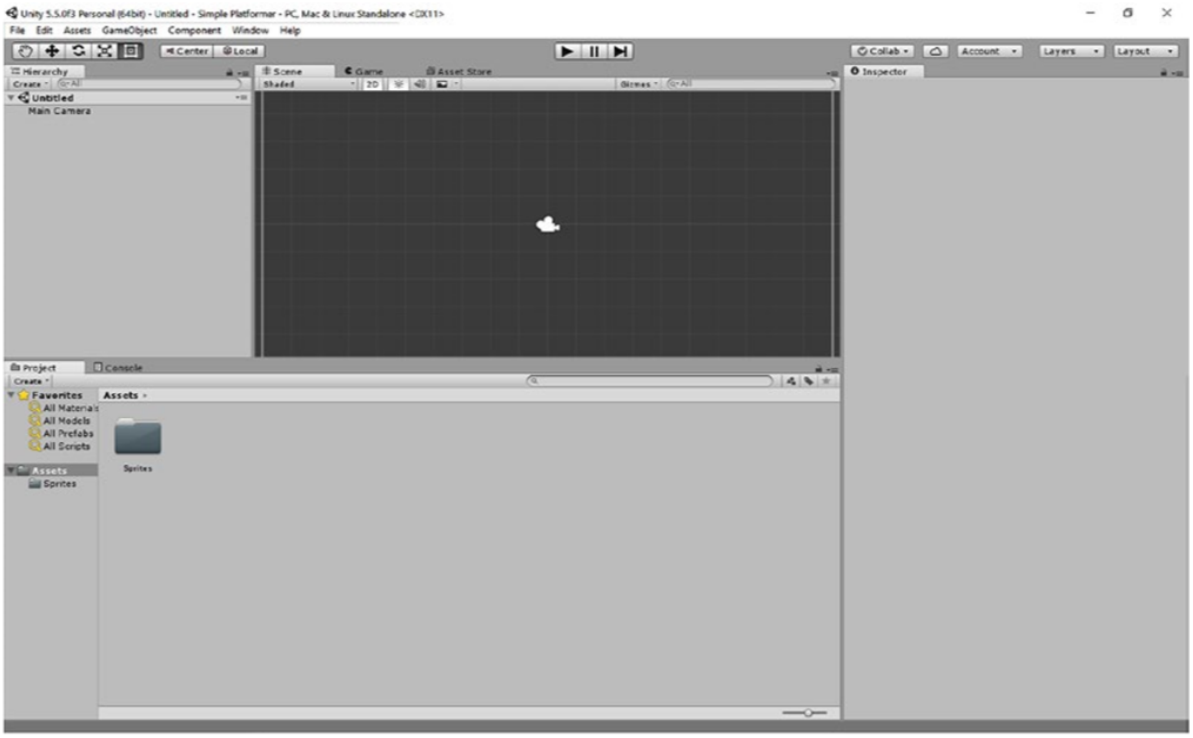
Menambahkan Sprite Untuk memulai, mari kita mulai dengan menambahkan GameObject pertama kita. Itu akan menjadi persegi 2D sederhana. Tidak seperti bekerja dengan objek 3D, tidak ada bentuk sederhana untuk Anda sisipkan dalam 2D. Itu berarti bahwa objek 2D apa pun yang Anda perkenalkan harus dibuat terlebih dahulu sebagai sprite. Membuat persegi cukup sederhana, meskipun: kita benar-benar dapat memulai file MSPaint baru, mengubah ukurannya menjadi 50 x 50 piksel, dan kemudian hanya mengisi ruang dengan satu warna blok. Simpan itu sebagai file PNG dan drag dari mana pun Anda menyimpannya ke folder proyek Anda. Sebut saja Square untuk kesederhanaan.

Gambar 2.5 Square, Grafik tidak cukup triple-A

Untuk membantu kami membiasakan diri sejak dini, pertama-tama kami akan membuat folder di proyek kami khusus untuk sprite. Dan dalam semangat kebiasaan baik dan tata nama yang baik, kami akan memanggil folder Sprite. Untuk melakukannya, klik kanan folder Aset Anda di jendela Proyek, lalu buka Buat Folder dan beri nama folder Sprite.



Gambar 2.6 Membuat folder baru sangat sederhana



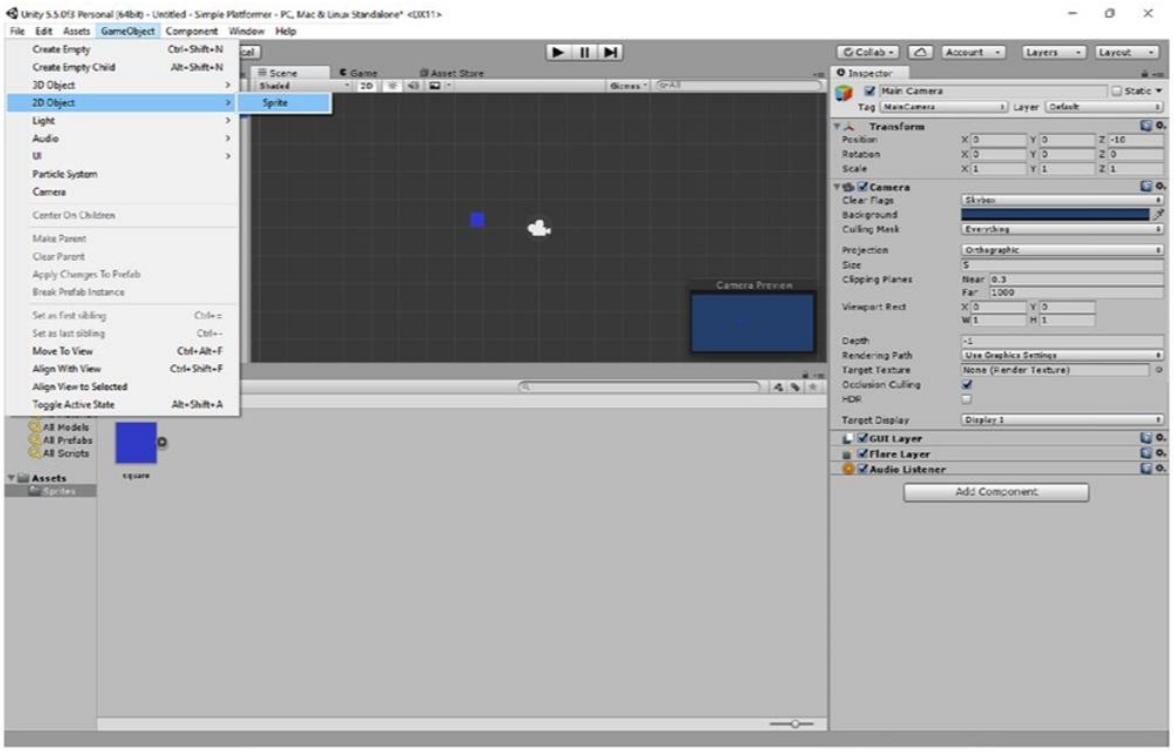
Gambar 2.7 Folder Sprite Anda akan terlihat seperti ini. Ini adalah folder. Disebut sprite.

Seiring berjalannya waktu, kami akan membuat lebih banyak folder untuk skrip, suara, scene, dan banyak lagi—pintar untuk membuat banyak folder untuk membantu menjaga semuanya tetap terpisah sehingga kami dapat dengan cepat mengambil jenis file yang kami butuhkan kapan saja. Setelah Anda membuat folder itu, Anda cukup menyeret file square.png dari Windows Explorer ke folder Sprite Anda. Perhatikan bahwa kapan saja, Anda juga dapat mengeklik kanan jendela Project dan memilih Show in Explorer. Ini akan menunjukkan kepada Anda direktori Aset dalam proyek Anda, dan apa pun yang Anda masukkan di sini akan muncul di jendela Proyek setelah Anda menyegarkan tampilan. Dengan itu, sprite sekarang menjadi bagian dari proyek Anda. Dan Anda dapat menggunakan proses yang sama persis apakah Anda ingin menambahkan sprite pohon, koleksi, musuh, atau apa pun ke dalam level Anda. Ini benar-benar sederhana.

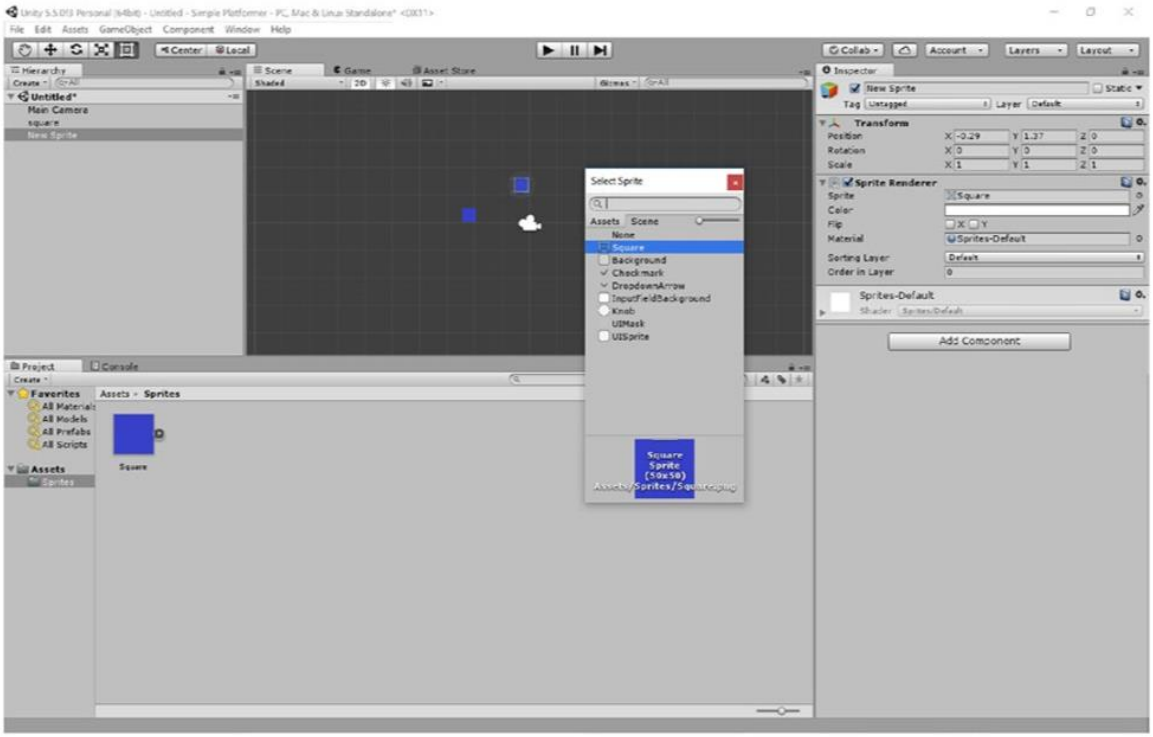
2.2 DUA CARA UNTUK MEMPERKENALKAN GAMEOBJECTS

Ada dua cara untuk menambahkan GameObject sederhana ini ke dalam scene, dan kita akan membahas keduanya di sini karena saya merasa ini adalah kesempatan belajar yang baik. Cara paling mudah adalah dengan mengklik sprite di jendela Project Anda dan kemudian menyeretnya langsung ke scene Anda dan menjatuhkannya. Anda kemudian akan melihat bahwa itu muncul di tampilan Scene Anda dan juga terdaftar di Hierarchy Anda di sebelah kanan. Jika dipilih, detail tentang kotak juga akan ditampilkan di jendela Inspector Anda. Klik tampilan Game Anda dan Anda akan melihat kotak itu dengan warna biru yang berbeda di latar belakang.

Cara lain untuk menambahkan sprite seperti ini ke Scene Anda adalah menuju ke menu atas dan kemudian klik GameObject > Objek 2D > Sprite. Saat Anda melakukan ini, Sprite Baru akan muncul di jendela Hierarchy Anda, tetapi Anda tidak akan dapat melihatnya di tampilan Scene karena saat ini tidak memiliki file gambar yang terkait dengannya. Namun, itu akan memiliki lingkaran tembus pandang di sekitar koordinatnya, setiap kali dipilih.

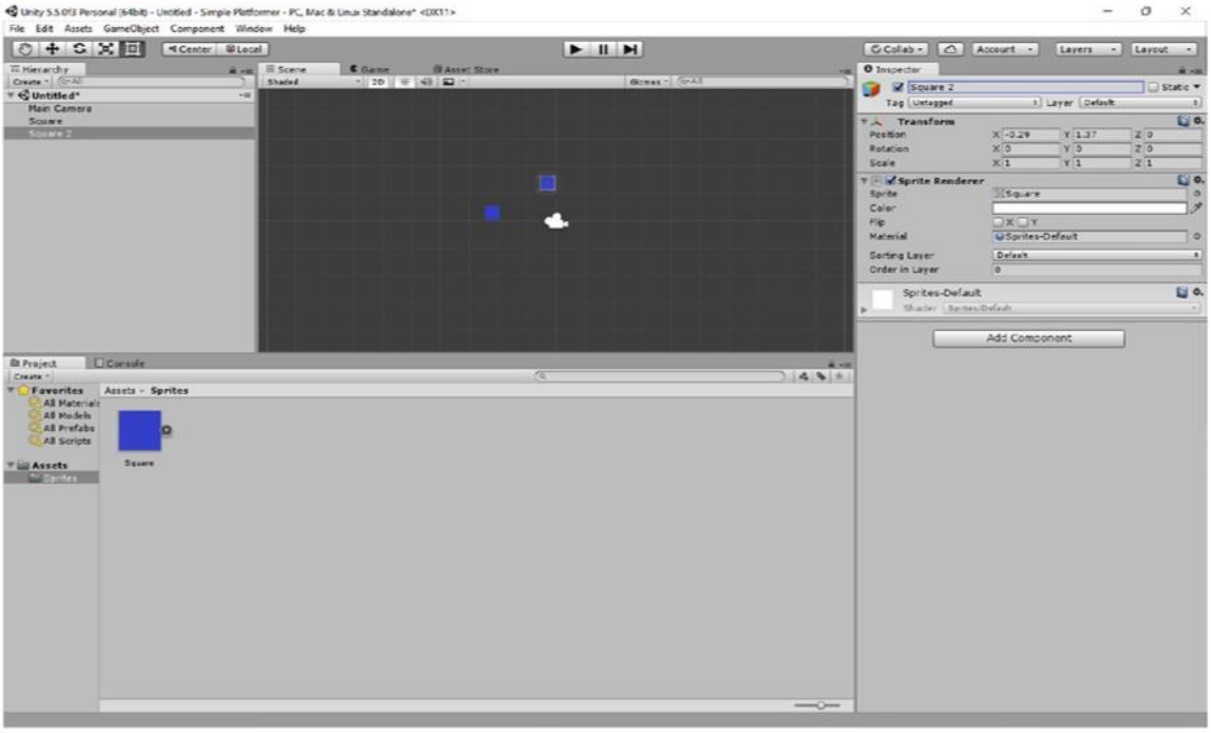


Gambar 2.8 Cara kedua untuk memasukkan sprite Anda



Gambar 2.9 Menggunakan Renderer Sprite

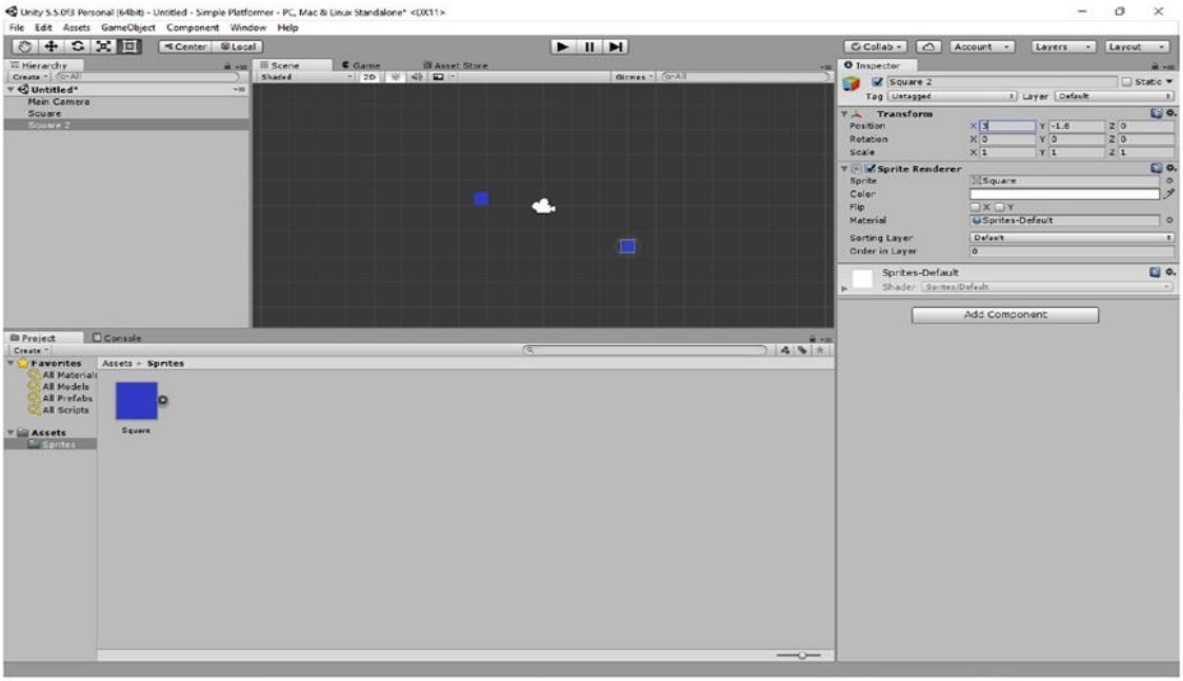
Saat Anda berada di Inspector, mengapa tidak mengambil kesempatan ini juga untuk mengubah nama sprite kedua Anda (saat ini Sprite Baru) menjadi sesuatu yang lebih menarik... seperti Kotak 2. Coba kedua metode ini dan Anda akan memiliki dua sprite yang berbeda di layar di jendela Scene Anda: Kotak 1 dan Kotak 2.



Gambar 2.9 Jika terlihat seperti ini, Anda melakukannya dengan baik.

Memanipulasi GameObjects

Seperti yang mungkin sudah Anda duga, Anda dapat dengan mudah memindahkan sprite baru Anda di dalam tampilan Scene hanya dengan mengkliknya dan menyeretnya di sekitar layar. Saat Anda melakukan ini, Anda akan melihat bahwa koordinat berubah di Inspector. Di bawah Transform, nilai X dan Y dari Posisi berubah saat Anda memindahkan kotak.



Gambar 2.10 Ubah koordinat X dan Y di Inspector untuk memindahkan sprite Anda

Itu berarti Anda dapat dengan mudah mengubah posisi dengan memasukkan angka di Inspector. Ini akan berguna jika Anda perlu mengatur semuanya dengan sempurna atau mengaturnya di posisi yang sangat spesifik (sesuatu yang akan sering Anda lakukan). Kita akan lihat nanti bahwa ada juga cara yang lebih efektif untuk memastikan bahwa semuanya tetap berbaris dengan baik dan terkunci ke grid saat kita bekerja. "Bagaimana dengan Z?" Saya mendengar Anda bertanya! Nah, sumbu Z sebagian besar berlebihan saat membuat game 2D, meskipun tidak sepenuhnya. Kita akan melihat nanti bahwa Anda dapat menggunakan opsi ini untuk membuat efek pengguliran paralaks dan itu juga dapat berguna untuk menentukan sprite mana yang harus dirender untuk dilihat player ketika ada beberapa yang terletak di tempat yang sama. Ini juga dapat dikontrol dengan mengubah urutan Z, yang juga akan kita lihat nanti.

2.3 ROTATION AND SCALE

Anda mungkin telah memperhatikan bahwa Anda juga memiliki dua opsi yang lebih menarik di bawah Transform di sini di Inspector: ini adalah Rotasi dan Scale. Mereka cukup banyak melakukan apa yang mereka katakan dan memungkinkan Anda untuk mengubah rotasi dan ukuran sprite Anda. Kami akan mengabaikan Rotasi untuk saat ini karena kami tidak akan membutuhkannya untuk sementara waktu. Tetapi Anda akan menemukan bahwa jika Anda mengubah Scale X dan Y dari 1 menjadi 2, ukuran sprite Anda menjadi dua kali lipat.

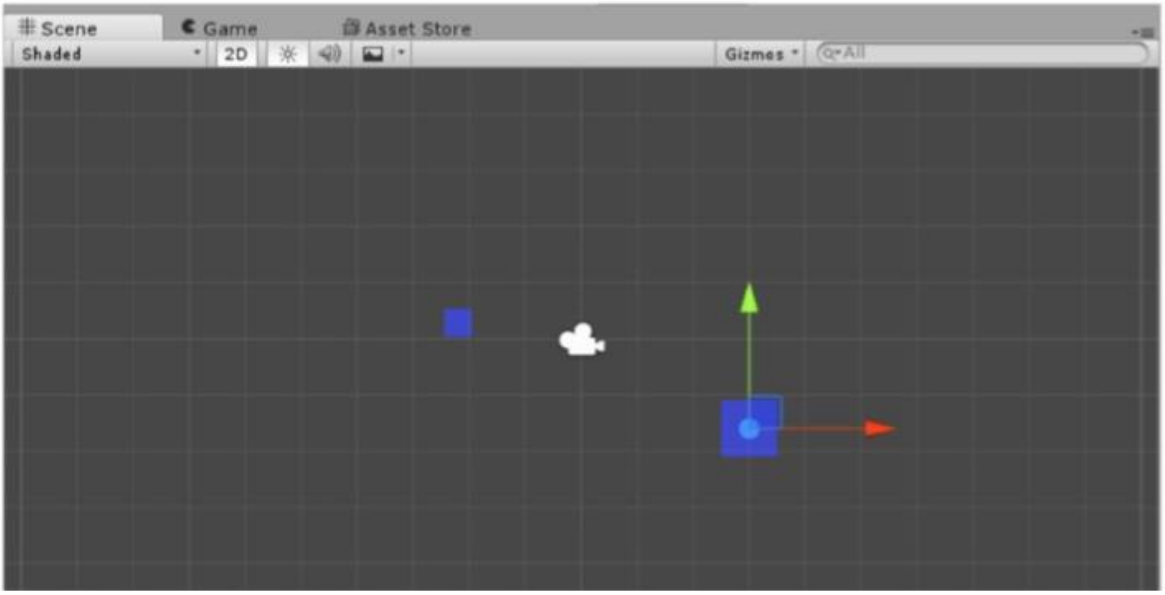
Memanipulasi GameObjects di Scene View

Jika Anda ingin melakukan ini secara bebas, Anda dapat memilih untuk menggunakan alat di bagian atas layar, di sebelah kiri tepat di atas jendela Hierarchy. Alat-alat ini termasuk tangan dan berbagai panah melakukan hal yang berbeda

Gambar 2.11 Alat yang akan Anda gunakan dalam tampilan Scene

Alat ini mengubah cara Anda berinteraksi dengan tampilan Scene. Anda cukup mengklik alat yang ingin Anda indentasi dan memilihnya:

- Tangan di kiri terjauh memungkinkan Anda menyeret layar ke sekeliling dan memindahkan tampilan Anda, yang berguna untuk menggulir melalui tingkat yang besar.
- Alat yang terlihat seperti empat panah pada kompas adalah yang Anda gunakan untuk memindahkan Objek Game tertentu di sekitar layar (lihat Gambar 2.12). Anda cukup mengklik GameObject dan mulai memindahkannya dengan bebas di tampilan Scene, atau Anda dapat memilihnya lalu menyeret panah merah atau hijau untuk memindahkannya hanya di sumbu X atau Y.

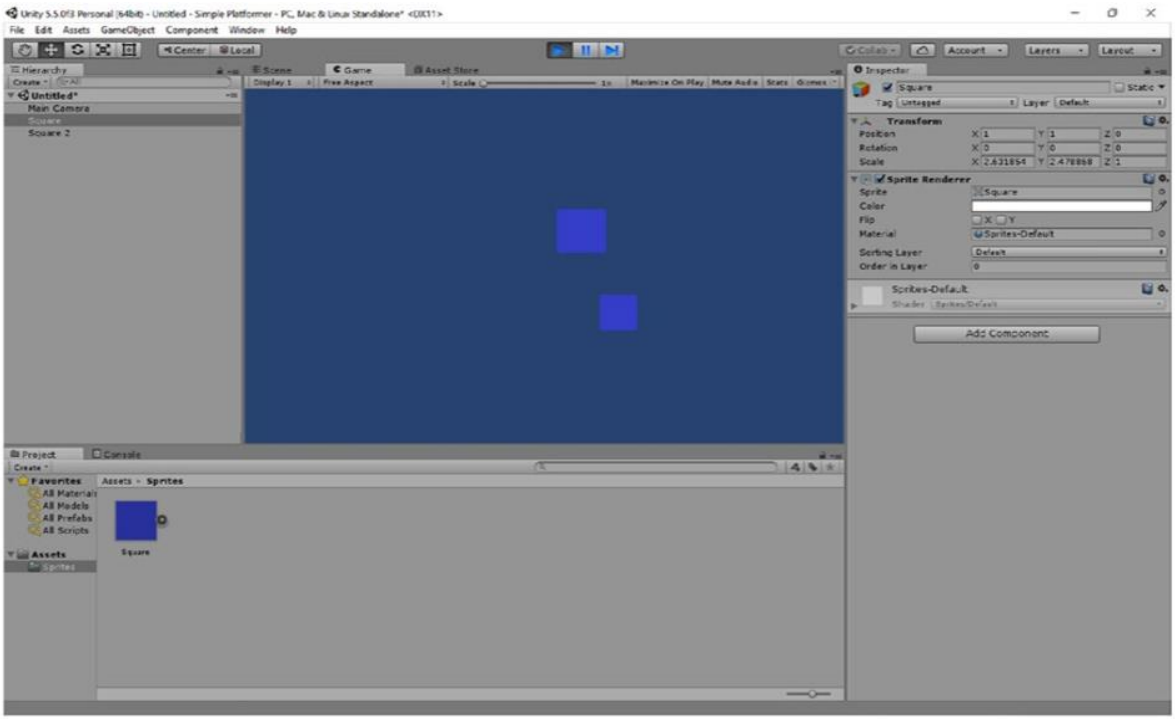


Gambar 2.12 Alat tarik

- Dua panah melengkung menunjukkan alat rotasi. Pilih ini dan sebuah lingkaran akan muncul di sekitar objek permainan yang dipilih yang memungkinkan Anda memutarnya dalam dua dimensi.
- Kemudian Anda memiliki alat scale, yang sekali lagi memberi Anda dua panah yang dapat Anda gunakan untuk scaling objek di sepanjang setiap sumbu.
- Alat terakhir adalah jack of all trades Anda. Ini memungkinkan Anda menyeret GameObjects, mengubah ukurannya dengan menarik sudutnya, atau menggambar kotak untuk memilih beberapa objek sekaligus.

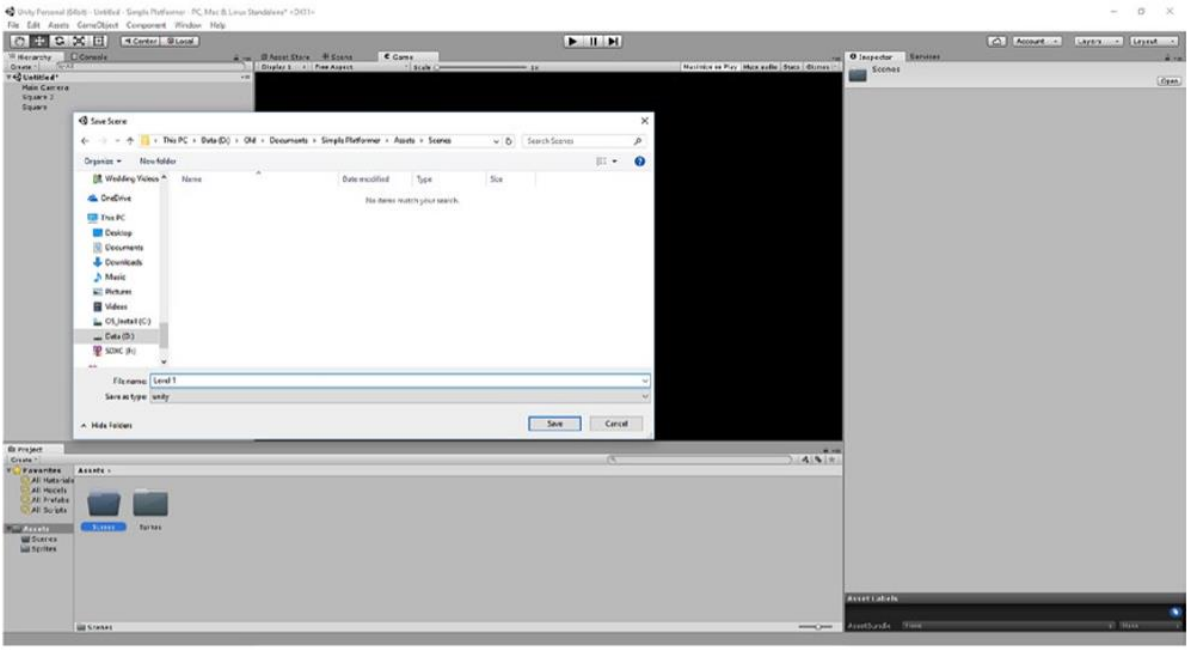
Baik Anda memindahkan objek secara bebas dengan alat atau dengan mengubah angka di Inspector, Anda dapat mengatur sprite sesuka Anda dan membuat beberapa lanskap yang tampak bagus. Tapi jangan sampai kita mendahului.

Menguji Game dan Menggunakan Kamera Jika ada satu kontrol di UI yang mungkin menarik perhatian Anda, itu adalah tombol putar. Kabar baiknya adalah ini melakukan persis seperti yang Anda harapkan: memungkinkan Anda untuk menguji permainan. Tekan tombol rotate dan game Anda akan berjalan di jendela Game, menunjukkan kepada Anda dua kotak yang terletak di latar belakang biru muda. Ini mungkin tidak akan mengejutkan dunia, tetapi selamat, Anda baru saja menjalankan program kerja pertama Anda. Ini adalah "Halo Dunia" kami, dan semuanya menjadi lebih menyenangkan dari sini.



Gambar 2.13 "Permainan" pertama Anda—selamat

Saat game sedang dimainkan, Anda dapat melihatnya di tampilan Game dan Anda dapat terus mengedit melalui Inspector atau di jendela Scene. Ingatlah bahwa ketika Anda melakukan ini, tidak ada yang akan menyelamatkan. Jika Anda memindahkan sprite saat permainan sedang berjalan, sprite akan melompat kembali ke posisi terakhirnya segera setelah Anda menghentikannya lagi. Gunakan ini untuk melihat preview perubahan "langsung" tetapi tidak untuk membuat perubahan permanen pada kode Anda. Coba dan masukkan ini ke kepala Anda sekarang. Bukan hal yang aneh untuk memindahkan banyak hal dan mengubah banyak kode, hanya untuk mengetahui bahwa Anda lupa menghentikan permainan agar tidak berjalan terlebih dahulu dan kehilangan semuanya. Untuk menghentikan permainan kapan saja, cukup tekan tombol rotate lagi. Perhatikan bahwa jika Anda ingin game menjadi layar penuh saat Anda menekan tombol play, Anda dapat mengklik Maksimalkan Saat Bermain. Ini berguna saat Anda menguji gim Anda dengan benar, atau Anda hanya ingin mencobanya dengan baik. Gambar 3-16 adalah tampilan game kami saat ini saat diledakkan.



Gambar 2.14 Game yang sama, hanya masif

2.4 KAMERA

Pengamat di antara pembaca mungkin telah memperhatikan bahwa ada lebih dari dua GameObjects dalam tampilan Hierarchy. Objek ketiga disebut Kamera Utama, dan jika Anda memilihnya, Anda akan melihat bahwa ini adalah ikon kamera putih yang mengambang di tampilan Scene Anda. Kamera Utama adalah GameObject, seperti sprite Anda, tetapi dengan komponen Kamera dan bukan sprite. Saat Anda menambahkan salah satu dari ini ke sebuah scene, itu menentukan di mana perspektif player dan apa yang akan mereka lihat di layar. Coba gerakkan kamera di sekitar dan kemudian tekan tombol putar, dan Anda akan menemukan bahwa itu berubah di mana kotak muncul di layar — kotak tidak bergerak, tetapi perspektif Anda telah berubah.

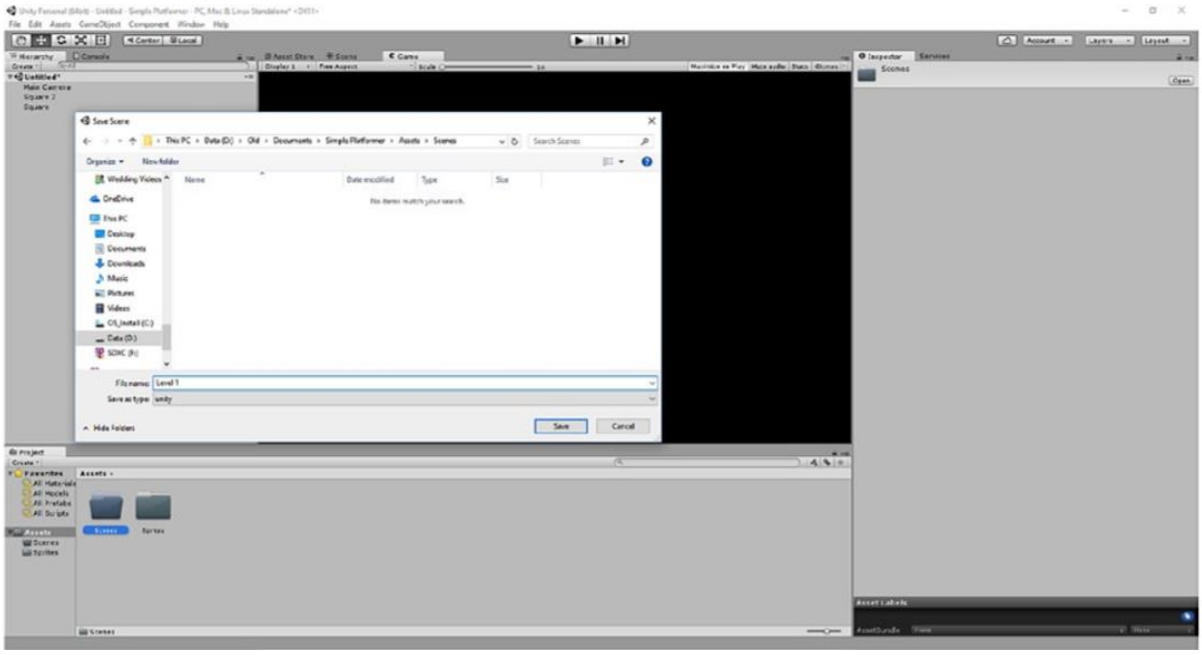
Mungkin terasa aneh bahwa kamera Anda diperlakukan dengan cara yang sama persis seperti sprite, tetapi inilah yang perlu Anda pahami tentang Unity: semuanya adalah GameObject. Skrip tidak berjalan kecuali jika dilampirkan ke GameObjects, dan ini mungkin mengharuskan Anda untuk memikirkan kembali cara Anda mendekati kode jika Anda terbiasa dengan bahasa lain. Tapi begitu Anda bisa mengatasinya, ini adalah cara kerja yang kuat dan fleksibel. P.S. Ini bukan apa yang dimaksud dengan pemrograman berorientasi objek. Ini terkait ... tapi saya akan menjelaskan lebih lanjut di bab berikutnya. Karena kita sedang melihat ke kamera, mari edit sesuatu yang mungkin membuat Anda frustrasi: warna latar belakang tampilan Game Anda. Klik kamera dan Anda akan melihat pengaturan yang disebut Latar Belakang di Inspector, yang saat ini berwarna biru. Jika Anda memilih warna, Anda akan diberi kesempatan untuk mengatur ini ke warna baru. Pilih hitam—dengan begitu, kotak biru Anda akan terlihat lebih jelas.

2.5 SIMPAN PROYEK DAN SCENE

Karena Anda mungkin sangat senang dengan kreasi menakjubkan ini, inilah saatnya Anda menyimpannya untuk memastikan tidak ada hal buruk yang terjadi padanya. Sebenarnya ada dua hal yang perlu Anda simpan di sini: proyek dan scene. Scene adalah apa yang Anda lihat melalui jendela Scene Anda dan itu mencakup segala sesuatu di Hierarchy Anda sekarang (dua kotak dan kamera). Untuk semua tingkat dan tujuan, scene adalah level

dalam banyak kasus, meskipun itu juga bisa merujuk ke layar judul atau menu opsi. Ini pada dasarnya adalah kumpulan GameObjects dan skrip yang ingin Anda muat di beberapa titik dalam gim Anda.

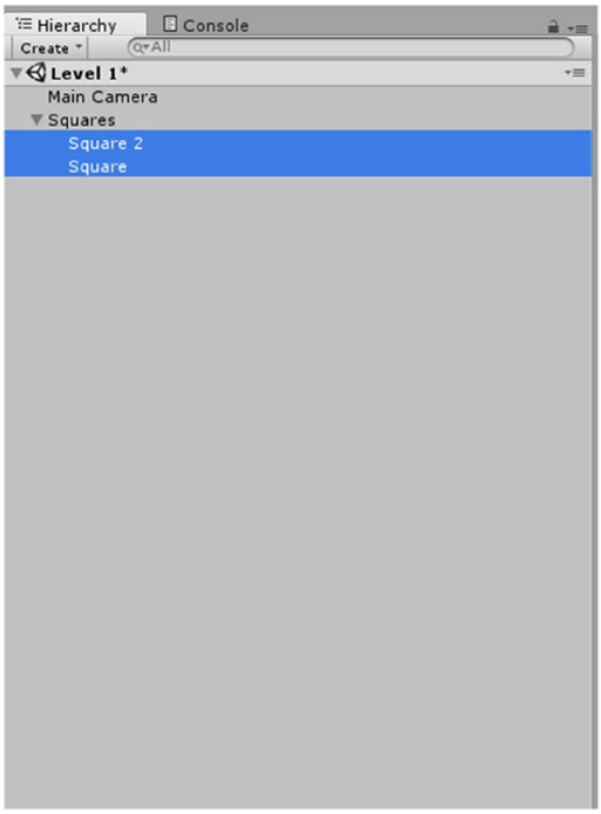
Untuk menyimpan proyek Anda, gunakan menu di bagian atas dan pilih File - Simpan Proyek. Untuk menyimpan scene, pertama-tama Anda ingin membuat subfolder baru di folder Aset Anda—sebut saja Scene. Sekarang pergi ke File - Save Scene. Saat dialog muncul, pilih folder Scene yang baru saja Anda buat dan panggil file Level 1. Ketika Anda memiliki beberapa scene dalam proyek Anda, Anda akan dapat beralih di antara mereka hanya dengan mengklik dua kali dari folder Scene.



Gambar 2.15 Menyimpan scene kami

Sedikit Lebih Banyak Organisasi

Karena kita sangat terorganisir dan terbiasa dengan kebiasaan baik sejak dini, mari buat satu folder lagi sebelum melanjutkan. Klik kanan ruang kosong di Hierarchy Anda dan klik Create Empty. Ini akan membuat GameObject "kosong", yang disebut GameObject. Seharusnya sudah dipilih, jadi pergilah ke Hierarchy dan beri nama Squares. Anda secara teoritis dapat mengubah GameObject kosong ini menjadi GameObject jenis lain dengan mengklik Tambahkan Komponen dan kemudian memilih komponen Kamera atau Sprite Renderer, misalnya. Alih-alih, kami membiarkannya kosong, yang memungkinkan kami melampirkan GameObjects lain ke dalamnya dan dengan demikian menggunakannya sebagai pengelompokan darurat

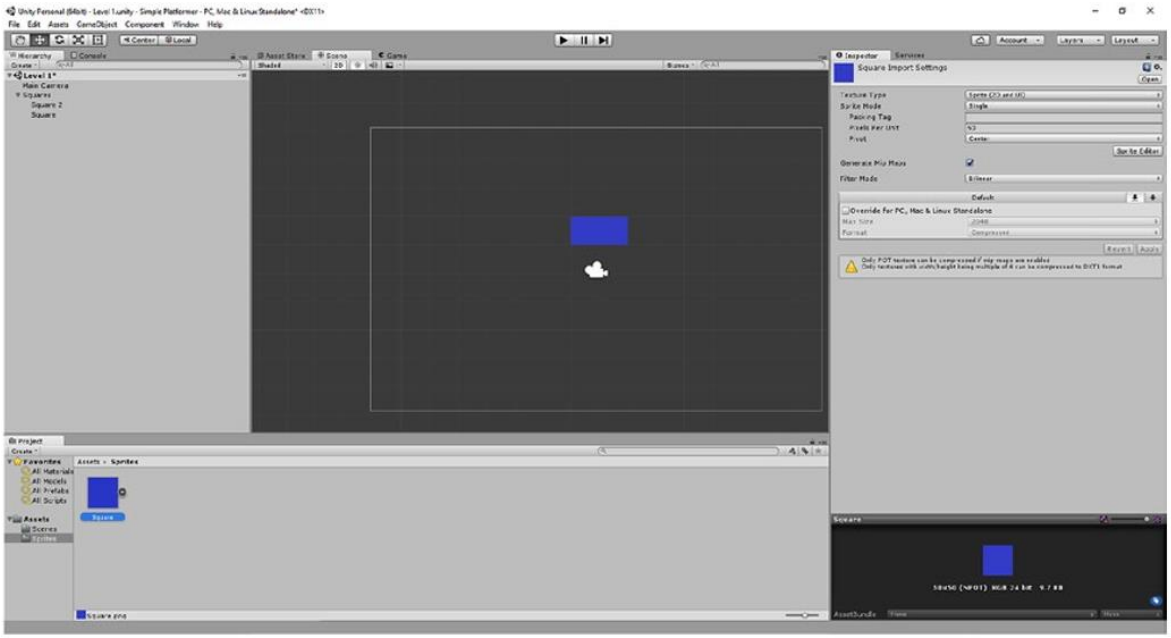


Gambar 2.16 Mulai atur Hierarchy Anda sekarang dan Anda akan sangat senang melakukannya

Drag dua kotak Anda dari tempat mereka berada ke GameObject kosong Anda, dan mereka sekarang akan disimpan di bawahnya. Panah tepat di sebelah GameObject yang kosong sekarang juga akan memungkinkan Anda untuk memperluas dan menciutkan item tersebut. Ini tidak benar-benar diperlukan pada saat ini, tetapi percayalah, ketika Anda memiliki 200 koin yang dapat dikoleksi, 30 musuh, dan 700 ubin di layar Anda, Anda akan senang untuk organisasi.

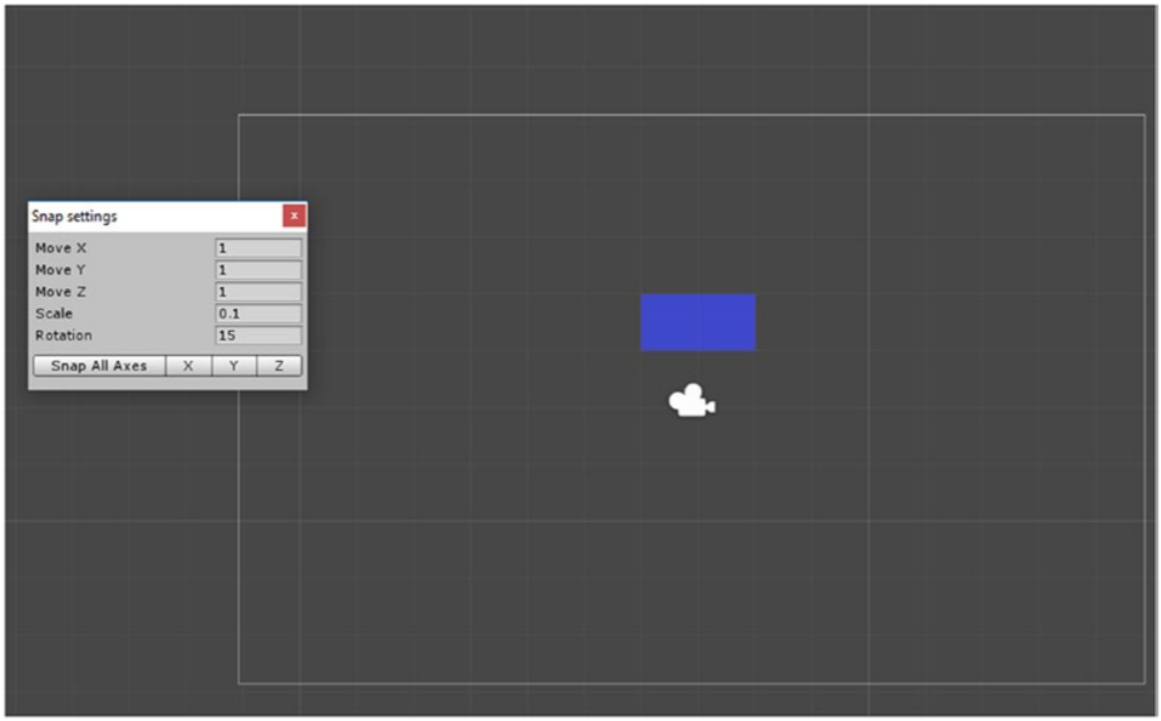
Satu Hal Terakhir yang Harus Disiapkan: Pengaturan Snap Grid

Anda mungkin telah memperhatikan bahwa ada kisi-kisi dalam tampilan Scene Anda dan bertanya-tanya tentang apa itu semua. Kisi ini terdiri dari unit, yang dapat mewakili apa pun yang Anda inginkan untuk membantu Anda mengatur sprite di sekitar layar. Apa yang Anda sebut program yang penuh dengan unit? Unity! (Oke, sebenarnya bukan dari mana nama itu berasal....) Saat ini, Anda akan melihat bahwa kuadrat dan satuan Anda tidak ada hubungannya. Untuk mengubahnya, pertama atur scale X dan scale Y pada kedua kotak kembali ke 1. Sekarang buka folder Sprite Anda dan pilih sprite Square sehingga terbuka di Inspector Anda (jangan klik salah satu Square GameObjects, karena itu tidak akan berhasil). Anda akan melihat opsi untuk Pixels Per Unit, yang mungkin disetel ke 100 secara default. Ubah itu menjadi 50—jangan lupa klik Terapkan di kanan bawah—dan Anda akan menemukan bahwa kedua kotak sekarang segera berubah menjadi ukuran yang sama dengan kotak pada kisi (seperti pada Gambar 3-19). Ingat, saat kami membuat sprite ini, kami membuatnya berukuran 50 x 50 piksel. Dengan menjadikannya 1 unit = 50 piksel, kita sekarang memiliki kotak berukuran sempurna.



Gambar 2.17 Pastikan untuk mengatur piksel Anda per unit untuk setiap sprite baru

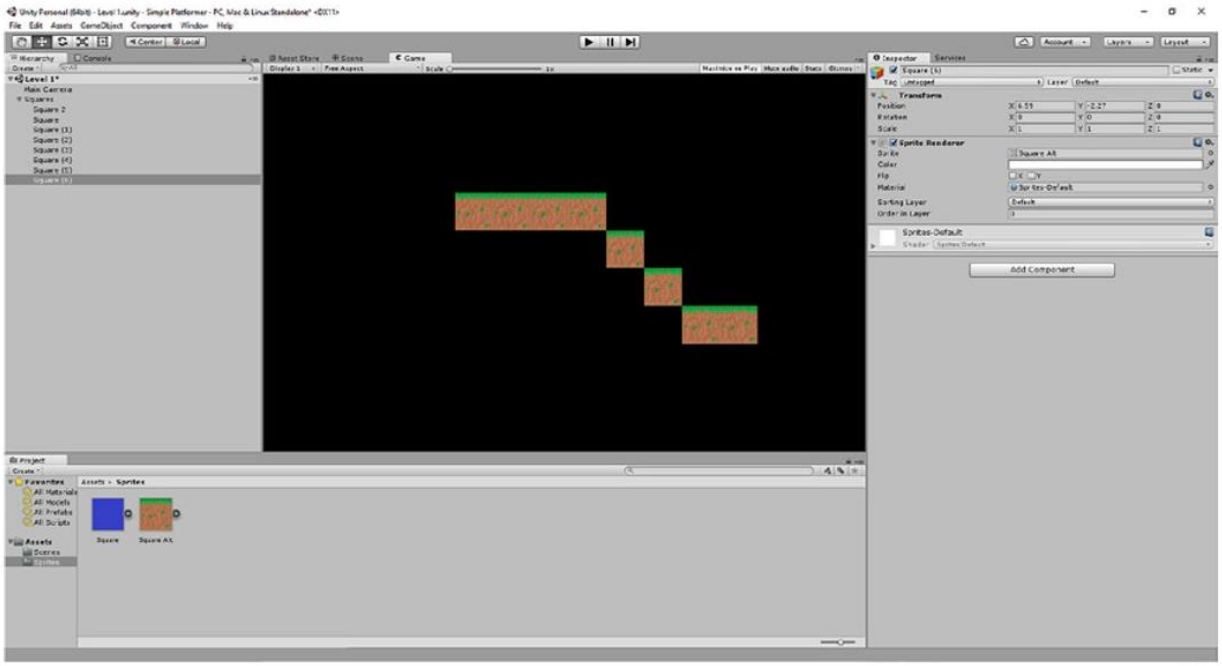
Trik berguna lainnya yang dapat kita gunakan sekarang adalah memasang sprite pada tempatnya. Saat menyeret sprite, tahan Ctrl dan Anda akan menemukan bahwa sprite melompat dari satu titik ke titik lain daripada bergerak dengan mulus. Anda dapat mengubah seberapa jauh jarak titik tersebut dengan memilih Edit - Pengaturan Snap. Anda akan menemukan bahwa ini mungkin diatur ke 1 untuk sumbu X dan Y masing-masing. Jika tidak, ubahlah menjadi



Gambar 2.18 Snap Setting

Tutup dialog itu dan gerakkan bebas salah satu kotak Anda sehingga diposisikan tepat di dalam batas salah satu kotak kotak. Sekarang tekan Ctrl + C (salin) dan kemudian Ctrl + V

(tempel). Ini akan menduplikasi Square GameObject Anda untuk membuat salinan persis di tempat yang sama persis. Tahan Ctrl dan drag kotak Anda ke kanan. Itu harus bergerak tepat satu lebar ubin untuk duduk rata di sebelah ubin Anda sebelumnya. Anda juga dapat menyalin dan menempelkan GameObjects dengan mengklik kanan dan memilih Salin dari tampilan Hierarki. Lakukan ini beberapa kali dan Anda dapat membuat tangga dan struktur lainnya. Ini mungkin terasa sangat tidak menarik pada saat ini, tetapi sebenarnya ini adalah keterampilan yang sangat penting. Saat membuat level dalam game, sangat penting bahwa semua ubin Anda diposisikan dengan sempurna tepat di samping satu sama lain dan tidak memiliki celah kecil selebar piksel di antara mereka. Mengapa tidak mencoba menggunakan sprite yang sedikit lebih rumit dan membuat sesuatu yang terlihat seperti dasar dari sebuah level?



Gambar 2.19 Kita akan belajar cara membuat sprite dan bahkan seni piksel di bab-bab selanjutnya

BAB 3
MEMULAI DEMO 3D DENGAN CODING DAN MENAMBAHKAN FISIKA

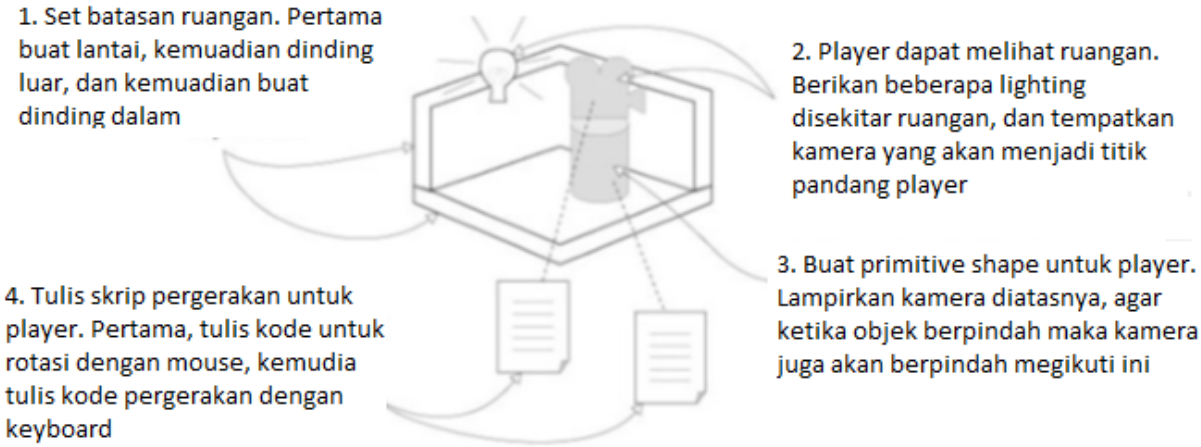
3.1 MEMBANGUN DEMO YANG MENEMPATKAN ANDA DI RUANG 3D

Unity memudahkan pendatang baru untuk memulai, tetapi mari kita bahas beberapa poin sebelum Anda membangun scene yang lengkap. Bahkan ketika bekerja dengan alat yang sefleksibel Unity, Anda perlu memiliki pemahaman tentang tujuan yang Anda tuju. Anda juga perlu memahami bagaimana koordinat 3D beroperasi atau Anda bisa tersesat segera setelah Anda mencoba memposisikan objek di scene.

Merencanakan proyek

Sebelum Anda mulai memprogram apa pun, Anda selalu ingin berhenti sejenak dan bertanya pada diri sendiri, "Jadi, apa yang saya bangun di sini?" Desain game adalah topik besar tersendiri, dengan banyak buku besar yang mengesankan berfokus pada cara mendesain game. Untungnya untuk tujuan kami, Anda hanya memerlukan garis besar singkat dari demo sederhana ini untuk mengembangkan proyek pembelajaran dasar. Proyek awal ini tidak akan menjadi desain yang terlalu rumit, untuk menghindari mengalihkan perhatian Anda dari mempelajari konsep pemrograman; Anda dapat (dan harus!) khawatir tentang masalah desain tingkat tinggi setelah Anda menguasai dasar-dasar developeran game.

Untuk proyek pertama ini, Anda akan membuat scene FPS (first-person shooter) dasar. Akan ada ruang untuk bernavigasi, player akan melihat dunia dari sudut pandang karakter mereka, dan player dapat mengontrol karakter menggunakan mouse dan keyboard. Semua kerumitan yang menarik dari permainan yang lengkap dapat dihilangkan untuk saat ini untuk berkonsentrasi pada mekanik inti: bergerak dalam ruang 3D. Gambar 2.2 menggambarkan peta jalan untuk proyek ini, pada dasarnya meletakkan daftar periksa mental yang saya buat di kepala saya:



Gambar 3.1 Peta jalan untuk demo 3D

1. Mengatur ruangan: buat lantai, dinding luar, dan dinding dalam.
2. Tempatkan lampu dan kamera.
3. Buat objek pemutar (termasuk memasang kamera di atas).
4. Tulis skrip gerakan: rotate dengan mouse dan gerakkan dengan keyboard.

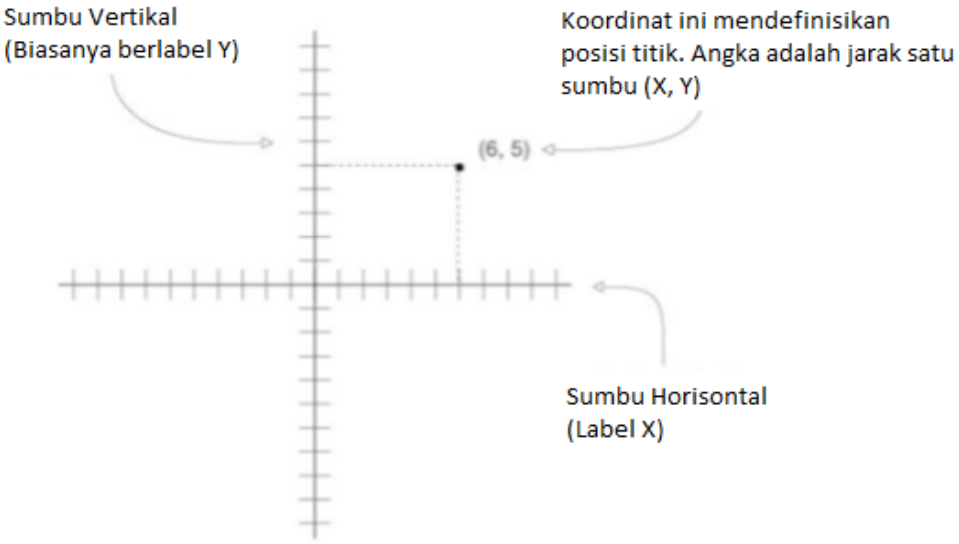
Jangan takut dengan semua yang ada di peta jalan ini! Kedengarannya seperti ada banyak hal dalam bab ini, tetapi Unity membuatnya mudah. Bagian mendatang tentang skrip gerakan sangat luas hanya karena kita akan membahas setiap baris untuk memahami semua konsep secara detail. Proyek ini adalah demo orang pertama untuk menjaga agar persyaratan seni

tetap sederhana; karena Anda tidak dapat melihat diri Anda sendiri, tidak apa-apa jika "Anda" berbentuk silinder dengan kamera di atasnya! Sekarang Anda hanya perlu memahami cara kerja koordinat 3D, dan akan mudah untuk menempatkan semuanya di editor visual.

3.2 MEMAHAMI RUANG KOORDINAT 3D

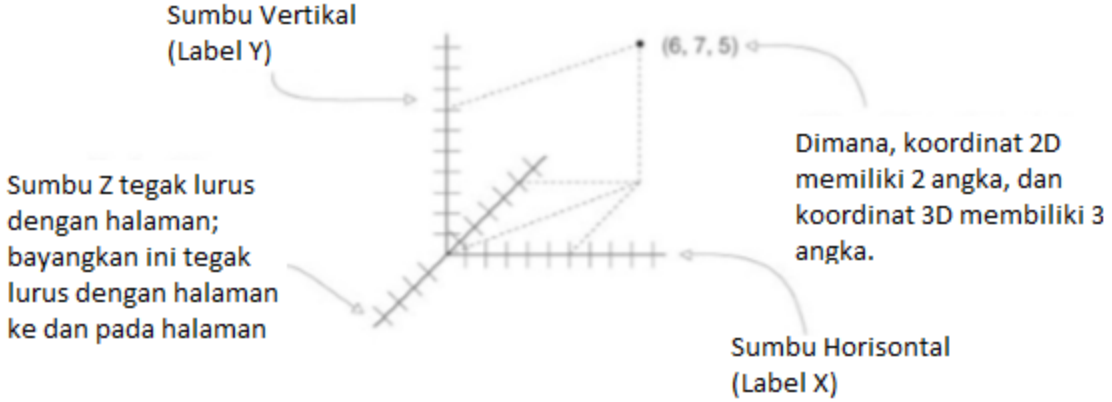
Jika Anda memikirkan rencana sederhana yang kami mulai, ada tiga aspek di dalamnya: ruangan, scene, dan kontrol. Semua item tersebut bergantung pada Anda untuk memahami bagaimana posisi dan gerakan direpresentasikan dalam simulasi komputer 3D, dan jika Anda baru bekerja dengan grafik 3D, Anda mungkin belum mengetahui hal itu.

Semuanya bermuara pada angka-angka yang menunjukkan titik-titik dalam ruang, dan cara angka-angka itu berkorelasi dengan ruang adalah melalui sumbu koordinat. Jika Anda mengingat kembali kelas matematika, Anda mungkin pernah melihat dan menggunakan sumbu X dan Y (lihat gambar 3.2) untuk menetapkan koordinat ke titik pada halaman, yang disebut sebagai sistem koordinat Cartesian.



Gambar 3.2 Koordinat sepanjang sumbu X dan Y menentukan titik 2D.

Dua sumbu memberi Anda koordinat 2D, dengan semua titik pada bidang yang sama. Tiga sumbu digunakan untuk mendefinisikan ruang 3D. Karena sumbu X menyusuri halaman secara horizontal dan sumbu Y menyusuri halaman secara vertikal, sekarang kita membayangkan sumbu ketiga yang menempel lurus ke dalam dan keluar halaman, tegak lurus terhadap sumbu X dan Y. Gambar 2.4 menggambarkan sumbu X-, Y-, dan Z untuk ruang koordinat 3D. Segala sesuatu yang memiliki posisi tertentu dalam scene akan memiliki koordinat XYZ: posisi player, penempatan dinding, dan sebagainya.



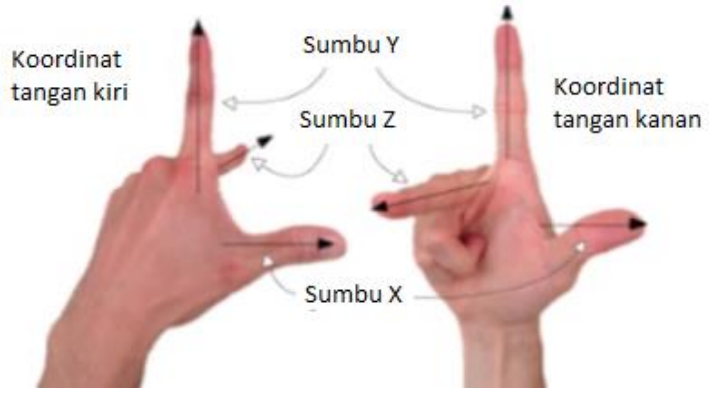
Gambar 3.3 Koordinat sepanjang sumbu X-, Y-, dan Z menentukan titik 3D.

Dalam tampilan Unity's Scene Anda dapat melihat ketiga sumbu ini ditampilkan, dan di Inspector Anda dapat mengetikkan tiga angka untuk memposisikan objek. Anda tidak hanya akan menulis kode untuk memposisikan objek menggunakan koordinat tiga angka ini, tetapi Anda juga dapat menentukan gerakan sebagai jarak untuk bergerak di sepanjang setiap sumbu.

Koordinat tangan kiri vs. tangan kanan

Arah positif dan negatif masing-masing sumbu berubah-ubah, dan koordinatnya tetap bekerja ke mana pun arah sumbunya. Anda hanya perlu tetap konsisten dalam alat grafis 3D yang diberikan (alat animasi, alat developeran game, dan sebagainya).

Tetapi di hampir semua kasus, X bergerak ke kanan dan Y naik; apa yang membedakan antara alat yang berbeda adalah apakah Z masuk atau keluar dari halaman. Kedua arah ini disebut sebagai "tangan kiri" atau "tangan kanan"; seperti yang ditunjukkan gambar ini, jika Anda mengarahkan ibu jari Anda di sepanjang sumbu X dan jari telunjuk Anda di sepanjang sumbu Y, maka jari tengah Anda menunjuk di sepanjang sumbu Z.

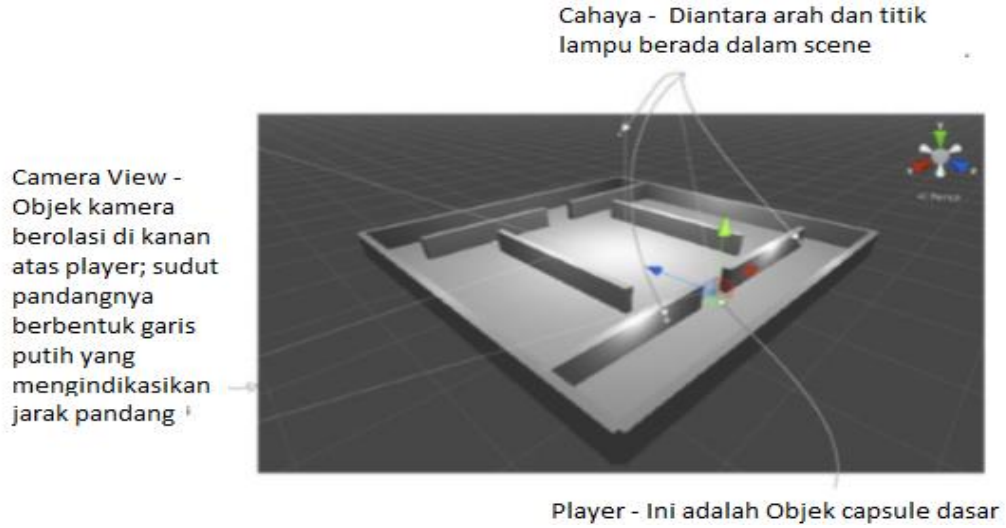


Gambar 3.4 Sumbu Z menunjuk ke arah yang berbeda di tangan kiri versus tangan kanan.

Unity menggunakan sistem koordinat tangan kiri, seperti halnya banyak aplikasi seni 3D. Banyak alat lain menggunakan sistem koordinat tangan kanan (OpenGL, misalnya), jadi jangan bingung jika Anda pernah melihat arah koordinat yang berbeda.

Mulailah proyek: tempatkan objek di scene

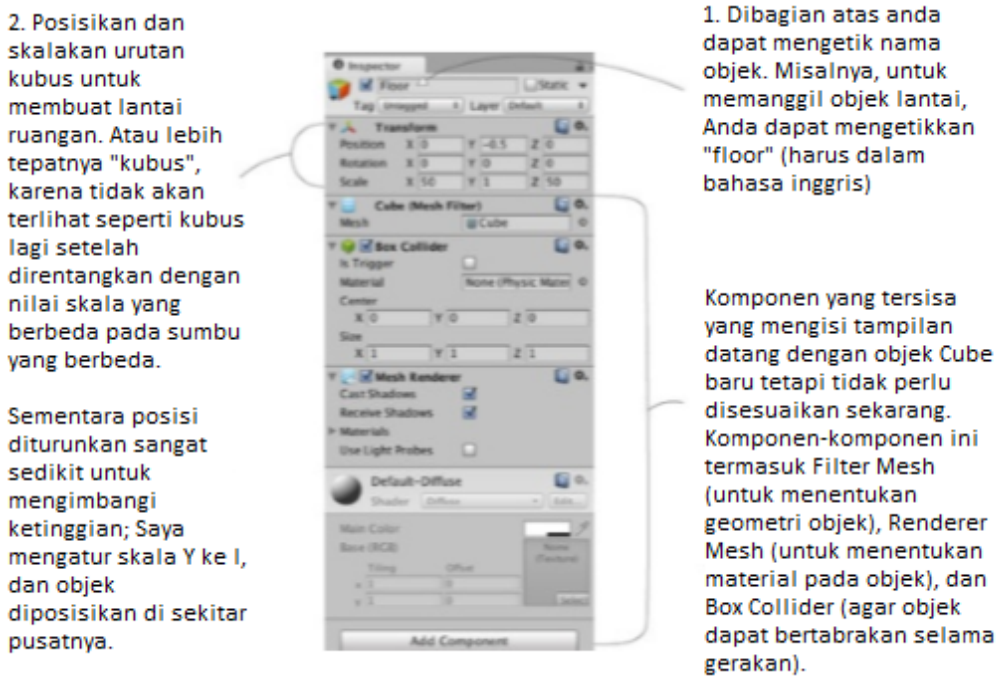
Baiklah, mari kita buat dan tempatkan objek di dalam scene. Pertama, Anda akan menyiapkan semua scene statis—lantai dan dinding. Kemudian Anda akan menempatkan lampu di sekitar scene dan memposisikan kamera. Terakhir Anda akan membuat objek yang akan menjadi player, objek yang akan Anda lampirkan skrip untuk berjalan di sekitar scene. Gambar 2.5 menunjukkan seperti apa tampilan editor dengan semua yang ada di tempatnya.



Gambar 3.5 Scene di Editor dengan lantai, dinding, lampu, kamera, dan pemutar

Scene: lantai, dinding luar, dinding dalam

Pilih menu GameObject di bagian atas layar, lalu arahkan kursor ke Objek 3D untuk melihat menu tarik-turun itu. Pilih Cube untuk membuat objek kubus baru di scene (nanti kita akan menggunakan bentuk lain seperti Sphere dan Capsule). Sesuaikan posisi dan scale objek ini, serta namanya, untuk membuat lantai; gambar 2.6 menunjukkan nilai apa yang harus ditetapkan lantai di Inspector (awalnya hanya kubus, sebelum Anda merentangkannya).

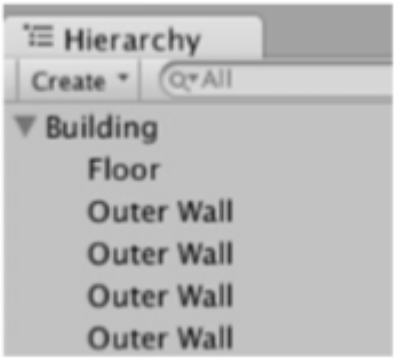


Gambar 3.6 Tampilan inspector untuk lantai

Catatan, Angka untuk posisi dapat berupa unit apa pun yang Anda inginkan, selama Anda konsisten di seluruh scene. Pilihan paling umum untuk satuan adalah meter dan itulah yang biasanya saya pilih, tetapi saya juga kadang-kadang menggunakan kaki dan saya bahkan pernah melihat orang lain memutuskan bahwa angkanya adalah inci!

Ulangi langkah yang sama untuk membuat dinding luar ruangan. Anda dapat membuat kubus baru setiap kali, atau Anda dapat menyalin dan menempelkan objek yang ada menggunakan pintasan standar. Gerakkan, putar, dan scalekan dinding untuk membentuk keliling di sekitar lantai. Bereksperimenlah dengan angka yang berbeda (misalnya, 1, 4, 50 untuk scale) atau gunakan alat transformasi yang pertama kali terlihat di bagian 1.2.2 (ingat bahwa istilah matematika untuk bergerak dan berputar dalam ruang 3D adalah "transform"). Tip Ingat juga kontrol navigasi sehingga Anda dapat melihat scene dari sudut yang berbeda atau memperkecil tampilan sekilas. Jika Anda tersesat dalam scene, tekan F untuk mengatur ulang tampilan pada objek yang dipilih saat ini.

Nilai transformasi yang tepat pada akhir dinding akan bervariasi tergantung pada bagaimana Anda memutar dan scaling kubus agar pas, dan pada bagaimana objek dihubungkan bersama dalam tampilan Hierarki. Misalnya, pada gambar 2.7 semua dinding adalah anak dari objek root yang kosong, sehingga daftar Hierarki akan terlihat terorganisir. Jika Anda memerlukan contoh untuk menyalin nilai kerja, unduh proyek sampel dan lihat dinding di sana.



Gambar 3.7 Tampilan Hierarchy menunjukkan dinding dan lantai yang diatur di bawah objek kosong

Tip Drag objek di atas satu sama lain dalam tampilan Hierarki untuk membuat hubungan. Objek yang memiliki objek lain yang melekat disebut sebagai induk; benda yang melekat pada benda lain disebut sebagai anak. Ketika objek induk dipindahkan (atau diputar atau discalekan), objek anak ditransformasikan bersamanya.

Tip Objek permainan kosong dapat digunakan untuk mengatur scene dengan cara ini. Dengan menautkan objek yang terlihat ke objek root, daftar Hierarchynya dapat diciutkan. Berhati-hatilah: sebelum menautkan objek anak apa pun ke sana, Anda ingin memposisikan objek root kosong pada 0, 0, 0 untuk menghindari keanehan posisi nanti.

Apa itu GameObject?

Semua objek scene adalah instance dari kelas GameObject, mirip dengan bagaimana semua komponen skrip mewarisi dari kelas MonoBehaviour. Fakta ini lebih eksplisit dengan objek kosong yang sebenarnya bernama GameObject tetapi tetap benar terlepas dari apakah objek tersebut bernama Lantai, Kamera, atau Player.

GameObject sebenarnya hanyalah wadah untuk sekumpulan komponen. Tujuan utama GameObject adalah agar MonoBehaviour memiliki sesuatu untuk dilampirkan. Apa sebenarnya objek dalam scene tergantung pada komponen apa yang telah ditambahkan ke GameObject itu. Objek Cube memiliki komponen Cube, objek Sphere memiliki komponen Sphere, dan seterusnya. Setelah dinding luar terpasang, buat beberapa dinding dalam untuk bernavigasi. Posisikan dinding bagian dalam sesuka Anda; idenya adalah untuk membuat beberapa lorong dan rintangan untuk dilalui begitu Anda menulis kode untuk gerakan. Sekarang scene memiliki ruangan di dalamnya, tetapi tanpa lampu apa pun, player tidak akan dapat melihatnya. Mari kita urus itu selanjutnya.

3.3 LAMPU DAN KAMERA

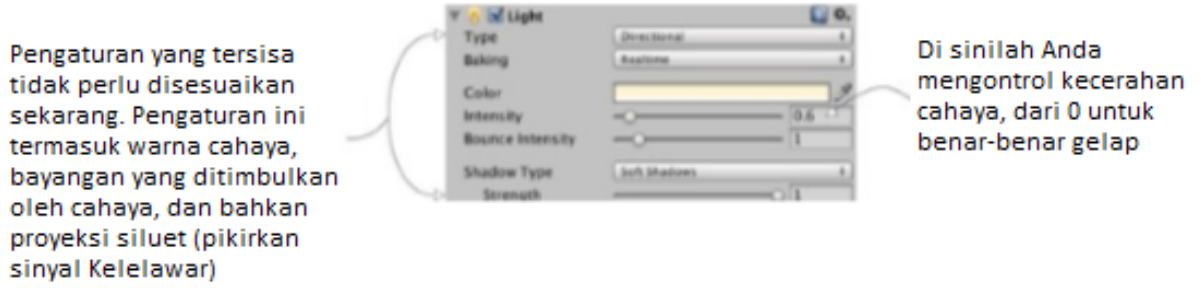
Biasanya Anda menyalakan scene 3D dengan cahaya terarah dan kemudian serangkaian lampu titik. Pertama mulai dengan lampu arah; scene mungkin sudah memilikinya secara default, tetapi jika tidak, buat satu dengan memilih GameObject > Light dan pilih Directional Light.

Jenis lampu

Anda dapat membuat beberapa jenis sumber cahaya, ditentukan oleh bagaimana dan di mana mereka memproyeksikan sinar cahaya. Tiga jenis utama adalah titik, titik, dan arah. *Point lights*adalah jenis sumber cahaya di mana semua sinar cahaya berasal dari satu titik dan diproyeksikan ke segala arah, seperti bola lampu di dunia nyata. Cahaya lebih terang dari dekat karena sinar cahaya berkumpul. *Spotlights*adalah jenis sumber cahaya di mana semua sinar cahaya berasal dari satu titik tetapi hanya diproyeksikan keluar dalam kerucut terbatas. Tidak ada lampu sorot yang digunakan dalam proyek saat ini, tetapi lampu ini biasanya digunakan

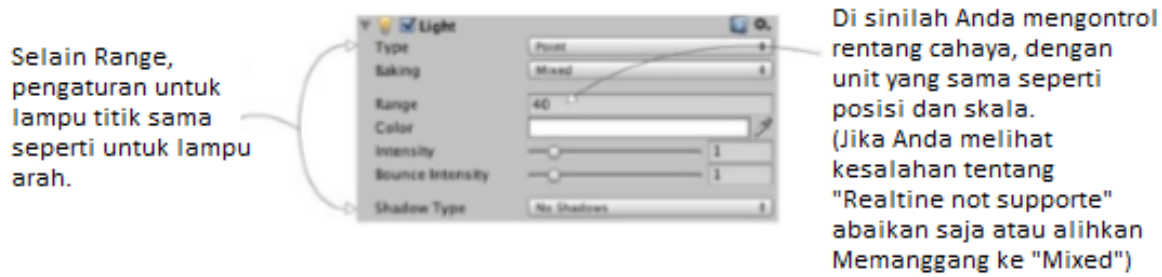
untuk menyorot bagian dari suatu level. *Directionallights*adalah sejenis sumber cahaya di mana semua sinar cahaya sejajar dan diproyeksikan secara merata, menerangi segala sesuatu di scene dengan cara yang sama. Ini seperti matahari di dunia nyata.

Posisi cahaya terarah tidak memengaruhi pancaran cahaya darinya, hanya rotasi sumber cahaya yang menghadap, jadi secara teknis Anda dapat menempatkan cahaya itu di mana saja dalam scene. Saya sarankan untuk menempatkannya tinggi di atas ruangan sehingga secara intuitif terasa seperti matahari dan tidak menghalangi saat Anda memanipulasi sisa scene. rotate lampu ini dan lihat efeknya pada ruangan; Saya sarankan untuk memutarnya sedikit pada sumbu X dan Y untuk mendapatkan efek yang baik. Anda dapat melihat pengaturan Intensitas saat Anda melihat di Inspector (lihat gambar 3.8). Sesuai namanya, pengaturan itu mengontrol kecerahan cahaya. Jika ini adalah satu-satunya cahaya, itu harus lebih intens, tetapi karena Anda akan menambahkan banyak lampu titik juga, cahaya terarah ini bisa sangat redup, seperti Intensitas 0,6.



Gambar 3.8 Pengaturan lampu arah di Inspector

Untuk lampu titik, buat beberapa menggunakan menu yang sama dan letakkan di sekitar ruangan di tempat gelap untuk memastikan semua dinding menyala. Anda tidak ingin terlalu banyak (kinerja akan menurun jika permainan memiliki banyak lampu), tetapi satu di dekat setiap sudut seharusnya baik-baik saja (saya sarankan menaikkannya ke puncak dinding), ditambah satu ditempatkan tinggi di atas scene (seperti a Y dari 18) untuk memberikan beberapa variasi pada cahaya di dalam ruangan. Perhatikan bahwa lampu titik memiliki pengaturan untuk Range yang ditambahkan ke Inspector (lihat gambar 3.9). Ini mengontrol seberapa jauh cahaya mencapai; sedangkan lampu arah memancarkan cahaya secara merata ke seluruh scene, lampu titik lebih terang saat objek lebih dekat. Titik lampu yang lebih rendah ke lantai harus memiliki jangkauan sekitar 18, tetapi lampu yang ditempatkan di atas harus memiliki jangkauan sekitar 40 untuk mencapai seluruh ruangan.



Gambar 3.9 Arahkan pengaturan lampu di Inspector

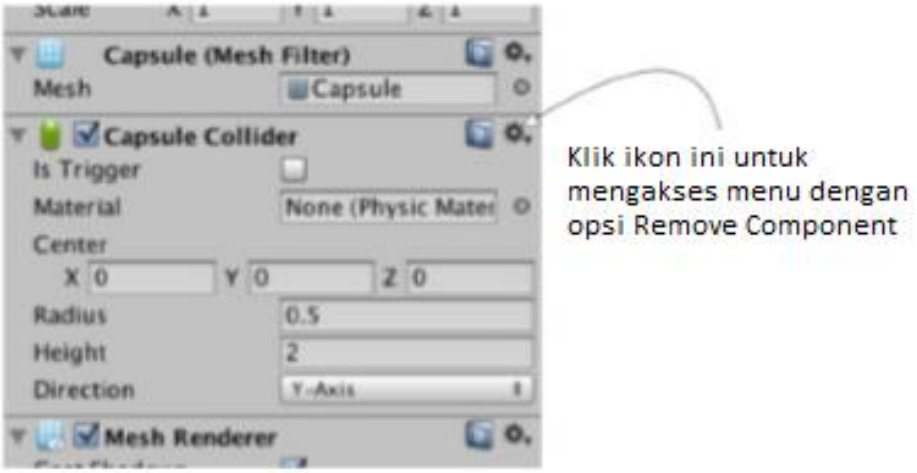
Jenis objek lain yang diperlukan agar player dapat melihat scene adalah kamera, tetapi scene "kosong" sudah datang dengan kamera utama, jadi Anda akan menggunakannya. Jika Anda perlu membuat kamera baru (seperti untuk tampilan layar terpisah dalam game

multiplayer), Kamera adalah pilihan lain di menu GameObject yang sama dengan Cube dan Lights. Kamera akan diposisikan di sekitar bagian atas pemutar sehingga tampilan tampak seperti mata player.

3.4 COLLIDER DAN SUDUT PANDANG PLAYER

Untuk proyek ini, bentuk primitif sederhana akan digunakan untuk mewakili player. Di menu GameObject (ingat, arahkan kursor ke Objek 3D untuk memperluas menu) klik Kapsul. Unity menciptakan bentuk silinder dengan ujung membulat; bentuk primitif ini akan mewakili player. Posisikan objek ini pada 1,1 pada sumbu Y (setengah tinggi objek, ditambah sedikit untuk menghindari tumpang tindih lantai). Anda dapat memindahkan objek pada X dan Z di mana pun Anda suka, selama berada di dalam ruangan dan tidak menyentuh dinding. Beri nama objek Player.

Di Inspector Anda akan melihat bahwa objek ini memiliki penumbuk kapsul yang ditugaskan padanya. Itu adalah pilihan default yang logis untuk objek kapsul, sama seperti objek kubus yang memiliki penumbuk kotak secara default. Tetapi objek khusus ini akan menjadi player dan karenanya membutuhkan jenis komponen yang sedikit berbeda dari kebanyakan objek. Lepaskan penumbuk kapsul dengan mengklik ikon roda gigi di kanan atas komponen tersebut, yang ditunjukkan pada gambar 2.10; yang akan menampilkan menu yang menyertakan opsi Hapus Komponen. Collider adalah mesh hijau yang mengelilingi objek, jadi Anda akan melihat mesh hijau menghilang setelah menghapus penumbuk kapsul.



Gambar 3. Menghapus komponen di Inspector

Alih-alih penumbuk kapsul, kita akan menetapkan controller karakter ke objek ini. Di bagian bawah Inspector ada tombol berlabel Tambah Komponen; klik tombol itu untuk membuka menu komponen yang dapat Anda tambahkan. Di bagian Fisika menu ini Anda akan menemukan Controller Karakter; pilih opsi itu. Sesuai dengan namanya, komponen ini akan memungkinkan objek berperilaku seperti karakter.

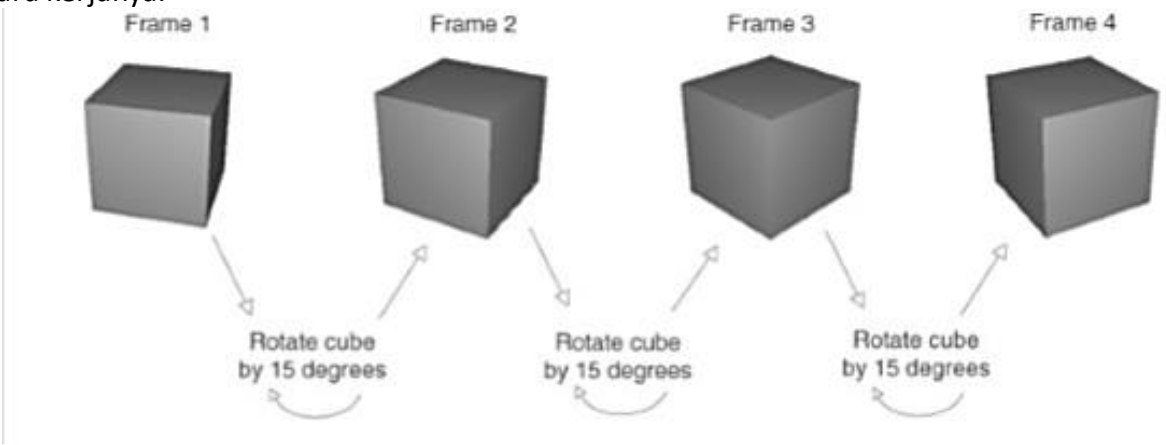
Anda harus menyelesaikan satu langkah terakhir untuk menyiapkan objek pemutar: memasang kamera. Seperti disebutkan di bagian sebelumnya tentang lantai dan dinding, objek dapat didrag satu sama lain dalam tampilan Hierarki. Drag objek kamera ke kapsul pemutar untuk memasang kamera ke pemutar. Sekarang posisikan kamera sehingga terlihat seperti mata player (saya sarankan posisi 0, 0,5, 0). Jika perlu, setel ulang rotasi kamera ke 0, 0, 0 (ini akan mati jika Anda memutar kapsul). Anda telah membuat semua objek yang diperlukan untuk scene ini. Yang tersisa adalah menulis kode untuk memindahkan objek player.

Membuat sesuatu bergerak: skrip yang menerapkan transformasi

Agar player berjalan di sekitar scene, Anda akan menulis skrip gerakan yang dilampirkan ke pemutar. Ingat, komponen adalah bit fungsionalitas modular yang Anda tambahkan ke objek, dan skrip adalah sejenis komponen. Akhirnya skrip tersebut akan merespons input keyboard dan mouse, tetapi pertama-tama buat pemutar berputar di tempatnya. Awal ini akan mengajarkan Anda bagaimana menerapkan transformasi dalam kode. Ingat bahwa tiga transformasi adalah Translate, Putar, dan Scalekan; memutar objek berarti mengubah rotasi. Tetapi ada lebih banyak yang perlu diketahui tentang tugas ini daripada hanya "ini melibatkan rotasi."

Membuat diagram bagaimana gerakan diprogram

Menganimasikan suatu objek (seperti membuatnya berputar) bermuara pada memindahkannya dalam jumlah kecil setiap frame, dengan frame yang diputar berulang-ulang. Dengan sendirinya, transformasi berlaku secara instan, sebagai lawan dari pergerakan yang terlihat dari waktu ke waktu. Tetapi menerapkan transformasi berulang kali menyebabkan objek terlihat bergerak, seperti serangkaian gambar diam di flipbook. Gambar 2.11 diagram cara kerjanya.



Gambar 3.11 Munculnya gerakan: proses siklus transformasi antara gambar diam

Ingat bahwa komponen skrip memiliki metode Update() yang menjalankan setiap frame. Untuk memutar kubus, tambahkan kode di dalam Update() yang memutar kubus sedikit. Kode ini akan berjalan berulang-ulang setiap frame. Kedengarannya cukup sederhana, bukan?

Menulis kode untuk mengimplementasikan diagram

Sekarang mari kita terapkan konsep yang baru saja dibahas. Buat skrip C# baru (ingat ada di submenu Buat di menu Aset), beri nama Spin, dan tulis kode dari daftar berikut (jangan lupa untuk menyimpan file setelah mengetik di dalamnya!).

```
using UnityEngine;
using System.Collections;

public class Spin : MonoBehaviour {
    public float speed = 3.0f;

    void Update() {
        transform.Rotate(0, speed, 0);
    }
}
```

Deklarasikan variabel publik untuk kecepatan atau rotasi

Letakkan perintah Rotate di sini sehingga menjalankan setiap frame

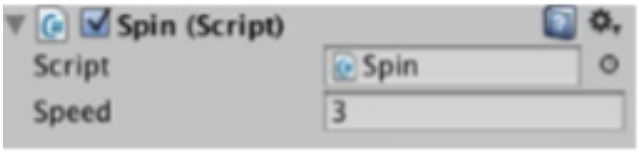
Gambar 3.12 Membuat benda berputar

Untuk menambahkan komponen skrip ke objek pemutar, drag skrip ke atas dari tampilan Proyek dan letakkan di Pemutar dalam tampilan Hierarki. Sekarang tekan Mainkan dan Anda akan melihat tampilan berputar; Anda telah menulis kode untuk membuat objek bergerak! Kode ini adalah template default untuk skrip baru ditambah dua baris baru yang ditambahkan, jadi mari kita periksa apa yang dilakukan kedua baris tersebut.

Pertama ada variabel untuk kecepatan yang ditambahkan ke bagian atas definisi kelas. Ada dua alasan untuk mendefinisikan kecepatan rotasi sebagai variabel: satu adalah aturan pemrograman standar "tidak ada angka ajaib", dan alasan kedua khusus untuk bagaimana Unity menampilkan variabel publik. Unity melakukan sesuatu yang berguna dengan variabel publik dalam komponen skrip, seperti yang dijelaskan dalam tip berikut.

Tip, Variabel publik diekspos di Inspector sehingga Anda dapat menyesuaikan nilai komponen setelah menambahkan komponen ke objek game. Ini disebut sebagai "serialisasi" nilai, karena Unity menyimpan status variabel yang dimodifikasi.

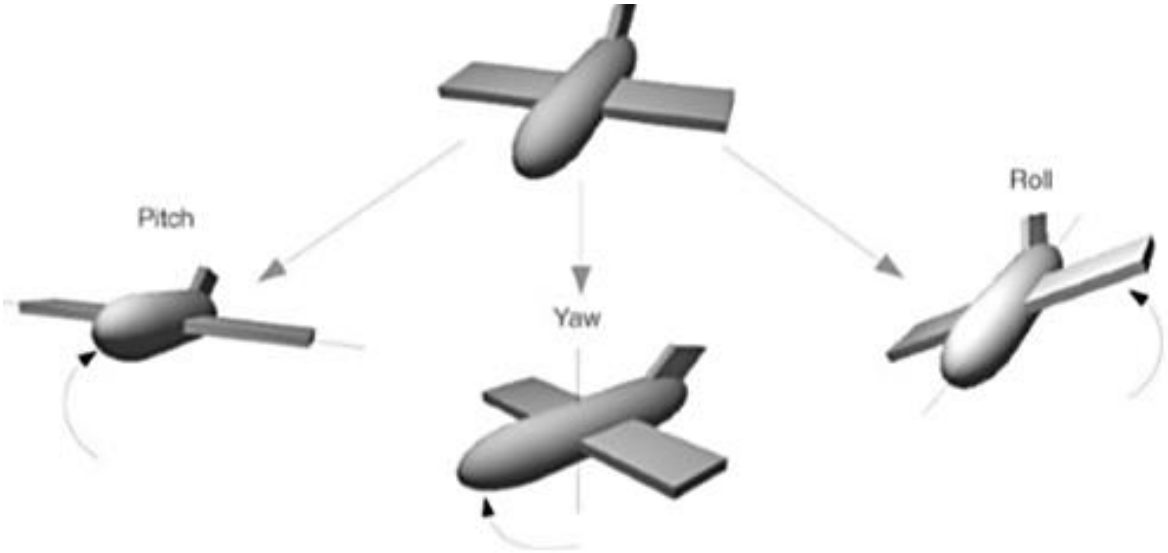
Gambar dibawah ini menunjukkan seperti apa komponen skrip di Inspector. Anda dapat mengetikkan nomor baru, lalu skrip akan menggunakan nilai itu alih-alih nilai default yang ditentukan dalam kode. Ini adalah cara praktis untuk menyesuaikan pengaturan komponen pada objek yang berbeda, bekerja di dalam editor visual alih-alih melakukan hardcoding setiap nilai.



Gambar 3.13 Inspector menampilkan variabel publik yang dideklarasikan dalam skrip

Baris kedua yang diperiksa dari daftar 2.1 adalah metode Rotate(). Itu ada di dalam Update() sehingga perintah menjalankan setiap frame. Rotate() adalah metode dari kelas Transform, jadi ini disebut dengan notasi titik melalui komponen transformasi objek ini (seperti dalam kebanyakan bahasa berorientasi objek, this.transform tersirat jika Anda mengetikkan transformasi). Transformasi diputar dengan derajat kecepatan setiap frame, menghasilkan gerakan berputar yang mulus. Tetapi mengapa parameter untuk Rotate() terdaftar sebagai (0, speed, 0) sebagai lawan, katakanlah, (speed, 0, 0)?

Ingatlah bahwa ada tiga sumbu dalam ruang 3D, berlabel X, Y, dan Z. Cukup intuitif untuk memahami bagaimana sumbu ini berhubungan dengan posisi dan gerakan, tetapi sumbu ini juga dapat digunakan untuk menggambarkan rotasi. Aeronautika menjelaskan rotasi dengan cara yang sama, sehingga programmer yang bekerja dengan grafik 3D sering menggunakan serangkaian istilah yang dipinjam dari aeronautika: pitch, yaw, dan roll. Gambar 2.13 mengilustrasikan apa arti istilah-istilah ini; pitch adalah rotasi di sekitar sumbu X, yaw adalah rotasi di sekitar sumbu Y, dan roll adalah rotasi di sekitar sumbu Z.



Gambar 3.14 Ilustrasi rotasi pitch, yaw, dan roll pesawat terbang

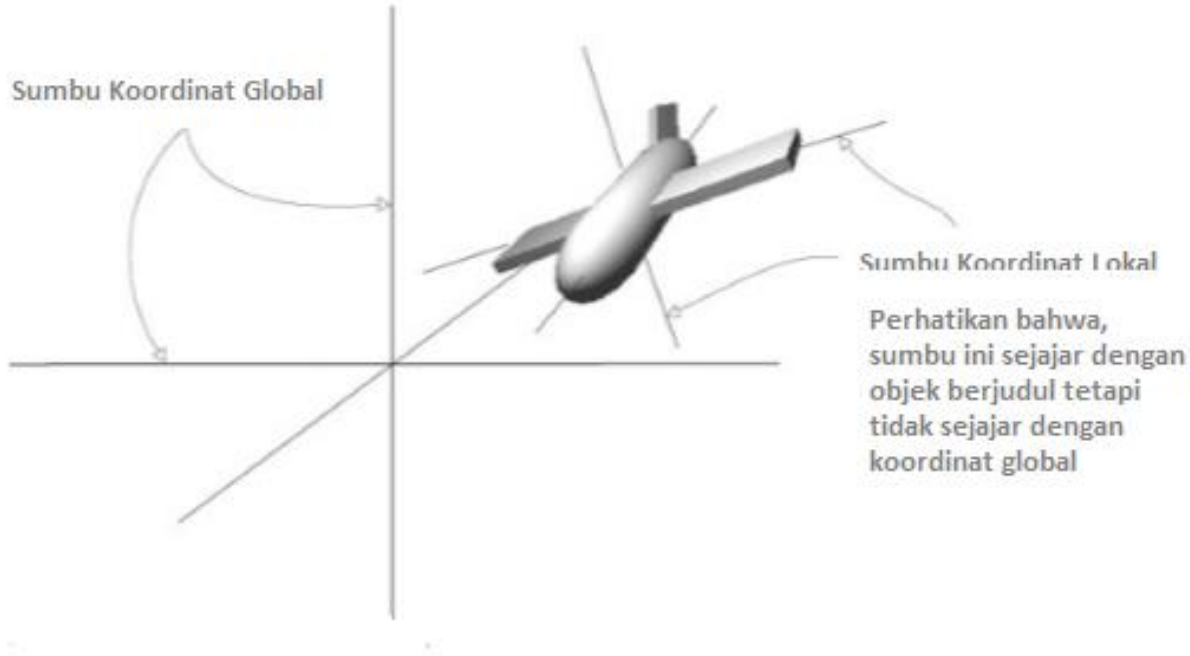
Mengingat bahwa kita dapat menggambarkan rotasi di sekitar sumbu X-, Y-, dan Z, itu berarti tiga parameter untuk Rotate() adalah rotasi X, Y, dan Z. Karena kami hanya ingin player berputar ke samping, bukan miring ke atas dan ke bawah, seharusnya hanya ada angka yang diberikan untuk rotasi Y, dan hanya 0 untuk rotasi X dan Z. Semoga Anda bisa menebak apa yang akan terjadi jika Anda mengubah parameter menjadi (kecepatan, 0, 0) dan kemudian memainkannya; coba itu sekarang! Ada satu poin penting lainnya yang perlu dipahami tentang rotasi dan sumbu koordinat 3D, yang diwujudkan dalam parameter keempat opsional untuk metode Rotate().

3.5 RUANG KOORDINAT LOKAL VS. GLOBAL

Secara default, metode Rotate() beroperasi pada apa yang disebut koordinat lokal. Jenis koordinat lain yang dapat Anda gunakan adalah global. Anda memberi tahu metode apakah akan menggunakan koordinat lokal atau global menggunakan parameter keempat opsional dengan menulis Space.Self atau Space.World seperti:

Rotasi (0, kecepatan, 0, Ruang.Dunia)

Lihat kembali penjelasan tentang ruang koordinat 3D, dan renungkan pertanyaan berikut: Di manakah (0, 0, 0) berada? Ke manakah arah sumbu X yang ditunjukkan? Bisakah sistem koordinat itu sendiri bergerak? Ternyata setiap objek memiliki titik asalnya sendiri, serta arahnya sendiri untuk ketiga sumbu, dan sistem koordinat ini bergerak bersama objek. Ini disebut sebagai koordinat lokal. Scene 3D keseluruhan juga memiliki titik asal dan arahnya sendiri untuk tiga sumbu, dan sistem koordinat ini tidak pernah bergerak. Ini disebut sebagai koordinat global. Jadi, ketika Anda menentukan lokal atau global untuk metode Rotate(), Anda memberi tahu sumbu X-, Y, dan Z-nya untuk berputar (lihat gambar 3.15).



Gambar 3.15 Sumbu koordinat lokal vs. Global

Jika Anda baru mengenal grafik 3D, ini adalah konsep yang membingungkan. Sumbu yang berbeda digambarkan pada Gambar 3.15 (perhatikan bagaimana "kiri" ke pesawat adalah arah yang berbeda dari "kiri" ke dunia) tetapi cara termudah untuk memahami lokal dan global adalah melalui sebuah contoh. Pertama, pilih objek player dan kemudian miringkan sedikit (seperti 30 untuk rotasi X). Ini akan membuang koordinat lokal, sehingga rotasi lokal dan global akan terlihat berbeda. Sekarang coba jalankan skrip Spin baik dengan dan tanpa Space.World ditambahkan ke parameter; jika terlalu sulit bagi Anda untuk memvisualisasikan apa yang terjadi, coba lepaskan komponen putaran dari objek pemutar dan alih-alih rotate kubus miring yang ditempatkan di depan pemutar. Anda akan melihat objek berputar di sekitar sumbu yang berbeda saat Anda mengatur perintah ke koordinat lokal atau global.

Komponen skrip untuk melihat-lihat: MouseLook

Sekarang Anda akan membuat rotasi merespons input dari mouse (yaitu, rotasi objek yang dilampirkan skrip ini, yang dalam hal ini adalah pemutarnya). Anda akan melakukan ini dalam beberapa langkah, secara bertahap menambahkan kemampuan gerakan baru ke karakter. Pertama player hanya akan berputar dari sisi ke sisi, dan kemudian player hanya akan berputar ke atas dan ke bawah. Akhirnya player akan dapat melihat sekeliling ke segala arah (berputar secara horizontal dan vertikal pada saat yang sama), perilaku yang disebut sebagai tampilan mouse. Mengingat bahwa akan ada tiga jenis perilaku rotasi yang berbeda (horizontal, vertikal, dan keduanya), Anda akan mulai dengan menulis kerangka kerja untuk mendukung ketiganya. Buat skrip C# baru, beri nama MouseLook, dan tulis kode dari daftar berikutnya.

```
using UnityEngine;
using System.Collections;

public class MouseLook : MonoBehaviour {
  public enum RotationAxes {
    MouseXAndY = 0,
    MouseX = 1,
    MouseY = 2
  }
  public RotationAxes axes = RotationAxes.MouseXAndY;

  void Update() {
    if (axes == RotationAxes.MouseX) {
      // horizontal rotation here
    }
    else if (axes == RotationAxes.MouseY) {
      // vertical rotation here
    }
    else {
    // both horizontal and vertical rotation here
    }
  }
}
```

Tentukan struktur data enum untuk mengaitkan nama dengan pengaturan

Deklarasikan variabel publik untuk disetel di editor Unity.

Letakkan kode di sini hanya untuk rotasi horizontal.

Letakkan kode di sini hanya untuk rotasi vertikal

Letakkan kode di sini untuk rotasi horizontal dan vertikal.

Gambar 3.16 Kerangka kerja MouseLook dengan enum untuk pengaturan Rotasi

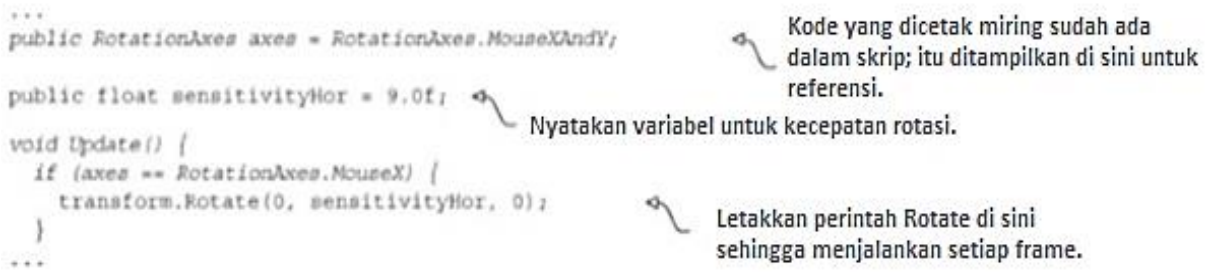
Perhatikan bahwa enum digunakan untuk memilih rotasi horizontal atau vertikal untuk skrip MouseLook. Mendefinisikan struktur data enum memungkinkan Anda untuk menetapkan nilai berdasarkan nama, daripada mengetik angka dan mencoba mengingat apa arti setiap angka (apakah 0 rotasi horizontal? Apakah 1?). Jika Anda kemudian mendeklarasikan variabel publik yang diketik ke enum itu, yang akan ditampilkan di Inspector sebagai menu tarik-turun yang berguna untuk memilih pengaturan.

Gambar 3.17 Inspector menampilkan variabel enum publik sebagai menu tarik-turun.

Hapus komponen Spin (dengan cara yang sama seperti Anda menghapus penumbuk kapsul sebelumnya) dan lampirkan skrip baru ini ke objek pemutar. Gunakan menu Sumbu untuk beralih di antara cabang kode saat mengerjakan kode. Dengan pengaturan rotasi horizontal/vertikal, Anda dapat mengisi kode untuk setiap cabang kondisional.

Rotasi horizontal yang melacak gerakan mouse

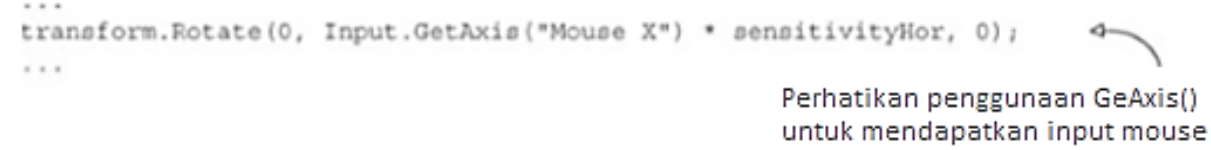
Cabang pertama dan paling sederhana adalah rotasi horizontal. Mulailah dengan menulis perintah rotasi yang sama yang Anda gunakan dalam daftar 2.1 untuk membuat objek berputar. Jangan lupa untuk mendeklarasikan variabel publik untuk kecepatan rotasi; mendeklarasikan variabel baru setelah sumbu tetapi sebelum Update(), dan memanggil variabel sensitivitasHor karena kecepatan terlalu umum nama setelah Anda memiliki beberapa rotasi yang terlibat. Tingkatkan nilai variabel menjadi 9 kali ini karena nilai itu harus lebih besar setelah kode mulai scalingnya (yang akan segera). Kode yang disesuaikan akan terlihat seperti daftar berikut.



Gambar 3.18 Rotasi horizontal, belum merespons mouse

Jika Anda memutar skrip sekarang, tampilan akan berputar seperti sebelumnya (hanya jauh lebih cepat, karena rotasi Y adalah 9 bukannya 3). Langkah selanjutnya adalah membuat rotasi bereaksi terhadap gerakan mouse, jadi mari kita perkenalkan metode baru: Input.GetAxis(). Kelas Input memiliki banyak metode untuk menangani perangkat input (seperti mouse) dan metode GetAxis() mengembalikan angka yang berkorelasi dengan pergerakan mouse (positif atau negatif, tergantung pada arah gerakan). GetAxis() mengambil nama sumbu yang diinginkan sebagai parameter, dan sumbu horizontal disebut Mouse X.

Jika Anda mengalikan kecepatan rotasi dengan nilai sumbu, rotasi akan merespons gerakan mouse. Kecepatan akan meningkat sesuai dengan gerakan mouse, menurun ke nol atau bahkan membalikkan arah. Perintah Rotate sekarang terlihat seperti daftar berikutnya.



Gambar 3.19 Putar perintah yang disesuaikan untuk merespons mouse

Tekan Mainkan lalu gerakkan mouse. Saat Anda menggerakkan mouse dari sisi ke sisi, tampilan akan berputar dari sisi ke sisi. Itu sangat keren! Langkah selanjutnya adalah memutar secara vertikal, bukan horizontal.

Rotasi vertikal dengan batas

Untuk rotasi horizontal kami telah menggunakan metode Rotate(), tetapi kami akan mengambil pendekatan yang berbeda dengan rotasi vertikal. Meskipun metode itu nyaman untuk menerapkan transformasi, itu juga agak tidak fleksibel. Ini hanya berguna untuk meningkatkan rotasi tanpa batas, yang baik untuk rotasi horizontal, tetapi rotasi vertikal membutuhkan batasan seberapa banyak tampilan dapat dimiringkan ke atas atau ke bawah. Daftar berikut menunjukkan kode rotasi vertikal untuk MouseLook; penjelasan rinci tentang kode akan datang setelahnya.

Gambar 3.20 Rotasi vertikal untuk MouseLook

Atur menu Axes dari komponen MouseLook ke rotasi vertikal dan mainkan skrip baru. Sekarang tampilan tidak akan berputar ke samping, tetapi akan miring ke atas dan ke bawah saat Anda menggerakkan mouse ke atas dan ke bawah. Kemiringan berhenti di batas atas dan bawah.

Ada beberapa konsep baru dalam kode ini yang perlu dijelaskan. Pertama, kita tidak menggunakan Rotate() kali ini, jadi kita memerlukan variabel (disebut _rotationX di sini, karena rotasi vertikal mengelilingi sumbu X) untuk menyimpan sudut rotasi. Metode Rotate() menambah rotasi saat ini, sedangkan kode ini menetapkan sudut rotasi secara langsung. Dengan kata lain, ini adalah perbedaan antara mengatakan "tambahkan 5 ke sudut" dan "atur sudut ke 30." Kita masih perlu menaikkan sudut rotasi, tapi itulah mengapa kode memiliki operator -=: untuk mengurangi nilai dari sudut rotasi, daripada mengatur sudut ke nilai itu. Dengan tidak menggunakan Rotate() kita dapat memanipulasi sudut rotasi dengan berbagai cara selain hanya menambahnya. Nilai rotasi dikalikan dengan Input.GetAxis() seperti pada kode untuk rotasi horizontal, kecuali sekarang kita meminta Mouse Y karena itu adalah sumbu vertikal mouse.

Sudut rotasi dimanipulasi lebih lanjut pada baris berikutnya. Kami menggunakan Mathf.Clamp() untuk menjaga sudut rotasi antara batas minimum dan maksimum. Batasan tersebut adalah variabel publik yang dideklarasikan sebelumnya dalam kode, dan mereka memastikan bahwa tampilan hanya dapat dimiringkan 45 derajat ke atas atau ke bawah. Metode Clamp() tidak khusus untuk rotasi, tetapi umumnya berguna untuk menjaga variabel angka di antara batas. Untuk melihat apa yang terjadi, coba komentari baris Clamp(); sekarang kemiringan tidak berhenti di batas atas dan bawah, memungkinkan Anda bahkan memutar sepenuhnya terbalik! Jelas, melihat dunia secara terbalik tidak diinginkan; oleh karena itu batas-batasnya. Karena properti sudut dari transformasi adalah Vektor3, kita perlu membuat Vektor3 baru dengan sudut rotasi yang diteruskan ke konstruktor. Metode Rotate() mengotomatiskan proses ini untuk kami, menambah sudut rotasi dan kemudian membuat vektor baru.

Definisi Vektor adalah beberapa angka yang disimpan bersama sebagai satu kesatuan. Misalnya, Vektor3 adalah 3 angka (berlabel x, y, z). *Peringatan* Alasan mengapa kita perlu membuat Vector3 baru alih-alih mengubah nilai dalam vektor yang ada dalam transformasi

adalah karena nilai tersebut hanya-baca untuk transformasi. Ini adalah kesalahan umum yang dapat membuat Anda tersandung.

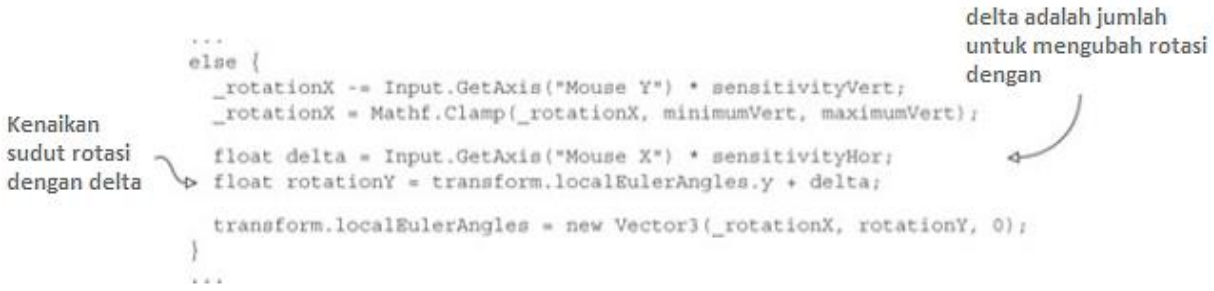
3.6 SUDUT EULER VS. QUATERNION

Anda mungkin bertanya-tanya mengapa properti ini disebut localEulerAngles dan bukan localRotation. Pertama, Anda perlu tahu tentang konsep yang disebut quaternions. Quaternions adalah konstruksi matematika yang berbeda untuk mewakili rotasi. Mereka berbeda dari sudut Euler, yang merupakan nama untuk pendekatan sumbu X-, Y-, Z yang telah kita ambil. Ingat seluruh diskusi pitch, yaw, dan roll? Nah, cara merepresentasikan rotasi itu adalah sudut Euler. Quaternions adalah ... berbeda. Sulit untuk menjelaskan apa itu quaternions, karena mereka adalah aspek yang tidak jelas dari matematika tingkat tinggi, yang melibatkan gerakan melalui empat dimensi. Jika Anda ingin penjelasan rinci, coba baca dokumen yang ada di sini:
www.flipcode.com/documents/matrfaq.html#Q47
Sedikit lebih mudah untuk menjelaskan mengapa angka empat digunakan untuk mewakili rotasi: interpolasi antara nilai rotasi (yaitu, melalui sekelompok nilai peralihan untuk secara bertahap berubah dari satu nilai ke nilai lainnya) terlihat lebih halus dan lebih alami saat menggunakan angka empat.

Kembali ke pertanyaan awal, karena localRotation adalah quaternion, bukan sudut Euler. Unity juga menyediakan properti sudut Euler untuk membuat manipulasi rotasi lebih mudah dipahami; properti sudut Euler dikonversi ke dan dari nilai quaternion secara otomatis. Unity menangani matematika yang lebih sulit untuk Anda di belakang layar, jadi Anda tidak perlu khawatir untuk menanganinya sendiri. Ada satu lagi pengaturan rotasi untuk MouseLook yang membutuhkan kode: rotasi horizontal dan vertikal secara bersamaan.

Rotasi horizontal dan vertikal secara bersamaan

Potongan kode terakhir ini juga tidak akan menggunakan Rotate(), karena alasan yang sama: sudut rotasi vertikal dijepit di antara batas setelah dinaikkan. Itu berarti rotasi horizontal perlu dihitung langsung sekarang. Ingat, Rotate() mengotomatiskan proses penambahan sudut rotasi.



Gambar 3.21 Tampilan Mouse Horizontal dan Vertikal

Beberapa baris pertama, yang berhubungan dengan _rotationX, sama persis seperti di bagian terakhir. Ingatlah bahwa berputar di sekitar sumbu X objek adalah rotasi vertikal. Karena rotasi horizontal tidak lagi ditangani menggunakan metode Rotate(), itulah yang dilakukan oleh garis delta dan rotasiY. Delta adalah istilah matematika umum untuk "jumlah perubahan", jadi perhitungan delta kami adalah jumlah rotasi yang harus diubah. Jumlah perubahan itu kemudian ditambahkan ke sudut rotasi saat ini untuk mendapatkan sudut rotasi baru yang diinginkan. Terakhir, kedua sudut, vertikal dan horizontal, digunakan untuk membuat vektor baru yang ditetapkan ke properti sudut komponen transformasi.

Larang rotasi fisika pada pemutar

Meskipun ini belum menjadi masalah untuk proyek ini, di sebagian besar game FPS modern ada simulasi fisika kompleks yang memengaruhi semua yang ada di scene. Ini akan menyebabkan benda terpental dan berjatuhan; perilaku ini terlihat dan berfungsi dengan baik untuk sebagian besar objek, tetapi rotasi player hanya perlu dikontrol oleh mouse dan tidak terpengaruh oleh simulasi fisika. Untuk alasan itu, skrip input mouse biasanya mengatur properti freezeRotation pada Rigidbody player. Tambahkan metode Start() ini ke skrip MouseLook:

```
...
void Start() {
    Rigidbody body = GetComponent<Rigidbody>();
    if (body != null)
        body.freezeRotation = true;
}
...
```

Periksa apakah komponen ini ada

Gambar 3.22 Skrip MouseLook yang sudah selesai

Jika Anda tersesat di mana harus membuat berbagai perubahan dan penambahan yang telah kita bahas, daftar berikutnya memiliki skrip lengkap yang telah selesai. Atau, unduh proyek contoh.

```
using UnityEngine;
using System.Collections;
public class MouseLook : MonoBehaviour { public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 } public RotationAxes axes = RotationAxes.MouseXAndY;
public float sensitivityHor = 9.0f; publicVert float sensitivity = 9.0f;
public float minimumVert = -45.0f; float maximumVert public = 45.0f;
private float _rotationX = 0;
void Start() { Rigid body = GetComponent<Rigidbody>(); if (body != null) body.freezeRotation = true; }
void Update() { if (axis == RotationAxes.MouseX) { transform.Rotate(0, Input.GetAxis("Mouse X") sensitivityHor, 0); } else if (axis == RotationAxes.MouseY) { _rotationX -= Input.GetAxis("Mouse Y") sensitivityVert; rotationX = Mathf.Clamp(rotationX, minimumVert, maximumVert);
float rotasiY = transform.localEulerAngles.y;
transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0); }
else {
        _rotationX -= Input.GetAxis("Mouse Y") sensitivitasVert;
        rotasiX = Mathf.Clamp(rotationX, minimumVert, maximumVert);
        float delta = Input.GetAxis("Mouse X") sensitivitasHor;
        float rotasiY = transform.localEulerAngles.y + delta;
        transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
                }
        }
}
```

Saat Anda menjalankan skrip baru, Anda dapat melihat ke segala arah sambil menggerakkan mouse. Besar! Tapi Anda masih terjebak di satu tempat, melihat sekeliling seolah-olah dipasang di menara. Langkah selanjutnya adalah bergerak di sekitar scene.

Komponen input keyboard: kontrol orang pertama

Melihat sekeliling sebagai respons terhadap input mouse adalah bagian penting dari kontrol orang pertama, tetapi Anda baru setengah jalan. Player juga perlu bergerak sebagai respons terhadap input keyboard. Mari kita menulis komponen kontrol keyboard untuk melengkapi komponen kontrol mouse; buat skrip C# baru bernama FPSInput dan lampirkan itu ke pemutar (bersama skrip MouseLook). Untuk saat ini setel komponen MouseLook ke rotasi horizontal saja. Kontrol keyboard dan mouse yang dijelaskan di sini dibagi menjadi skrip terpisah. Anda tidak perlu menyusun kode dengan cara ini, dan Anda dapat menggabungkan semuanya menjadi satu skrip "kontrol pemutar", tetapi sistem komponen (seperti yang ada di Unity) cenderung paling fleksibel dan karenanya paling berguna saat Anda memiliki fungsionalitas yang dibagi menjadi beberapa komponen yang lebih kecil.

Kode yang Anda tulis di bagian sebelumnya hanya memengaruhi rotasi, tetapi sekarang kita akan mengubah posisi objek. Seperti yang ditunjukkan pada daftar 2.8, lihat kembali kode rotasi sebelum kita menambahkan input mouse; ketik itu ke dalam FPSInput, tetapi ubah Rotate() menjadi Translate(). Saat Anda menekan Putar, tampilan meluncur ke atas alih-alih berputar. Coba ubah nilai parameter untuk melihat bagaimana gerakan berubah (khususnya, coba tukar angka pertama dan kedua); setelah bereksperimen dengan itu sebentar, Anda dapat melanjutkan untuk menambahkan input keyboard.

```
using UnityEngine;
using System.Collections;

public class FPSInput : MonoBehaviour {
  public float speed = 6.0f;

  void Update() {
    transform.Translate(0, speed, 0);
  }
}
```

Tidak diperlukan, tetapi Anda mungkin ingin meningkatkan kecepatan

Mengubah Rotate() menjadi Translate()

Gambar 3.23 Putar kode dari daftar pertama, dengan beberapa perubahan kecil

Menanggapi penekanan tombol

Kode untuk bergerak menurut penekanan tombol (ditunjukkan dalam daftar berikut) mirip dengan kode untuk berputar menurut mouse. Metode GetAxis() juga digunakan di sini, dan dengan cara yang sangat mirip. Daftar berikut menunjukkan cara menggunakan perintah itu.

```
...
void Update() {
  float deltaX = Input.GetAxis("Horizontal") * speed;
  float deltaZ = Input.GetAxis("Vertical") * speed;
  transform.Translate(deltaX, 0, deltaZ);
}
...
```

"Horizontal" dan "Vertikal" adalah nama tidak langsung untuk pemetaan keyboard

Gambar 3.24 Gerakan posisi menanggapi penekanan tombol

Seperti sebelumnya, nilai GetAxis() dikalikan dengan kecepatan untuk menentukan jumlah gerakan. Padahal sebelumnya sumbu yang diminta selalu "Mouse sesuatu," sekarang kita lewati baik Horizontal atau Vertikal. Nama-nama ini adalah abstraksi untuk pengaturan input di Unity; jika Anda melihat di menu Edit di bawah Pengaturan Proyek dan kemudian melihat di bawah Input, Anda akan menemukan daftar nama input abstrak dan kontrol yang

tepat dipetakan ke nama-nama itu. Tombol panah kiri/kanan dan huruf A/D dipetakan ke Horizontal, sedangkan tombol panah atas/bawah dan huruf W/S dipetakan ke Vertikal.

Perhatikan bahwa nilai pergerakan diterapkan pada koordinat X dan Z. Seperti yang mungkin Anda perhatikan saat bereksperimen dengan metode Translate(), koordinat X bergerak dari sisi ke sisi dan koordinat Z bergerak maju dan mundur.

Masukkan kode gerakan baru ini dan Anda akan dapat bergerak dengan menekan tombol panah atau tombol huruf WASD, standar di sebagian besar game FPS. Skrip gerakan hampir selesai, tetapi kami memiliki beberapa penyesuaian lagi untuk dilakukan.

3.7 MENYETEL LAJU GERAKAN YANG TIDAK BERGANTUNG PADA KECEPATAN KOMPUTER

Itu tidak jelas sekarang karena Anda hanya menjalankan kode di satu komputer (milik Anda), tetapi jika Anda menjalankannya di mesin yang berbeda, itu akan berjalan pada kecepatan yang berbeda. Itu karena beberapa komputer dapat memproses kode dan grafik lebih cepat daripada yang lain. Saat ini player akan bergerak dengan kecepatan yang berbeda di komputer yang berbeda karena kode gerakan terkait dengan kecepatan komputer. Itu disebut sebagai frame rate dependen, karena kode gerakan bergantung pada frame rate game. Misalnya, bayangkan Anda menjalankan demo ini di dua komputer yang berbeda, yang satu mendapat 30 fps (frame per detik) dan satu yang mendapat 60 fps. Itu berarti Update() akan dipanggil dua kali lebih sering pada komputer kedua, dan nilai kecepatan 6 yang sama akan diterapkan setiap waktu. Pada 30 fps, kecepatan gerakan akan menjadi 180 unit/detik, dan gerakan pada 60 fps akan menjadi 360 unit/detik. Untuk sebagian besar game, kecepatan gerakan yang bervariasi seperti ini akan menjadi berita buruk.

Solusinya adalah menyesuaikan kode gerakan agar frame rate independen. Itu berarti kecepatan gerakan tidak tergantung pada frame rate game. Cara untuk mencapainya adalah dengan tidak menerapkan nilai kecepatan yang sama pada setiap kecepatan bingkai. Sebagai gantinya, scale nilai kecepatan lebih tinggi atau lebih rendah tergantung pada seberapa cepat komputer berjalan. Ini dicapai dengan mengalikan nilai kecepatan dengan nilai lain yang disebut deltaTime, seperti yang ditunjukkan pada daftar berikutnya.

```
...
        void Update() {
        float deltaX = Input.GetAxis("Horizontal")  speed;
        float deltaZ = Input.GetAxis("Vertical")  speed;
        transform.Translate(deltaX Time.deltaTime, 0, deltaZ
Time.deltaTime);
}
...
```

Itu adalah perubahan yang sederhana. Kelas Time memiliki sejumlah properti dan metode yang berguna untuk pengaturan waktu, dan salah satu properti tersebut adalah deltaTime. Karena kita tahu bahwa delta berarti jumlah perubahan, itu berarti deltaTime adalah jumlah perubahan waktu. Secara khusus, deltaTime adalah jumlah waktu antara frame. Waktu antar bingkai bervariasi pada kecepatan bingkai yang berbeda (misalnya, 30 fps adalah deltaTime 1/30 detik), jadi mengalikan nilai kecepatan dengan deltaTime akan scaling nilai kecepatan pada komputer yang berbeda. Sekarang kecepatan gerakan akan sama di semua komputer. Tapi script gerakannya masih belum selesai; ketika Anda bergerak di sekitar ruangan Anda dapat melewati dinding, jadi kami perlu menyesuaikan kode lebih lanjut untuk mencegahnya.

3.8 MEMINDAHKAN CHARACTERCONTROLLER UNTUK DETEKSI COLLISION

Mengubah transformasi objek secara langsung tidak menerapkan deteksi collision, sehingga karakter akan menembus dinding. Untuk menerapkan deteksi collision, yang ingin kita lakukan adalah menggunakan CharacterController. CharacterController adalah komponen yang membuat objek bergerak lebih seperti karakter dalam game, termasuk menabrak dinding. Ingat itu kembali ketika kami mengatur pemutar, kami memasang CharacterController, jadi sekarang kami akan menggunakan komponen itu dengan kode gerakan di FPSInput (lihat daftar berikut).

Gambar 3.25 Memindahkan CharacterController alih-alih Transform

Kutipan kode ini memperkenalkan beberapa konsep baru. Konsep pertama yang ditunjukkan adalah variabel untuk mereferensikan CharacterController. Variabel ini hanya membuat referensi lokal ke objek (objek kode, yaitu—jangan dikelirukan dengan objek scene); beberapa skrip dapat memiliki referensi ke instance CharacterController yang satu ini.

Variabel itu mulai kosong, jadi sebelum Anda dapat menggunakan referensi, Anda perlu menetapkan objek untuk dirujuk. Di sinilah GetComponent() berperan; metode itu mengembalikan komponen lain yang dilampirkan ke GameObject yang sama. Daripada melewatkan parameter di dalam tanda kurung, Anda menggunakan sintaks C# untuk mendefinisikan tipe di dalam kurung sudut, <>.

Setelah Anda memiliki referensi ke CharacterController, Anda dapat memanggil Move() pada controller. Berikan vektor ke metode itu, mirip dengan cara kode rotasi mouse menggunakan vektor untuk nilai rotasi. Juga mirip dengan bagaimana nilai rotasi dibatasi, gunakan Vector3.ClampMagnitude() untuk membatasi besarnya vektor ke kecepatan gerakan; penjepit digunakan karena jika tidak, gerakan diagonal akan memiliki magnitudo yang lebih besar daripada gerakan langsung sepanjang sumbu (gambarkan sisi dan sisi miring segitiga siku-siku).

Tapi ada satu aspek rumit dari vektor gerakan di sini, dan itu ada hubungannya dengan lokal versus global, seperti yang telah kita bahas sebelumnya untuk rotasi. Kami akan membuat vektor dengan nilai untuk dipindahkan, katakanlah, ke kiri. Itu kiri player, meskipun, yang mungkin menjadi arah yang sama sekali berbeda dari kiri dunia. Artinya, kita berbicara tentang yang tersisa di ruang lokal, bukan ruang global. Kita perlu meneruskan vektor gerakan yang didefinisikan dalam ruang global ke metode Move(), jadi kita perlu mengubah vektor ruang lokal menjadi ruang global. Melakukan konversi itu adalah matematika yang sangat

rumit, tetapi untungnya bagi kami Unity menangani matematika itu untuk kami, dan kami hanya perlu memanggil metode TransformDirection() untuk mengubah arah.

Definisi Transform digunakan sebagai kata kerja berarti mengubah dari satu ruang koordinat ke ruang koordinat lainnya (lihat kembali bagian 2.3.3 jika Anda tidak ingat apa itu ruang koordinat). Jangan bingung dengan definisi lain dari transformasi, termasuk komponen Transform dan tindakan memindahkan objek di sekitar scene. Ini semacam istilah yang berlebihan, karena semua makna ini merujuk pada konsep dasar yang sama.

Uji memainkan kode gerakan sekarang. Jika Anda belum melakukannya, atur komponen MouseLook ke rotasi horizontal dan vertikal. Anda dapat melihat sekeliling scene sepenuhnya dan terbang di sekitar scene menggunakan kontrol keyboard. Ini cukup bagus jika Anda ingin player terbang di sekitar scene, tetapi bagaimana jika Anda ingin player berjalan-jalan di tanah?

Menyesuaikan komponen untuk berjalan alih-alih terbang

Sekarang deteksi collision berfungsi, skrip dapat memiliki gravitasi dan player akan tetap menempel di lantai. Deklarasikan variabel gravitasi dan kemudian gunakan nilai gravitasi tersebut untuk sumbu Y, seperti yang ditunjukkan pada daftar berikutnya.

```
...
public float gravity = -9.8f;
...
void Update() {
  ...
  movement = Vector3.ClampMagnitude(movement, speed);
  movement.y = gravity;
  ...
```

Gunakan nilai gravitasi, bukan hanya 0

Gambar 3.26 Menambahkan gravitasi ke kode gerakan

Sekarang ada gaya ke bawah yang konstan pada player, tetapi tidak selalu mengarah lurus ke bawah, karena objek player dapat dimiringkan ke atas dan ke bawah dengan mouse. Untungnya semua yang perlu kita perbaiki sudah ada di tempatnya, jadi kita hanya perlu melakukan beberapa penyesuaian kecil tentang cara mengatur komponen pada pemutar. Pertama-tama atur komponen MouseLook pada objek pemutar ke rotasi horizontal saja. Selanjutnya tambahkan komponen MouseLook ke objek kamera, dan setel itu ke rotasi vertikal saja. Betul sekali; Anda akan memiliki dua objek berbeda yang merespons mouse!

Karena objek player sekarang hanya berputar secara horizontal, tidak ada lagi masalah dengan gaya gravitasi ke bawah yang dimiringkan. Objek kamera di-parent ke objek pemutar (ingat ketika kita melakukannya dalam tampilan Hierarki?), jadi meskipun ia berputar secara vertikal terlepas dari pemutar, kamera berputar secara horizontal bersama pemutar.

3.9 MEMOLES SKRIP YANG SUDAH JADI

Gunakan metode RequireComponent() untuk memastikan bahwa komponen lain yang diperlukan oleh skrip juga dilampirkan. Terkadang komponen lain bersifat opsional (yaitu, kode yang mengatakan "Jika komponen lain ini juga dilampirkan, maka ..."), tetapi terkadang Anda ingin membuat komponen lain menjadi wajib. Tambahkan metode ke bagian atas skrip untuk menerapkan ketergantungan itu dan berikan komponen yang diperlukan sebagai parameter.

Demikian pula, jika Anda menambahkan metode AddComponentMenu() ke atas skrip Anda, skrip itu akan ditambahkan ke menu komponen di editor Unity. Beri tahu perintah nama item menu yang ingin Anda tambahkan, lalu skrip dapat dipilih saat Anda mengklik Tambahkan Komponen di bagian bawah Inspector. Berguna!

Skrip dengan kedua metode ditambahkan ke atas akan terlihat seperti ini:

```
using UnityEngine; using System.Collections;
[RequireComponent(typeof(CharacterController))]
[AddComponentMenu("Control  Script/FPS Input")]
public class FPSInput : MonoBehaviour {
...
```

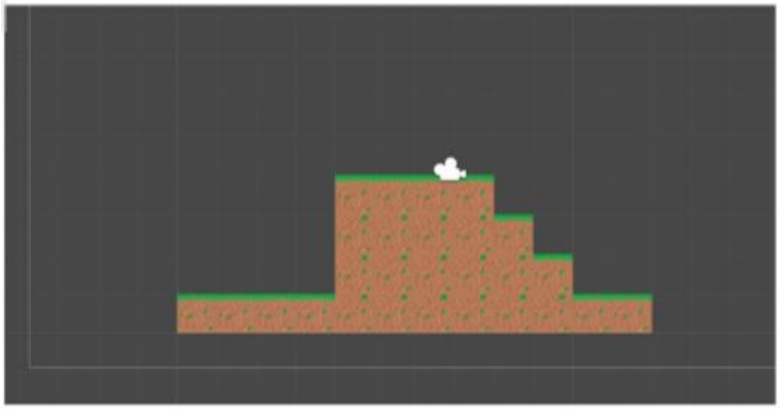
gerakan terbang dengan mengatur Gravity ke 0 di Inspector.

Daftar Skrip Input FPS yang sudah selesai

```
using UnityEngine; using System.Collections;
[RequireComponent(typeof(CharacterController))] [AddComponentMenu("Control Script/FPS Input")] public class FPSInput : MonoBehaviour { public float speed = 6.0f; public float gravity = -9.8f;
private CharacterController charController;
void Start() { charController = GetComponent<CharacterController>();
}
        void Update() { float deltaX = Input.GetAxis("Horizontal") speed; float deltaZ = Input.GetAxis("Vertical") speed; Vector3 movement = new Vector3(deltaX, 0, deltaZ);
        movement = Vector3.ClampMagnitude(movement, speed);
        movement.y = gravity;
        movement *= Time.deltaTime; movement = transform.TransformDirection(movement); _charController.Move(movement);
}
}
```

Selamat membangun proyek 3D ini! Kami membahas banyak hal dalam bab ini, dan sekarang Anda sudah berpengalaman tentang cara membuat kode gerakan di Unity. Semenarik demo pertama ini, ini masih jauh dari menjadi game yang lengkap. Lagi pula, rencana proyek menggambarkan ini sebagai scene FPS dasar, dan apa itu penembak jika Anda tidak bisa menembak? Jadi beri diri Anda tepukan yang layak untuk proyek bab ini, dan kemudian bersiaplah untuk langkah berikutnya.

Jadi, kami memiliki sprite ubin yang memiliki pola berumput, dan untuk sedikit mencampuradukkannya, saya juga membuat sprite untuk mewakili tanah di bawah rumput, sehingga saya bisa sedikit lebih kreatif dengan desain level saya.



Gambar 3.27 Beberapa ubin baru saja siap untuk dinaiki!

Saya membuat sprite kotoran kedua dengan cara yang sama persis seperti yang pertama dan kemudian menyalinnya di mana pun saya membutuhkan tanah di bawah permukaan rumput saya. Anda dapat melakukan hal yang sama—ingatlah untuk menahan Ctrl

untuk menjaga jarak tetap sempurna menggunakan fitur snap to grid. Sekarang kita akan melakukan sesuatu yang lebih menarik: buat karakter game dan jatuhkan dia ke dunia kita. Untuk memulainya, kita akan membuat sprite karakter paling sederhana yang kita bisa, yang akan menjadi kotak lain. Untuk membuat segalanya sedikit lebih menarik, kami juga akan memberinya mata. Temui Squarey.

Gambar 3.27 Squarey, sobat, kamu tidak terlihat begitu baik

Persegi dapat menjadi ukuran apa pun yang Anda suka, tetapi untuk kesederhanaan saya sarankan menjaga dimensi yang sama seperti yang Anda gunakan untuk ubin tanah. Sekarang tambahkan Squarey ke folder Sprite Anda seperti yang Anda lakukan dengan yang lain. Pastikan Anda ingat untuk mengatur Pixels Per Unit ke 50 lagi dan kemudian drag dia ke scene Anda. Beri nama Game Object Player baru ini dan pisahkan dari folder Squares. Tidak masalah di level mana Anda ingin menyeretnya, tetapi saya memilih untuk menempatkannya di puncak bukit yang saya buat.

Gambar 3.28 Squarey mensurvei domainnya

Selamat! Anda sekarang memiliki karakter utama. Masalahnya, dia sebenarnya belum melakukan apa-apa. Jadi, hal selanjutnya yang ingin kita lakukan adalah menerapkan beberapa fisika sehingga gravitasi dan hal-hal lain akan mempengaruhi karakter kita. Dan Anda tidak akan percaya betapa mudahnya Unity membuat ini sedikit.

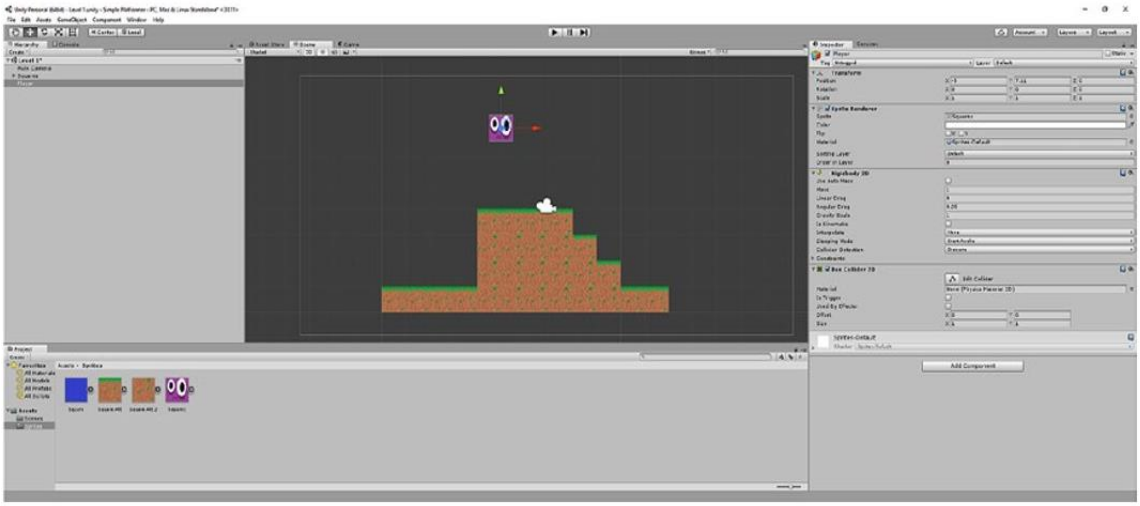
Menggunakan RigidBody 2D

Seperti yang telah disebutkan, inti dari penggunaan mesin fisika seperti Unity adalah agar kita dapat mengakses skrip dan elemen yang sudah jadi untuk menghindari pengkodean semuanya sendiri dari awal. Ini membuatnya sangat sederhana untuk menambahkan sesuatu seperti gravitasi: yang harus kita lakukan hanyalah menyeret skrip ke GameObject kita agar bisa diterapkan. Saat pemutar Anda dipilih, klik Tambahkan Komponen di Inspector. Sekarang klik Fisika 2D RigidBody 2D. RigidBody 2D adalah nama skrip yang bekerja pada sprite 2D dan menerapkan semua fisika dasar yang kita inginkan, seperti gravitasi, gesekan, momentum, tumbukan, dan banyak lagi. Namun, cara termudah untuk melihat apa yang dilakukannya adalah dengan menerapkannya. Jadi drag Squarey sedikit lebih tinggi di jendela Scene lalu klik

play. Jika semuanya berjalan sesuai rencana, kita sekarang memiliki Squarey yang tampak sedih yang jatuh begitu saja dari atas layar dan dari bawah. Squarey, temui gravitasi!

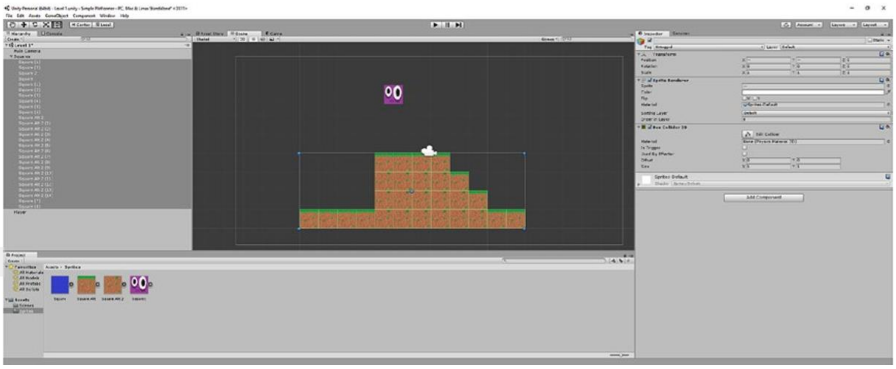
Menggunakan Collider

Yang cerdik akan mencatat bahwa ada sesuatu yang hilang di sini. Tentu saja, kami sebenarnya tidak ingin Squarey jatuh ke tanah di bawahnya; kami ingin dia mendarat di atasnya. Untungnya, ini juga merupakan perbaikan yang mudah. Cukup klik Squarey lagi dan kemudian Tambahkan Komponen Fisika 2D Box Collider. Anda sekarang seharusnya dapat melihat RigidBody 2D dan Box Collider 2D di Inspector.



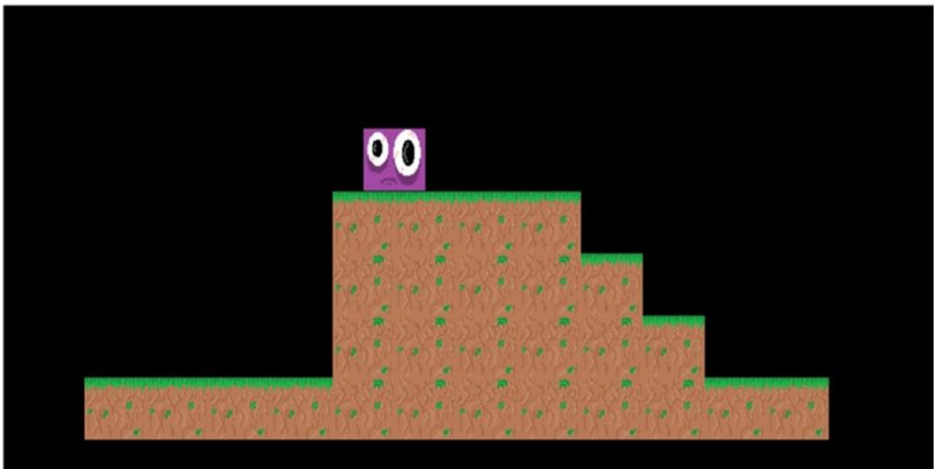
Gambar 3.29 Garis hijau menunjukkan Collider Squarey

Terlebih lagi, Anda juga harus memperhatikan garis hijau tipis di sekitar Squarey. Ini adalah collider, yang pada dasarnya mendefinisikan batas sprite Anda dan memberi tahu RigidBody 2D di mana massa fisik yang membentuk karakter dimulai dan berhenti. Klik mainkan dan Anda akan melihat bahwa karakternya masih jatuh di tanah di bawah. Mudah-mudahan, Anda sudah menebak bahwa ini karena ubin kami juga harus memiliki penabrakan yang terpasang. Untuk melakukan ini, drag kotak di sekitar ubin di tampilan Scene Anda menggunakan mouse untuk memilih semuanya sekaligus. Anda dapat melakukannya dengan cara yang sama seperti memilih sekumpulan ikon di desktop (Gambar 4-6), atau dengan mengeklik item pertama dalam hierarki, lalu item terakhir sambil menahan Shift. Anda juga dapat menggunakan Ctrl untuk membuat pilihan massal, seperti yang Anda lakukan di sebagian besar program Windows. Ini adalah trik yang akan sering berguna, meskipun ada metode lain untuk membuat perubahan massal yang akan segera kita bahas. Pastikan untuk membatalkan pilihan kamera dan kemudian menambahkan penumbuk di Inspector. Ini sekarang akan diterapkan ke setiap GameObject yang dipilih sekaligus.



Gambar 3.30 Nanti kita akan membahas menggunakan prefab untuk menghindari harus memilih beberapa GameObjects

Perhatikan bahwa kami tidak menambahkan RigidBody 2D ke ubin tanah kami. Itu karena kami tidak ingin mereka jatuh dari layar. Sekarang klik play, dan jika semuanya berjalan sesuai rencana, karakter kita akan jatuh dan mendarat di tanah.

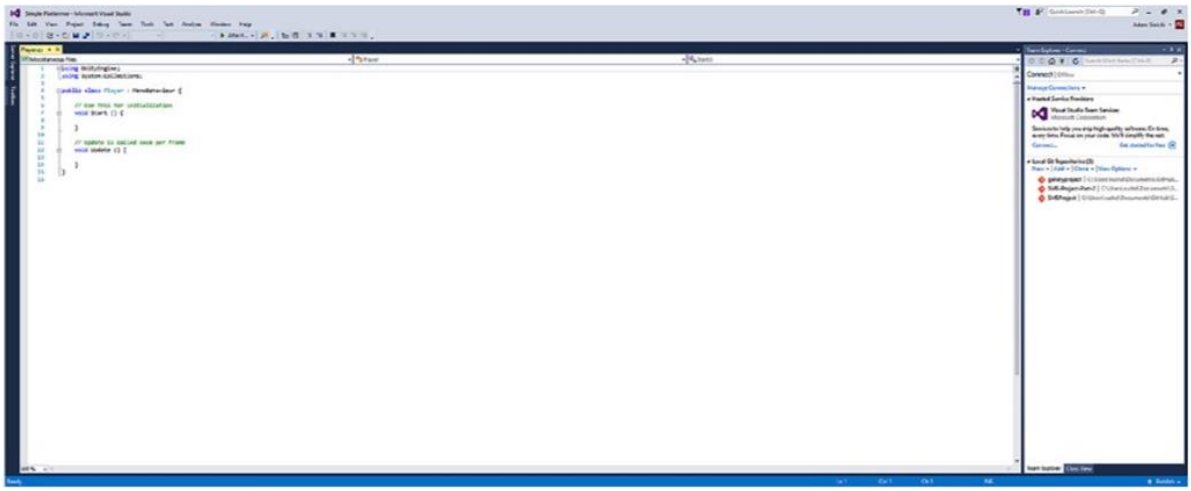


Gambar 3.31 Berhasil

Jika Anda ingin menguji seberapa detail RigidBody 2D, posisikan Squarey sehingga dia sebagian menggantung di tepi salah satu langkah, lalu klik rotate lagi. Ketika dia mendarat, dia harus tip dan benar-benar berguling menuruni tangga. Pada titik ini, kami sekarang sudah memiliki permainan yang bertindak seperti yang kami harapkan. Langkah selanjutnya mungkin harus membuatnya interaktif. Bersiaplah: di sinilah pengkodean yang sebenarnya masuk.

3.10 MEMULAI CODING DALAM C#

Sebelum Anda memulai coding, pertama-tama kita perlu membuat folder untuk memuat semua skrip kita di dalam direktori Assets. Klik kanan pada tampilan Proyek dan pilih Buat Folder, seperti yang Anda lakukan saat membuat folder Sprite dan Scene. Panggil folder baru ini Scripts dan kemudian buka. Di sini, Anda kembali klik kanan dan kali ini pilih Create C# Script. Panggil Player ini dan kemudian klik dua kali untuk membukanya. Anda sekarang akan membuka Visual Studio untuk pertama kalinya. Tetapi pertama-tama, Anda harus masuk. Untuk melakukannya, Anda cukup menggunakan akun Microsoft Anda (yang mungkin Anda gunakan untuk masuk ke Windows dan Hotmail). Jika Anda tidak memilikinya, Anda akan diberi kesempatan untuk membuatnya. Setelah Anda masuk



Gambar 3.32 Halaman Awal C#

Untuk saat ini, kami berkonsentrasi pada jendela besar di tengah, di mana kami dapat memasukkan dan mengedit kode. Kami memilih untuk membuat kode dalam C# di sini karena agak mirip dengan Java—bahasa yang digunakan di sebagian besar developeran Android—hanya sedikit lebih sederhana. Satu hal yang perlu diingat ketika coding dalam C# adalah bahwa setiap baris harus diakhiri dengan titik koma atau kurung kurawal terbuka atau tertutup. Jika Anda melewatkan ini, Anda akan mendapatkan kesalahan. Anda mungkin akan menemukan bahwa sangat mudah untuk melewatkan detail penting ini dan kemudian menghabiskan waktu lama untuk merutekan kode Anda mencoba mencari tahu mengapa itu tidak berjalan. Apa yang mungkin juga Anda perhatikan di sini adalah bahwa dokumen tersebut sebenarnya tidak kosong tetapi sudah memiliki beberapa baris kode di dalamnya. Ini adalah fungsi, yang merupakan kumpulan kode mandiri yang dipanggil pada waktu tertentu. Dua fungsi yang kita miliki di sini akan hadir di setiap skrip yang kita buat dan membantu memberikan sedikit struktur untuk apa yang akan kita lakukan. Semuanya akan terlihat seperti berikut:

```
using UnityEngine; using System.Collections;
public class Player : MonoBehaviour {
    // Use this for initialization void Start() { }
 // Update is called once per frame void Update() {
}
}
```

Jangan khawatir tentang dua baris teratas dulu. Baris ketiga yang membaca kelas publik sebenarnya hanya menamai skrip yang telah kita buat, dan dua bagian berikut (void Start dan void Update) adalah fungsi kita. Dua hal lain yang perlu diperhatikan adalah garis miring ke depan. Setiap kali sebuah baris dimulai dengan dua garis miring di C#, itu berarti itu adalah komentar dan tidak akan berpengaruh pada cara kode berjalan. Dengan kata lain, ini adalah bagaimana Anda dapat menulis pesan untuk diri sendiri jika Anda lupa apa yang dilakukan oleh satu baris kode. Ketika tim pemrogram bekerja sama, berkomentar seperti ini sangat penting untuk memastikan bahwa setiap anggota mengetahui apa yang dilakukan semua orang.

apa yang dilakukan semuanya. Dua komentar yang sudah ada di sini menjelaskan apa fungsinya. Yang pertama mengatakan "Gunakan ini untuk inisialisasi", jadi void Start akan berjalan setiap kali skrip pertama kali diperkenalkan ke TKP. Komentar kedua mengatakan "Pembaruan disebut sekali per bingkai", jadi batal Pembaruan berjalan berulang kali dengan kecepatan yang sangat cepat saat permainan disegarkan. Untuk mendemonstrasikan ini, mari kita coba dan buat karakter kita bergerak tepat di seberang layar. Untuk melakukan itu, pertama-tama kita perlu merujuk ke beberapa elemen penting yang akan kita kerjakan dalam kode. Dalam hal ini, kita perlu bekerja dengan skrip RigidBody 2D yang dilampirkan ke Player GameObject kita. Dan untuk melakukan itu, kita perlu menambahkan kode berikut:

```
public class Controls : MonoBehaviour {
public Rigidbody2D rb;
        void Start() {
                rb = GetComponent<Rigidbody2D>();
}
```

Apa yang terjadi di sini adalah kita membuat referensi ke RigidBody 2D dan menyebutnya rb (kependekan dari RigidBody). Kemudian, ketika skrip diinisialisasi, kami memberi tahu bahwa instance RigidBody 2D untuk digunakan adalah GameObject tempat skrip kami dilampirkan

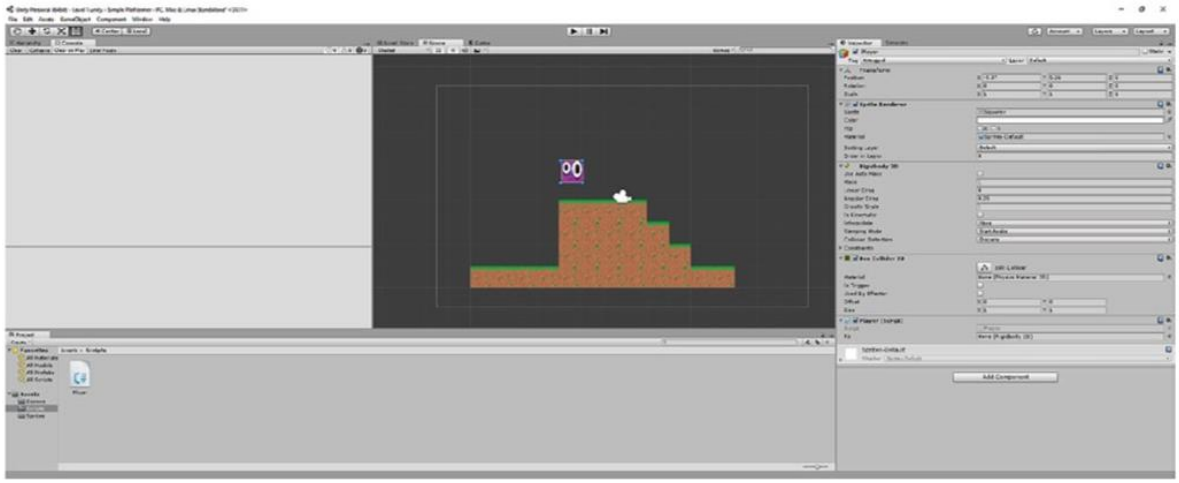
(dalam beberapa saat, kami akan melampirkan ini ke Player GameObject). Jangan khawatir jika ini semua sedikit membingungkan: Anda cukup menyalin dan menempelkan kode untuk saat ini dan itu akan lebih masuk akal nanti. Terakhir, kita akan menambahkan baris kode berikut ke fungsi Update kita:

```
void Update() {
rb.velocity = new Vector2(1, rb.velocity.y);
}
```

Ini hanya menambahkan kecepatan dengan nilai 1 ke koordinat X horizontal dari RigidBody (Vektor adalah koordinat). Karena ini ada di Pembaruan, itu berarti itu harus berjalan setiap kali scene disegarkan — yang terjadi sangat cepat. Semuanya akan terlihat seperti berikut:

```
using UnityEngine;
using System.Collections;
public class Player : MonoBehaviour {
public Rigidbody2D rb; // Use this for initialization void Start() { rb = GetComponent<Rigidbody2D>(); }
    // Update is called once per frame void Update() {
rb.velocity = new Vector2(1, rb.velocity.y);
}
}
```

Pastikan Anda ingat untuk menyimpan pekerjaan Anda! Sekarang yang tersisa untuk dilakukan adalah kembali ke Unity dan menambahkan skrip Player ke karakter Player. Kami melakukan ini sama seperti kami menambahkan RigidBody 2D: pilih Player GameObject, klik Add Component, lalu Scripts - Player. Sekarang klik mainkan dan Anda akan menemukan bahwa Squarey terus bergerak ke kanan dan kemudian menuruni tangga menuju azab pamungkasnya.



Gambar 3.33 Squarey sekarang memiliki Lemming AI

Hanya setelah Anda melampirkan skrip ke GameObject Anda, itu akan memiliki efek apa pun, dan Anda dapat dengan mudah menambahkan ini ke ubin tanah untuk efek yang sama.

Memperkenalkan Variabel Sekarang saatnya kita bermain dengan beberapa variabel. Variabel adalah konsep yang sangat penting dalam pengkodean dan di mana banyak logika dan keserbagunaan masuk. Pada dasarnya, variabel adalah singkatan untuk sepotong data yang dapat digunakan untuk mewakili data tersebut (seperti kesehatan atau nama player) di masa depan. Kemungkinannya adalah, jika Anda dapat mengingat kembali matematika di

sekolah, Anda akan menemukan bahwa Anda telah menemukan variabel di masa lalu. Ingat teka-teki seperti ini?

10 + x = 13, cari x

Nah, dalam kasus ini, x adalah variabel yang mewakili 3. Tetapi jika kita menulisnya seperti ini

10 + x = ?

yang memungkinkan kita untuk mengubah hasilnya, cukup dengan mengubah nilai x menggunakan kunci atau yang serupa. Kita dapat melakukan hal yang sama persis ketika bekerja dengan variabel dalam C#. Misalnya, kita dapat mengubah kecepatan gerakan karakter kita dengan mengganti 1 dengan variabel yang disebut kecepatan gerak. Tapi pertama-tama, kita perlu mendefinisikan apa yang dimaksud kecepatan bergerak dengan menginisialisasinya. Jadi sekarang kode kita terlihat seperti ini:

```
using UnityEngine;
using System.Collections;
public class Player : MonoBehaviour { public Rigidbody2D rb;
public int movespeed;
   // Use this for initialization void Start() {
   rb = GetComponent<Rigidbody2D>(); movespeed = 2;
   }
     // Update is called once per frame void Update() { rb.velocity = new Vector2(movespeed,
rb.velocity.y);
         }
}
```

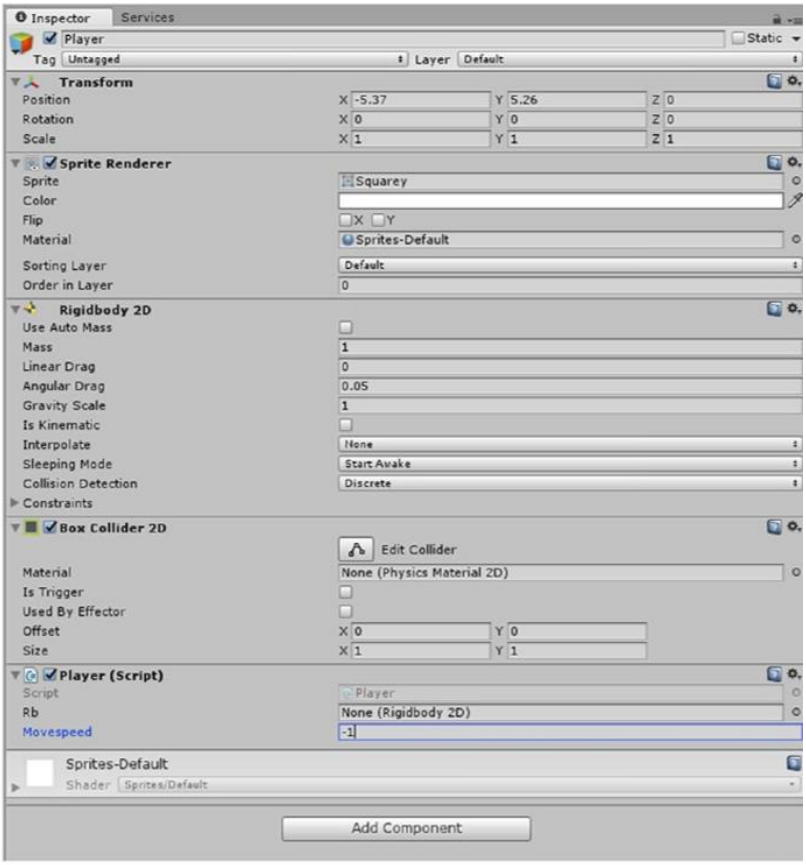
Squarey akan bergerak sama sebelumnya, tapi kali ini dengan kecepatan dua kali lipat. Ketika kami membuat variabel kami, saya menggunakan beberapa terminologi yang akan berguna untuk dipahami. Int, misalnya, adalah kependekan dari integer, yang merupakan jenis variabel yang menyimpan bilangan bulat. Setiap kali Anda mendefinisikan variabel, Anda selalu perlu memberi tahu C# jenis variabel apa yang sedang Anda kerjakan. Beberapa yang berguna untuk diketahui pada saat ini adalah:

- Integer: Semua bilangan bulat
- Float: Variabel floating point adalah angka dengan titik desimal
- Boolean: Variabel yang bisa benar atau salah, atau 1 atau 0
- String: Variabel teks

Sedangkan public artinya variabel tersebut dapat diakses dari luar script. Faktanya, ini berarti bahwa kita bahkan dapat mengedit kecepatan gerak kita dari UI Unity. Untuk melakukannya, hapus baris yang mengatakan movespeed = 2 lalu pilih Player GameObject di Unity. Anda akan melihat bahwa sekarang ada kotak Movespeed di Inspector.

Mengubah Kecepatan Bergerak di Inspector

Coba atur nilai ini ke -1 dan Anda akan melihat bahwa Squarey sekarang bergerak ke arah yang berlawanan, menjauh dari tangga dan menuju penurunan tiba-tiba. Jika Anda tidak menghapus baris movespeed = 2 dari skrip, ini hanya akan ditimpa setiap kali fungsi Start dipanggil. Jika Anda tidak ingin variabel tersebut dapat diakses di luar kode Anda, cukup gunakan kata privat alih-alih publik. Untuk saat ini, biarkan kecepatan bergerak sebagai 3, karena kami ingin Squarey menjadi sedikit lebih cepat untuk bit berikutnya. Meskipun Anda bisa melakukan hal yang sama tanpa menggunakan variabel, ini semua sangat berguna untuk diketahui dan Anda akan melihatnya berguna berulang kali saat Anda menambahkan lebih banyak elemen ke scene Anda.



Gambar 3.34 Mengubah Kecepatan Bergerak di Inspector

Coba atur nilai ini ke -1 dan Anda akan melihat bahwa Squarey sekarang bergerak ke arah yang berlawanan, menjauh dari tangga dan menuju penurunan tiba-tiba. Jika Anda tidak menghapus baris movespeed = 2 dari skrip, ini hanya akan ditimpa setiap kali fungsi Start dipanggil. Jika Anda tidak ingin variabel tersebut dapat diakses di luar kode Anda, cukup gunakan kata privat alih-alih publik. Untuk saat ini, biarkan kecepatan bergerak sebagai 3, karena kami ingin Squarey menjadi sedikit lebih cepat untuk bit berikutnya. Meskipun Anda bisa melakukan hal yang sama tanpa menggunakan variabel, ini semua sangat berguna untuk diketahui dan Anda akan melihatnya berguna berulang kali saat Anda menambahkan lebih banyak elemen ke scene Anda.

Mengontrol Karakter Player

Melihat karakter kita bergerak di sekitar level dan berinteraksi dengan scene cukup mengasyikkan, tetapi sebenarnya kita ingin dapat mengontrol karakter tersebut. Kabar baiknya adalah bahwa ini adalah sesuatu yang dapat kita lakukan dengan cukup mudah hanya dengan sedikit mengubah kode yang sudah kita miliki. Yang perlu kita lakukan adalah menambahkan beberapa pernyataan if, seperti pada kode berikut:

```
void Update() {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
        rb.velocity = new Vector2(-movespeed, rb.velocity.y);
}
        if (Input.GetKey(KeyCode.RightArrow))
        {
        rb.velocity = new Vector2(movespeed, rb.velocity.y);
}
```

}

Apa yang terjadi di sini adalah skrip memeriksa untuk melihat apakah ada input setiap kali game diperbarui. Tentu saja, KeyCode.LeftArrow dan KeyCode.RightArrow mengacu pada input keyboard masing-masing, dan kami kemudian memindahkan karakter dengan kecepatan gerak atau –kecepatan tergantung yang mana yang ditekan. Pernyataan "Jika" pada dasarnya memberi tahu bagian kode untuk dijalankan hanya ketika argumen tertentu benar. Kode di dalam kurung kurawal hanya akan berjalan jika baris di dalam kurung benar. Jika kita menulis ini dalam kode semu ("kode" palsu yang menggunakan terminologi bahasa Inggris normal agar lebih mudah dimengerti), maka itu akan ditranslate sebagai berikut:

If (Player is pressing right) {
Move character to the right
}

Itu selalu penting bahwa Anda ingat untuk menutup kurung kurawal di akhir pernyataan if. Jika Anda memiliki pengalaman dengan Excel atau software spreadsheet lainnya, menggunakan pernyataan if seperti ini mungkin sudah tidak asing lagi. Perhatikan jika Anda membuat apk (paket aplikasi untuk android - lebih lanjut tentang ini nanti) dan menjalankannya di android sekarang, itu akan benar-benar berfungsi dengan keyboard Bluetooth. Namun nanti, kita akan melihat cara menerapkan kontrol layar sentuh.

Jika Anda mengklik mainkan sekarang, Anda seharusnya memiliki kesempatan menarik untuk benar-benar mencoba mengendalikan Squarey—seperti game sungguhan. Meskipun kami harus menulis sedikit kode di sini, semoga Anda setuju bahwa ini sangat sederhana untuk semua hal: hanya dengan beberapa baris, kami sekarang memiliki dunia permainan yang bagus dan karakter yang dapat kami kendalikan di dalamnya. .

Lebih Lanjut Logika dan Memperkenalkan Jumping

Squarey dapat bergerak ke kiri dan ke kanan seperti pro sekarang dan cukup mahir melompat juga. Anda seharusnya merasa seperti orang tua yang bangga. Tetapi untuk platform dengan Mario, Sonic, dan yang terbaik dari mereka, dia juga perlu mempelajari beberapa keterampilan melompat. Sayangnya, itu sedikit lebih rumit daripada bergerak ke kiri dan ke kanan. Anda dapat mencoba dan menerapkan lompatan dengan cara yang sama seperti Anda menangani gerakan ke kiri dan ke kanan. Kode berikut akan memungkinkan pahlawan kita untuk melompat:

```
if (Input.GetKey(KeyCode.Space))
{
rb.velocity = new Vector2(rb.velocity.x, 5);
}
```

Satu-satunya masalah adalah bahwa kode ini juga memungkinkan dia untuk terbang. Itu karena kami akan menambahkan lebih banyak kecepatan ke atas setiap kali kami menekan bilah spasi, terlepas dari apakah dia berada di tanah atau di udara. Ini tidak baik. Jadi sebagai gantinya kita perlu memeriksa apakah dia ada di terra firma terlebih dahulu. Dan itu sedikit lebih rumit. Pertama, kita perlu membuat transformasi baru. Transformasi adalah suatu titik dalam ruang yang memiliki koordinat dan rotasi (sudut) tersendiri. Titik ini juga akan memiliki radius (yang akan menjadi float), dan kita juga membutuhkan layer mask. Kami sekarang juga membuat variabel Boolean pertama kami yang disebut onGround. Singkatnya, Anda menambahkan semua kode berikut ke skrip Anda:

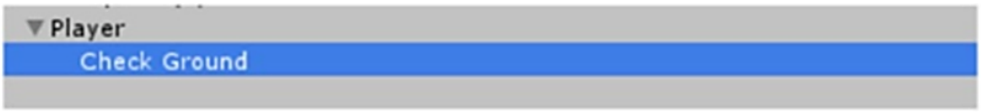
```
public Transform groundCheck;
public float groundCheckRadius;
```

```
public LayerMask whatIsGround;
private bool onGround;
```

Sekarang, itu mungkin tampak agak rumit, tetapi jangan khawatir, saya akan menjelaskan apa itu setiap bit dan apa fungsinya seiring berjalannya waktu. Jika itu tidak cukup menakutkan pada tahap ini, kami juga akan menambahkan fungsi lain ke kode kami, yang disebut FixedUpdate. FixedUpdate adalah fungsi seperti Update kecuali bahwa Update terikat pada kecepatan refresh layar, FixedUpdate lebih konsisten dan karena itu lebih dapat diandalkan untuk kode yang berkaitan dengan fisika. Di dalam fungsi ini, Anda akan menambahkan yang berikut:

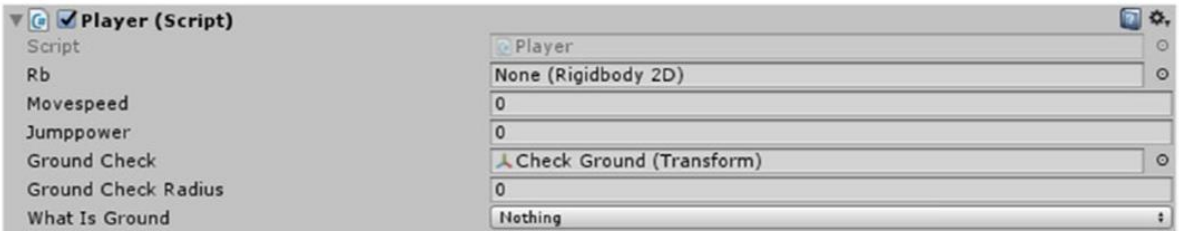
```
void FixedUpdate()
{
onGround = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius,
whatIsGround);
}
```

Jangan khawatir tentang apa yang dilakukan ini dulu. Kembali ke Unity di mana Anda sekarang dapat melihat float publik baru, Bool, dan layer mask yang kita buat di Inspector. Di sini Anda akan membuat GameObject kosong baru. Ini adalah GameObject seperti sprite atau kamera tetapi tanpa komponen. Klik kanan di Hierarchy dan pilih Create Empty. Anda akan menyebut GameObject Check Ground yang kosong ini dan menjadikannya anak dari Player.



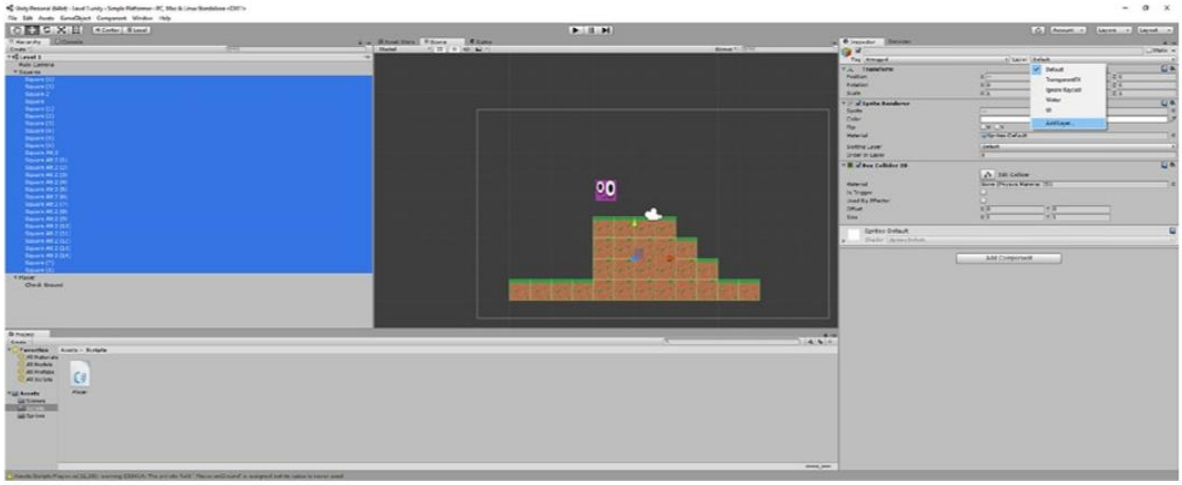
Gambar 3.35 Check Ground adalah GameObject dan anak Player yang kosong

Sekarang pilih player lagi dan di Hierarchy temukan di mana dikatakan Ground Check. Saat ini akan tertulis None (Transform), tetapi Anda akan mengubahnya dengan menyeret objek game Check Ground dan menjatuhkannya ke dalam kotak itu.



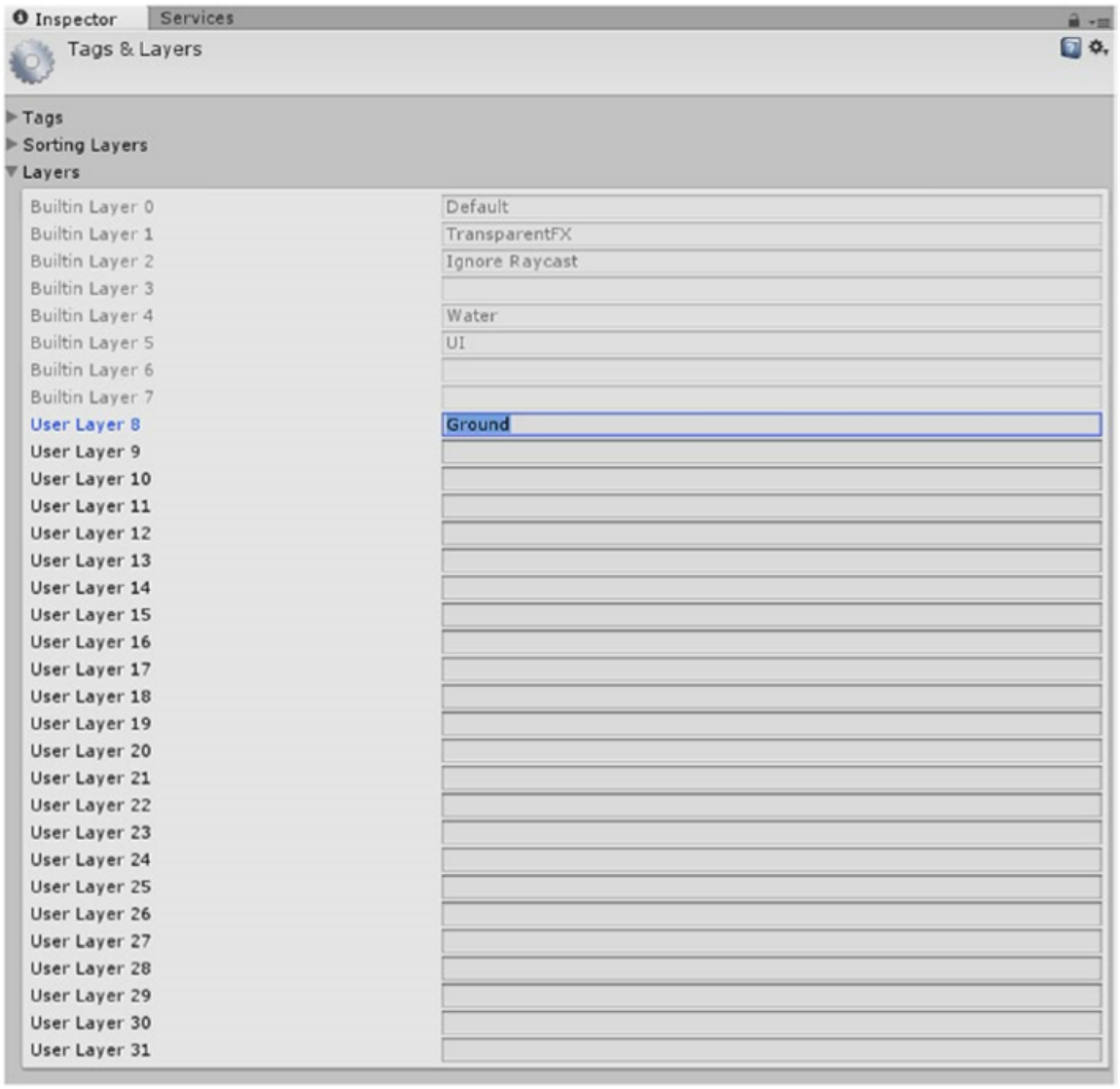
Gambar 3.36 Ground Check adalah transformasi yang sekarang didefinisikan sebagai koordinat Check Ground

Ingat: Ground Check adalah transformasi, yang berarti sekumpulan koordinat. Dengan menjatuhkan GameObject kosong ke sini, kami sekarang memberi tahu skrip kami untuk mengatur koordinat tersebut menjadi yang dilampirkan ke Check Ground. Dengan kata lain, kita telah mendefinisikan sebuah "titik", dan ini adalah bagaimana kita akan melihat apakah Squarey berdiri di atas tanah yang kokoh atau tidak. Sekarang atur nilai radius menjadi 0,1, artinya itu akan menjadi titik yang sangat kecil. Terakhir, pilih Check Ground dalam tampilan Scene dan gunakan move tool untuk memposisikannya tepat di bawah Squarey sehingga memeriksa ruang langsung di bawahnya. Satu hal lagi yang perlu kita lakukan adalah membuat layer baru dan beri nama Ground. Drag dan pilih semua ubin lantai Anda lagi lalu cari menu Layer di kiri atas Inspector. Di bagian bawah tarik-turun, Anda akan melihat opsi untuk Menambahkan Layer.



Gambar 3.37 Menambahkan layer Tanah

Anda kemudian akan diberi kesempatan untuk membuat layer baru Anda hanya dengan mengetikkan namanya di ruang berikutnya yang tersedia (beberapa layer sudah ditentukan oleh Unity)



Gambar 3.37 Panggil layer baru Ground

Ketik Ground ke dalam ruang kosong. Kemudian kembali ke Inspector untuk ubin Kotak Anda dan kali ini pilih Ground dari menu dropdown untuk mengatur semuanya ke nilai itu. Sekarang lihat Player GameObject di Inspector lagi dan kali ini atur What is Ground to Ground dengan menggunakan menu dropdown. Ini pada dasarnya berarti bahwa apa pun yang diatur ke layer Ground sekarang akan diperlakukan sebagai ground dalam skrip kita — artinya Squarey dapat melompat darinya. Dengan semua ini selesai, kita sekarang dapat menambahkan baris kode terakhir ke skrip kita:

```
if (Input.GetKey(KeyCode.Space)  && onGround)
{
rb.velocity = new Vector2(rb.velocity.x, 5);
}
```

Sekarang tekan play dan Anda akan menemukan bahwa Squarey dapat melompat—tetapi hanya jika dia berada di tanah yang kokoh.

Sedikit Penjelasan Lebih Lanjut

Anda mungkin masih sedikit bingung dengan apa yang sebenarnya terjadi di sini. Mari kita rekap apa yang terjadi. Kuncinya adalah baris ini, yang telah kami tempatkan di fungsi FixedUpdate kami:

```
onGround = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround);
```

Pernyataan Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround) adalah pernyataan benar atau salah yang menanyakan apakah transformasi groundCheck tumpang tindih dengan apa pun yang kita definisikan sebagai Ground (whatIsGround). onGround benar jika ada tumpang tindih dan salah jika tidak ada tumpang tindih—ingat, ini adalah variabel yang hanya bisa benar atau salah. Karena baris ini dalam FixedUpdate, nilainya akan terus berubah tergantung pada apa yang terjadi di dalam game.

Dalam kode semu, kami mengatakan ini:
Jika lingkaran di bawah pemutar tumpang tindih dengan material tanah, maka onGround benar. Jika tidak, onGround salah.

Kemudian, dalam fungsi Pembaruan kami, kami memeriksa apakah onGround benar setiap kali player menekan spasi: di C#, && berarti dan. Dengan menggunakan && di dalam pernyataan if, kita menguji apakah dua argumen benar. Jadi

```
if (Input.GetKey(KeyCode.Space)  && onGround)
{
rb.velocity = new Vector2(rb.velocity.x, 5);
}
```

Sebenarnya berarti:
Jika player menekan jump dan 'onGround' benar maka tambahkan kecepatan ke atas.

Kita juga bisa mengganti 5 dengan variabel seperti yang kita lakukan dengan kecepatan bergerak. Sebut saja daya lompat. Untuk kenyamanan Anda, seluruh skrip Player sekarang akan terlihat seperti ini:

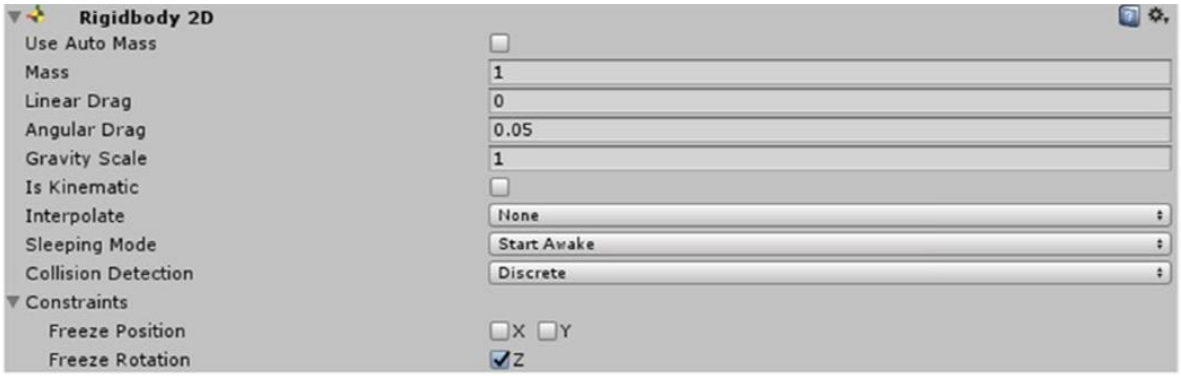
```
using UnityEngine;
using System.Collections;
public class Player : MonoBehaviour {
        public Rigidbody2D rb;
```

```
        public int movespeed;
        public int jumppower;
        public Transform groundCheck;
        public float groundCheckRadius;
        public LayerMask whatIsGround;
        private bool onGround;
        void Start () {
        rb = GetComponent<Rigidbody2D>();
        movespeed = 3;
        jumppower = 5;
}
void FixedUpdate()
{
onGround = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround);
}
        void Update ()
        {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
        rb.velocity = new Vector2(-movespeed, rb.velocity.y);
        }
        if (Input.GetKey(KeyCode.RightArrow))
        {
        rb.velocity = new Vector2(movespeed, rb.velocity.y);
        }
        if (Input.GetKey(KeyCode.Space)  && onGround)
        {
        rb.velocity = new Vector2(rb.velocity.x, jumppower);
        }
        }
}
```

Jika Anda menyalin ini untuk saat ini, Anda selalu dapat mencoba mengubah garis dan merekayasa baliknya saat Anda mencoba dan memahaminya. Anda harus menemukan bahwa banyak dari itu sebenarnya cukup jelas.

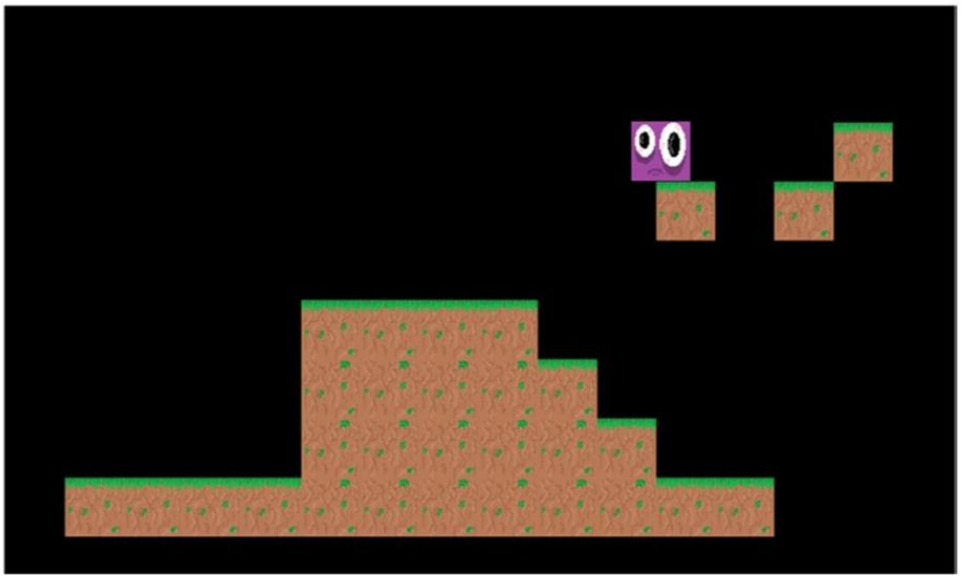
Satu Sentuhan Terakhir: Menjaga agar Player Tetap Tegak

Kecuali ada satu masalah kecil: Squarey saat ini jatuh dan berputar menuruni tangga seolah-olah dia minum terlalu banyak vodka. Jika Anda melompat dan kemudian mendarat di kepala Anda, Anda tidak akan bisa lepas landas lagi karena GameObject Check Ground sekarang akan mengarah ke udara. Untuk memperbaikinya, klik Squarey dan temukan opsi Constraints di bawah komponen Rigidbody di Inspector. Di sini, Anda akan menemukan opsi untuk Freeze Rotation Z. Centang kotak itu dan sudut Squarey sekarang akan terkunci di tempatnya.



Gambar 3.38 Centang Rotasi Beku dan karakter Anda akan berhenti jatuh

Anda juga akan melihat beberapa opsi lain di sini, seperti Gravity Scale (yang mengontrol bagaimana Squarey dipengaruhi oleh gravitasi) dan beberapa lainnya. Kami akan bermain-main dengan ini di Bab 5. Sekarang Anda dapat mencoba membuat beberapa platform mengambang dan sedikit bersenang-senang mencoba melewatinya tanpa jatuh dari layar. Anda mungkin perlu sedikit menyesuaikan jumppower Anda untuk itu atau Scale Gravitasi Anda, jadi bersenang-senanglah dengannya. Dan luangkan waktu sekarang untuk merenungkan apa yang telah Anda capai: hanya sebentar dan Anda sudah memiliki karakter yang dapat melompati layar dan level yang cukup menyenangkan untuk dinavigasi. Kami baru saja memulai, jadi pikirkan saja apa lagi yang dapat Anda lakukan sebelum kami selesai!



Gambar 3.39 Game sudah terlihat hampir jadi

BAB 4
MENAMBAHKAN MUSUH DAN MENGEMBANGKAN GRAFIS GAME 3D

Demo gerakan dari bab sebelumnya cukup keren tapi masih belum benar-benar game. Mari kita ubah demo gerakan itu menjadi Sniper pertama. Jika Anda berpikir tentang apa lagi yang kita butuhkan sekarang, itu bermuara pada kemampuan untuk menembak, dan hal-hal untuk menembak. Pertama kita akan menulis skrip yang memungkinkan player untuk menembak objek di scene. Kemudian kita akan membangun musuh untuk mengisi scene, termasuk kode untuk berkeliaran tanpa tujuan dan bereaksi saat dipukul. Akhirnya kita akan memungkinkan musuh untuk melawan, memancarkan bola api ke player. Tidak ada skrip dari bab 2 yang perlu diubah; sebagai gantinya, kami akan menambahkan skrip ke proyek—skrip yang menangani fitur tambahan.

Saya telah memilih Sniper pertama untuk proyek ini karena beberapa alasan. Salah satunya adalah karena game FPS sangat populer; orang-orang suka game menembak, jadi mari kita buat game menembak. Alasan yang lebih halus berkaitan dengan teknik yang akan Anda pelajari; proyek ini adalah cara yang bagus untuk mempelajari beberapa konsep dasar dalam simulasi 3D. Misalnya, permainan menembak adalah cara yang bagus untuk mengajarkan raycasting. Sedikit kita akan membahas secara spesifik apa itu raycasting, tetapi untuk saat ini Anda hanya perlu tahu bahwa itu adalah alat yang berguna untuk banyak tugas berbeda dalam simulasi 3D. Meskipun raycasting berguna dalam berbagai situasi, kebetulan menggunakan raycasting paling intuitif untuk pemotretan.

Saya telah memilih Sniper pertama untuk proyek ini karena beberapa alasan. Salah satunya adalah karena game FPS sangat populer; orang-orang suka game menembak, jadi mari kita buat game menembak. Alasan yang lebih halus berkaitan dengan teknik yang akan Anda pelajari; proyek ini adalah cara yang bagus untuk mempelajari beberapa konsep dasar dalam simulasi 3D. Misalnya, permainan menembak adalah cara yang bagus untuk mengajarkan raycasting. Sedikit kita akan membahas secara spesifik apa itu raycasting, tetapi untuk saat ini Anda hanya perlu tahu bahwa itu adalah alat yang berguna untuk banyak tugas berbeda dalam simulasi 3D. Meskipun raycasting berguna dalam berbagai situasi, kebetulan menggunakan raycasting paling intuitif untuk pemotretan. pendekatan untuk mengirim pesan yang ditunjukkan dalam proyek ini juga berguna di tempat lain. Di bab-bab selanjutnya Anda akan melihat aplikasi lain untuk teknik ini, dan bahkan dalam satu proyek ini kita akan membahas situasi alternatif.

Pada akhirnya kami akan mendekati proyek ini satu fitur baru pada satu waktu, dengan permainan selalu dapat dimainkan di setiap langkah tetapi juga selalu merasa seperti ada bagian yang hilang untuk dikerjakan selanjutnya. Peta jalan ini memecah langkah-langkah menjadi perubahan kecil yang dapat dimengerti, dengan hanya satu fitur baru yang ditambahkan di setiap langkah:

1. Tulis kode yang memungkinkan player untuk menembak ke dalam scene.
2. Buat target statis yang bereaksi saat dipukul.
3. Buat target berkeliaran.
4. Memunculkan target pengembara secara otomatis.
5. Aktifkan target/musuh untuk menembakkan bola api ke arah player.

4.1 MEMOTRET MELALUI RAYCAST

Fitur baru pertama yang diperkenalkan ke demo 3D adalah pemotretan. Melihat sekeliling dan bergerak tentu saja merupakan fitur penting untuk Sniper pertama, tetapi ini

bukan permainan sampai player dapat memengaruhi simulasi dan menerapkan keterampilan mereka. Pemotretan dalam game 3D dapat diimplementasikan dengan beberapa pendekatan berbeda, dan salah satu pendekatan terpenting adalah raycasting.

Light (Cahaya)

Sebelum kita membahas lebih jauh tentang raycasting, mari kita pelajari dulu tentang Lighting.

Dalam segala bentuk media visual, cahaya sangat berperan dalam menentukan bagaimana suatu scene dipersepsikan. Cahaya terang dan sedikit kuning dapat membuat scene terlihat cerah dan hangat. Ambil scene yang sama dan berikan cahaya biru intensitas rendah, dan itu akan terlihat menakutkan dan membingungkan. Warna lampu juga akan berpadu dengan warna skybox untuk memberikan hasil yang tampak lebih realistis. Sebagian besar scene yang mengupayakan realisme atau efek dramatis menggunakan setidaknya satu cahaya (dan seringkali banyak). Di jam-jam sebelumnya, Anda secara singkat bekerja dengan lampu untuk menyorot elemen lain. Di bagian ini, Anda bekerja dengan lampu lebih langsung.

Ulangi Properti

Lampu yang berbeda memiliki banyak sifat yang sama. Jika lampu memiliki properti yang telah tercakup di bawah jenis lampu yang berbeda, itu tidak tercakup lagi. Ingatlah bahwa jika dua jenis cahaya yang berbeda memiliki properti dengan nama yang sama, properti tersebut melakukan hal yang sama.

Apa Itu Light/Cahaya?

Dalam Unity, lampu bukanlah objek itu sendiri. Sebaliknya, lampu adalah komponen. Ini berarti bahwa saat Anda menambahkan cahaya ke scene, Anda sebenarnya hanya menambahkan objek game dengan komponen Cahaya, yang dapat berupa jenis cahaya apa pun yang dapat Anda gunakan.

Baking Versus Waktu Nyata

Sebelum Anda mulai benar-benar bekerja dengan lampu, Anda perlu memahami dua cara utama menggunakan cahaya: dipanggang dan waktu nyata. Hal yang perlu diingat adalah bahwa semua cahaya dalam game kurang lebih bersifat komputasional. Cahaya harus ditentukan oleh mesin dalam tiga langkah: 1. Warna, arah, dan jangkauan sinar cahaya yang disimulasikan dihitung dari sumber cahaya. 2. Ketika sinar cahaya mengenai permukaan, mereka menerangi dan mengubah warna permukaan. 3. Sudut benturan dengan permukaan dihitung, dan cahaya memantul. Langkah 1 dan 2 diulang lagi dan lagi (tergantung pada pengaturan cahaya). Dengan setiap pantulan, sifat sinar cahaya diubah oleh permukaan yang mereka tabrak (seperti di kehidupan nyata). Proses ini diulang untuk setiap cahaya di setiap bingkai dan menciptakan iluminasi global (di mana objek menerima cahaya dan warna berdasarkan objek di sekitarnya). Proses ini sedikit terbantu oleh fitur Precomputed Realtime GI, yang diaktifkan secara default dan tidak memerlukan usaha dari Anda. Dengan fitur ini, sebagian dari proses kalkulasi cahaya yang dijelaskan di atas dihitung sebelum scene dimulai sehingga hanya sebagian kalkulasi yang perlu dilakukan saat runtime. Anda mungkin sudah melihat ini dalam operasi. Jika Anda pernah membuka scene Unity untuk pertama kalinya dan memperhatikan bahwa elemen scene gelap untuk sesaat, Anda telah melihat proses pra-perhitungan. Baking, di sisi lain, mengacu pada proses pra-perhitungan cahaya dan bayangan sepenuhnya untuk tekstur dan objek selama pembuatan. Anda dapat melakukan ini dengan Unity atau dengan editor grafis. Misalnya, jika Anda membuat tekstur dinding dengan bintik gelap di atasnya yang menyerupai bayangan manusia dan kemudian meletakkan model manusia di sebelah dinding tempat ia berada, model itu akan tampak seperti membuat bayangan di dinding. Namun, kenyataannya adalah bahwa bayangan itu "dipanggang" ke dalam tekstur. Baking dapat membuat game Anda berjalan lebih cepat karena mesin tidak

perlu menghitung cahaya dan bayangan setiap frame; namun, baking tidak terlalu diperlukan untuk kebutuhan Anda saat ini karena permainan yang dibahas dalam buku ini tidak cukup rumit untuk membutuhkannya.

Point Lights Jenis lampu pertama yang akan Anda kerjakan adalah point light. Pikirkan titik cahaya sebagai bola lampu. Semua cahaya dipancarkan dari satu lokasi pusat ke segala arah. Lampu titik adalah jenis lampu yang paling umum untuk menerangi area interior. Untuk menambahkan titik cahaya ke scene, pilih GameObject > Light > Point Light. Setelah berada di scene, objek game Point Light dapat dimanipulasi seperti yang lainnya. Tabel 5.1 menjelaskan sifat titik cahaya.

Tabel 4.1 Tabel Properti Titik Cahaya

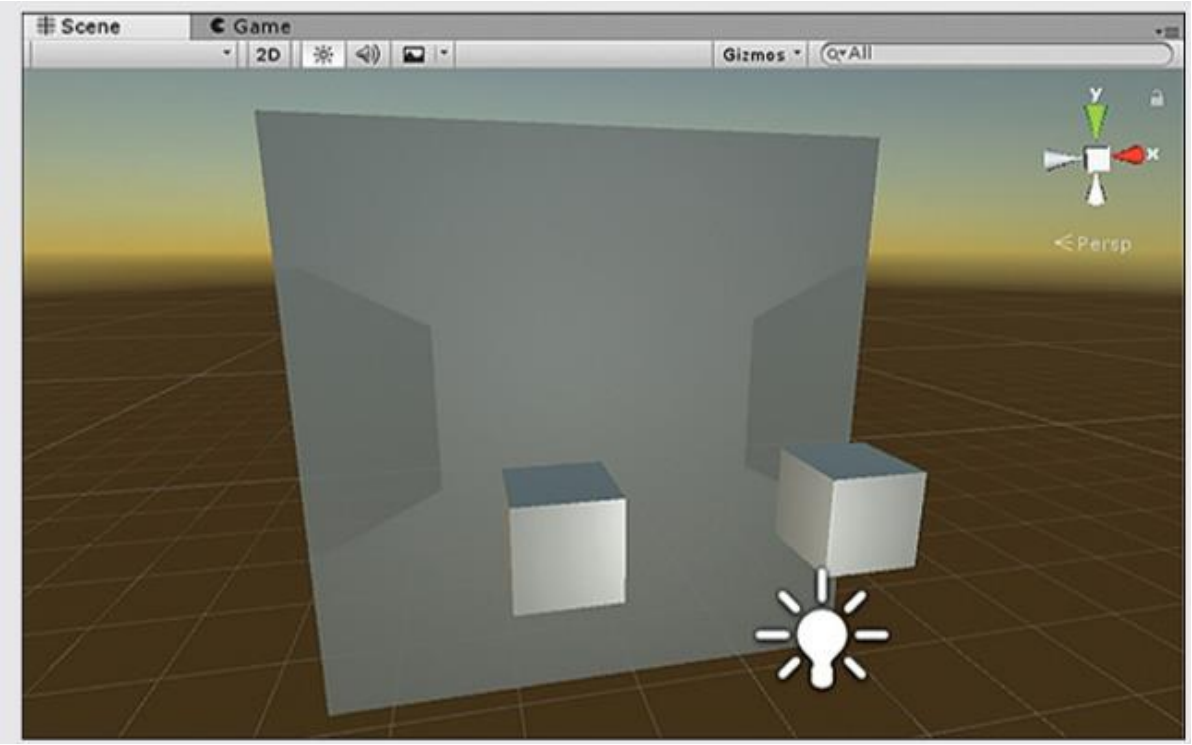
Properti	**Deskripsi**
Type	Menentukan jenis cahaya yang dikeluarkan komponen. Karena ini adalah point light, maka tipenya harus Point. Mengubah properti Type mengubah jenis cahaya.
Range	Mendikte seberapa jauh cahaya bersinar. Penerangan memudar secara merata dari sumber cahaya ke kisaran yang ditentukan.
Color	Menentukan warna cahaya bersinar. Warna adalah aditif, yang berarti jika Anda menyinari lampu merah pada objek biru, itu akan menjadi ungu.
Mode	Menentukan apakah cahaya adalah cahaya waktu nyata, cahaya yang dipanggang, atau campuran keduanya.
Intensity	Mendikte seberapa terang cahaya bersinar. Perhatikan bahwa cahaya masih bersinar hanya sejauh yang ditentukan oleh properti Range.
Indirect Multiplier	Menentukan seberapa terang cahaya setelah memantul dari benda. (Unity mendukung Global Illumination, artinya ia menghitung hasil pantulan cahaya.)
Shadow Type	Menentukan bagaimana bayangan dihitung untuk sumber ini dalam sebuah scene. Bayangan lembut lebih realistis tetapi juga lebih intensif kinerja.
Cookie	Menerima peta kubus (seperti kotak langit) yang menentukan pola agar cahaya dapat menembus. Cookie dibahas secara lebih rinci nanti dalam jam ini.
Draw Halo	Menentukan apakah halo bercahaya muncul di sekitar cahaya. Halo dibahas secara lebih rinci nanti di jam ini.
Render Mode	Menerima aset suar cahaya dan mensimulasikan efek cahaya terang yang menyinari lensa kamera.
Culling Mask	Menentukan pentingnya cahaya ini. Tiga pengaturan tersebut adalah Otomatis, Penting, dan Tidak Penting. Cahaya yang penting dirender dalam kualitas yang lebih tinggi, sedangkan cahaya yang kurang penting dirender dengan lebih cepat. Gunakan pengaturan Otomatis untuk saat ini.

Menambahkan Titik Cahaya ke Scene

Ikuti langkah-langkah berikut untuk membuat scene dengan beberapa pencahayaan titik dinamis:

1. Buat proyek atau scene baru dan hapus lampu arah yang ada secara default.

2. Tambahkan pesawat ke scene (dengan memilih GameObject > 3D Object > Plane). Pastikan bahwa pesawat diposisikan pada (0, .5, 0) dan diputar ke (270, 0, 0). Bidang harus terlihat oleh kamera, tetapi hanya dari satu sisi dalam tampilan Scene.
3. Tambahkan dua kubus ke scene. Posisikan mereka di (-1.5, 1, -5) dan (1.5, 1, -5).
4. Tambahkan titik cahaya ke scene (dengan memilih GameObject > Light > Point Light). Posisikan titik lampu pada (0, 1, -7). Perhatikan bagaimana cahaya menerangi sisi dalam kubus dan bidang latar belakang (lihat Gambar 4.1).
5. Atur jenis bayangan cahaya ke Hard Shadows dan coba pindahkan. Lanjutkan menjelajahi sifat-sifat cahaya. Pastikan untuk bereksperimen dengan warna, jangkauan, dan intensitas cahaya.



Gambar 4.1 Hasil dari Ini Coba Sendiri.

Spotlight/Lampu Sorot

Lampu sorot sangat mirip dengan lampu depan mobil atau senter. Cahaya lampu sorot dimulai di titik pusat dan memancar keluar dalam bentuk kerucut. Dengan kata lain, lampu sorot menerangi apa pun yang ada di depan mereka sambil membiarkan yang lainnya dalam kegelapan. Sedangkan titik cahaya mengirimkan cahaya ke segala arah, Anda dapat mengarahkan lampu sorot. Untuk menambahkan sorotan ke scene, pilih GameObject > Create Other > Spotlight. Alternatifnya, jika Anda sudah memiliki cahaya di scene Anda, Anda dapat mengubah jenisnya menjadi Spot, dan itu menjadi sorotan. Lampu sorot hanya memiliki satu properti yang belum tercakup: Sudut Titik. Properti Spot Angle menentukan radius kerucut cahaya yang dipancarkan oleh lampu sorot.

Menambahkan Spotlight ke Scene

Sekarang Anda memiliki kesempatan untuk bekerja dengan lampu sorot di Unity. Untuk singkatnya, latihan ini menggunakan proyek yang dibuat di sebelumnya untuk lampu titik. Jika Anda belum menyelesaikan latihan itu, lakukan dan lanjutkan dengan langkah-langkah berikut:

1. Gandakan scene Point Light dari proyek sebelumnya (dengan memilih Edit > Duplicate) dan beri nama Spotlight scene baru.

2. Klik kanan Point light pada tampilan Hierarchy dan pilih Rename. Ganti nama objek Spotlight. Di Inspector, ubah properti Type menjadi Spot. Posisikan objek cahaya pada (0, 1, -13).
3. Bereksperimenlah dengan sifat-sifat lampu sorot. Perhatikan bagaimana rentang, intensitas, dan sudut titik membentuk dan mengubah efek cahaya.

Lampu Arah

Jenis lampu terakhir yang akan Anda gunakan pada jam ini adalah lampu arah. Lampu terarah mirip dengan lampu sorot karena dapat diarahkan. Namun, tidak seperti lampu sorot, lampu arah menerangi seluruh scene. Anda dapat menganggap cahaya terarah mirip dengan matahari. Faktanya, Anda menggunakan cahaya terarah sebagai matahari di Jam 4, "Medan dan Lingkungan." Cahaya dari cahaya terarah memancar secara merata dalam garis paralel di seluruh scene. Scene baru hadir dengan lampu arah secara default. Untuk menambahkan cahaya arah baru ke scene, pilih GameObject > Light > Directional Light. Alternatifnya, jika Anda sudah memiliki cahaya dalam sebuah scene, Anda dapat mengubah jenisnya ke Directional, dan itu menjadi cahaya terarah. Lampu penunjuk arah memiliki satu properti tambahan yang belum tercakup: Ukuran Cookie. Cookie akan dibahas kemudian dalam jam ini, tetapi pada dasarnya properti ini mengontrol seberapa besar cookie dan dengan demikian berapa kali diulang di seluruh scene.

Menambahkan Cahaya Terarah ke Scene

Sekarang Anda akan menambahkan cahaya terarah ke scene Unity. Sekali lagi, latihan ini dibangun di atas proyek yang dibuat di sebelumnya. Jika Anda belum menyelesaikan latihan itu, lakukan, lalu ikuti langkah-langkah berikut:

1. Gandakan scene Spotlight dari proyek sebelumnya (dengan memilih Edit > Duplicate) dan beri nama scene baru Directional Light.
2. Klik kanan Spotlight pada tampilan Hierarchy dan pilih Rename. Ubah nama objek Directional Light. Di Inspector, ubah properti Type menjadi Directional.
3. Ubah putaran lampu menjadi (75, 0, 0). Perhatikan bagaimana langit berubah saat Anda memutar lampu. Hal ini disebabkan scene menggunakan skybox prosedural. Skybox dibahas lebih detail di Jam 6, "Game 1: Pembalap Luar Biasa."
4. Perhatikan bagaimana cahaya terlihat pada objek di scene. Sekarang ubah posisi lampu menjadi (50, 50, 50). Perhatikan bahwa cahaya tidak berubah. Karena arah cahaya berada pada garis sejajar, maka posisi cahaya tidak menjadi masalah. Hanya rotasi cahaya arah yang penting.
5. Percobaan dengan sifat-sifat cahaya terarah. Tidak ada jangkauan (karena jangkauan tidak terbatas), tetapi perhatikan bagaimana warna dan intensitas mempengaruhi scene.

Area Lights dan Emissive Materials

Ada dua jenis cahaya lagi yang tidak tercakup dalam teks ini: area lights dan material emissive. Area light adalah fitur yang ada untuk proses yang disebut lightmap baking. Topik-topik ini lebih maju daripada yang Anda butuhkan untuk proyek game dasar sehingga tidak tercakup dalam buku ini. Jika Anda ingin mempelajari lebih lanjut tentang ini, lihat kekayaan dokumentasi online Unity. Bahan memancarkan adalah bahan yang diterapkan pada objek yang benar-benar mentransmisikan cahaya. Jenis lampu ini bisa sangat berguna untuk layar TV, lampu indikator, dan sebagainya.

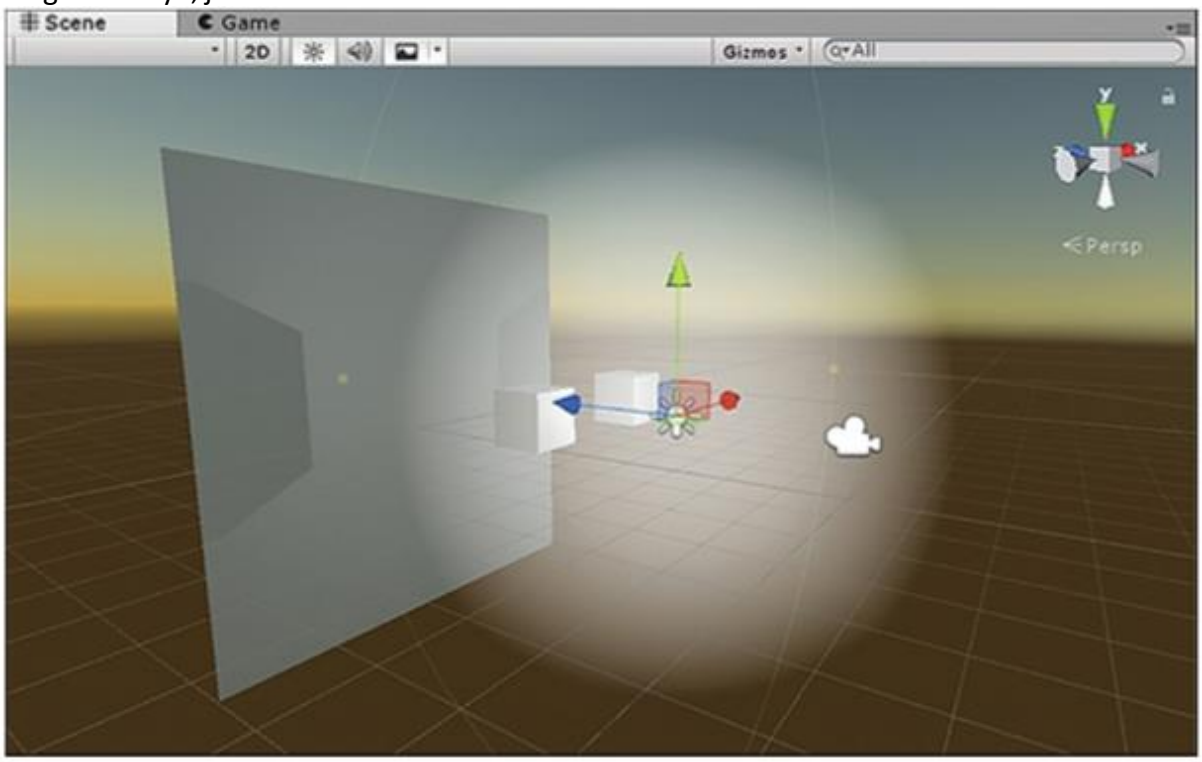
4.2 MEMBUAT CAHAYA DARI OBJEK

Karena lampu di Unity adalah komponen, objek apa pun dalam scene bisa menjadi cahaya. Untuk menambahkan cahaya ke objek, pertama pilih objek. Kemudian, pada tampilan

Inspector, klik tombol Add Component. Daftar baru akan muncul. Pilih Rendering lalu Light. Sekarang objek Anda memiliki komponen ringan. Cara alternatif untuk menambahkan cahaya ke objek adalah dengan memilih objek dan pilih Component > Rendering > Light. Perhatikan beberapa hal tentang menambahkan lampu ke objek. Pertama, sebuah benda tidak akan menghalangi cahaya. Ini berarti bahwa menempatkan cahaya di dalam kubus tidak akan menghentikan cahaya dari memancar. Kedua, menambahkan cahaya ke suatu objek tidak membuatnya bersinar. Objek itu sendiri tidak akan terlihat seperti memancarkan cahaya, tetapi memang begitu.

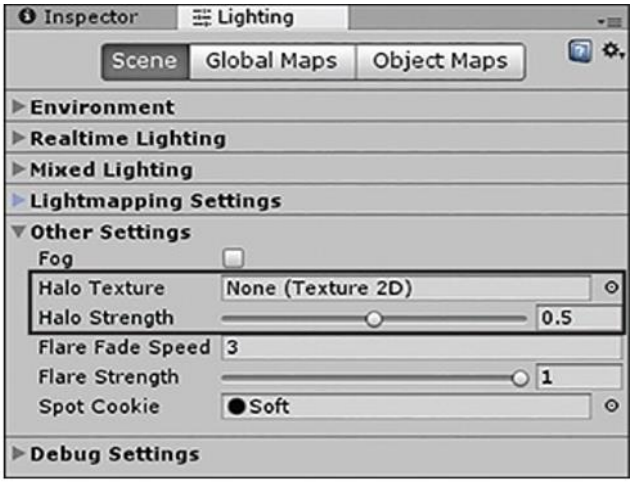
Halo

Halo adalah lingkaran bercahaya yang muncul di sekitar lampu dalam kondisi berkabut atau berawan (lihat Gambar 4.2). Mereka terjadi karena cahaya memantul dari partikel kecil di sekitar sumber cahaya. Di Unity, Anda dapat dengan mudah menambahkan lingkaran cahaya ke lampu. Setiap lampu memiliki kotak centang Draw Halo. Jika dicentang, lingkaran cahaya digambar. Jika Anda tidak dapat melihat lingkaran cahaya, Anda mungkin terlalu dekat dengan cahaya, jadi cobalah mundur sedikit.



Gambar 4.2 Sebuah lingkaran cahaya di sekitar cahaya.

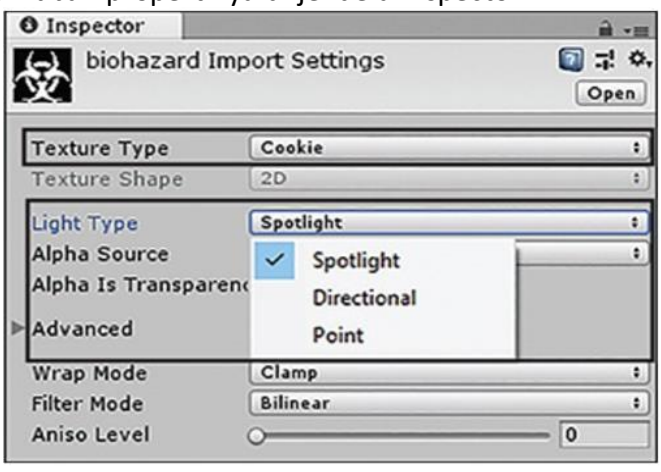
Ukuran lingkaran cahaya ditentukan oleh jangkauan cahaya. Semakin besar jangkauannya, semakin besar lingkaran cahayanya. Unity juga menyediakan beberapa properti yang berlaku untuk semua lingkaran cahaya dalam sebuah scene. Anda dapat mengakses properti ini dengan memilih Window > Lighting > Settings. Perluas Pengaturan Lainnya, dan pengaturan kemudian muncul di tampilan Inspector



Gambar 4.3 Pengaturan pencahayaan scene.

Properti Halo Strength menentukan seberapa besar halo, berdasarkan jangkauan cahaya. Misalnya, jika sebuah cahaya memiliki jangkauan 10 dan kekuatan 1, halo akan meluas ke seluruh 10 unit. Jika kekuatannya 0,5, lingkaran cahaya hanya memanjang 5 unit (10 × .5 = 5). Properti Halo Texture memungkinkan Anda untuk menentukan bentuk yang berbeda untuk halo dengan memberikan tekstur baru. Jika Anda tidak ingin menggunakan tekstur khusus untuk lingkaran cahaya, Anda dapat membiarkannya kosong, dan lingkaran default digunakan. Cookies

Jika Anda pernah menyorotkan cahaya ke dinding dan kemudian meletakkan tangan Anda di antara lampu dan dinding, Anda mungkin memperhatikan bahwa tangan Anda menghalangi sebagian cahaya, meninggalkan bayangan berbentuk tangan di dinding. Anda dapat mensimulasikan efek ini di Unity dengan menggunakan cookie. Cookie adalah tekstur khusus yang dapat Anda tambahkan ke lampu untuk menentukan bagaimana cahaya memancar. Cookie sedikit berbeda untuk lampu titik, titik, dan arah. Lampu sorot dan lampu arah keduanya menggunakan tekstur datar hitam-putih untuk cookies. Lampu sorot tidak mengulangi cookie, tetapi lampu arah melakukannya. Lampu titik juga menggunakan tekstur hitam-putih, tetapi jenis cahaya ini harus ditempatkan dalam peta kubus. Cubemap adalah enam tekstur yang ditempatkan bersama untuk membentuk kotak (seperti skybox). Menambahkan cookie ke lampu adalah proses yang cukup mudah. Anda cukup menerapkan tekstur ke properti Cookie dari cahaya. Trik untuk membuat cookie berfungsi adalah menyiapkan tekstur dengan benar sebelumnya. Untuk mengatur tekstur dengan benar, pilih di Unity dan kemudian ubah propertinya di jendela Inspector.

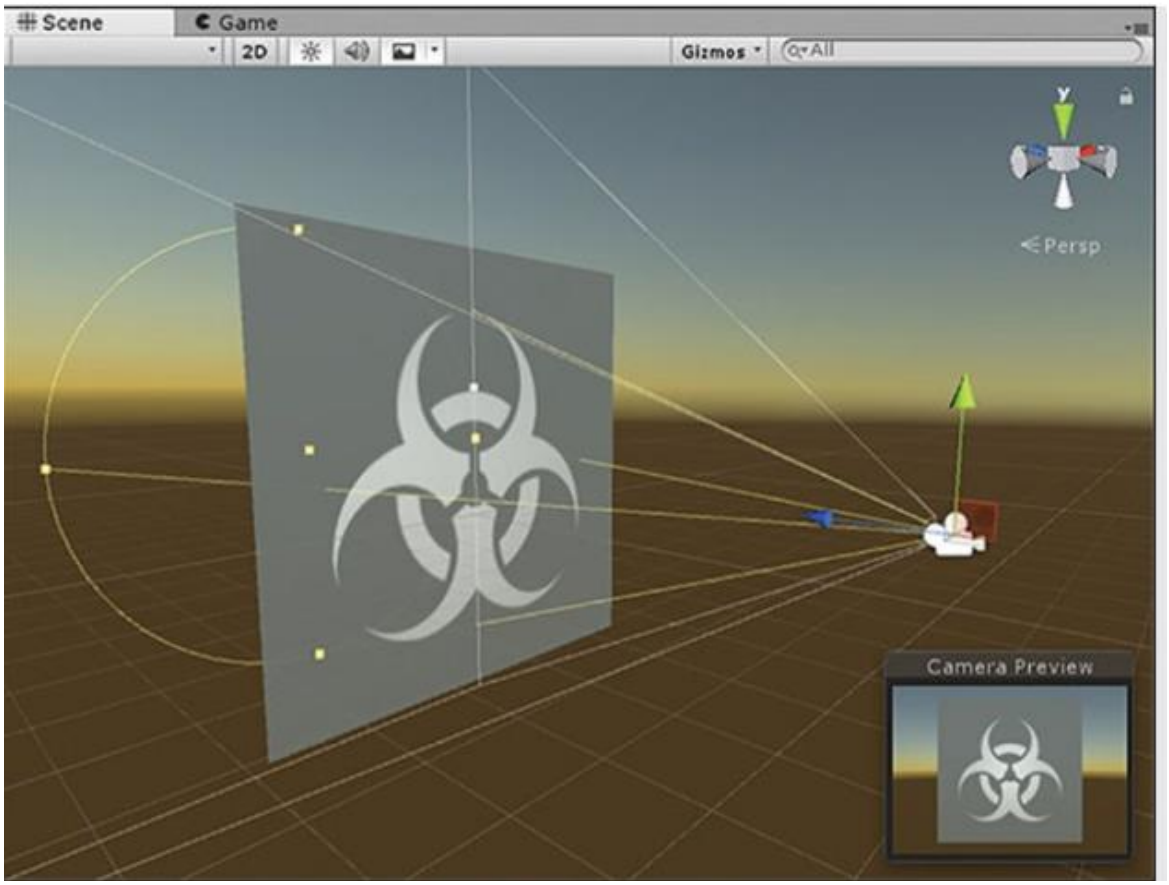


Gambar 4.4 Sifat tekstur cookie untuk lampu titik, titik, dan arah.

Menambahkan Cookie ke Spotlight

Latihan ini memerlukan gambar biohazard.png, tersedia di aset buku untuk Jam 5. Ikuti langkah-langkah ini untuk menambahkan cookie ke sorotan sehingga Anda dapat melihat proses dari awal hingga akhir:

1. Buat a proyek atau scene baru. Hapus cahaya arah dari scene.
2. Tambahkan bidang ke scene dan posisikan di (0, 1, 0) dengan rotasi (270, 0, 0).
3. Tambahkan sorotan ke Kamera Utama dengan memilih Kamera Utama lalu klik Komponen > Rendering > Cahaya dan ubah jenisnya menjadi Spot. Atur rentang ke 18, sudut titik ke 40, dan intensitas ke 3.
4. Drag tekstur biohazard.png dari aset buku ke tampilan Proyek Anda. Pilih tekstur, dan dalam tampilan Inspector, ubah tipe tekstur ke Cookie, atur jenis cahaya ke Spotlight, dan atur sumber alfa ke From Grayscale. Ini membuat blok cookies menjadi terang di tempat yang hitam.
5. Dengan Kamera Utama dipilih, klik dan drag tekstur biohazard ke properti Cookie dari komponen cahaya. Anda akan melihat simbol biohazard diproyeksikan ke pesawat (lihat Gambar 4.5).
6. Bereksperimenlah dengan rentang dan intensitas cahaya yang berbeda. rotate bidang dan lihat bagaimana simbol melengkung dan terdistorsi.



Gambar 4.5 Spotlight dengan Cookies

4.3 KAMERA

Kamera adalah pandangan player ke dunia. Ini memberikan perspektif dan mengontrol bagaimana hal-hal tampak kepada player. Setiap game di Unity memiliki setidaknya satu kamera. Bahkan, kamera selalu ditambahkan untuk Anda setiap kali Anda membuat scene

baru. Kamera selalu muncul dalam hierarki sebagai Kamera Utama. Di bagian ini, Anda akan mempelajari semua tentang kamera dan cara menggunakannya untuk efek yang menarik.

Anatomi Kamera Semua kamera memiliki seperangkat properti yang sama yang menentukan bagaimana mereka berperilaku. Tabel 5.2 menjelaskan semua properti kamera.

Tabel 4.2 Properti Kamera

Properti	**Deskripsi**
Clear Flags	Menentukan apa yang ditampilkan kamera di area di mana tidak ada objek game. Standarnya adalah Skybox. Jika tidak ada skybox, default kamera adalah warna solid. Hanya Kedalaman harus digunakan hanya jika ada beberapa kamera. Jangan Hapus menyebabkan goresan dan hanya boleh digunakan jika Anda menulis shader khusus.
Backgrounf	Menentukan warna latar belakang jika tidak ada skybox.
Culling Mask	Menentukan layer apa yang diambil oleh kamera. Secara default, kamera melihat semuanya. Anda dapat menghapus centang pada layer tertentu (lebih lanjut tentang layer nanti dalam jam ini), dan layer tersebut tidak akan terlihat oleh kamera.
Projection	Menentukan bagaimana kamera melihat dunia. Dua opsi tersebut adalah Perspektif dan Ortografi. Kamera perspektif melihat dunia dalam 3D, dengan objek yang lebih dekat menjadi lebih besar dan objek yang lebih jauh menjadi lebih kecil. Ini adalah pengaturan untuk digunakan jika Anda ingin kedalaman dalam permainan Anda. Pengaturan Ortografis mengabaikan kedalaman dan memperlakukan semuanya sebagai datar.
Field of View	Menentukan seberapa lebar area yang dapat dilihat kamera.
Clipping Planes	Menentukan rentang di mana objek terlihat oleh kamera. Objek yang lebih dekat dari bidang dekat atau lebih jauh dari bidang jauh tidak akan terlihat.
View Port Recat	Menetapkan bagian mana dari layar sebenarnya yang diproyeksikan oleh kamera. (View Port Rect adalah kependekan dari View Port Rectangle.) Secara default, x dan y keduanya diatur ke 0, yang menyebabkan kamera mulai di kiri bawah layar. Lebar dan tinggi keduanya disetel ke 1, yang menyebabkan kamera menutupi 100% layar secara vertikal dan horizontal. Ini dibahas lebih rinci nanti di jam ini.
Depth	Menentukan prioritas untuk beberapa kamera. Angka yang lebih rendah ditarik terlebih dahulu, yang berarti angka yang lebih tinggi dapat ditarik di atas dan dapat menyembunyikannya secara efektif.
Redering Path	Menentukan bagaimana kamera merender. Itu harus dibiarkan diatur ke Gunakan Pengaturan Player
Target Texture	Memungkinkan Anda menentukan tekstur untuk menggambar kamera alih-alih layar.
Occlusion Culling	Menonaktifkan rendering objek saat objek tersebut tidak terlihat oleh kamera karena objek tersebut dikaburkan (ditutupi) oleh objek lain.
Allow HDR	Menentukan apakah penghitungan cahaya internal Unity terbatas pada rentang warna dasar. (HDR adalah singkatan dari Hyper-

	Dynamic Range.) Properti ini memungkinkan efek visual tingkat lanjut.
Allow MSAA	Mengaktifkan jenis antialiasing dasar, namun efisien, yang disebut antialiasing MultiSample. Antialiasing adalah metode menghilangkan tepi berpiksel saat merender grafik.
Allow Dynamic Resolution	Memungkinkan penyesuaian resolusi dinamis untuk game konsol.

Kamera memiliki banyak properti, tetapi Anda dapat mengatur sebagian besar dan kemudian melupakannya. Kamera juga memiliki beberapa komponen tambahan. Layer suar memungkinkan kamera untuk melihat suar lensa dari lampu, dan pendengar audio memungkinkan kamera untuk menangkap suara. Jika Anda menambahkan lebih banyak kamera ke sebuah scene, Anda harus menghapus pendengar audionya. Hanya ada satu pendengar audio per scene.

Beberapa Kamera

Banyak efek dalam game modern tidak akan mungkin terjadi tanpa banyak kamera. Untungnya, Anda dapat memiliki kamera sebanyak yang Anda inginkan dalam scene Unity. Untuk menambahkan kamera baru ke scene, pilih GameObject > Kamera. Atau, Anda dapat menambahkan komponen kamera ke objek game yang sudah ada di scene Anda. Untuk melakukannya, pilih objek dan klik Add Component di Inspector. Pilih Rendering > Camera untuk menambahkan komponen kamera. Ingatlah bahwa menambahkan komponen kamera ke objek yang ada tidak secara otomatis memberi Anda layer suar atau komponen pendengar audio.

Beberapa Pendengar Audio

Seperti disebutkan sebelumnya, sebuah scene hanya dapat memiliki satu pendengar audio. Dalam versi Unity yang lebih lama, memiliki dua atau lebih pendengar akan menyebabkan kesalahan dan mencegah scene berjalan. Sekarang, jika Anda memiliki banyak pendengar, Anda hanya melihat pesan peringatan, meskipun audio mungkin tidak terdengar dengan benar. Topik ini dibahas secara rinci di Jam 21, “Audio.”

Bekerja dengan Banyak Kamera

Cara terbaik untuk memahami bagaimana banyak kamera berinteraksi adalah dengan berlatih menggunakannya. Latihan ini berfokus pada manipulasi kamera dasar:

1. Buat proyek atau scene baru dan tambahkan dua kubus. Letakkan kubus di (-2, 1, -5) dan (2, 1, -5).
2. Pindahkan Kamera Utama ke (-3, 1, -8) dan ubah rotasinya ke (0, 45, 0).
3. Tambahkan kamera baru ke scene (dengan memilih GameObject > Camera) dan posisikan di (3, 1, -8). Ubah rotasinya menjadi (0, 315, 0). Pastikan untuk menonaktifkan pendengar audio untuk kamera dengan menghapus centang pada kotak di sebelah komponen. 4. Jalankan scene. Perhatikan bahwa kamera kedua adalah satu-satunya yang ditampilkan. Ini karena kamera kedua memiliki kedalaman yang lebih tinggi daripada Kamera Utama. Kamera Utama digambar ke layar terlebih dahulu, lalu kamera kedua digambar di atasnya. Ubah kedalaman kamera utama menjadi 1 lalu jalankan scene lagi. Perhatikan bahwa Kamera Utama sekarang satu-satunya yang terlihat.

Split-Screen dan Picture-in-Picture

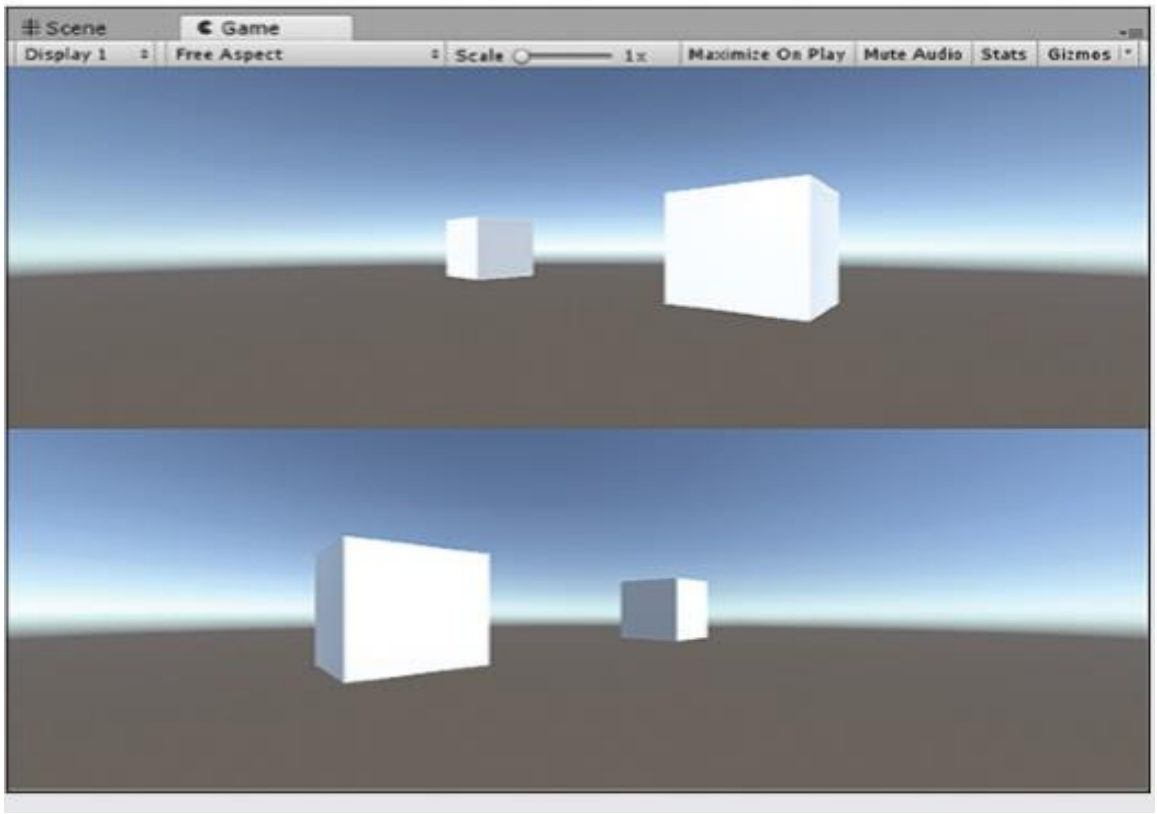
Seperti yang Anda lihat di awal jam ini, memiliki beberapa kamera dalam satu scene tidak banyak berguna jika salah satunya hanya menarik yang lain. Di bagian ini, Anda akan

belajar menggunakan properti Normalized View Port Rect untuk mencapai efek layar terpisah dan gambar-dalam-gambar. Port tampilan yang dinormalisasi pada dasarnya memperlakukan layar sebagai persegi panjang sederhana. Sudut kiri bawah persegi panjang adalah (0, 0), dan sudut kanan atas adalah (1, 1). Ini tidak berarti bahwa layar harus berbentuk persegi yang sempurna. Sebaliknya, pikirkan koordinat sebagai persentase dari ukuran. Jadi, koordinat 1 berarti 100%, dan koordinat 0,5 berarti 50%. Mengetahui hal ini, menempatkan kamera di layar menjadi mudah. Secara default, kamera memproyeksikan dari (0, 0) dengan lebar dan tinggi 1 (atau 100%). Ini menyebabkan mereka mengambil seluruh layar. Namun, jika Anda mengubah angka-angka itu, Anda akan mendapatkan efek yang berbeda.

Membuat Sistem Kamera Layar Terpisah

Latihan ini membahas pembuatan sistem kamera split-cell. Jenis sistem ini umum dalam permainan dua player di mana para player harus berbagi layar yang sama. Latihan ini dibangun di atas Coba Sendiri sebelumnya untuk beberapa kamera di awal jam ini. Ikuti langkah-langkah berikut ini:

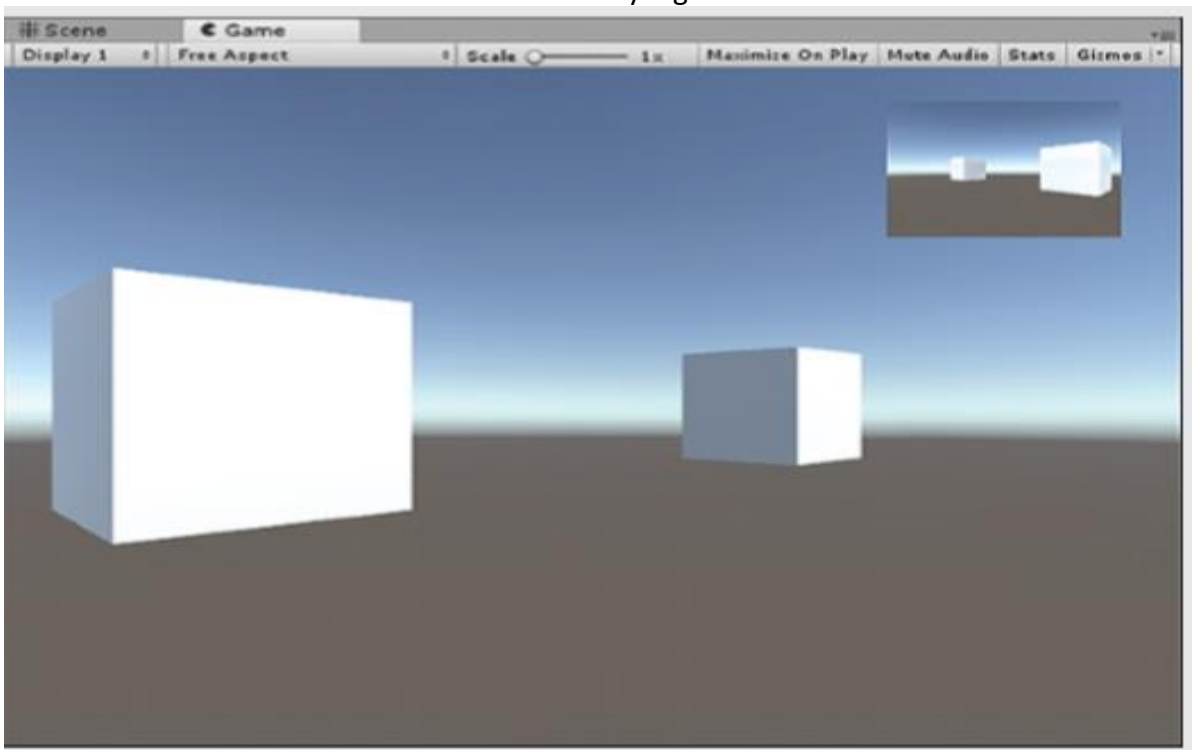
1. Buka project yang telah Anda buat sebelumnya
2. Pastikan Kamera Utama memiliki kedalaman -1. Pastikan bahwa properti X dan Y dari properti View Port Rect kamera keduanya 0. Atur properti W dan H masing-masing ke 1 dan .5 (yaitu, 100% lebar dan 50% tinggi).
3. Pastikan kamera kedua juga memiliki kedalaman -1. Atur properti X dan Y dari port tampilan ke (0, .5). Ini menyebabkan kamera mulai menggambar di tengah layar. Atur properti W dan H masing-masing menjadi 1 dan .5.
4. Jalankan scene dan perhatikan bahwa kedua kamera sekarang memproyeksikan pada layar secara bersamaan (lihat Gambar 4.6). Anda dapat membagi layar seperti ini sebanyak yang Anda inginkan.



Gambar 4.6 Efek split-cell.

Membuat Efek Gambar-dalam-Gambar Gambar-dalam-gambar biasanya digunakan untuk membuat efek seperti peta mini. Dengan efek ini, satu kamera menarik kamera lainnya di area tertentu. Latihan ini dibangun di atas Coba Sendiri sebelumnya untuk beberapa kamera di awal jam ini:

1. Buka proyek yang Anda buat di "Bekerja dengan Banyak Kamera"
2. Pastikan Kamera Utama memiliki kedalaman -1. Pastikan properti X dan Y dari properti Normalized View Port Rect kamera keduanya 0 dan properti W dan H keduanya 1.
3. Pastikan kedalaman kamera kedua adalah 0. Atur properti X dan Y dari port tampilan ke (.75, .75) dan atur nilai W dan H ke .2.
4. Jalankan scene. Perhatikan bahwa kamera kedua muncul di sudut kanan atas layar (lihat Gambar 4.7). Bereksperimenlah dengan pengaturan port tampilan yang berbeda untuk membuat kamera muncul di sudut yang berbeda.



Gambar 4.7 Efek gambar-dalam-gambar.

4.4 LAYERS

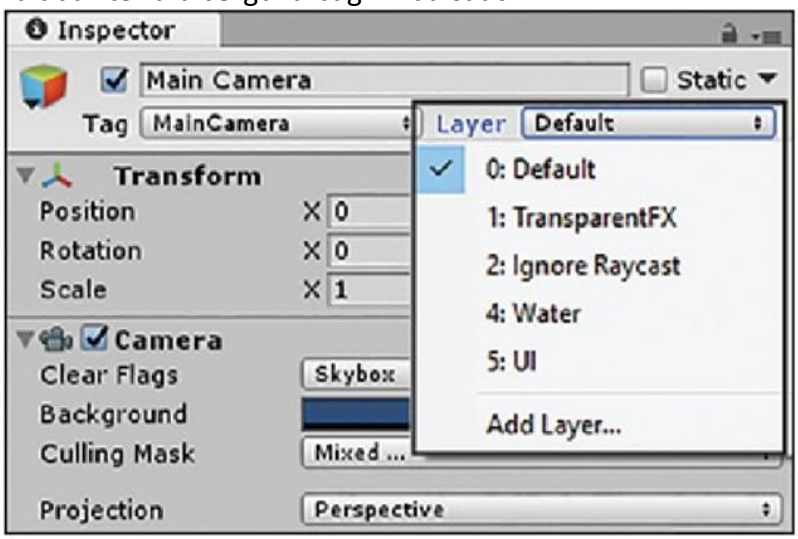
Seringkali sulit untuk mengatur banyak objek dalam sebuah proyek dan dalam sebuah scene. Terkadang Anda ingin item hanya dapat dilihat oleh kamera tertentu atau hanya diterangi oleh lampu tertentu. Terkadang Anda ingin collision hanya terjadi di antara jenis objek tertentu. Unity memungkinkan Anda untuk mengatur dengan menggunakan layer. default, ada 8 layer bawaan dan 24 layer untuk ditentukan pengguna.

Layer Overload

Menambahkan layer bisa menjadi cara yang bagus untuk mencapai perilaku kompleks tanpa melakukan banyak pekerjaan. Sebuah kata peringatan, meskipun: Jangan membuat layer untuk item kecuali Anda perlu. Terlalu sering, orang secara sewenang-wenang membuat layer saat menambahkan objek ke scene dengan pemikiran bahwa mereka mungkin membutuhkannya nanti. Pendekatan ini dapat menyebabkan mimpi buruk organisasi saat Anda mencoba mengingat untuk apa setiap layer dan apa fungsinya. Singkatnya, tambahkan layer saat Anda membutuhkannya. Jangan mencoba menggunakan layer hanya karena Anda bisa.

Bekerja dengan Layers

Setiap objek game dimulai pada layer Default. Artinya, objek tidak memiliki layer khusus untuk dimiliki, sehingga digabungkan dengan yang lainnya. Anda dapat dengan mudah menambahkan objek ke layer dalam tampilan Inspector. Dengan objek yang dipilih, klik drop-down Layer di Inspector dan pilih layer baru untuk objek yang akan menjadi bagiannya (lihat Gambar 4.7). Secara default, ada lima layer untuk dipilih: Default, TransparentFX, Abaikan Raycast, Air, dan UI. Anda dapat dengan aman mengabaikan sebagian besar dari ini untuk saat ini karena mereka tidak terlalu berguna bagi Anda saat ini



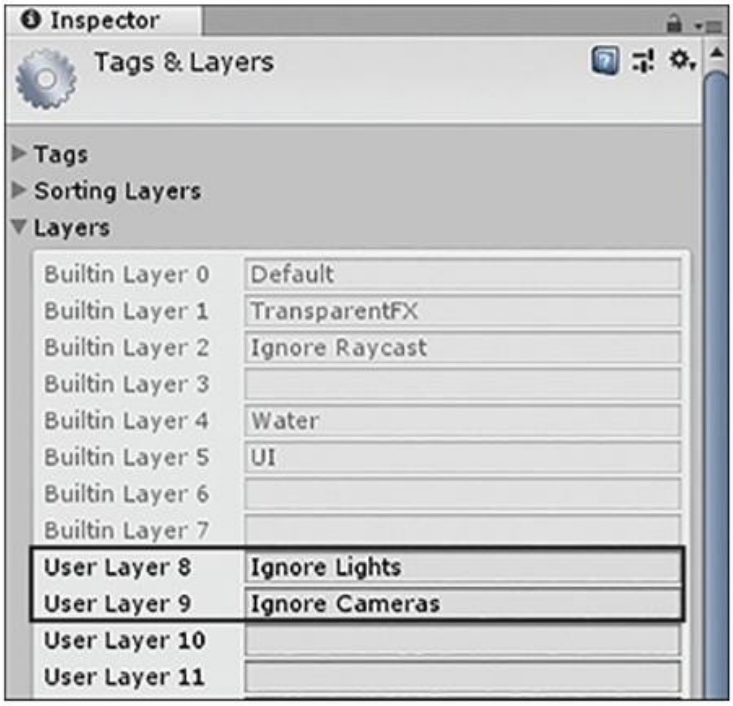
Gambar 4.7 Menu tarik-turun Layer.

Meskipun layer built-in saat ini tidak terlalu berguna bagi Anda, Anda dapat dengan mudah membukanya dengan tiga cara berbeda:

- Dengan objek yang dipilih, klik drop-down Layer dan pilih Add.
- Di menu di bagian atas editor, klik Edit > Pengaturan Proyek > Tag dan Layer.
- Klik pemilih Layers di toolbar scene dan pilih Edit Layers.

Gambar 4.8 Pemilih Layer di toolbar scene.

Di Pengelola Tag & Layer, klik di sebelah kanan salah satu layer pengguna untuk memberinya nama. Gambar 4.8 mengilustrasikan proses ini, menunjukkan dua layer baru yang ditambahkan. (Mereka ditambahkan untuk gambar ini, dan Anda tidak akan memilikinya kecuali Anda menambahkannya sendiri.)



Gambar 4.9 Menambahkan layer baru ke Tag & Layers Manager.

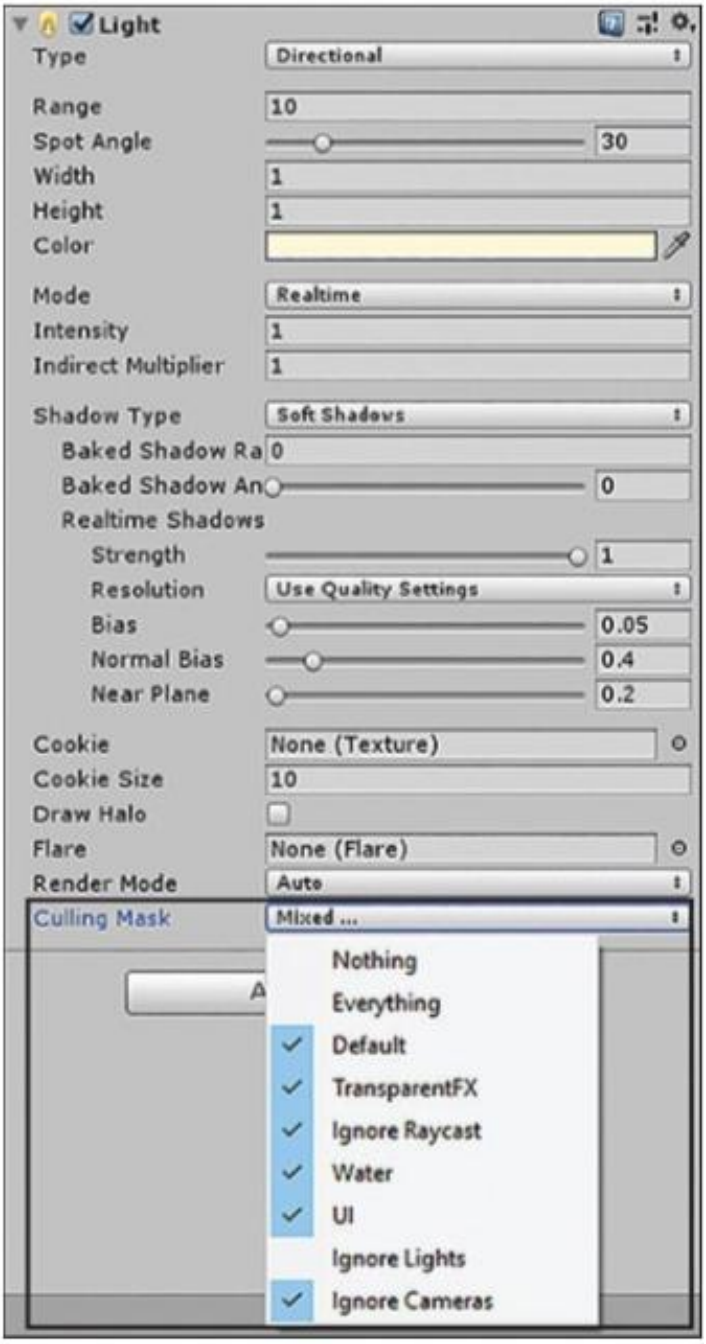
Menggunakan Layers

Ada banyak kegunaan layer. Kegunaan layer hanya dibatasi oleh apa yang dapat Anda pikirkan untuk dilakukan dengannya. Bagian ini mencakup empat penggunaan umum. Salah satu penggunaan umum layer adalah menyembunyikannya dari tampilan Scene. Dengan mengklik pemilih Layers di toolbar Scene view (lihat Gambar 5.9), Anda dapat memilih layer mana yang muncul di Scene view dan mana yang tidak. Secara default, scene diatur untuk menampilkan semuanya

Item Scene Tak Terlihat

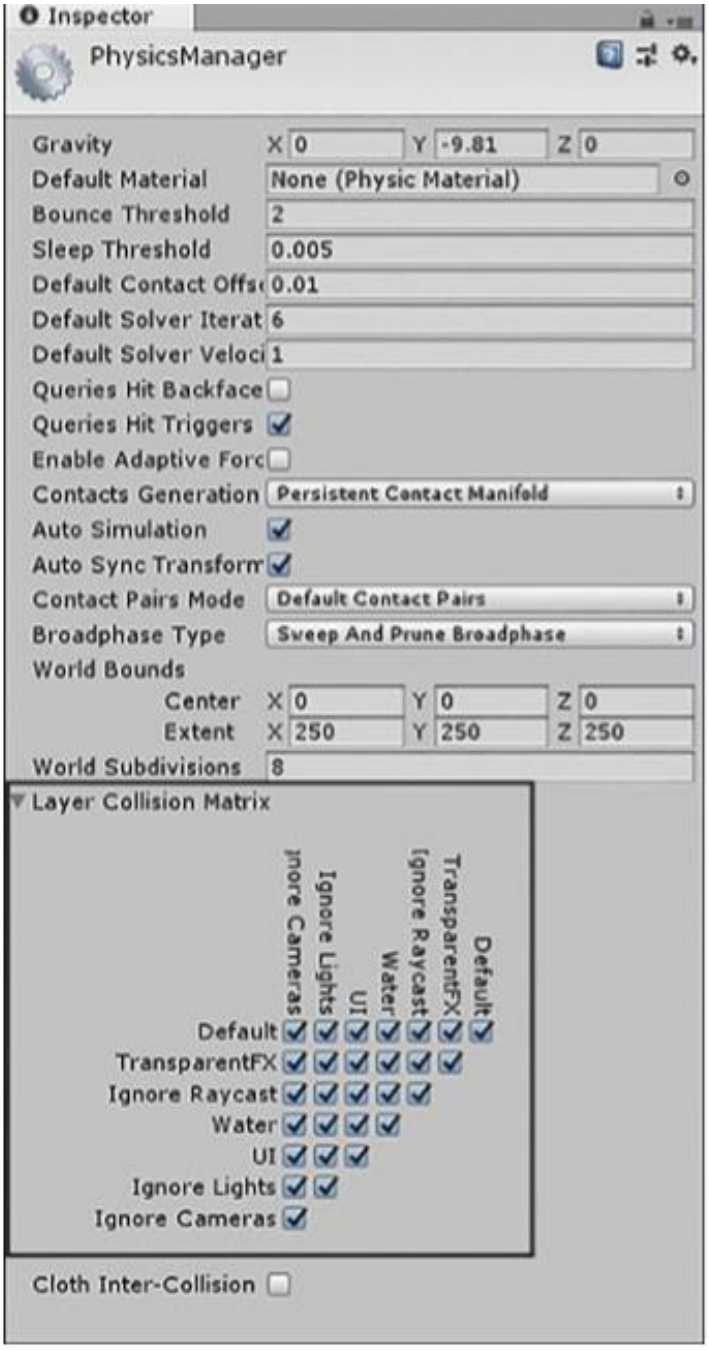
Satu kesalahan umum adalah secara tidak sengaja mengubah layer apa yang terlihat dalam tampilan Scene. Jika Anda tidak terbiasa dengan kemampuan untuk membuat layer tidak terlihat, ini bisa sangat membingungkan. Perhatikan saja bahwa jika suatu saat item tidak muncul dalam tampilan Scene ketika seharusnya, Anda harus memeriksa pemilih Layers untuk memastikan bahwa itu diatur untuk menampilkan semuanya.

Cara kedua untuk menggunakan layer adalah dengan mengecualikan objek agar tidak diterangi oleh cahaya. Ini berguna jika Anda membuat antarmuka pengguna khusus, membuat sistem bayangan, atau menggunakan sistem pencahayaan yang kompleks. Untuk mencegah layer diterangi oleh cahaya, pilih cahaya dan kemudian, dalam tampilan Inspector, klik properti Culling mask dan batalkan pilihan layer yang ingin Anda abaikan.



Gambar 4.10 Properti Culling mask.

Hal ketiga yang dapat Anda gunakan untuk layer adalah memberi tahu Unity objek fisika mana yang berinteraksi satu sama lain. Anda dapat memilih ini di Edit > Project Settings > Physics dan cari Layer Collision Matrix



Gambar 4.11 Matriks Collision Layer.

Hal terakhir yang perlu diketahui tentang layer adalah Anda dapat menggunakannya untuk menentukan apa yang dapat dan tidak dapat dilihat oleh kamera. Ini berguna jika Anda ingin membuat efek visual khusus menggunakan beberapa kamera untuk satu penampil. Sama seperti yang dijelaskan sebelumnya, untuk mengabaikan layer, cukup klik menu tarik-turun Culling Mask pada komponen kamera dan batalkan pilihan apa pun yang tidak ingin Anda tampilkan.

Mengabaikan Lampu dan Kamera

Ikuti langkah-langkah ini untuk bekerja secara singkat dengan layer untuk lampu dan kamera:

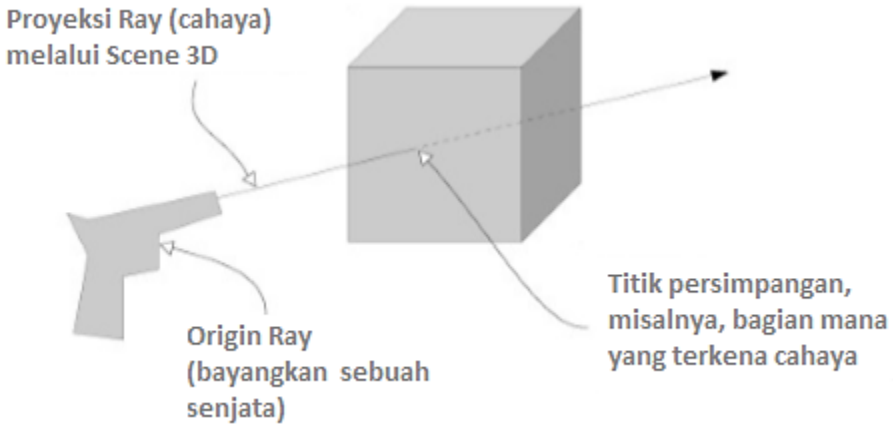
1. Buat proyek atau scene baru. Tambahkan dua kubus ke scene dan posisikan di (2, 1, -5) dan (2, 1, -5).

2. Masuk ke Tag & Layers Manager menggunakan salah satu dari tiga metode yang terdaftar sebelumnya dan tambahkan dua layer baru: Abaikan Lampu dan Abaikan Kamera
3. Pilih salah satu kubus dan tambahkan ke layer Ignore Lights. Pilih kubus lainnya dan tambahkan ke layer Abaikan Kamera.
4. Pilih Directional Light di scene, dan di properti Culling Mask, batalkan pilihan layer Ignore Lights. Perhatikan bahwa sekarang hanya satu kubus yang menyala. Yang lain telah diabaikan karena layernya.
5. Pilih Main Camera dan hapus layer Ignore Cameras dari properti Culling Mask. Jalankan scene dan perhatikan bahwa hanya satu kubus yang tidak diterangi yang muncul. Yang lain telah diabaikan oleh kamera.

4.5 RAYTRACING - APA ITU RAYCASTING?

Seperti namanya, raycasting adalah saat Anda memancarkan sinar ke dalam scene. Jelas, bukan? Nah, oke, jadi apa sebenarnya sinar itu?

Definisi Ray adalah garis imajiner atau tidak terlihat dalam scene yang dimulai pada beberapa titik asal dan memanjang ke arah tertentu. Raycasting adalah saat Anda membuat sinar dan kemudian menentukan apa yang memotong sinar itu; Gambar 3.1 mengilustrasikan konsep tersebut. Pertimbangkan apa yang terjadi ketika Anda menembakkan peluru dari pistol: peluru dimulai pada posisi pistol dan kemudian terbang ke depan dalam garis lurus sampai mengenai sesuatu. Sinar dianalogikan dengan jalur peluru, dan pancaran sinar analog dengan menembakkan peluru dan melihat di mana ia mengenai.



Gambar 4.12 Raycasting adalah garis imajiner, dan raycasting adalah menemukan di mana garis itu berpotongan.

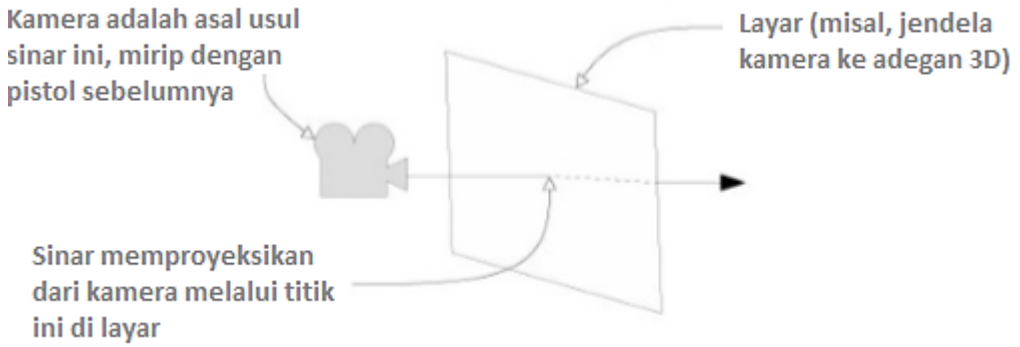
Seperti yang dapat Anda bayangkan, matematika di balik raycasting sering kali menjadi rumit. Tidak hanya sulit untuk menghitung perpotongan garis dengan bidang 3D, tetapi Anda juga perlu melakukannya untuk semua poligon dari semua objek mesh dalam scene (ingat, objek mesh adalah visual 3D yang dibuat dari banyak garis yang terhubung dan bentuk). Untungnya, Unity menangani matematika sulit di balik raycasting, tetapi Anda masih harus khawatir tentang masalah tingkat yang lebih tinggi seperti dari mana ray dipancarkan dan mengapa.

Dalam proyek ini jawaban atas pertanyaan terakhir (mengapa) adalah dengan mensimulasikan peluru yang ditembakkan ke scene. Untuk Sniper pertama, sinar biasanya dimulai pada posisi kamera dan kemudian meluas melalui bagian tengah tampilan kamera. Dengan kata lain, Anda memeriksa objek langsung di depan kamera; Unity menyediakan perintah untuk membuat tugas itu sederhana. Mari kita lihat perintah-perintah ini.

Menggunakan perintah ScreenPointToRay untuk pemotretan

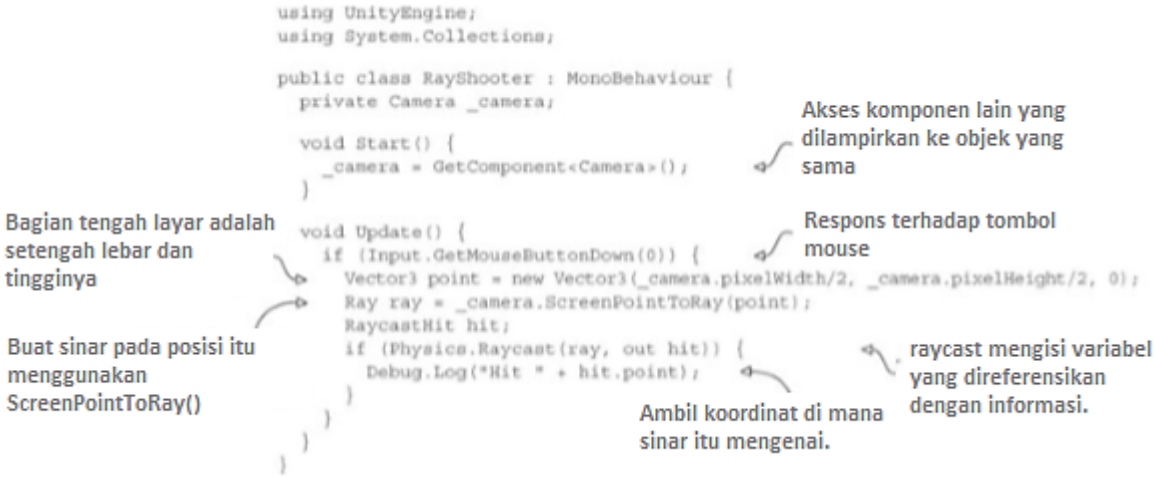
Anda akan menerapkan pemotretan dengan memproyeksikan sinar yang dimulai dari kamera dan meluas ke depan melalui pusat tampilan. Memproyeksikan sinar melalui pusat tampilan kamera adalah kasus khusus dari tindakan yang disebut sebagai memetik mouse.

Definisi Memilih mouse adalah tindakan memilih tempat dalam scene 3D langsung di bawah kursor mouse. Unity menyediakan metode ScreenPointToRay() untuk melakukan tindakan ini. Gambar 3.2 mengilustrasikan apa yang terjadi. Metode ini menciptakan sinar yang dimulai pada kamera dan diproyeksikan pada sudut yang melewati koordinat layar yang diberikan. Biasanya koordinat posisi mouse digunakan untuk memilih mouse, tetapi untuk pemotretan orang pertama digunakan bagian tengah layar. Setelah Anda memiliki sinar, itu dapat diteruskan ke metode Physics.Raycast() untuk melakukan raycasting menggunakan sinar itu.



Gambar 4.13 ScreenPointToRay() memproyeksikan sinar dari kamera melalui koordinat layar yang diberikan.

Mari kita menulis beberapa kode yang menggunakan metode yang baru saja kita bahas. Di Unity buat skrip C# baru, lampirkan skrip itu ke kamera (bukan objek pemutar), lalu tulis kode dari daftar berikutnya di dalamnya.



Gambar 4.14 Skrip RayShooter untuk dilampirkan ke kamera

Anda harus mencatat beberapa hal dalam daftar kode ini. Pertama, komponen kamera diambil di Start(), seperti CharacterController di bab sebelumnya. Kemudian sisa kode dimasukkan ke dalam Update() karena perlu memeriksa mouse berulang kali, bukan hanya satu kali. Metode Input.GetMouseButtonDown() mengembalikan nilai true atau false tergantung pada apakah mouse telah diklik, jadi menempatkan perintah tersebut dalam kondisi berarti kode terlampir hanya berjalan ketika mouse telah diklik. Anda ingin menembak

ketika player mengklik mouse; maka cek kondisional tombol mouse. Sebuah vektor dibuat untuk menentukan koordinat layar untuk sinar (ingat bahwa vektor adalah beberapa angka terkait yang disimpan bersama). Nilai pixelWidth dan pixelHeight kamera memberi Anda ukuran layar, jadi membagi nilai tersebut menjadi dua memberi Anda bagian tengah layar. Meskipun koordinat layar adalah 2D, dengan hanya komponen horizontal dan vertikal dan tanpa kedalaman, Vector3 dibuat karena ScreenPointToRay() memerlukan tipe data tersebut (mungkin karena penghitungan sinar melibatkan aritmatika pada vektor 3D). ScreenPointToRay() dipanggil dengan set koordinat ini, menghasilkan objek Ray (objek kode, yaitu, bukan objek game; keduanya terkadang membingungkan).

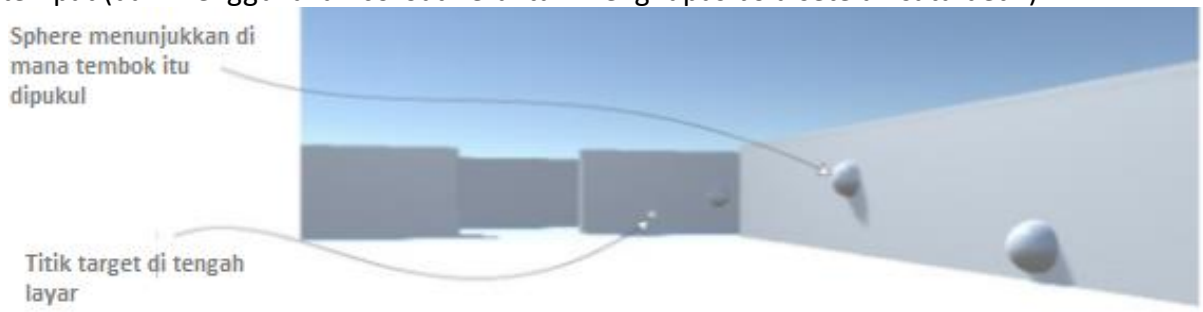
Sinar tersebut kemudian diteruskan ke metode Raycast(), tetapi itu bukan satu-satunya objek yang dilewatkan. Ada juga struktur data RaycastHit; RaycastHit adalah kumpulan informasi tentang perpotongan sinar, termasuk di mana perpotongan itu terjadi dan objek apa yang berpotongan. Sintaks C# out memastikan bahwa struktur data yang dimanipulasi di dalam perintah adalah objek yang sama yang ada di luar perintah, sebagai lawan dari objek yang menjadi salinan terpisah dalam cakupan fungsi yang berbeda.

Akhirnya kode memanggil metode Physics.Raycast(). Metode ini memeriksa persimpangan dengan sinar yang diberikan, mengisi data tentang persimpangan, dan mengembalikan true jika sinar itu mengenai sesuatu. Karena nilai Boolean dikembalikan, metode ini dapat dimasukkan ke dalam pemeriksaan bersyarat, sama seperti Anda menggunakan Input.GetMouseButtonDown() sebelumnya.

Untuk saat ini kode memancarkan pesan konsol untuk menunjukkan saat terjadi persimpangan. Pesan konsol ini menampilkan koordinat 3D dari titik di mana sinar itu mengenai (nilai XYZ yang kita bahas di bab 2). Tetapi mungkin sulit untuk memvisualisasikan di mana tepatnya sinar itu mengenai; demikian pula, sulit untuk mengetahui di mana pusat layar berada (yaitu, di mana sinar menembus). Mari tambahkan indikator visual untuk mengatasi kedua masalah tersebut.

Menambahkan indikator visual untuk membidik dan memukul

Langkah kita selanjutnya adalah menambahkan dua jenis indikator visual: titik bidik di tengah layar, dan tanda di tempat di mana sinar itu mengenai. Untuk Sniper pertama yang terakhir biasanya lubang peluru, tetapi untuk saat ini Anda akan meletakkan bola kosong di tempat (dan menggunakan coroutine untuk menghapus bola setelah satu detik).



Gambar 4.15 Memotret berulang kali setelah menambahkan indikator visual untuk membidik dan memukul

Coroutine adalah cara khusus Unity untuk menangani tugas yang dijalankan secara bertahap dari waktu ke waktu, berbeda dengan cara sebagian besar fungsi membuat program menu nggu hingga selesai. Pertama mari kita tambahkan indikator untuk menandai di mana sinar itu mengenai. Listing 3.2 menunjukkan script setelah melakukan penambahan ini. Berlari di sekitar lokasi syuting; cukup menyenangkan melihat indikator bola!

```
using UnityEngine;
using System.Collections;

public class RayShooter : MonoBehaviour {
  private Camera _camera;

  void Start() {
    _camera = GetComponent<Camera>();
  }

  void Update() {
    if (Input.GetMouseButtonDown(0)) {
      Vector3 point = new Vector3(_camera.pixelWidth/2, _camera.pixelHeight/2, 0);
      Ray ray = _camera.ScreenPointToRay(point);
      RaycastHit hit;
      if (Physics.Raycast(ray, out hit)) {
        StartCoroutine(SphereIndicator(hit.point));

      }
    }
  }

  private IEnumerator SphereIndicator(Vector3 pos) {
    GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    sphere.transform.position = pos;

    yield return new WaitForSeconds(1);

    Destroy(sphere);
  }
}
```

Fungsi ini sebagian besar adalah kode raycasting yang sama

Luncurkan coroutine sebagai tanggapan atas hit

Coroutine menggunakan fungsi IEenumerator

Kata kunci hasil memberi tahu coroutine tempat jeda

Hapus GameObject ini dan bersihkan memorinya

Gambar 4.16 Skrip RayShooter dengan indikator bola ditambahkan

Metode baru adalah SphereIndicator(), ditambah modifikasi satu baris dalam metode Update() yang ada. Metode ini membuat bola pada suatu titik dalam scene dan kemudian menghapus bola itu sedetik kemudian. Memanggil SphereIndicator() dari kode raycasting memastikan bahwa akan ada indikator visual yang menunjukkan dengan tepat di mana sinar itu mengenai. Fungsi ini didefinisikan dengan IEnumerator, dan tipe itu terikat dengan konsep coroutine.

Secara teknis, coroutine tidak asinkron (operasi asinkron tidak menghentikan sisa kode untuk berjalan; pikirkan untuk mengunduh gambar dalam skrip situs web), tetapi melalui penggunaan enumerator yang cerdas, Unity membuat coroutine berperilaku mirip dengan fungsi asinkron . Saus rahasia dalam coroutine adalah kata kunci hasil; kata kunci itu menyebabkan coroutine berhenti sementara, mengembalikan aliran program dan mengambil lagi dari titik itu di bingkai berikutnya. Dengan cara ini, coroutine tampaknya berjalan di latar belakang program, melalui siklus berulang yang berjalan di tengah jalan dan kemudian kembali ke program lainnya.

Sesuai dengan namanya, StartCoroutine() menggerakkan coroutine. Setelah coroutine dimulai, coroutine terus berjalan sampai fungsi selesai; itu hanya berhenti di sepanjang jalan. Perhatikan poin halus namun signifikan bahwa metode yang diteruskan ke StartCoroutine() memiliki serangkaian tanda kurung yang mengikuti namanya: sintaks ini berarti Anda memanggil fungsi itu, bukan meneruskan namanya. Fungsi yang dipanggil berjalan hingga mencapai perintah hasil, di mana fungsi berhenti.

SphereIndicator() membuat bola pada titik tertentu, berhenti sejenak untuk pernyataan hasil, lalu menghancurkan bola setelah coroutine dilanjutkan. Panjang jeda

dikendalikan oleh nilai yang dikembalikan pada hasil. Beberapa jenis nilai pengembalian yang berbeda berfungsi di coroutine, tetapi yang paling mudah adalah mengembalikan jangka waktu tertentu untuk menunggu. Mengembalikan WaitForSeconds(1) menyebabkan coroutine berhenti sejenak selama satu detik. Buat bola, jeda selama satu detik, lalu hancurkan bola: urutan itu menyiapkan indikator visual sementara.

```
...
void Start() {
  _camera = GetComponent<Camera>();

  Cursor.lockState = CursorLockMode.Locked;
  Cursor.visible = false;
}

void OnGUI() {
  int size = 12;
  float posX = _camera.pixelWidth/2 - size/4;
  float posY = _camera.pixelHeight/2 - size/2;
  GUI.Label(new Rect(posX, posY, size, size), "*");
}
...
```

Sembunyikan mouse crusor di tengah layar

Command GUI.Label() menampilkan teks di layar

Gambar 4.17 Daftar Indikator visual untuk membidik

Metode baru lainnya telah ditambahkan ke kelas RayShooter, yang disebut OnGUI(). Unity hadir dengan sistem antarmuka pengguna (UI) dasar dan yang lebih canggih; karena sistem dasar memiliki banyak keterbatasan, kami akan membangun UI lanjutan yang lebih fleksibel di bab mendatang, tetapi untuk saat ini menampilkan titik di tengah layar menggunakan UI dasar jauh lebih mudah. Sama seperti Start() dan Update(), setiap MonoBehaviour secara otomatis merespons metode OnGUI(). Fungsi itu menjalankan setiap bingkai tepat setelah scene 3D dirender, menghasilkan semua yang digambar selama OnGUI() muncul di atas scene 3D (bayangkan stiker diterapkan pada lukisan lanskap).

Definisi Render adalah tindakan komputer menggambar piksel scene 3D. Meskipun scene ditentukan menggunakan koordinat XYZ, tampilan sebenarnya pada monitor Anda adalah kisi 2D piksel berwarna. Jadi untuk menampilkan scene 3D, komputer perlu menghitung warna semua piksel dalam kisi 2D; menjalankan algoritma itu disebut sebagai rendering.

Di dalam OnGUI() kode mendefinisikan koordinat 2D untuk tampilan (digeser sedikit untuk memperhitungkan ukuran label) dan kemudian memanggil GUI.Label(). Metode itu menampilkan label teks; karena string yang diteruskan ke label adalah tanda bintang (*), Anda berakhir dengan karakter yang ditampilkan di tengah layar. Sekarang jauh lebih mudah untuk membidik di game FPS kami yang baru lahir! Selalu ingat bahwa Anda dapat menekan Esc untuk membuka kunci kursor mouse. Saat kursor mouse terkunci, tidak mungkin mengklik tombol Play dan menghentikan permainan.

4.6 MEMBUAT SKRIP TARGET REAKTIF

Mampu menembak semuanya baik dan bagus, tetapi saat ini player tidak memiliki apa pun untuk ditembak. Kami akan membuat objek target dan memberikan skrip yang akan merespons jika dipukul. Atau lebih tepatnya, kami akan sedikit memodifikasi kode pemotretan untuk memberi tahu target saat terkena, dan kemudian skrip pada target akan bereaksi saat diberi tahu.

Menentukan apa yang dipukul

Pertama, Anda perlu membuat objek baru untuk menembak. Buat objek kubus baru (GameObject > 3D Object > Cube) lalu scalekan secara vertikal dengan mengatur scale Y

menjadi 2 dan biarkan X dan Z pada 1. Posisikan objek baru pada 0, 1, 0 untuk meletakkannya di lantai di tengah ruangan, dan beri nama objek Musuh. Buat skrip baru bernama ReactiveTarget dan lampirkan ke kotak yang baru dibuat. Anda akan segera menulis kode untuk skrip ini, tetapi biarkan default untuk saat ini; Anda hanya membuat file skrip karena daftar kode berikutnya mengharuskannya ada untuk dikompilasi. Kembali ke RayShooter.cs dan ubah kode raycasting sesuai dengan daftar berikut. Jalankan kode baru dan tembak target baru; pesan debug muncul di konsol alih-alih indikator bola di scene.

```
...
if (Physics.Raycast(ray, out hit)) {
  GameObject hitObject = hit.transform.gameObject;
  ReactiveTarget target = hitObject.GetComponent<ReactiveTarget>();
  if (target != null) {
    Debug.Log("Target hit");
  } else {
    StartCoroutine(SphereIndicator(hit.point));
  }
}
...
```

Ambil kembali objek yang terkena sinar

Periksa komponen ReactiveTarget pada objek

Gambar 4.18 Daftar Mendeteksi apakah objek target terkena.

Perhatikan bahwa Anda mengambil objek dari RaycastHit, seperti koordinat yang diambil untuk indikator bola. Secara teknis, informasi hit tidak mengembalikan hit objek game; ini menunjukkan hit komponen Transform. Anda kemudian dapat mengakses gameObject sebagai properti transformasi.

Kemudian, Anda menggunakan metode GetComponent() pada objek untuk memeriksa apakah itu target reaktif (yaitu, jika skrip ReactiveTarget terpasang). Seperti yang Anda lihat sebelumnya, metode itu mengembalikan komponen dari tipe tertentu yang dilampirkan ke GameObject. Jika tidak ada komponen jenis itu yang dilampirkan ke objek, maka GetComponent() tidak akan mengembalikan apa pun. Anda memeriksa apakah null dikembalikan dan menjalankan kode yang berbeda dalam setiap kasus. Jika objek yang terkena adalah target reaktif, kode memancarkan pesan debug alih-alih memulai coroutine untuk indikator bola. Sekarang mari kita beri tahu objek target tentang pukulan sehingga dapat bereaksi.

Peringatkan target yang terkena

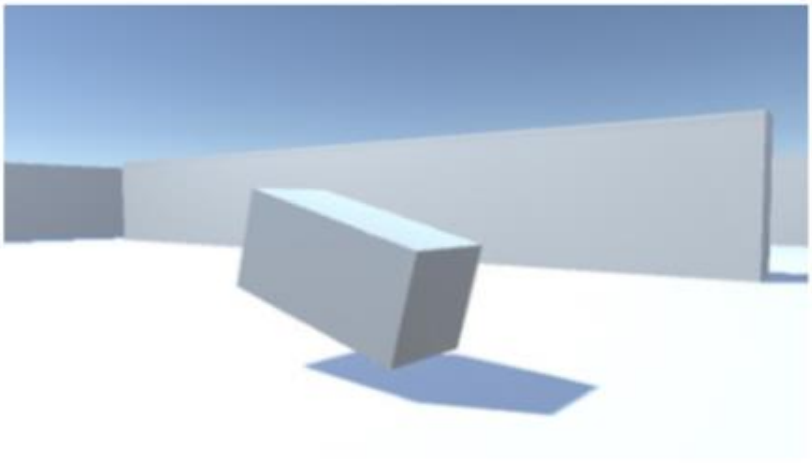
Semua yang diperlukan dalam kode adalah perubahan satu baris, seperti yang ditunjukkan pada daftar berikut.

```
...
if (target != null) {
  target.ReactToHit();
} else {
  StartCoroutine(SphereIndicator(hit.point));
}
...
```

Panggil metode target alih-alih hanya memancarkan pesan debug

Gambar 4.19 Daftar Mengirim pesan ke objek target

Sekarang kode pemotretan memanggil metode target, jadi mari kita tulis metode target itu. Dalam skrip ReactiveTarget, tulis kode dari daftar berikutnya. Objek target akan jatuh dan menghilang saat Anda menembaknya; lihat gambar 4.20.



Gambar 4.20 Objek target jatuh saat dipukul

```
using UnityEngine;
using System.Collections;

public class ReactiveTarget : MonoBehaviour {

  public void ReactToHit() {
    StartCoroutine(Die());
  }

  private IEnumerator Die() {
    this.transform.Rotate(-75, 0, 0);

    yield return new WaitForSeconds(1.5f);

    Destroy(this.gameObject);
  }
}
```

Metode yang disebut oleh skrip pemotretan

Jatuhkan musuh, tunggu 1,5 detik lalu hancurkan musuh

Objek dapat menghancurkan dirinya sendiri seperti objek terpisah

Gambar 4.21 Daftar Skrip ReactiveTarget yang mati saat dipukul

Sebagian besar kode ini seharusnya sudah Anda kenal dari skrip sebelumnya, jadi kami hanya akan membahasnya secara singkat. Pertama, Anda mendefinisikan metode ReactToHit(), karena itulah nama metode yang dipanggil dalam skrip pemotretan. Metode ini memulai coroutine yang mirip dengan kode indikator bola dari sebelumnya; perbedaan utamanya adalah ia beroperasi pada objek skrip ini daripada membuat objek terpisah. Ekspresi seperti this.gameObject merujuk ke GameObject tempat skrip ini dilampirkan (dan kata kunci this adalah opsional, sehingga kode dapat merujuk ke gameObject tanpa apa pun di depannya).

Transformasi diterapkan secara instan, tetapi Anda mungkin lebih suka melihat gerakan saat objek terguling. Setelah Anda mulai mencari topik yang lebih lanjut di luar buku ini, Anda mungkin ingin mencari tween, sistem yang digunakan untuk membuat objek bergerak dengan lancar seiring waktu. Baris kedua dari metode ini menggunakan kata kunci hasil yang sangat penting bagi coroutine, menjeda fungsi di sana dan mengembalikan jumlah detik untuk menunggu sebelum melanjutkan. Akhirnya, objek game menghancurkan dirinya sendiri di baris terakhir fungsi. Destroy(this.gameObject) dipanggil setelah waktu tunggu, seperti kode yang disebut Destroy(sphere) sebelumnya.

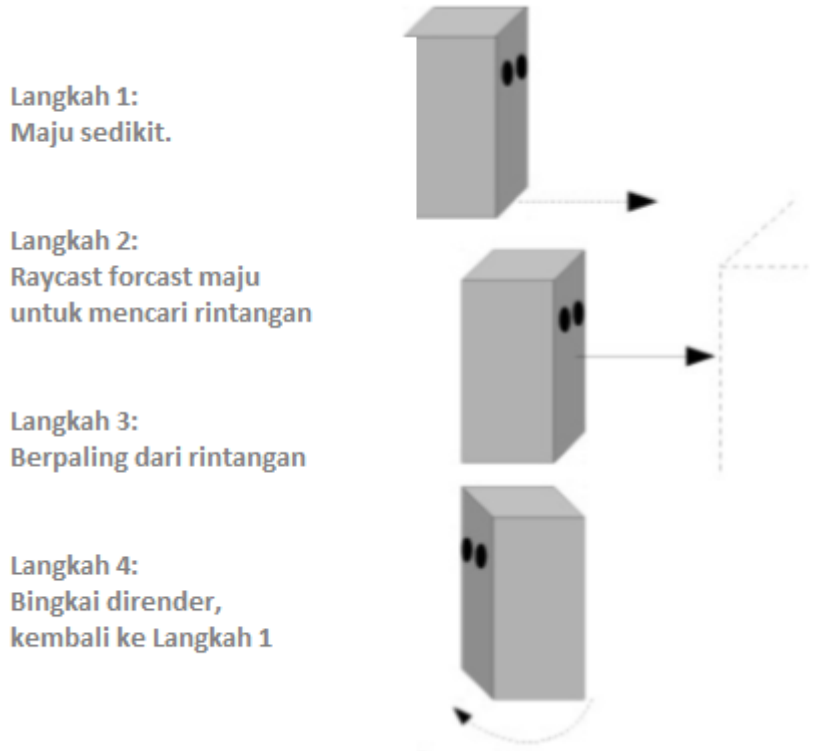
Pastikan untuk memanggil Destroy() di this.gameObject dan bukan hanya ini! Jangan bingung antara keduanya; this hanya merujuk ke komponen skrip ini, sedangkan this.gameObject merujuk ke objek yang dilampirkan skrip. Target sekarang bereaksi saat ditembak; Bagus! Tapi itu tidak melakukan hal lain sendiri, jadi mari tambahkan lebih banyak perilaku untuk membuat target ini menjadi karakter musuh yang tepat.

AI pengembara dasar

Target statis tidak terlalu menarik, jadi mari kita menulis kode yang akan membuat musuh berkeliaran. Kode untuk berkeliaran adalah contoh AI yang paling sederhana; kecerdasan buatan (AI) mengacu pada entitas yang dikendalikan komputer. Dalam hal ini entitas adalah musuh dalam sebuah game, tetapi bisa juga berupa robot di dunia nyata, atau suara yang bermain catur, misalnya.

Membuat diagram cara kerja AI dasar

Ada sejumlah pendekatan berbeda untuk AI (serius, kecerdasan buatan adalah bidang penelitian utama bagi ilmuwan komputer), tetapi untuk tujuan kami, kami akan tetap menggunakan pendekatan sederhana. Saat Anda menjadi lebih berpengalaman dan permainan Anda menjadi lebih canggih, Anda mungkin ingin menjelajahi berbagai pendekatan untuk AI.



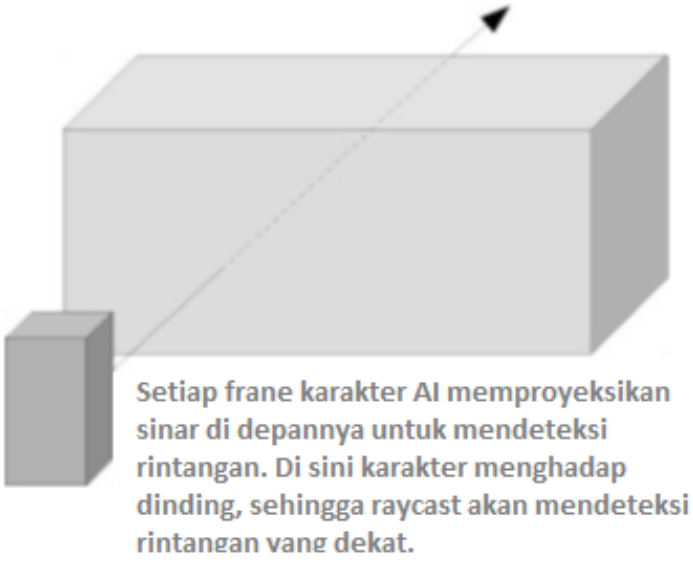
Gambar 4.22 Kecerdasan buatan

Kode sebenarnya akan terlihat cukup familiar, karena itu menggerakkan musuh ke depan menggunakan perintah yang sama seperti menggerakkan player ke depan. Kode AI juga akan menggunakan raycasting, mirip dengan tetapi dalam konteks yang berbeda dari pemotretan.

"Melihat" rintangan dengan raycast

Seperti yang Anda lihat di pengantar bab ini, raycasting adalah teknik yang berguna untuk sejumlah tugas dalam simulasi 3D. Salah satu tugas yang mudah dipahami adalah memotret, tetapi raycasting tempat lain yang berguna adalah untuk memindai di sekitar scene. Mengingat bahwa pemindaian di sekitar scene adalah langkah dalam kode AI, itu berarti raycasting digunakan dalam kode AI.

Sebelumnya Anda membuat sinar yang berasal dari kamera, karena dari sanalah player melihat; kali ini Anda akan membuat sinar yang berasal dari musuh. Sinar pertama ditembakkan melalui bagian tengah layar, tetapi kali ini sinar akan menembak ke depan di depan karakter; Gambar 3.6 mengilustrasikan hal ini. Kemudian seperti kode penembakan yang menggunakan informasi RaycastHit untuk menentukan apakah sesuatu terkena dan di mana, kode AI akan menggunakan informasi RaycastHit untuk menentukan apakah ada sesuatu di depan musuh dan, jika demikian, seberapa jauh.



Gambar 4.23 Menggunakan raycasting untuk "melihat" rintangan

Satu perbedaan antara raycasting untuk pemotretan dan raycasting untuk AI adalah radius sinar yang terdeteksi. Untuk pemotretan sinar diperlakukan sebagai sangat tipis, tetapi untuk AI sinar akan diperlakukan sebagai memiliki penampang yang besar; dalam hal kode, ini berarti menggunakan metode SphereCast() bukan Raycast(). Alasan perbedaan ini adalah pelurunya kecil, sedangkan untuk memeriksa rintangan di depan karakter kita perlu memperhitungkan lebar karakter. Buat skrip baru bernama WanderingAI, lampirkan itu ke objek target (bersama skrip ReactiveTarget), dan tulis kode dari daftar berikutnya. Mainkan scene sekarang dan Anda akan melihat musuh berkeliaran di sekitar ruangan; Anda masih bisa menembak target dan bereaksi dengan cara yang sama seperti sebelumnya.

Gambar 4.24 Skrip dasar WanderingAI

Daftar tersebut menambahkan beberapa variabel untuk mewakili kecepatan gerakan dan dari seberapa jauh untuk bereaksi terhadap rintangan. Kemudian metode Translate() ditambahkan dalam metode Update() untuk bergerak maju terus menerus (termasuk penggunaan deltaTime untuk pergerakan frame rate-independen). Di Update() Anda juga akan melihat kode raycasting yang sangat mirip dengan skrip pemotretan sebelumnya; sekali lagi, teknik raycasting yang sama digunakan di sini untuk melihat alih-alih memotret. Sinar dibuat menggunakan posisi dan arah musuh, bukan menggunakan kamera.

Seperti yang telah dijelaskan sebelumnya, perhitungan raycasting dilakukan dengan menggunakan metode Physics.SphereCast(). Metode ini mengambil parameter radius untuk menentukan seberapa jauh di sekitar sinar untuk mendeteksi persimpangan, tetapi dalam segala hal itu persis sama dengan Physics.Raycast(). Kesamaan ini termasuk bagaimana perintah mengisi informasi hit, memeriksa persimpangan seperti sebelumnya, dan menggunakan properti jarak untuk memastikan untuk bereaksi hanya ketika musuh mendekati rintangan (sebagai lawan dari dinding di seberang ruangan).

Ketika musuh memiliki rintangan terdekat tepat di depannya, kode memutar karakter dalam jumlah semi-acak ke arah yang baru. Saya katakan "semi-acak" karena nilainya dibatasi pada nilai minimum dan maksimum yang masuk akal untuk situasi ini. Secara khusus, kami menggunakan metode Random.Range() yang disediakan Unity untuk memperoleh nilai acak di antara batasan. Dalam hal ini batasannya hanya sedikit di luar belokan kiri atau kanan yang tepat, memungkinkan karakter untuk berbelok secukupnya untuk menghindari rintangan.

Melacak keadaan karakter

Satu keanehan dari perilaku saat ini adalah musuh terus bergerak maju setelah jatuh karena dipukul. Itu karena sekarang metode Translate() menjalankan setiap frame apa pun yang terjadi. Mari kita buat sedikit penyesuaian pada kode untuk melacak apakah karakter itu hidup atau tidak—atau dengan cara lain (lebih teknis), kita ingin melacak status karakter "hidup". Membuat kode melacak dan merespons secara berbeda terhadap keadaan objek saat ini adalah pola kode umum di banyak bidang pemrograman, bukan hanya AI. Implementasi yang lebih canggih dari pendekatan ini disebut sebagai mesin negara, atau bahkan mungkin mesin negara yang terbatas.

Definisi Finite state machine (FSM) adalah struktur kode di mana status objek saat ini dilacak, transisi yang terdefinisi dengan baik ada di antara status, dan kode berperilaku berbeda berdasarkan status.

Kami tidak akan menerapkan FSM penuh, tetapi bukan kebetulan bahwa tempat umum untuk melihat inisial FSM adalah dalam diskusi tentang AI. FSM penuh akan memiliki banyak status untuk semua perilaku berbeda dari AI yang canggih, tetapi dalam AI dasar ini kita hanya perlu melacak apakah karakter itu hidup atau tidak. Daftar berikutnya menambahkan nilai Boolean, _alive, ke bagian atas skrip, dan kode memerlukan pemeriksaan kondisional sesekali dari nilai itu. Dengan pemeriksaan itu, kode gerakan hanya berjalan saat musuh masih hidup.

```
...
private bool _alive;                                  Nilai Boolean untuk melacak
                                                      apakah musuh masih hidup
void Start() {
  _alive = true;                    Inisialisasi nilai itu
}
void Update() {
  if (_alive) {                                                Hanya bergerak jika
    transform.Translate(0, 0, speed * Time.deltaTime);         karakternya hidup
    ...
  }
}

public void SetAlive(bool alive) {        Metode publik yang memungkinkan
  _alive = alive;                         kode luar memengaruhi status
}                                         "hidup"
...
```

Gambar 4.25 Daftar Skrip WanderingAI dengan status "hidup" ditambahkan

Skrip ReactiveTarget sekarang dapat memberi tahu skrip WanderingAI saat musuh hidup atau tidak (lihat daftar berikut).

```
...
public void ReactToHit() {
    WanderingAI behavior = GetComponent<WanderingAI>();
    if (behavior != null) {                          Periksa apakah karakternya
        behavior.SetAlive(false);                    memiliki skrip WanderingAI;
    }                                                mungkin tidak.
    StartCoroutine(Die());
}
...
```

Gambar 4.26 Daftar ReactiveTarget memberi tahu WanderingAI saat mati

4.7 STRUKTUR KODE AI

Kode AI dalam bab ini terkandung dalam satu kelas sehingga pembelajaran dan pemahamannya mudah. Struktur kode ini baik-baik saja untuk kebutuhan AI sederhana, jadi jangan takut bahwa Anda telah melakukan sesuatu yang "salah" dan bahwa struktur kode yang lebih kompleks merupakan persyaratan mutlak. Untuk kebutuhan AI yang lebih kompleks (seperti game dengan beragam karakter yang sangat cerdas), struktur kode yang lebih kuat dapat membantu memfasilitasi developeran AI.

Seperti yang disinggung dalam contoh bab 1 untuk komposisi versus pewarisan, terkadang Anda ingin membagi potongan AI menjadi skrip terpisah. Melakukannya akan memungkinkan Anda untuk mencampur dan mencocokkan komponen, menghasilkan perilaku unik untuk setiap karakter. Pikirkan tentang persamaan dan perbedaan antara karakter Anda, dan perbedaan itu akan memandu Anda saat Anda mendesain arsitektur kode Anda. Misalnya, jika gim Anda memiliki beberapa musuh yang bergerak dengan menyerang player dengan cepat dan beberapa yang menyelinap dalam bayang-bayang, Anda mungkin ingin menjadikan Locomotion sebagai komponen terpisah. Kemudian Anda dapat membuat skrip untuk LocomotionCharge dan LocomotionSlink, dan menggunakan komponen Locomotion yang berbeda pada musuh yang berbeda. Struktur kode AI yang Anda inginkan bergantung pada desain game spesifik Anda; tidak ada satu cara yang "benar" untuk melakukannya. Unity memudahkan untuk merancang arsitektur kode fleksibel seperti ini.

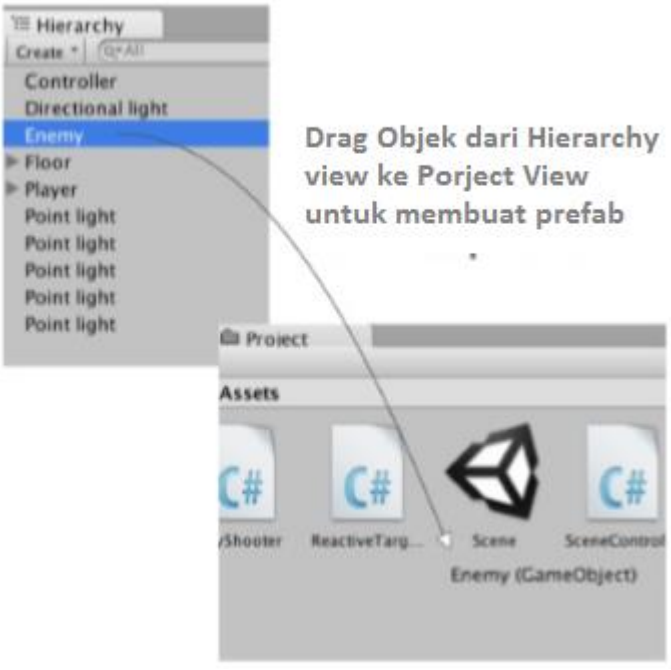
Spawning musuh prefab

Saat ini hanya ada satu musuh di scene, dan ketika mati, tempat itu kosong. Mari kita buat game menelurkan musuh sehingga setiap kali musuh mati, musuh baru muncul. Ini mudah dilakukan di Unity menggunakan konsep yang disebut prefab. Seperti yang dijelaskan si bab sebelumnya, Prefab adalah pendekatan yang fleksibel untuk mendefinisikan objek interaktif secara visual. Singkatnya, prefab adalah objek game yang sepenuhnya disempurnakan (dengan komponen yang sudah terpasang dan diatur) yang tidak ada dalam scene tertentu melainkan ada sebagai aset yang dapat disalin ke scene mana pun. Penyalinan ini dapat dilakukan secara manual, untuk memastikan bahwa objek musuh (atau prefab lainnya) sama di setiap scene. Lebih penting lagi, Prefab juga dapat dihasilkan dari kode; Anda dapat menempatkan salinan objek ke dalam scene menggunakan perintah dalam skrip dan tidak hanya dengan melakukannya secara manual di editor visual.

Definisi Aset adalah file apa pun yang muncul di tampilan Proyek; ini bisa berupa gambar 2D, model 3D, file kode, scene, dan sebagainya. Saya menyebutkan istilah aset secara singkat di bab 1, tetapi saya tidak menekankannya sampai sekarang. Istilah untuk salah satu salinan prefab ini adalah instance, analog dengan bagaimana kata instance mengacu pada objek kode tertentu yang dibuat dari suatu kelas. Cobalah untuk menjaga terminologi tetap lurus; prefab mengacu pada objek game yang ada di luar scene apa pun, sedangkan instance mengacu pada salinan objek yang ditempatkan di sebuah scene.

Membuat Prefab musuh

Untuk membuat prefab, pertama buat objek di scene yang akan menjadi prefab. Karena objek musuh kita akan menjadi prefab, kita sudah melakukan langkah pertama ini. Sekarang yang kita lakukan hanyalah menyeret objek ke bawah dari tampilan Hierarchy dan meletakkannya di tampilan Project; ini akan secara otomatis menyimpan objek sebagai Prefab. Kembali ke tampilan Hierarki, nama objek asli akan berubah menjadi biru untuk menandakan bahwa itu sekarang ditautkan ke prefab. Jika Anda ingin mengedit prefab lebih lanjut (seperti dengan menambahkan komponen baru), Anda akan membuat perubahan tersebut pada objek dalam scene dan kemudian pilih GameObject > Apply Changes To Prefab. Tapi kita tidak ingin objek di scene lagi (kita akan spawn prefab, tidak menggunakan instance yang sudah ada di scene), jadi hapus objek musuh sekarang.



Gambar 4.27 Drag objek dari Hierarki ke Proyek untuk membuat Prefab.

Peringatan Antarmuka untuk bekerja dengan Prefab agak canggung, dan hubungan antara Prefab dan contoh mereka dalam scene bisa rapuh. Misalnya, Anda sering kali harus menyeret prefab ke dalam scene untuk mengeditnya, lalu menghapus objek setelah Anda selesai mengedit. Dalam bab pertama saya menyebutkan ini sebagai kelemahan Unity, dan saya berharap alur kerja dengan Prefab meningkat di versi Unity yang akan datang. Sekarang kita memiliki objek prefab yang sebenarnya untuk dimunculkan di scene, jadi mari kita menulis kode untuk membuat instance dari prefab.

Membuat instance dari SceneController yang tidak terlihat

Meskipun Prefab itu sendiri tidak ada di scene, harus ada beberapa objek di scene untuk dilampirkan kode spawning musuh. Apa yang akan kita lakukan adalah membuat objek game kosong; kita dapat melampirkan skrip untuk itu, tetapi objek tidak akan terlihat di scene. Tip, Penggunaan GameObjects kosong untuk melampirkan komponen skrip adalah pola umum dalam developeran Unity. Trik ini digunakan untuk tugas abstrak yang tidak berlaku untuk objek tertentu di scene. Skrip kesatuan dimaksudkan untuk dilampirkan ke objek yang terlihat, tetapi tidak setiap tugas masuk akal seperti itu.

Pilih *GameObject > Create Empty*, ganti nama objek baru menjadi Controller, lalu atur posisinya menjadi 0, 0, 0 (secara teknis posisinya tidak masalah karena objek tidak terlihat, tetapi meletakkannya di asal akan membuat hidup lebih sederhana jika Anda pernah menjadi orang tua untuk itu). Buat skrip bernama SceneController, seperti yang ditunjukkan pada daftar berikut.

```
using UnityEngine;
using System.Collections;

public class SceneController : MonoBehaviour {
  [SerializeField] private GameObject enemyPrefab;
  private GameObject _enemy;

  void Update() {
    if (_enemy == null) {
      _enemy = Instantiate(enemyPrefab) as GameObject;
      _enemy.transform.position = new Vector3(0, 1, 0);
      float angle = Random.Range(0, 360);
      _enemy.transform.Rotate(0, angle, 0);
    }
  }
}
```

Variabel serial untuk menautkan ke objek cetakan

Variabel pribadi untuk melacak instance musuh di tempat kejadian

Hanya menelurkan musuh baru jika belum ada satu di tempat kejadian

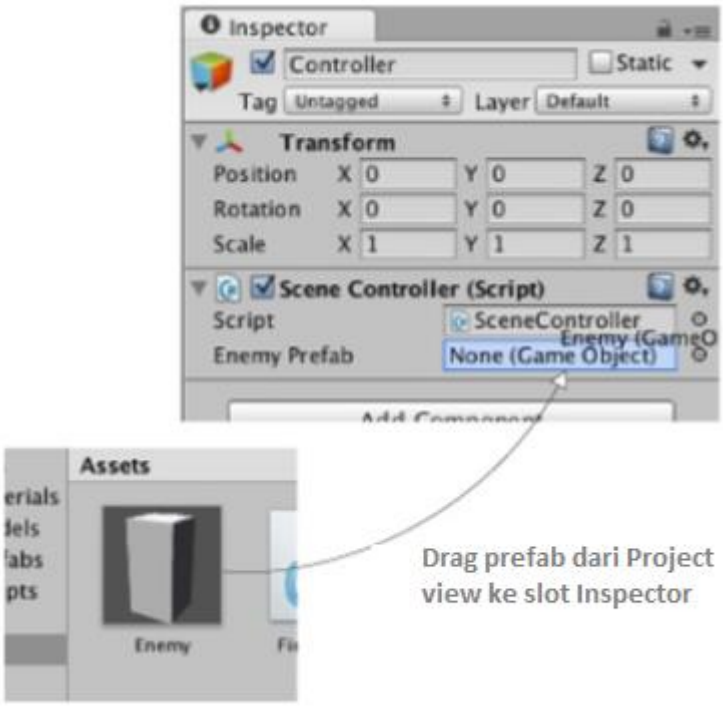
Metode yang menyalin objek cetakan

Gambar 4.28 SceneController yang memunculkan Prefab musuh

Lampirkan skrip ini ke objek controller, dan di Inspector Anda akan melihat slot variabel untuk Prefab musuh. Ini bekerja mirip dengan variabel publik, tetapi ada perbedaan penting (lihat peringatan berikut). Peringatan Saya merekomendasikan variabel personal dengan SerializeField ke objek referensi di editor Unity karena Anda ingin mengekspos variabel itu di Inspector tetapi tidak ingin nilainya diubah oleh skrip lain. Seperti yang dijelaskan di bab 2, variabel publik muncul di Inspector secara default (dengan kata lain, variabel tersebut diserialkan oleh Unity), jadi sebagian besar tutorial dan kode contoh yang Anda lihat menggunakan variabel publik untuk semua nilai yang diserialisasi. Tetapi variabel-variabel ini juga dapat dimodifikasi oleh skrip lain (bagaimanapun juga, ini adalah variabel publik); dalam banyak kasus, Anda tidak ingin nilai diubah dalam kode tetapi hanya ditetapkan di Inspector.

Drag aset Prefab dari Proyek ke slot variabel kosong; ketika mouse mendekat, Anda akan melihat sorot slot untuk menunjukkan bahwa objek dapat dihubungkan di sana. Setelah Prefab musuh ditautkan ke skrip SceneController, mainkan scene untuk melihat kode beraksi. Musuh akan muncul di tengah ruangan sama seperti sebelumnya, tetapi sekarang jika Anda

menembak musuh maka akan digantikan oleh musuh baru. Jauh lebih baik daripada hanya satu musuh yang hilang selamanya!



Gambar 4.29 Tarik prefab musuh dari Project ke slot Enemy Prefab di Inspector.

Tip Pendekatan menyeret objek ke slot variabel Inspector ini adalah teknik praktis yang muncul di banyak skrip berbeda. Di sini kami menautkan prefab ke skrip, tetapi Anda juga dapat menautkan ke objek dalam scene, atau bahkan komponen tertentu (karena kode perlu memanggil metode publik dalam komponen spesifik itu). Dalam bab-bab selanjutnya kita akan menggunakan teknik ini lagi.

Tip Pendekatan menyeret objek ke slot variabel Inspector ini adalah teknik praktis yang muncul di banyak skrip berbeda. Di sini kami menautkan prefab ke skrip, tetapi Anda juga dapat menautkan ke objek dalam scene, atau bahkan komponen tertentu (karena kode perlu memanggil metode publik dalam komponen spesifik itu). Dalam bab-bab selanjutnya kita akan menggunakan teknik ini lagi.

Inti dari skrip ini adalah metode Instantiate(), jadi perhatikan baris itu. Saat kami membuat instance prefab, itu membuat salinan di scene. Secara default, Instantiate() mengembalikan objek baru sebagai tipe Object generik, tetapi Object sangat tidak berguna secara langsung dan kita perlu menanganinya sebagai GameObject. Di C#, gunakan kata kunci as untuk typecasting untuk mengkonversi dari satu jenis objek kode ke jenis lain (ditulis dengan sintaks objek asli sebagai tipe baru).

Objek instantiated disimpan di _enemy, variabel personal dari tipe GameObject (dan sekali lagi, tetap luruskan perbedaan antara prefab dan instance dari prefab; musuhPrefab menyimpan prefab sedangkan _enemy menyimpan instance). Pernyataan if yang memeriksa objek yang disimpan memastikan bahwa Instantiate() dipanggil hanya ketika _enemy kosong (atau null, dalam coderspeak). Variabel mulai kosong, sehingga kode instantiating berjalan sekali sejak awal sesi. Objek yang dikembalikan oleh Instantiate() kemudian disimpan di _enemy sehingga kode instantiate tidak akan berjalan lagi. Karena musuh menghancurkan dirinya sendiri saat ditembak, itu mengosongkan variabel _enemy dan menyebabkan Instantiate() dijalankan lagi. Dengan cara ini, selalu ada musuh di scene.

Tip Pendekatan menyeret objek ke slot variabel Inspector ini adalah teknik praktis yang muncul di banyak skrip berbeda. Di sini kami menautkan prefab ke skrip, tetapi Anda juga dapat menautkan ke objek dalam scene, atau bahkan komponen tertentu (karena kode perlu memanggil metode publik dalam komponen spesifik itu). Dalam bab-bab selanjutnya kita akan menggunakan teknik ini lagi.

Inti dari skrip ini adalah metode Instantiate(), jadi perhatikan baris itu. Saat kami membuat instance prefab, itu membuat salinan di scene. Secara default, Instantiate() mengembalikan objek baru sebagai tipe Object generik, tetapi Object sangat tidak berguna secara langsung dan kita perlu menanganinya sebagai GameObject. Di C#, gunakan kata kunci as untuk typecasting untuk mengkonversi dari satu jenis objek kode ke jenis lain (ditulis dengan sintaks objek asli sebagai tipe baru).

Objek instantiated disimpan di _enemy, variabel personal dari tipe GameObject (dan sekali lagi, tetap luruskan perbedaan antara prefab dan instance dari prefab; musuhPrefab menyimpan prefab sedangkan _enemy menyimpan instance). Pernyataan if yang memeriksa objek yang disimpan memastikan bahwa Instantiate() dipanggil hanya ketika _enemy kosong (atau null, dalam coderspeak). Variabel mulai kosong, sehingga kode instantiating berjalan sekali sejak awal sesi. Objek yang dikembalikan oleh Instantiate() kemudian disimpan di _enemy sehingga kode instantiate tidak akan berjalan lagi. Karena musuh menghancurkan dirinya sendiri saat ditembak, itu mengosongkan variabel _enemy dan menyebabkan Instantiate() dijalankan lagi. Dengan cara ini, selalu ada musuh di scene.

Menghancurkan GameObjects dan manajemen memori

Agak tidak terduga bahwa referensi yang ada menjadi nol ketika sebuah objek menghancurkan dirinya sendiri. Dalam bahasa pemrograman yang dikelola memori seperti C#, biasanya Anda tidak dapat menghancurkan objek secara langsung; Anda hanya dapat melakukan dereferensi sehingga dapat dihancurkan secara otomatis. Ini masih berlaku di Unity, tetapi cara GameObjects ditangani di belakang layar membuatnya terlihat seperti dihancurkan secara langsung.

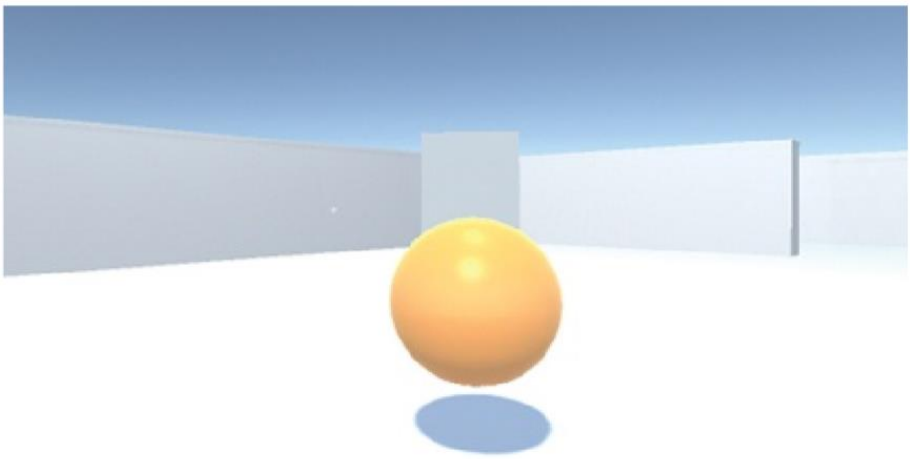
Untuk menampilkan objek dalam scene, Unity harus memiliki referensi ke semua objek dalam grafik scenenya. Jadi, bahkan jika Anda menghapus semua referensi ke GameObject dalam kode Anda, masih akan ada referensi grafik scene ini yang mencegah objek dihancurkan secara otomatis. Karena itu, Unity menyediakan metode Destroy() untuk memberi tahu game engine "Hapus objek ini dari grafik scene." Sebagai bagian dari fungsionalitas di balik layar itu, Unity juga membebani operator == untuk mengembalikan nilai true saat memeriksa null. Secara teknis objek itu masih ada di memori, tetapi mungkin juga tidak ada lagi, jadi Unity membuatnya tampak nol. Anda dapat mengonfirmasi ini dengan memanggil GetInstanceID() pada objek yang dihancurkan.

Namun, perhatikan bahwa developer Unity sedang mempertimbangkan untuk mengubah perilaku ini ke manajemen memori yang lebih standar. Jika ya, maka kode spawning juga perlu diubah, mungkin dengan menukar centang (_enemy==null) dengan parameter baru seperti (_enemy.isDestroyed). Lihat blog/halaman Facebook mereka:
https://www.facebook.com/unity3d/posts/10152271098591773
(Jika sebagian besar dari diskusi ini adalah bahasa Yunani bagi Anda, maka jangan khawatir; ini adalah diskusi teknis tangensial untuk orang-orang yang tertarik dengan detail yang tidak jelas ini.)

Memotret melalui pembuatan objek

Baiklah, mari tambahkan sedikit fungsionalitas ke musuh. Sama seperti yang kita lakukan dengan player, pertama-tama kita buat mereka bergerak—sekarang mari kita buat mereka menembak! Seperti yang saya sebutkan kembali saat memperkenalkan raycasting, itu

hanyalah salah satu pendekatan untuk menerapkan pemotretan. Pendekatan lain melibatkan pembuatan prefab, jadi mari kita lakukan pendekatan itu untuk membuat musuh membalas. Tujuan dari bagian ini adalah untuk melihat gambar 4.30 saat bermain.



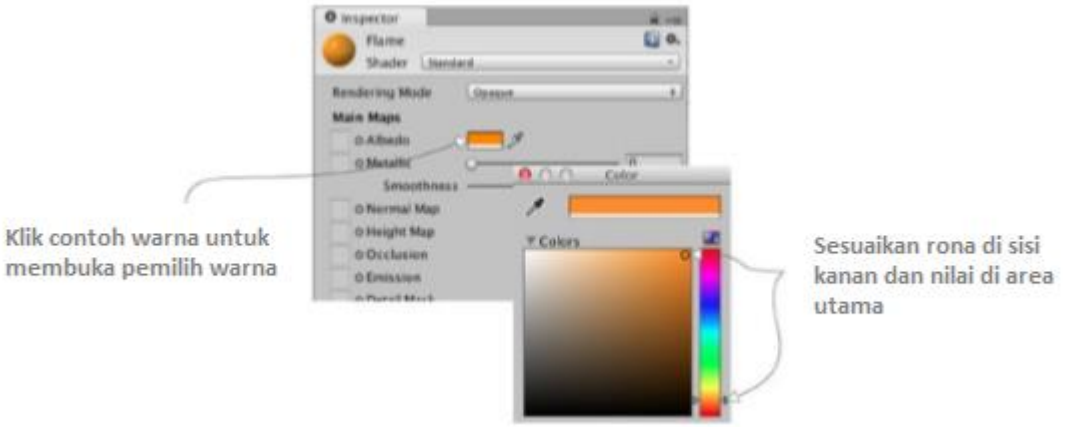
Gambar 4.30 Musuh menembakkan "bola api" ke player

Membuat Prefab proyektil

Jika pemotretan sebelumnya tidak melibatkan proyektil yang sebenarnya di scene, pemotretan kali ini akan melibatkan proyektil di scene. Menembak dengan raycasting pada dasarnya instan, mencatatkan pukulan saat mouse diklik, tapi kali ini musuh akan memancarkan bola api yang terbang di udara. Memang, mereka akan bergerak cukup cepat, tetapi itu tidak akan instan, memberi player kesempatan untuk menghindar. Alih-alih menggunakan raycasting untuk mendeteksi hit, kami akan menggunakan deteksi collision (sistem collision yang sama yang mencegah player yang bergerak melewati dinding).

Kode akan menelurkan bola api dengan cara yang sama seperti musuh bertelur: dengan membuat Prefab. Seperti yang sudah dijelaskan di bagian sebelumnya, langkah pertama saat membuat prefab adalah membuat objek di scene yang akan menjadi prefab, jadi mari kita buat bola api. Untuk memulai, pilih GameObject > 3D Object > Sphere. Ganti nama objek baru Fireball. Sekarang buat skrip baru, juga disebut Fireball, dan lampirkan skrip itu ke objek ini. Akhirnya kami akan menulis kode dalam skrip ini, tetapi biarkan default untuk saat ini sementara kami mengerjakan beberapa bagian lain dari objek bola api. Agar tampak seperti bola api dan bukan hanya bola abu-abu, kita akan memberi objek warna oranye terang. Sifat permukaan seperti warna dikendalikan menggunakan bahan. *Definisi*, Bahan adalah paket informasi yang mendefinisikan sifat permukaan objek 3D apa pun yang dilampirkan material. Sifat permukaan ini dapat mencakup warna, kilau, dan bahkan kekasaran yang halus.

Pilih Aset > Buat > Bahan. Beri nama materi baru seperti Flame. Pilih material di tampilan Project untuk melihat properti material di Inspector. Seperti yang ditunjukkan gambar 3.10, klik contoh warna berlabel Albedo (itu adalah istilah teknis yang mengacu pada warna utama permukaan). Mengklik yang akan memunculkan pemilih warna di jendelanya sendiri; geser kedua bilah berwarna pelangi di sisi kanan dan area pengambilan utama untuk mengatur warnanya menjadi oranye.



Gambar 4.31 Mengatur warna material

Kami juga akan mencerahkan material agar terlihat lebih seperti api. Sesuaikan nilai Emission (salah satu atribut lain di Inspector). Ini default ke 0, jadi ketik .3 untuk mencerahkan materi. Sekarang Anda dapat mengubah objek bola api menjadi prefab dengan menyeret objek ke bawah dari Hierarchy ke Project, seperti yang Anda lakukan dengan prefab musuh. Bagus, kami memiliki Prefab baru untuk digunakan sebagai proyektil! Selanjutnya adalah menulis kode untuk menembak menggunakan proyektil itu.

Menembak proyektil dan bercollision dengan target

Mari kita membuat beberapa penyesuaian pada musuh untuk mengeluarkan bola api. Karena kode untuk mengenali player akan memerlukan skrip baru (seperti ReactiveTarget yang diperlukan oleh kode untuk mengenali target), pertama buat skrip baru dan beri nama skrip itu PlayerCharacter. Lampirkan skrip ini ke objek pemutar di scene.

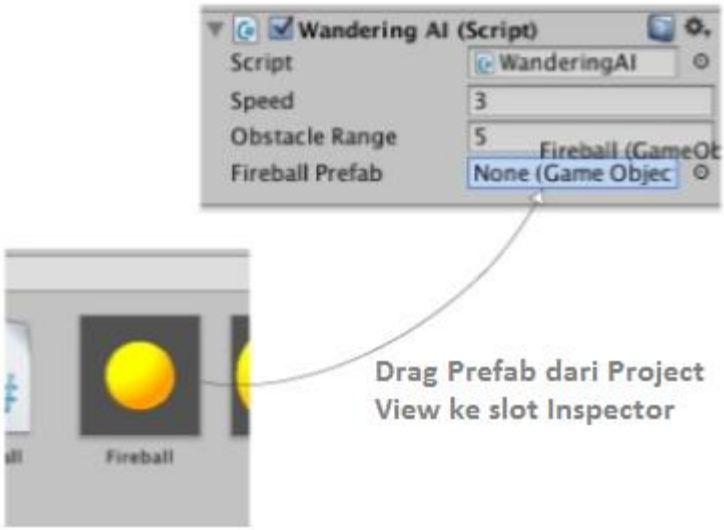
Sekarang buka WanderingAI dan tambahkan kode dari daftar berikut.

Gambar 4.32 Daftar Penambahan WanderingAI untuk memancarkan bola api

Anda akan melihat bahwa semua anotasi dalam daftar ini merujuk pada bit yang serupa (atau sama) di skrip sebelumnya. Daftar kode sebelumnya sudah menunjukkan semua yang diperlukan untuk memancarkan bola api; sekarang kami sedang menumbuk bersama-sama dan remixing bit kode agar sesuai dengan konteks baru. Sama seperti di SceneController, Anda perlu menambahkan dua bidang GameObject di bagian atas skrip: variabel serial untuk menautkan prefab ke, dan variabel personal untuk melacak instance yang disalin oleh kode. Setelah melakukan raycast, kode memeriksa PlayerCharacter pada objek yang dipukul; ini berfungsi seperti ketika kode pemotretan memeriksa ReactiveTarget pada objek yang dipukul. Kode yang membuat bola api ketika belum ada satu pun di scene berfungsi seperti kode yang

memberi contoh musuh. Namun, pemosisian dan rotasinya berbeda; kali ini, Anda menempatkan instance tepat di depan musuh dan mengarahkannya ke arah yang sama.

Setelah semua kode baru terpasang, slot Fireball Prefab baru akan muncul saat Anda melihat komponen di Inspector, seperti slot Enemy Prefab di komponen SceneController. Klik Prefab musuh di tampilan Proyek dan Inspector akan menunjukkan komponen objek itu, seolah-olah Anda telah memilih objek di scene. Meskipun peringatan sebelumnya tentang kecanggungan antarmuka sering berlaku saat mengedit Prefab, antarmuka memudahkan untuk menyesuaikan komponen pada objek, dan hanya itu yang kami lakukan. Seperti yang ditunjukkan pada gambar 3.11, drag Prefab bola api dari Project ke slot Fireball Prefab di Inspector (sekali lagi, seperti yang Anda lakukan dengan SceneController).



Gambar 4.33 Drag Prefab bola api dari Project ke slot Fireball Prefab di Inspector.

Sekarang musuh akan menembak player saat player berada tepat di depannya...yah, coba tembak; bola oranye terang muncul di depan musuh, tetapi hanya duduk di sana karena kami belum menulis scriptnya. Ayo lakukan sekarang. Daftar berikutnya menunjukkan kode untuk skrip Fireball.

```
using UnityEngine;
using System.Collections;

public class Fireball : MonoBehaviour {
  public float speed = 10.0f;
  public int damage = 1;

  void Update() {
    transform.Translate(0, 0, speed * Time.deltaTime);
  }

  void OnTriggerEnter(Collider other) {
    PlayerCharacter player = other.GetComponent<PlayerCharacter>();
    if (player != null) {
      Debug.Log("Player hit");
    }
    Destroy(this.gameObject);
  }
}
```

Fungsi ini dipanggil ketika benda bertabrakan degnan trigger

Cek jika objek lainnya adalah PlayerCharacter

Gambar 4.34 Daftar Skrip bola api yang bereaksi terhadap collision

Bit baru yang penting untuk kode ini adalah metode OnTriggerEnter(). Metode itu dipanggil secara otomatis ketika objek mengalami collision, seperti menabrak dinding atau dengan player. Saat ini kode ini tidak akan berfungsi sepenuhnya; jika Anda menjalankannya, bola api akan terbang ke depan berkat baris Translate(), tetapi pemicunya tidak akan berjalan, mengantrekan bola api baru dengan menghancurkan yang sekarang. Perlu ada beberapa penyesuaian lain yang dilakukan pada komponen pada objek bola api. Perubahan pertama adalah menjadikan collider sebagai pemicu. Untuk menyesuaikannya, klik kotak centang Is Trigger di komponen Sphere Collider.

Tip Komponen Collider yang ditetapkan sebagai pemicu masih akan bereaksi terhadap sentuhan/tumpang tindih objek lain, tetapi tidak akan lagi menghentikan objek lain untuk lewat secara fisik. Bola api juga membutuhkan Rigidbody, komponen yang digunakan oleh sistem fisika di Unity. Dengan memberi bola api komponen Rigidbody, Anda memastikan bahwa sistem fisika mampu mendaftarkan pemicu collision untuk objek itu. Di Inspector, klik Add Component dan pilih Physics > Rigidbody. Pada komponen yang ditambahkan, hapus pilihan Use Gravity (lihat gambar 4.35) agar bola api tidak tertarik ke bawah karena gravitasi.

Gambar 4.35 Matikan gravitasi di komponen Rigidbody.

Mainkan sekarang, dan bola api dihancurkan ketika mereka mengenai sesuatu. Karena kode pemancar bola api berjalan setiap kali tidak ada bola api di scene, musuh akan menembakkan lebih banyak bola api ke player. Sekarang hanya ada satu lagi yang tersisa untuk menembak player: membuat player bereaksi saat dipukul.

Damage player

Sebelumnya Anda membuat skrip PlayerCharacter tetapi membiarkannya kosong. Sekarang Anda akan menulis kode untuk membuatnya bereaksi saat dipukul, seperti yang ditunjukkan daftar berikut.

```
using UnityEngine;
using System.Collections;

public class PlayerCharacter : MonoBehaviour {
  private int _health;

  void Start() {
    _health = 5;
  }

  public void Hurt(int damage) {
    _health -= damage;
    Debug.Log("Health: " + _health);
  }
}
```

Inisialisasi nilai kesehatan

Pengurangan kesehatan player

Gambar 4.36 Daftar Player yang dapat menerima kerusakan

Daftar tersebut mendefinisikan bidang untuk kesehatan player dan mengurangi kesehatan sesuai perintah. Di bab selanjutnya kita akan membahas tampilan teks untuk menampilkan informasi di layar, tetapi untuk saat ini kita hanya dapat menampilkan informasi tentang kesehatan player menggunakan pesan debug. Sekarang kita perlu kembali ke skrip Fireball untuk memanggil metode Hurt() player. Ganti baris debug dalam skrip Fireball dengan player.Hurt(damage) untuk memberi tahu player bahwa mereka telah terkena. Dan itu adalah bagian terakhir dari kode yang kami butuhkan!

Kami sebagian besar berfokus pada bagaimana game berfungsi dan tidak terlalu fokus pada tampilan game. Itu bukan kebetulan—buku ini kebanyakan tentang pemrograman game di Unity. Namun, penting untuk memahami cara mengerjakan dan meningkatkan visual. Sebelum kita kembali ke fokus utama buku ini pada pengkodean berbagai bagian permainan, mari kita habiskan satu bab untuk mempelajari seni permainan sehingga proyek Anda tidak akan selalu berakhir hanya dengan kotak kosong yang meluncur. Semua konten visual dalam game terdiri dari apa yang disebut aset seni. Tapi apa sebenarnya artinya itu?

4.8 MEMAHAMI ASET SENI

Aset seni adalah unit individual dari informasi visual (biasanya file) yang digunakan oleh game. Ini adalah istilah umum untuk semua konten visual; file gambar adalah aset seni, model 3D adalah aset seni, dan seterusnya. Memang, istilah aset seni hanyalah kasus spesifik dari aset, yang telah Anda pelajari adalah file apa pun yang digunakan oleh game (seperti skrip)—maka folder Aset utama di Unity. Membuat seni untuk game baru umumnya dimulai dengan gambar 2D atau model 3D karena aset tersebut membentuk dasar yang menjadi sandaran segala hal lainnya. Seperti namanya, gambar 2D adalah dasar dari grafik 2D, sedangkan model 3D adalah dasar dari grafik 3D. Secara khusus, gambar 2D adalah gambar datar; bahkan jika Anda sebelumnya tidak terbiasa dengan seni permainan, Anda mungkin sudah terbiasa dengan gambar 2D dari grafik yang digunakan di situs web. Model tiga dimensi, di sisi lain, mungkin asing bagi pendatang baru, jadi saya memberikan definisi berikut.

Definisi Animasi adalah paket informasi yang mendefinisikan pergerakan objek terkait. Karena gerakan-gerakan ini dapat didefinisikan secara independen dari objek itu sendiri, gerakan-gerakan ini dapat digunakan dengan cara mencampur dan mencocokkan dengan banyak objek. Sebagai contoh konkret, pikirkan tentang karakter yang berjalan-jalan. Posisi keseluruhan karakter ditangani oleh kode permainan (misalnya, skrip gerakan yang Anda tulis di bab 2). Namun detail gerakan kaki yang menyentuh tanah, lengan yang berayun, dan pinggul yang berputar adalah rangkaian animasi yang sedang diputar ulang; bahwa urutan animasi adalah aset seni.

Untuk membantu Anda memahami bagaimana animasi dan model 3D saling berhubungan, mari kita buat analogi dengan pewayangan: model 3D adalah boneka, animator adalah dalang yang membuat boneka bergerak, dan animasi adalah rekaman gerakan boneka. Gerakan yang didefinisikan dengan cara ini dibuat sebelumnya dan biasanya gerakan scale kecil yang tidak mengubah posisi keseluruhan objek. Ini berbeda dengan jenis gerakan scale besar yang dilakukan dalam kode di bab-bab sebelumnya.

Definisi Sistem partikel adalah mekanisme yang teratur untuk menciptakan dan mengendalikan sejumlah besar objek bergerak. Benda bergerak ini biasanya berukuran kecil—karenanya dinamakan partikel—tetapi tidak harus begitu. Sistem partikel berguna untuk menciptakan efek visual, seperti api, asap, atau penyemprotan air. Partikel (yaitu, objek individu di bawah kendali sistem partikel) dapat berupa objek mesh apa pun yang Anda pilih,

tetapi untuk sebagian besar efek partikel akan menjadi persegi yang menampilkan gambar (percikan api atau kepulan asap, misalnya).

Sebagian besar pekerjaan membuat seni permainan dilakukan dalam software eksternal, bukan di dalam Unity itu sendiri. Bahan dan sistem partikel dibuat dalam Unity, tetapi aset seni lainnya dibuat menggunakan software eksternal. Lihat lampiran B untuk mempelajari lebih lanjut tentang alat eksternal; berbagai aplikasi seni digunakan untuk membuat model dan animasi 3D. Model tiga dimensi yang dibuat di alat eksternal kemudian disimpan sebagai aset seni yang diimpor oleh Unity. Saya menggunakan Blender di lampiran C ketika menjelaskan cara membuat model (unduh dari www.blender.org), tapi itu hanya karena Blender adalah open source dan dengan demikian tersedia untuk semua pembaca. Catatan Unduhan proyek untuk bab ini menyertakan folder bernama "scratch." Meskipun folder itu berada di tempat yang sama dengan proyek Unity, itu bukan bagian dari proyek Unity; di situlah saya meletakkan file eksternal tambahan.

Saat Anda mengerjakan proyek untuk bab ini, Anda akan melihat contoh sebagian besar jenis aset seni ini (animasi agak terlalu rumit untuk saat ini dan akan dibahas nanti di buku ini). Anda akan membuat scene yang menggunakan gambar 2D, model 3D, material, dan sistem partikel. Dalam beberapa kasus, Anda akan membawa aset seni yang sudah ada dan mempelajari cara mengimpornya ke Unity, tetapi di lain waktu (terutama dengan sistem partikel) Anda akan membuat aset seni dari awal dalam Unity. Bab ini hanya menggores permukaan dari pembuatan game art. Karena buku ini berfokus pada bagaimana melakukan pemrograman di Unity, cakupan disiplin seni yang luas akan mengurangi seberapa banyak yang bisa dicakup oleh buku ini. Membuat seni permainan adalah topik besar dengan sendirinya, dengan mudah dapat mengisi beberapa buku. Dalam kebanyakan kasus, seorang programmer game perlu bermitra dengan seniman game yang berspesialisasi dalam disiplin itu. Karena itu, sangat berguna bagi programmer game untuk memahami bagaimana Unity bekerja dengan aset seni dan bahkan mungkin membuat standin kasar mereka sendiri untuk diganti nanti (umumnya dikenal sebagai seni programmer).

Catatan Tidak ada dalam bab ini yang secara langsung membutuhkan proyek dari bab-bab sebelumnya. Tetapi Anda ingin memiliki skrip gerakan seperti yang ada di bab 2 sehingga Anda dapat berjalan di sekitar scene yang akan Anda buat; jika perlu, Anda dapat mengambil objek pemutar dan skrip dari unduhan proyek. Demikian pula, bab ini diakhiri dengan objek bergerak yang mirip dengan yang dibuat di bab-bab sebelumnya.

Membangun scene 3D dasar: whiteboxing

Topik pembuatan konten pertama yang akan kita bahas adalah whiteboxing. Proses ini biasanya merupakan langkah pertama dalam membangun level di komputer (setelah mendesain level di atas kertas). Sesuai namanya, Anda memblokir dinding scene dengan geometri kosong (yaitu, kotak putih). Melihat daftar aset seni yang berbeda, scene kosong ini adalah jenis model 3D yang paling dasar, dan menyediakan dasar untuk menampilkan gambar 2D. Jika Anda mengingat kembali scene primitif yang Anda buat di bab 2, itu pada dasarnya adalah whiteboxing (Anda belum mempelajari istilahnya). Beberapa bagian ini akan menjadi pengulangan dari pekerjaan yang dilakukan di awal bab 2, tetapi kali ini kita akan membahas prosesnya lebih cepat serta membahas lebih banyak terminologi baru. Catatan Istilah lain yang sering digunakan adalah greyboxing. Ini berarti hal yang sama. Saya cenderung menggunakan whiteboxing karena itulah istilah yang pertama kali saya pelajari, tetapi yang lain menggunakan greyboxing dan istilah itu juga diterima. Warna sebenarnya yang digunakan bervariasi, mirip dengan bagaimana cetak biru tidak selalu biru.

Whiteboxing

Memblokir scene dengan geometri kosong memiliki beberapa tujuan. Pertama, proses ini memungkinkan Anda dengan cepat membuat "sketsa" yang akan disempurnakan secara bertahap dari waktu ke waktu. Aktivitas ini terkait erat dengan desain level dan/atau desainer level. Definisi Desain level adalah disiplin perencanaan dan pembuatan scene dalam game (atau level). Desainer level adalah praktisi desain level.

Seiring bertambahnya ukuran tim developeran game dan anggota tim menjadi lebih terspesialisasi, alur kerja pembangunan level yang umum adalah bagi perancang level untuk membuat versi pertama level melalui whiteboxing. Level kasar ini kemudian diserahkan kepada tim seni untuk dipoles visual. Tetapi bahkan di tim kecil, di mana orang yang sama mendesain level dan menciptakan seni untuk game, alur kerja pertama melakukan whiteboxing dan kemudian memoles visual umumnya bekerja paling baik; Anda harus memulai dari suatu tempat, dan whiteboxing memberikan dasar yang jelas untuk membangun visual.

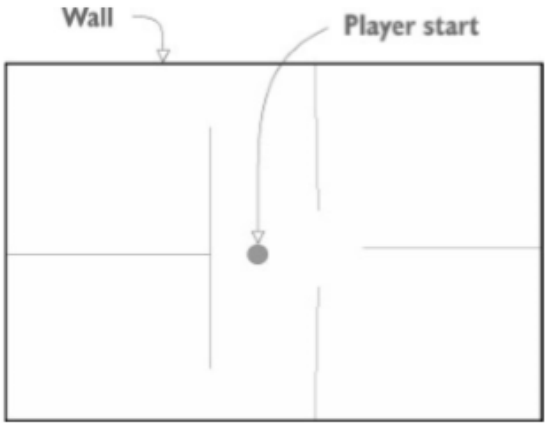
Tujuan kedua yang dilayani oleh whiteboxing adalah bahwa level tersebut mencapai status yang dapat dimainkan dengan sangat cepat. Ini mungkin belum selesai (memang, level setelah whiteboxing masih jauh dari selesai) tetapi versi kasar ini fungsional dan dapat mendukung gameplay. Minimal, player dapat berjalan di sekitar scene (pikirkan demo dari bab 2). Dengan cara ini Anda dapat menguji untuk memastikan levelnya menyatu dengan baik (misalnya, apakah ukuran ruangan yang tepat untuk game ini?) sebelum menginvestasikan banyak waktu dan energi untuk pekerjaan yang mendetail. Jika ada yang tidak beres (misalnya Anda menyadari bahwa ruang harus lebih besar), maka akan lebih mudah untuk mengubah dan menguji ulang saat Anda berada pada tahap whiteboxing.

Selain itu, dapat memainkan level yang sedang dibangun adalah dorongan moral yang sangat besar. Jangan abaikan manfaat ini: membangun semua visual untuk sebuah scene bisa memakan banyak waktu, dan itu bisa mulai terasa seperti kerja keras jika Anda harus menunggu lama sebelum Anda bisa merasakan pekerjaan itu di dalam game. . Whiteboxing membangun level yang lengkap (jika primitif) segera, dan sangat menyenangkan untuk memainkan game ini karena terus meningkat. Baiklah, jadi Anda mengerti mengapa level dimulai dengan whiteboxing; sekarang mari kita benar-benar membangun level!

Menggambar denah lantai untuk level

Membangun level di komputer mengikuti merancang level di atas kertas. Kami tidak akan masuk ke diskusi besar tentang desain level; seperti yang dicatat Bab 2 tentang desain game, desain level (yang merupakan bagian dari desain game) adalah disiplin besar yang dapat mengisi seluruh buku dengan sendirinya. Untuk tujuan kami, kami akan menggambar tingkat dasar dengan sedikit "desain" masuk ke dalam rencana, untuk memberi kami target untuk dikerjakan.

Gambar dibawah ini adalah gambar top-down dari tata letak sederhana dengan empat kamar yang dihubungkan oleh lorong tengah. Itu saja yang kita butuhkan untuk sebuah rencana sekarang: sekumpulan area terpisah dan dinding interior untuk ditempatkan. Dalam permainan nyata, rencana Anda akan lebih luas dan mencakup hal-hal seperti musuh dan item.

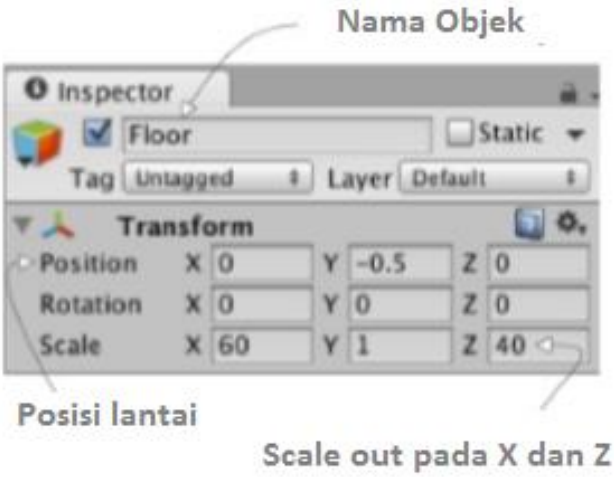


Gambar 4.37 Denah lantai untuk tingkat: empat kamar dan koridor tengah

Anda bisa berlatih whiteboxing dengan membuat denah lantai ini, atau Anda bisa menggambar level sederhana Anda sendiri untuk mempraktikkan langkah itu juga. Spesifik tata letak ruangan tidak terlalu penting untuk latihan ini. Yang penting untuk tujuan kita adalah membuat denah lantai sehingga kita bisa bergerak maju dengan langkah berikutnya.

Meletakkan primitif sesuai dengan rencana

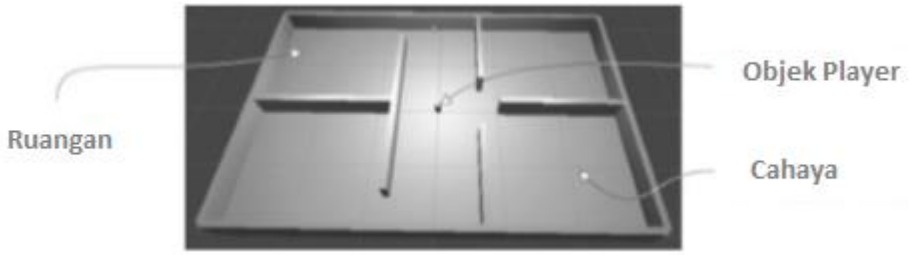
Membangun tingkat kotak putih sesuai dengan denah lantai yang digambar melibatkan penempatan dan penscalean sekelompok kotak kosong untuk menjadi dinding dalam diagram. Seperti yang dijelaskan di bagian 2.2.1, pilih GameObject > 3D Object > Cube untuk membuat kotak kosong yang dapat Anda posisikan dan scalekan sesuai kebutuhan. Catatan Ini tidak diperlukan, tetapi alih-alih objek kubus, Anda mungkin ingin menggunakan objek QuadsBox dalam unduhan proyek. Objek ini berbentuk kubus yang terdiri dari enam bagian terpisah untuk memberi Anda lebih banyak fleksibilitas saat menerapkan bahan. Apakah Anda menggunakan objek ini atau tidak tergantung pada alur kerja yang Anda inginkan; misalnya, saya tidak peduli dengan QuadsBox karena semua geometri kotak putih akan diganti dengan seni baru nanti.



Gambar 4.38 Tampilan inspector dari kotak yang diposisikan dan discalekan untuk lantai

Ulangi langkah-langkah ini untuk membuat dinding scene. Anda mungkin ingin membersihkan tampilan Hierarchy dengan menjadikan dinding sebagai anak-anak dari objek dasar yang sama (ingat, posisikan objek root pada 0, 0, 0, lalu drag objek ke dalam Hierarchy), tetapi itu tidak diperlukan. Juga letakkan beberapa lampu sederhana di sekitar scene sehingga Anda dapat melihatnya; merujuk kembali ke bab 2, buat lampu dengan memilihnya di

submenu Light dari menu GameObject. Levelnya akan terlihat seperti gambar 4.3 setelah Anda selesai dengan whiteboxing.



Gambar 4.39 Tingkat kotak putih dari denah lantai

Siapkan objek pemutar atau kamera Anda untuk bergerak (buat pemutar dengan controller karakter dan skrip gerakan; lihat bab 2 jika Anda memerlukan penjelasan lengkap). Sekarang Anda dapat berjalan di sekitar scene primitif untuk mengalami pekerjaan Anda dan mengujinya. Dan begitulah cara Anda melakukan whiteboxing! Cukup sederhana—tetapi yang Anda miliki saat ini hanyalah geometri kosong, jadi mari kita hiasi geometri dengan gambar di dinding.

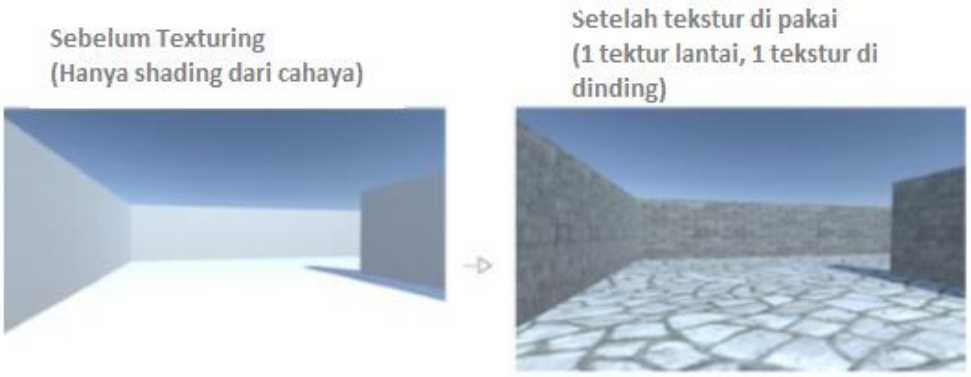
Mengekspor geometri kotak putih ke alat seni eksternal

Sebagian besar pekerjaan saat menambahkan polesan visual ke level dilakukan dalam aplikasi seni 3D eksternal seperti Blender. Karena itu, Anda mungkin ingin memiliki geometri kotak putih di alat seni Anda untuk dirujuk. Secara default tidak ada opsi ekspor untuk primitif yang diletakkan di dalam Unity. Tetapi skrip pihak ketiga tersedia yang menambahkan fungsi ini ke editor. Kebanyakan skrip semacam itu memungkinkan Anda untuk memilih geometri dalam scene dan kemudian menekan tombol Ekspor (Bab 1 menyebutkan bahwa skrip dapat menyesuaikan editor). Skrip kustom ini biasanya mengekspor geometri sebagai file OBJ (OBJ adalah salah satu dari beberapa jenis file yang dibahas nanti dalam bab ini). Di situs Unity3D, klik tombol cari dan ketik obj eksportir. Atau Anda bisa pergi ke sini untuk satu contoh: http://wiki.unity3d.com/index.php?title=ObjExporter

4.9 TEKSTUR SCENE DENGAN GAMBAR 2D

Level pada titik ini adalah sketsa kasar. Ini dapat dimainkan, tetapi jelas lebih banyak pekerjaan yang harus dilakukan pada tampilan visual scene. Langkah selanjutnya dalam meningkatkan tampilan level adalah menerapkan tekstur. Definisi Tekstur adalah gambar 2D yang digunakan untuk menyempurnakan grafik 3D. Itu secara harfiah totalitas dari apa arti istilah itu; jangan bingung dengan berpikir bahwa salah satu dari berbagai penggunaan tekstur adalah bagian dari bagaimana istilah tersebut didefinisikan. Tidak peduli bagaimana gambar digunakan, itu masih disebut sebagai tekstur.

Catatan Kata tekstur secara rutin digunakan sebagai kata kerja dan kata benda. Selain definisi kata benda, kata tersebut menjelaskan tindakan menggunakan gambar 2D dalam grafik 3D. Tekstur memiliki sejumlah kegunaan dalam grafik 3D, tetapi penggunaan yang paling mudah adalah untuk ditampilkan pada permukaan model 3D. Nanti di bab ini kita akan membahas cara kerjanya untuk model yang lebih kompleks, tetapi untuk level kotak putih kita, gambar 2D akan bertindak sebagai wallpaper yang menutupi dinding (lihat gambar 4.40).



Gambar 4.40 Membandingkan level sebelum dan sesudah tekstur

Seperti yang Anda lihat dari perbandingan pada gambar 4.4, tekstur mengubah apa yang merupakan konstruksi digital yang jelas tidak nyata menjadi dinding bata. Kegunaan lain untuk tekstur termasuk mask untuk memotong bentuk dan peta normal untuk membuat permukaan bergelombang; nanti Anda mungkin ingin mencari informasi lebih lanjut tentang tekstur dalam sumber daya yang disebutkan dalam lampiran D.

Memilih format file

Berbagai format file tersedia untuk menyimpan gambar 2D, jadi mana yang harus Anda gunakan? Unity mendukung penggunaan banyak format file yang berbeda, sehingga Anda dapat memilih salah satu dari yang ditunjukkan pada tabel 4.2.

Definisi Saluran alfa digunakan untuk menyimpan informasi transparansi dalam sebuah gambar. Warna yang terlihat datang dalam tiga "saluran" informasi: Merah, Hijau, dan Biru. Alfa adalah saluran informasi tambahan yang tidak terlihat tetapi mengontrol visibilitas gambar.Meskipun Unity akan menerima salah satu gambar yang ditampilkan pada tabel 4.2 untuk diimpor dan digunakan sebagai tekstur, berbagai format file sangat bervariasi dalam fitur yang didukungnya. Dua faktor khususnya penting untuk gambar 2D yang diimpor sebagai tekstur: bagaimana gambar dikompresi, dan apakah gambar itu memiliki saluran alfa? Saluran alfa adalah pertimbangan langsung: karena saluran alfa sering digunakan dalam grafik 3D, lebih baik jika gambar memiliki saluran alfa. Kompresi gambar adalah pertimbangan yang sedikit lebih rumit, tetapi bermuara pada "kompresi lossy buruk": tidak ada kompresi dan kompresi lossless menjaga kualitas gambar, sedangkan kompresi lossy mengurangi kualitas gambar (maka istilah lossy) sebagai bagian dari pengurangan ukuran file.

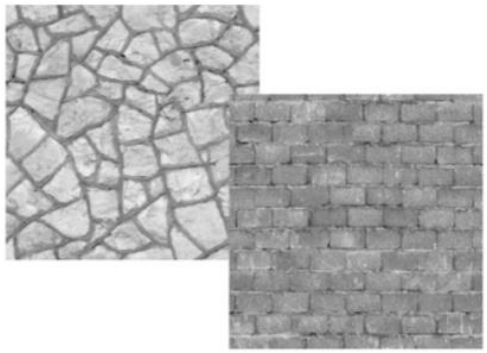
Di antara dua pertimbangan ini, dua format file yang saya rekomendasikan untuk tekstur Unity adalah PNG atau TGA. Targas (TGA) dulunya adalah format file favorit untuk membuat tekstur grafik 3D, sebelum PNG digunakan secara luas di internet; hari ini PNG hampir setara secara teknologi tetapi jauh lebih luas, karena berguna baik di web maupun sebagai tekstur. PSD juga umumnya direkomendasikan untuk tekstur Unity, karena ini adalah format file tingkat lanjut dan nyaman jika file yang sama yang Anda kerjakan di Photoshop juga berfungsi di Unity. Tapi saya cenderung lebih suka menyimpan file kerja terpisah dari file "selesai" yang diekspor ke Unity (pola pikir yang sama ini muncul lagi nanti dengan model 3D). Hasilnya adalah semua gambar yang saya berikan dalam contoh proyek adalah PNG, dan saya menyarankan Anda bekerja dengan format file itu juga. Dengan keputusan ini, saatnya untuk membawa beberapa gambar ke Unity dan menerapkannya ke scene kosong.

Mengimpor file gambar

Mari mulai membuat/menyiapkan tekstur yang akan kita gunakan. Gambar yang digunakan untuk tingkat tekstur biasanya dapat dibuat ubin sehingga dapat diulang di seluruh permukaan besar seperti lantai. Definisi Sebuah gambar tileable (kadang-kadang disebut

sebagai ubin mulus) adalah gambar di mana tepi yang berlawanan cocok ketika ditempatkan berdampingan. Dengan cara ini gambar dapat diulang tanpa jahitan yang terlihat di antara pengulangan. Konsep untuk tekstur 3D sama seperti wallpaper di halaman web.

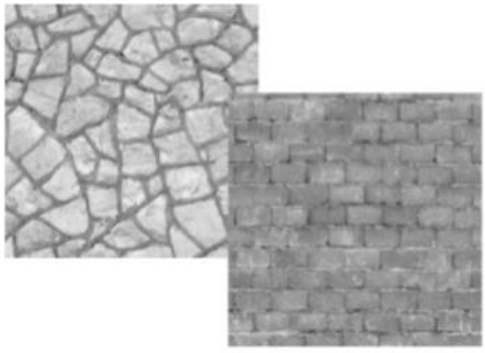
Anda dapat memperoleh gambar yang dapat diberi ubin dengan beberapa cara berbeda, seperti memanipulasi foto atau bahkan mengecatnya dengan tangan. Tutorial dan penjelasan tentang teknik ini dapat ditemukan di berbagai buku dan situs web, tetapi kami tidak ingin terjebak dengan itu sekarang. Sebagai gantinya, mari ambil beberapa gambar yang dapat dipasangi ubin dari salah satu dari banyak situs web yang menawarkan katalog gambar tersebut untuk digunakan oleh seniman 3D. Sebagai contoh, saya memperoleh beberapa gambar dari www.cgtextures.com (lihat gambar 4.41) untuk diterapkan pada dinding dan lantai lantai; temukan beberapa gambar yang menurut Anda bagus untuk lantai dan dinding.



Gambar 4.41 ubin batu dan bata yang mulus diperoleh dari CGTextures.com

Unduh gambar yang Anda inginkan dan siapkan untuk digunakan sebagai tekstur. Secara teknis, Anda dapat menggunakan gambar secara langsung saat diunduh, tetapi gambar tersebut tidak ideal untuk digunakan sebagai tekstur. Meskipun mereka tentu saja dapat dipasangi ubin (aspek penting mengapa kami menggunakan gambar-gambar ini), ukurannya tidak tepat dan format filenya salah. Tekstur harus berukuran dalam pangkat 2. Untuk alasan efisiensi teknis, chip grafis ingin menangani tekstur dalam ukuran 2N: 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 (angka berikutnya adalah 4096, tetapi pada saat itu gambar terlalu besar untuk digunakan sebagai tekstur). Di editor gambar Anda (Photoshop, GIMP, atau apa pun; lihat lampiran B) scalekan gambar yang diunduh ke 256x256, dan simpan sebagai PNG.

Sekarang drag file dari lokasinya di komputer ke tampilan Proyek di Unity. Ini akan menyalin file ke proyek Unity Anda (lihat gambar 4.42), di mana mereka diimpor sebagai tekstur dan dapat digunakan dalam scene 3D. Jika menyeret file ke atas akan canggung, Anda dapat mengklik kanan di Project dan memilih Impor Aset Baru untuk mendapatkan pemilih file.

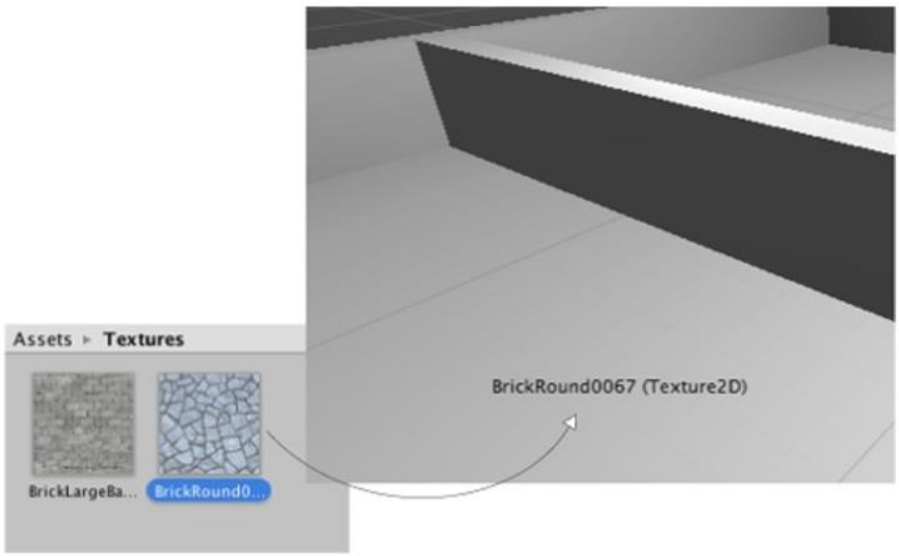


Gambar 4.42 Drag gambar dari luar Unity untuk mengimpornya ke tampilan Proyek.

Tip Mengatur aset Anda ke dalam folder terpisah mungkin merupakan ide yang baik karena proyek Anda mulai menjadi lebih kompleks; dalam tampilan Proyek, buat folder untuk Skrip dan Tekstur, lalu pindahkan aset ke folder yang sesuai. Cukup drag file ke folder barunya. Warning Unity memiliki beberapa kata kunci yang ditanggapinya dalam nama folder, dengan cara khusus menangani konten folder khusus ini. Kata kunci tersebut adalah Sumber Daya, Plugin, Editor, dan Gizmos. Nanti di buku ini kita akan membahas apa yang dilakukan beberapa folder khusus ini, tetapi untuk saat ini hindari memberi nama folder apa pun dengan kata-kata itu. Sekarang gambar diimpor ke Unity sebagai tekstur, siap digunakan. Tapi bagaimana kita menerapkan tekstur ke objek dalam scene?

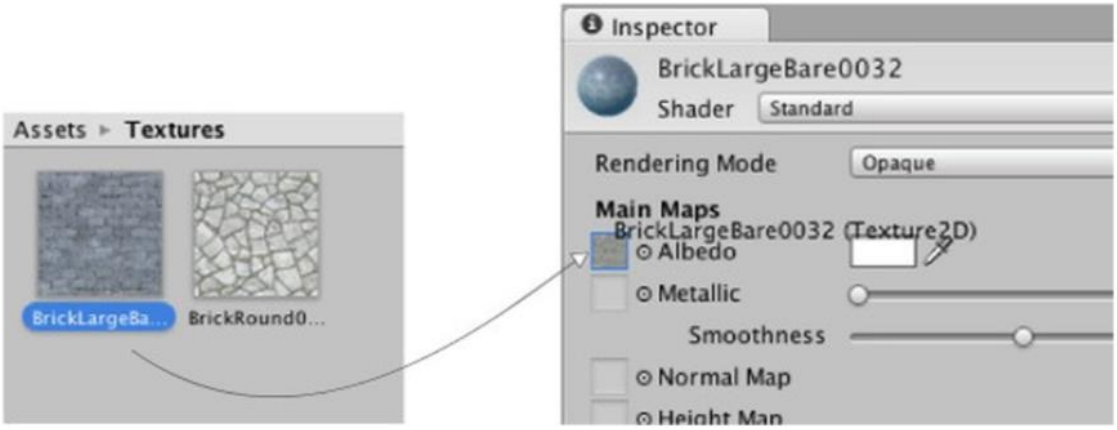
Menerapkan gambar

Secara teknis, tekstur tidak diterapkan pada geometri secara langsung. Sebaliknya, tekstur dapat menjadi bagian dari material, dan material diterapkan pada geometri. Seperti yang dijelaskan dalam pendahuluan, material adalah sekumpulan informasi yang mendefinisikan sifat-sifat permukaan; informasi itu dapat mencakup tekstur untuk ditampilkan di permukaan itu. Tipuan ini penting karena tekstur yang sama dapat digunakan dengan banyak bahan. Karena itu, biasanya setiap tekstur menggunakan bahan yang berbeda, jadi untuk kenyamanan, Unity memungkinkan Anda menjatuhkan tekstur ke objek dan kemudian membuat bahan baru secara otomatis. Jika Anda menyeret tekstur dari tampilan Proyek ke objek dalam scene, Unity akan membuat materi baru dan menerapkan materi baru ke objek; Gambar dibawah ini mengilustrasikan manuver tersebut. Coba sekarang dengan tekstur untuk lantai.



Gambar 4.43 Salah satu cara untuk menerapkan tekstur adalah dengan menyeretnya dari Project ke objek Scene

Selain cara mudah membuat material secara otomatis, cara yang “tepat” untuk membuat material adalah melalui submenu Create pada menu Assets; aset baru akan muncul di tampilan Proyek. Sekarang pilih material untuk menunjukkan propertinya di Inspector dan drag tekstur ke slot tekstur utama; pengaturannya disebut Albedo (itu istilah teknis untuk warna dasar) dan slot teksturnya adalah persegi di samping panel. Sementara itu, drag materi ke atas dari Proyek ke objek di scene untuk menerapkan materi ke objek itu. Coba langkah-langkah ini sekarang dengan tekstur untuk dinding: buat material baru, drag tekstur dinding ke dalam material ini, dan drag material ke dinding dalam scene.



Gambar 4.44 di bawah ini Pilih bahan untuk melihatnya di Inspector, lalu tarik tekstur ke properti material.

Anda sekarang akan melihat gambar batu dan bata muncul di permukaan objek lantai dan dinding, tetapi gambar terlihat agak melebar dan buram. Apa yang terjadi adalah gambar tunggal sedang direntangkan untuk menutupi seluruh lantai. Yang Anda inginkan adalah agar gambar berulang beberapa kali di atas permukaan lantai. Anda dapat mengatur ini menggunakan properti Tiling dari material; pilih material di Project dan kemudian ubah nomor Tiling di Inspector (dengan nilai X dan Y terpisah untuk ubin di setiap arah). Pastikan Anda mengatur ubin peta utama dan bukan peta sekunder (materi ini mendukung peta tekstur sekunder untuk efek lanjutan). Ubin default adalah 1 (itu bukan ubin, dengan gambar direntangkan ke seluruh permukaan); ubah nomornya menjadi sekitar 8 dan lihat apa yang terjadi di scene. Ubah angka di kedua bahan menjadi ubin yang terlihat bagus. Bagus, sekarang scene memiliki tekstur yang diterapkan ke lantai dan dinding! Anda juga dapat menerapkan tekstur ke langit scene; mari kita lihat proses itu.

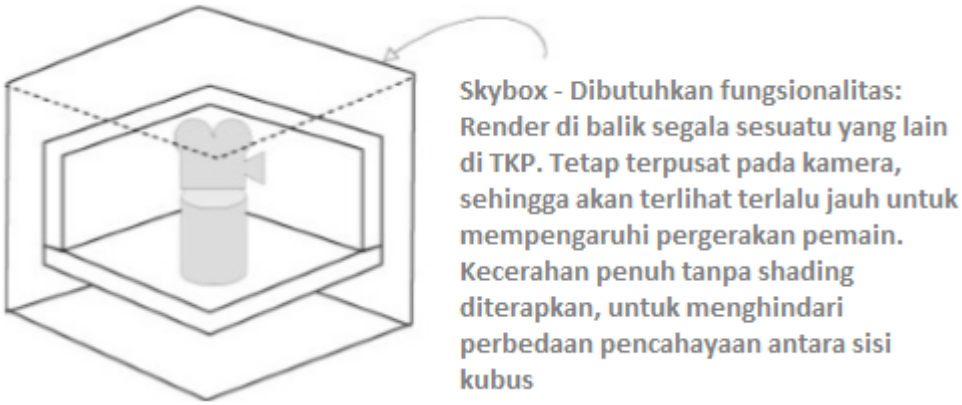
Menghasilkan visual langit menggunakan gambar tekstur

Tekstur batu bata dan batu memberikan tampilan yang jauh lebih alami pada dinding dan lantai. Namun langit saat ini kosong dan tidak alami; kami juga menginginkan tampilan langit yang realistis. Pendekatan paling umum untuk tugas ini adalah jenis tekstur khusus menggunakan gambar langit.

4.10 APA ITU SKYBOX?

Secara default, warna latar belakang kamera adalah biru tua. Biasanya warna itu mengisi area kosong apa pun dari tampilan (misalnya, di atas dinding scene ini), tetapi dimungkinkan untuk membuat gambar langit sebagai latar belakang. Di sinilah konsep skybox masuk.

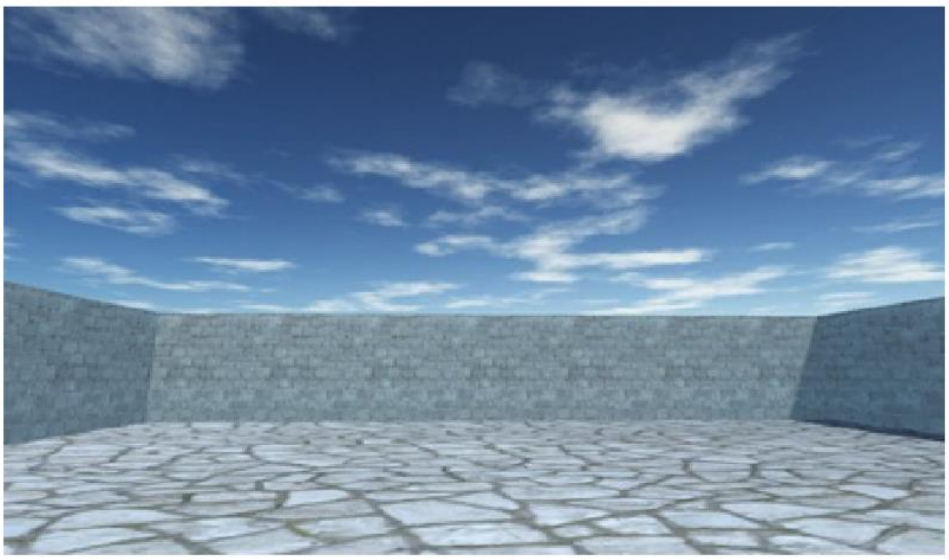
Definisi Skybox adalah kubus yang mengelilingi kamera dengan gambar langit di setiap sisinya. Tidak peduli ke arah mana kamera menghadap, itu melihat gambar langit. Menerapkan skybox dengan benar bisa jadi rumit; Gambar 4.45 menunjukkan diagram cara kerja skybox. Ada sejumlah trik rendering yang diperlukan agar skybox akan muncul sebagai latar belakang yang jauh. Untungnya Unity sudah mengurus semua itu untuk Anda.



Gambar 4.45 Diagram sebuah skybox

Scene baru sebenarnya datang dengan skybox yang sangat sederhana yang sudah ditetapkan. Inilah sebabnya mengapa langit memiliki gradien dari biru muda ke biru tua, bukannya biru tua datar. Jika Anda membuka jendela pencahayaan (Window > Lighting) pengaturan pertama adalah Skybox dan slot untuk pengaturan itu mengatakan Default. Pengaturan ini ada di panel Pencahayaan Lingkungan; jendela ini memiliki sejumlah panel pengaturan yang terkait dengan sistem pencahayaan canggih di Unity, tetapi untuk saat ini kami hanya peduli dengan pengaturan pertama.

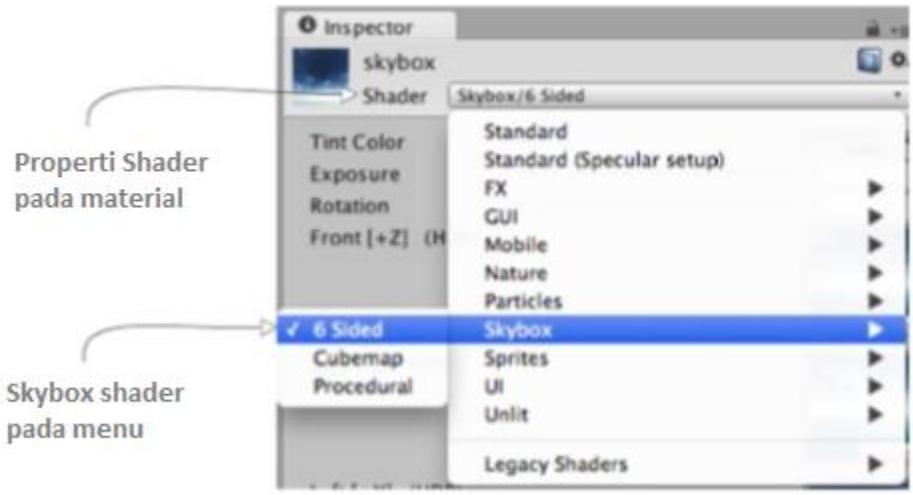
Sama seperti tekstur bata tadi, gambar skybox bisa didapatkan dari berbagai website. Cari tekstur skybox; misalnya, saya memperoleh beberapa skybox hebat dari www.93i.de, termasuk set TropicalSunnyDay. Setelah skybox ini diterapkan ke scene, Anda akan melihat sesuatu seperti gambar dibawah ini.



Gambar 4.46 Scene dengan gambar latar langit

Membuat materi skybox baru

Pertama, buat materi baru (seperti biasa, klik kanan dan Buat, atau pilih Buat dari menu Aset) dan pilih untuk melihat pengaturan di Inspector. Selanjutnya Anda perlu mengubah shader yang digunakan oleh bahan ini. Bagian atas pengaturan material memiliki menu ShaderDi bagian 4.3 kami cukup mengabaikan menu ini karena default berfungsi dengan baik untuk sebagian besar tekstur standar, tetapi skybox memerlukan shader khusus.



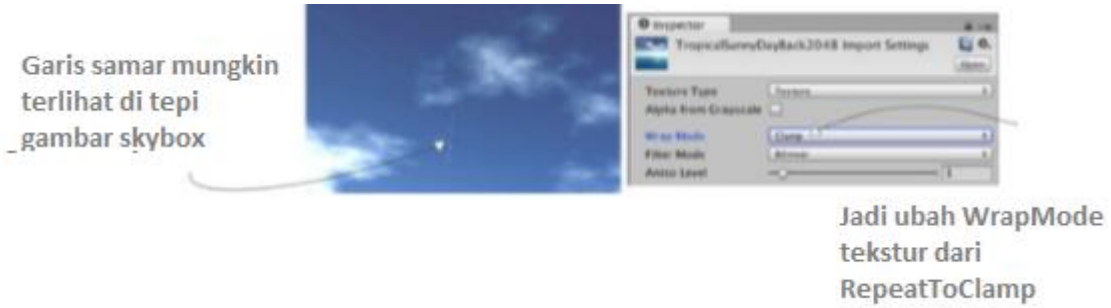
Gambar 4.47 Menu tarik-turun dari shader yang tersedia

Definisi Shader adalah program singkat yang menguraikan instruksi tentang cara menggambar permukaan, termasuk apakah akan menggunakan tekstur apa pun. Komputer menggunakan instruksi ini untuk menghitung piksel saat merender gambar. Shader yang paling umum mengambil warna material dan menggelapkannya sesuai dengan cahaya, tetapi shader juga dapat digunakan untuk semua jenis efek visual. Setiap bahan memiliki shader yang mengontrolnya (Anda bisa menganggap bahan sebagai contoh shader). Material baru diatur ke Standard shader secara default. Shader ini menampilkan warna material (termasuk tekstur) sambil menerapkan warna dasar gelap dan terang di seluruh permukaan.

Untuk skybox ada shader yang berbeda. Klik menu untuk melihat daftar dropdown dari semua shader yang tersedia. Pindah ke bagian Skybox dan pilih 6 Sided di submenu. Dengan shader aktif ini, material sekarang memiliki enam slot tekstur besar (bukan hanya slot tekstur Albedo kecil yang dimiliki shader standar). Keenam slot tekstur ini sesuai dengan enam sisi kubus, jadi gambar-gambar ini harus cocok di tepinya agar tampak mulus.

Gambar 4.48 Enam sisi skybox—gambar untuk atas, bawah, depan, belakang, kiri, dan kanan

Impor gambar skybox ke Unity dengan cara yang sama seperti Anda membawa tekstur bata: drag file ke tampilan Proyek atau klik kanan di Proyek dan pilih Impor Aset Baru. Ada satu pengaturan impor halus untuk diubah; klik tekstur yang diimpor untuk melihat propertinya di Inspector, dan ubah pengaturan Wrap Mode (ditunjukkan pada gambar 4.13) dari Repeat to Clamp (jangan lupa klik Apply setelah selesai). Biasanya tekstur dapat diberi ubin berulang kali di atas permukaan; agar ini tampak mulus, tepi gambar yang berlawanan menyatu. Namun perpaduan tepi ini dapat membuat garis samar di langit tempat gambar bertemu, jadi pengaturan Clamp (mirip dengan fungsi Clamp() di bab 2) akan membatasi batas tekstur dan menghilangkan pencampuran ini.



Gambar 4.49 Perbaiki garis tepi yang samar dengan menyesuaikan mode Bungkus.

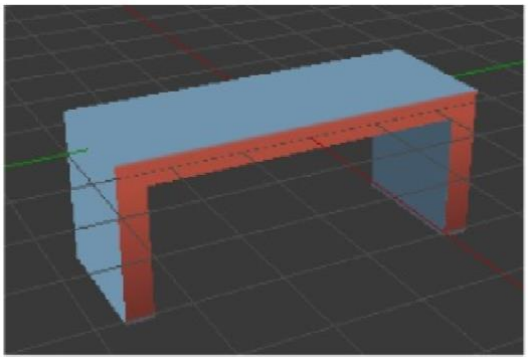
Sekarang Anda dapat menyeret gambar-gambar ini ke slot tekstur material skybox. Nama gambar sesuai dengan slot tekstur yang akan ditetapkan (seperti kiri atau depan). Setelah keenam tekstur terhubung, Anda dapat menggunakan material baru ini sebagai skybox untuk scene. Buka jendela pencahayaan lagi dan atur materi baru ini ke slot Skybox; drag materi ke slot itu, atau klik ikon lingkaran kecil untuk membuka pemilih file.

Tip Secara default, Unity akan menampilkan skybox (atau setidaknya warna utamanya) dalam tampilan Scene editor. Anda mungkin menemukan warna ini mengganggu saat mengedit objek, sehingga Anda dapat mengaktifkan atau menonaktifkan skybox. Di bagian atas panel tampilan Scene adalah tombol yang mengontrol apa yang terlihat; cari tombol Efek untuk mengaktifkan atau menonaktifkan skybox.

Bekerja dengan model 3D khusus

Pada bagian sebelumnya kita telah melihat penerapan tekstur pada dinding datar besar dan lantai pada tingkat tersebut. Tapi bagaimana dengan objek yang lebih detail? Bagaimana jika kita ingin, katakanlah, furnitur menarik di dalam ruangan? Kita dapat melakukannya dengan membangun model 3D di aplikasi seni 3D eksternal. Ingat definisi dari pengantar bab ini: Model 3D adalah objek mesh dalam game (yaitu, bentuk tiga dimensi). Nah, kita akan mengimpor mesh 3D dari bangku sederhana.

Aplikasi yang banyak digunakan untuk memodelkan objek 3D termasuk Maya Autodesk dan 3ds Max. Keduanya adalah alat komersial yang mahal, jadi contoh untuk bab ini menggunakan aplikasi open source Blender. Unduhan sampel menyertakan file .blend yang dapat Anda gunakan; gambar 4.14 menggambarkan model bangku di Blender. Jika Anda tertarik untuk mempelajari cara memodelkan objek Anda sendiri, Anda akan menemukan latihan di lampiran C tentang memodelkan bangku ini di Blender.



Gambar 4.50 Bench Model di Blender

Selain model yang dibuat khusus yang dibuat oleh Anda sendiri atau seniman yang bekerja sama dengan Anda, banyak model 3D tersedia untuk diunduh dari situs web game art.

Salah satu sumber daya yang bagus untuk model 3D adalah Unity's Asset Store di sini: https://www.assetstore.unity3d.com

Format file mana yang harus dipilih?

Sekarang setelah Anda membuat model di Blender, Anda perlu mengekspor aset dari software itu. Sama seperti gambar 2D, sejumlah format file berbeda tersedia untuk Anda gunakan saat mengekspor model 3D, dan jenis file ini memiliki berbagai kelebihan dan kekurangan. Tabel dibawah ini mencantumkan format file 3D yang didukung Unity.

Memilih di antara opsi-opsi ini bermuara pada apakah file tersebut mendukung animasi atau tidak. Karena Collada dan FBX adalah satu-satunya dua opsi yang menyertakan data animasi, itu adalah dua opsi yang harus dipilih. Kapan pun tersedia (tidak semua alat 3D memilikinya sebagai opsi ekspor), ekspor FBX cenderung berfungsi paling baik, tetapi jika Anda menggunakan alat tanpa ekspor FBX, maka Collada juga berfungsi dengan baik. Dalam kasus kami, Blender mendukung ekspor FBX jadi kami akan menggunakan format file itu.

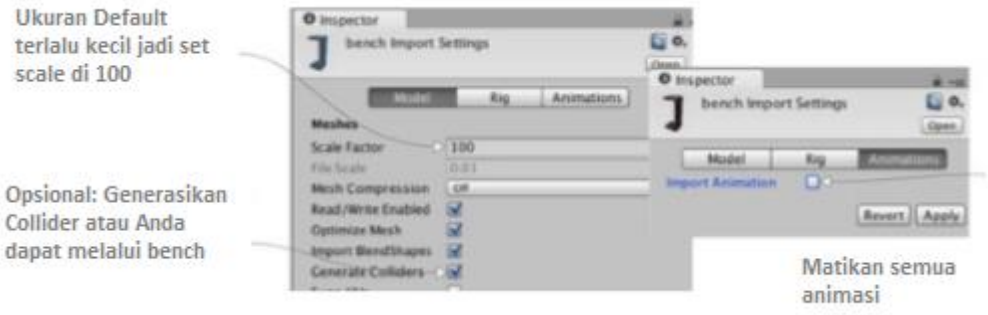
Perhatikan bahwa bagian bawah tabel 4.3 mencantumkan beberapa aplikasi seni 3D. Unity memungkinkan Anda untuk langsung memasukkan file aplikasi tersebut ke dalam proyek Anda, yang tampaknya berguna pada awalnya, tetapi fungsi itu memiliki beberapa peringatan. Sebagai permulaan, Unity tidak memuat file aplikasi tersebut secara langsung; sebagai gantinya, itu mengekspor model di belakang layar dan memuat file yang diekspor itu. Karena model sedang diekspor ke FBX atau Collada, sebaiknya lakukan langkah itu secara eksplisit. Selanjutnya, ekspor ini mengharuskan Anda menginstal aplikasi yang relevan. Persyaratan ini sangat merepotkan jika Anda berencana untuk berbagi file di antara banyak komputer (misalnya, tim developer yang bekerja bersama). Saya tidak menyarankan menggunakan file Blender (atau Maya atau apa pun) langsung di Unity.

Mengekspor dan mengimpor model

Baiklah, saatnya mengekspor model dari Blender dan kemudian mengimpornya ke Unity. Pertama buka bangku di Blender lalu pilih File > Export > FBX. Setelah file disimpan, impor ke Unity dengan cara yang sama seperti Anda mengimpor gambar. Drag file FBX dari komputer ke tampilan Proyek Unity atau klik kanan di Proyek dan pilih Impor Aset Baru. Model 3D akan disalin ke dalam proyek Unity dan siap ditampilkan untuk ditampilkan.

Catatan Unduhan sampel menyertakan file .blend sehingga Anda dapat berlatih mengekspor file FBX dari Blender; bahkan jika Anda tidak membuat model apa pun sendiri, Anda mungkin perlu mengonversi model yang diunduh ke dalam format yang diterima Unity. Jika Anda ingin melewati semua langkah yang melibatkan Blender, gunakan file FBX yang disediakan.

Ada beberapa pengaturan default yang digunakan untuk mengimpor model yang ingin Anda ubah segera. Pertama, Unity secara default mengimpor model ke scale yang sangat kecil, ubah Scale Factor menjadi 100 untuk melawan sebagian dari .01 File Scale. Anda mungkin juga ingin mengklik kotak centang Hasilkan Collider, tetapi itu opsional; tanpa collision Anda bisa berjalan melalui bangku. Kemudian beralih ke tab Animation di pengaturan impor dan batalkan pilihan Impor Animasi (Anda tidak menganimasikan model ini).



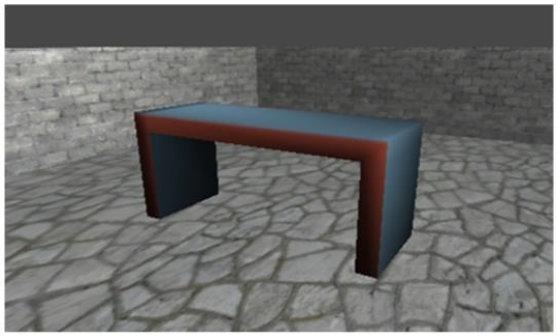
Gambar 4.51 Sesuaikan pengaturan impor untuk model 3D.

Itu menangani mesh yang diimpor. Sekarang untuk teksturnya; ketika Unity mengimpor file FBX, itu juga membuat bahan untuk bangku. Bahan ini secara default kosong (sama seperti bahan baru lainnya), jadi tetapkan tekstur bangku (gambar pada gambar 4.16) dengan cara yang sama seperti Anda menetapkan batu bata ke dinding sebelumnya: drag gambar tekstur ke Project untuk mengimpornya ke Unity, lalu drag tekstur yang diimpor ke slot tekstur bahan bangku. Gambar terlihat agak aneh, dengan bagian gambar yang berbeda muncul di bagian bangku yang berbeda; koordinat tekstur model diedit untuk menentukan pemetaan image-tomesh ini.

Gambar 4.52 Gambar 2D untuk tekstur bangku

Definisi Koordinat tekstur adalah kumpulan nilai tambahan untuk setiap simpul yang menetapkan poligon ke area gambar tekstur. Anggap saja seperti kertas kado; model 3D adalah kotak yang sedang dibungkus, teksturnya adalah kertas pembungkusnya, dan koordinat teksturnya menunjukkan ke mana pada kertas pembungkus itu setiap sisi kotak akan pergi.

Catatan Bahkan jika Anda tidak ingin memodelkan bangku, Anda mungkin ingin membaca penjelasan rinci tentang koordinat tekstur di lampiran C. Konsep koordinat tekstur (serta istilah terkait lainnya seperti UV dan pemetaan) dapat berguna untuk diketahui saat memprogram game. Material baru seringkali terlalu mengkilat, jadi Anda mungkin ingin mengurangi pengaturan Smoothness (permukaan yang lebih halus lebih mengkilap) ke 0. Terakhir, setelah menyesuaikan semuanya sesuai kebutuhan, Anda dapat menempatkan bangku di scene. Drag model ke atas dari tampilan Proyek dan letakkan di satu ruangan di tingkat tersebut; saat Anda menyeret mouse, Anda akan melihatnya di scene. Setelah Anda meletakkannya di tempatnya, Anda akan melihat sesuatu seperti gambar 4.17. Selamat; Anda membuat model bertekstur untuk level!



Gambar 4.53 Bangku impor di tingkat

Catatan Kami tidak akan melakukannya dalam bab ini, tetapi biasanya Anda juga akan mengganti geometri kotak putih dengan model yang dibuat di alat eksternal. Geometri baru mungkin terlihat pada dasarnya identik, tetapi Anda akan memiliki lebih banyak fleksibilitas untuk mengatur UV untuk teksturnya.

4.11 MENGANIMASIKAN KARAKTER DENGAN MECANIM

Model yang kami buat statis, duduk diam di tempat. Anda juga dapat menghidupkan di Blender dan kemudian memutar animasi di Unity. Proses pembuatan animasi 3D itu panjang dan rumit, tapi ini bukan buku tentang animasi jadi kita tidak akan membahasnya di sini. Seperti yang telah disebutkan untuk pemodelan, ada banyak sumber daya yang ada jika Anda ingin mempelajari lebih lanjut tentang animasi 3D. Tetapi berhati-hatilah: ini adalah topik yang sangat besar. Ada alasan mengapa "animator" adalah peran khusus dalam developeran game. Unity memiliki sistem yang canggih untuk mengelola animasi pada model, sistem yang disebut Mecanim. Nama khusus Mecanim mengidentifikasi sistem animasi yang lebih baru dan lebih canggih yang baru-baru ini ditambahkan ke Unity sebagai pengganti sistem animasi yang lebih lama. Sistem yang lebih lama masih ada, diidentifikasi sebagai animasi lawas. Tetapi sistem animasi lawas mungkin akan dihapus dalam versi Unity yang akan datang, di mana Mecanim akan menjadi sistem animasinya.

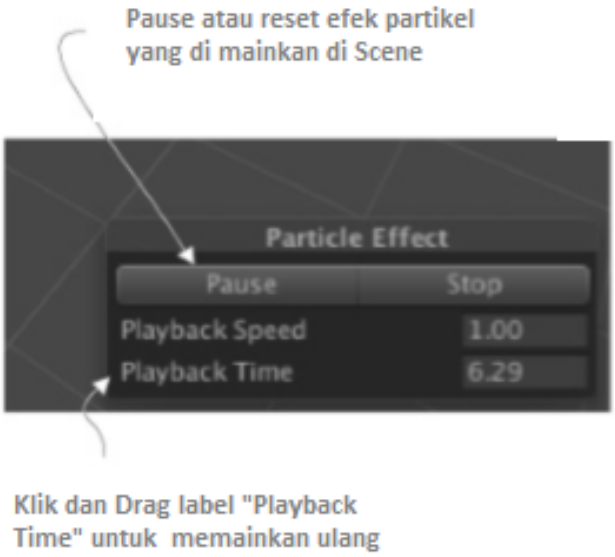
Membuat efek menggunakan sistem partikel

Selain gambar 2D dan model 3D, jenis konten visual lainnya yang dibuat oleh seniman game adalah sistem partikel. Definisi dalam pendahuluan bab ini menjelaskan bahwa sistem partikel adalah mekanisme yang teratur untuk menciptakan dan mengendalikan sejumlah besar objek bergerak. Sistem partikel berguna untuk menciptakan efek visual, seperti api, asap, atau penyemprotan air. Misalnya, efek api pada gambar 4.18 dibuat menggunakan sistem partikel.

Gambar 4.54 Efek api dibuat menggunakan sistem partikel

Sementara sebagian besar aset seni lainnya dibuat dalam alat eksternal dan diimpor ke dalam proyek, sistem partikel dibuat di dalam Unity itu sendiri. Unity menyediakan beberapa alat yang fleksibel dan kuat untuk menciptakan efek partikel. Catatan Sama seperti situasi dengan sistem animasi Mecanim, dulu ada sistem partikel lama dan sistem yang lebih baru memiliki nama khusus, Shuriken. Pada titik ini sistem partikel warisan sepenuhnya dihapus, sehingga nama terpisah tidak lagi diperlukan.

Untuk memulai, buat sistem partikel baru dan saksikan efek default diputar. Dari menu GameObject, pilih Particle System, dan Anda akan melihat puffballs putih dasar menyembur ke atas dari objek baru. Atau lebih tepatnya, Anda akan melihat partikel menyembur ke atas saat Anda memilih objek; saat Anda memilih sistem partikel, panel pemutaran partikel ditampilkan di sudut layar dan menunjukkan berapa lama waktu telah berlalu.

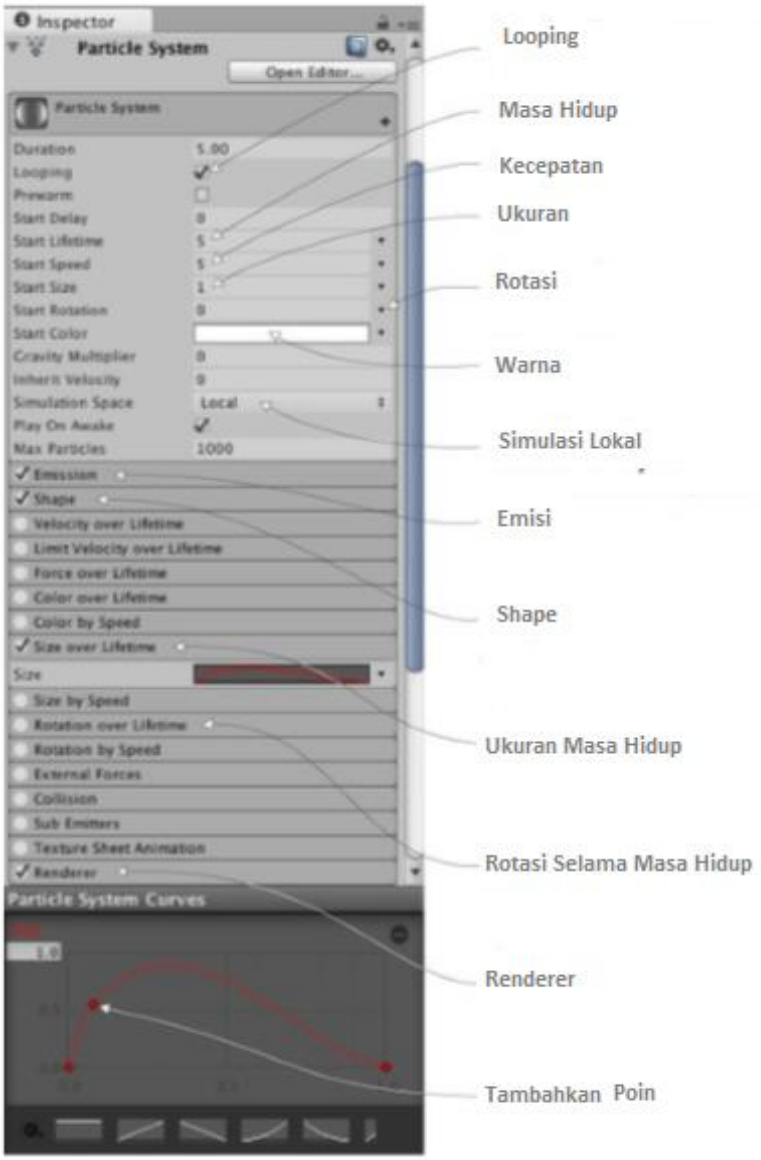


Gambar 4.55 Panel pemutaran untuk sistem partikel

Efek default sudah terlihat cukup rapi, tetapi mari kita lihat daftar parameter yang luas yang dapat Anda gunakan untuk menyesuaikan efeknya.

Menyesuaikan parameter pada efek default

Gambar dibawah inimenunjukkan seluruh daftar pengaturan untuk sistem partikel. Kami tidak akan membahas setiap pengaturan dalam daftar itu; sebagai gantinya, kita akan melihat pengaturan yang relevan untuk membuat efek api. Setelah Anda memahami bagaimana beberapa pengaturan bekerja, sisanya harus cukup jelas. Masing-masing label pengaturan sebenarnya adalah keseluruhan panel informasi. Awalnya hanya panel informasi pertama yang diperluas; sisa panel runtuh. Klik pada label pengaturan untuk memperluas panel informasi tersebut.



Gambar 4.56 Inspector menampilkan pengaturan untuk sistem partikel (menunjukkan pengaturan untuk efek api).

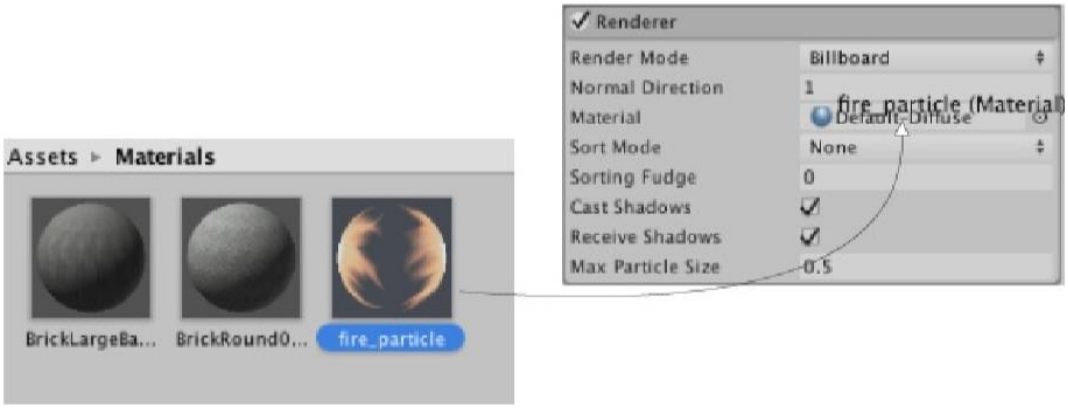
Tip Banyak pengaturan dikendalikan oleh kurva yang ditampilkan di bagian bawah Inspector. Kurva tersebut menunjukkan bagaimana nilai berubah dari waktu ke waktu: sisi kiri grafik adalah saat partikel pertama kali muncul, sisi kanan adalah saat partikel hilang, bagian bawah adalah nilai 0, dan bagian atas adalah nilai maksimum. Drag titik di sekitar grafik, dan klik dua kali atau klik kanan pada kurva untuk menyisipkan titik baru.

Menerapkan tekstur baru untuk api

Sekarang sistem partikel lebih terlihat seperti semburan api, tetapi efeknya masih membutuhkan partikel agar terlihat seperti api, bukan gumpalan putih. Itu membutuhkan mengimpor gambar baru ke Unity. Saya membuat titik oranye dan menggunakan alat Smudge untuk menggambar sulur api (dan kemudian saya menggambar hal yang sama dengan warna kuning). Baik Anda menggunakan gambar ini dari proyek sampel, menggambar sendiri, atau mengunduh yang serupa, Anda perlu mengimpor file gambar ke Unity. Seperti yang dijelaskan sebelumnya, drag file gambar ke tampilan Proyek, atau pilih Aset > Impor Aset Baru.

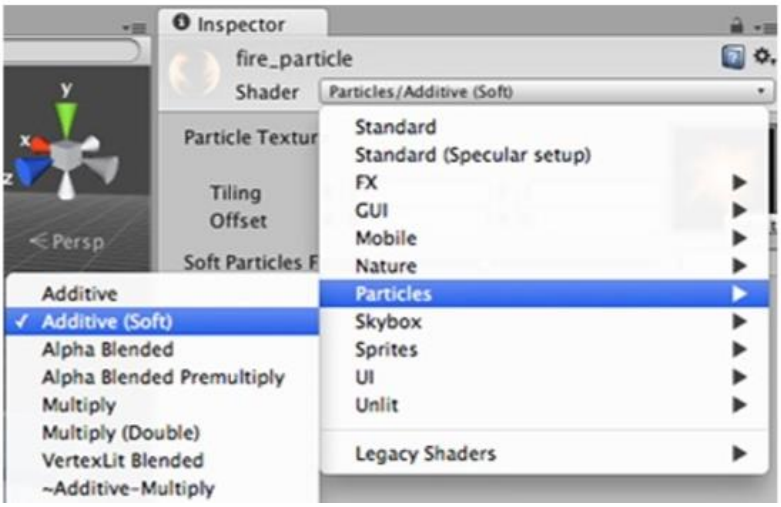
Gambar 4.57 Gambar yang digunakan untuk partikel api

Sama seperti model 3D, tekstur tidak diterapkan ke sistem partikel secara langsung; Anda menambahkan tekstur ke bahan dan menerapkan bahan itu ke sistem partikel. Buat material baru dan kemudian pilih untuk melihat propertinya di Inspector. Drag gambar api dari Proyek ke slot tekstur. Itu menghubungkan tekstur api ke material api, jadi sekarang Anda ingin menerapkan material ke sistem partikel. Gambar 4.22 menunjukkan bagaimana melakukan ini; pilih sistem partikel, luaskan Renderer di bagian bawah pengaturan, dan drag material ke slot Material.



Gambar 4.58 Tetapkan bahan ke sistem partikel

Seperti yang Anda lakukan untuk material skybox, Anda perlu mengubah shader untuk material partikel. Klik menu Shader di dekat bagian atas pengaturan material untuk melihat daftar shader yang tersedia. Alih-alih standar standar, bahan untuk partikel membutuhkan salah satu shader di bawah submenu Partikel. Seperti terlihat pada gambar 4.23, dalam hal ini kita menginginkan Additive (Lembut). Ini akan membuat partikel tampak kabur dan mencerahkan scene, seperti api.



Gambar 4.59 Mengatur shader untuk bahan partikel api

Definisi Aditif adalah shader yang menambahkan warna partikel ke warna di belakangnya, bukan mengganti piksel. Ini membuat piksel lebih cerah dan membuat hitam pada partikel menjadi tidak terlihat. Kebalikannya adalah Multiply, yang membuat segalanya lebih gelap; shader ini memiliki efek visual yang sama dengan efek layer Additive dan Multiply di Photoshop. Dengan bahan api yang ditetapkan untuk efek partikel api, sekarang akan terlihat seperti efek yang ditunjukkan sebelumnya pada gambar 4.18. Ini terlihat seperti semburan api yang cukup meyakinkan, tetapi efeknya tidak hanya bekerja saat duduk diam; selanjutnya mari kita tempelkan ke objek yang bergerak.

Melampirkan efek partikel ke objek 3D

Buat sebuah bola (ingat, GameObject > 3D Object > Sphere). Buat skrip baru bernama BackAndForth, seperti yang ditunjukkan pada daftar berikut, dan lampirkan ke sphere baru.
Daftar Memindahkan objek bolak-balik di sepanjang jalan yang lurus

```
using UnityEngine;
using System.Collections;

public class BackAndForth : MonoBehaviour {
    public float speed = 3.0f;
    public float maxZ = 16.0f;
    public float minZ = -16.0f;

    private int _direction = 1;

    void Update() {
        transform.Translate(0, 0, _direction * speed * Time.deltaTime);

        bool bounced = false;
        if (transform.position.z > maxZ || transform.position.z < minZ) {
            _direction = -_direction;
            bounced = true;
        }
        if (bounced) {
            transform.Translate(0, 0, _direction * speed * Time.deltaTime);
        }
    }
}
```

Jalankan skrip ini dan bola meluncur bolak-balik di koridor tengah level. Sekarang Anda dapat membuat sistem partikel menjadi anak dari bola dan api akan bergerak dengan bola. Sama seperti dengan dinding level, dalam tampilan Hierarki drag objek partikel ke objek bola. Peringatan Anda biasanya harus mengatur ulang posisi suatu objek setelah menjadikannya anak dari objek lain. Misalnya, kita ingin sistem partikel pada 0, 0, 0 (ini relatif terhadap induknya). Unity akan mempertahankan penempatan objek dari sebelum ditautkan sebagai anak.

Membangun game Memori menggunakan fungsionalitas 2D baru Unity dan Meletakkan GUI 2D ke Game 3D

Sampai sekarang kami telah bekerja dengan grafik 3D. Tetapi Anda juga dapat bekerja dengan grafik 2D di Unity, jadi dalam bab ini Anda akan membuat game 2D untuk mempelajarinya. Kami akan mengembangkan Memori permainan anak-anak klasik: kami akan menampilkan kisi-kisi bagian belakang kartu, memperlihatkan bagian depan kartu saat diklik, dan mencetak kecocokan. Mekanik ini mencakup dasar-dasar yang perlu Anda ketahui untuk mengembangkan game 2D di Unity.

Meskipun Unity berasal dari alat untuk game 3D, Unity juga sering digunakan untuk game 2D. Versi terbaru Unity (dimulai dengan versi 4.3, dirilis menjelang akhir tahun 2013) telah menambahkan kemampuan untuk menampilkan grafik 2D, tetapi bahkan sebelum itu game 2D sudah dikembangkan di Unity (terutama game seluler yang memanfaatkan lintas platform Unity alam). Di Unity versi sebelumnya, developer game memerlukan kerangka kerja pihak ketiga (seperti 2D Toolkit dari Unikron Software) untuk meniru grafik 2D dalam scene 3D Unity. Akhirnya editor inti dan game engine dimodifikasi untuk menggabungkan grafik 2D, dan bab ini akan mengajarkan Anda tentang fungsionalitas yang lebih baru itu.

Alur kerja 2D di Unity kurang lebih sama dengan alur kerja untuk mengembangkan game 3D: mengimpor aset seni, menyeretnya ke dalam scene, dan menulis skrip untuk dilampirkan ke objek. Jenis utama aset seni dalam grafik 2D disebut sprite.

Definisi Sprite adalah gambar 2D yang ditampilkan langsung di layar, bukan gambar ditampilkan pada permukaan model 3D (yaitu, tekstur). Anda dapat mengimpor gambar 2D ke Unity sebagai sprite dengan cara yang sama seperti Anda mengimpor gambar sebagai tekstur (lihat bab 4). Secara teknis sprite ini akan menjadi objek dalam ruang 3D, tetapi mereka akan menjadi permukaan datar yang berorientasi sepanjang sumbu Z. Karena mereka semua akan menghadap ke arah yang sama, Anda dapat mengarahkan kamera langsung ke sprite dan player hanya akan dapat melihat gerakan mereka di sepanjang sumbu X dan Y (yaitu, dua dimensi).

Menyiapkan semuanya untuk grafik 2D

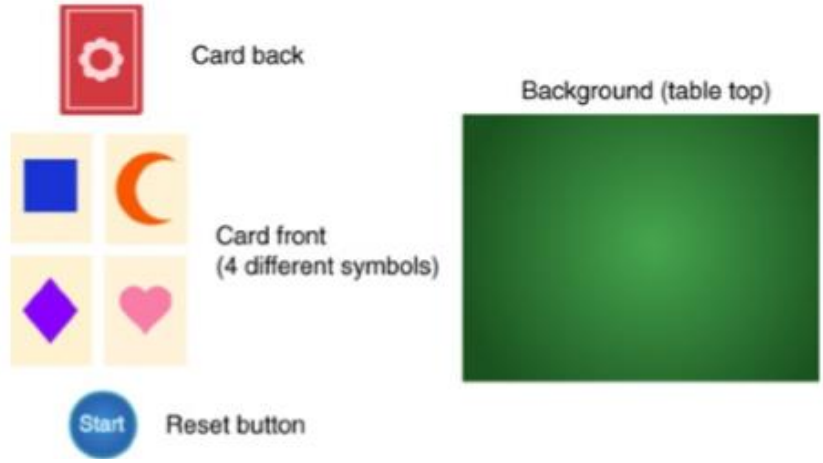
Kami akan membuat game klasik Memory. Bagi mereka yang tidak terbiasa dengan permainan ini, serangkaian kartu akan dibagikan tertutup. Setiap kartu akan memiliki kartu yang cocok yang terletak di tempat lain, tetapi player tidak dapat membedakan apa itu berbagai kartu. Player dapat membalik dua kartu sekaligus, mencoba menemukan kartu yang cocok; jika dua kartu yang dipilih tidak cocok, mereka akan membalik kembali dan kemudian player dapat menebak lagi.

Gambar 4.60 Mockup seperti apa game Memory itu

Perhatikan bahwa mockup kali ini menggambarkan dengan tepat apa yang akan dilihat player (sedangkan mockup untuk scene 3D menggambarkan ruang di sekitar player dan kemudian ke mana kamera mengarahkan player untuk melihat). Sekarang setelah Anda tahu apa yang akan kami bangun, saatnya untuk mulai bekerja.

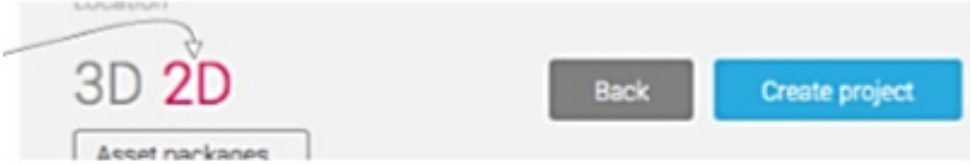
Mempersiapkan proyek

Langkah pertama adalah mengumpulkan dan menampilkan grafik untuk game kita. Dengan cara yang sama seperti membangun demo 3D sebelumnya, Anda ingin memulai game baru dengan menyusun set grafis minimum agar game dapat beroperasi, dan setelah itu Anda dapat mulai memprogram fungsionalitasnya.



Gambar 4.61 Aset seni yang dibutuhkan untuk permainan Memori

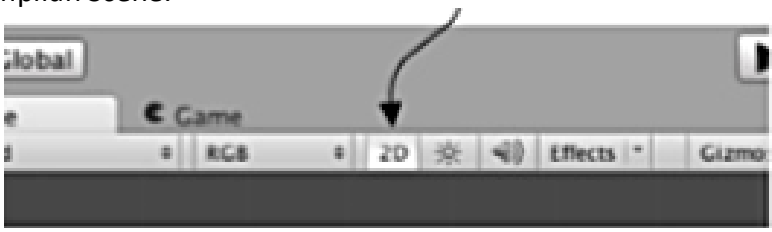
Kumpulkan gambar yang dibutuhkan, lalu buat new project di Unity. Di jendela New project yang muncul, Anda akan melihat beberapa tombol di bagian bawah yang memungkinkan Anda beralih antara mode 2D dan 3D. Dalam bab-bab sebelumnya, kami telah bekerja dengan grafik 3D, dan karena itu adalah nilai default, kami tidak memperhatikan pengaturan ini. Namun, dalam bab ini, Anda ingin beralih ke mode 2D saat membuat new project.



Gambar 4.62 Buat new project dalam mode 2D atau 3D dengan tombol ini.

Mode Editor 2D dan tampilan scene 2D

Pengaturan 2D/3D untuk new project menyesuaikan dua pengaturan berbeda dalam editor Unity, yang keduanya dapat Anda sesuaikan secara manual nanti jika diinginkan. Kedua pengaturan tersebut adalah mode Editor 2D dan tampilan Scene 2D. Tampilan Scene 2D mengontrol bagaimana scene ditampilkan dalam Unity; alihkan tombol 2D di sepanjang bagian atas tampilan Scene.



Gambar 4.63 Beralih tampilan adegan 2d

Anda mengatur mode Editor 2D dengan membuka menu Edit dan memilih Editor dari menu drop-down Project Settings. Dalam pengaturan tersebut, Anda akan melihat pengaturan Mode Perilaku Default dengan pilihan untuk 3D atau 2D.

Gambar 4.64 Pengaturan editor

Menyetel editor ke mode 2D menyebabkan gambar yang diimpor disetel ke Sprite; seperti yang Anda lihat di bab 4, biasanya gambar diimpor sebagai tekstur. Mode Editor 2D juga menyebabkan scene baru tidak memiliki pengaturan pencahayaan 3D default; pencahayaan ini tidak merusak scene 2D, tetapi itu tidak perlu. Jika Anda perlu menghapusnya secara manual, hapus lampu arah yang datang dengan scene baru dan matikan skybox di jendela pencahayaan (klik ikon lingkaran kecil untuk pemilih file dan pilih Tidak ada dari daftar). Dengan new project untuk bab ini yang dibuat dan disetel untuk 2D, kita dapat mulai menempatkan gambar kita ke dalam scene.

Menampilkan gambar 2D (alias sprite)

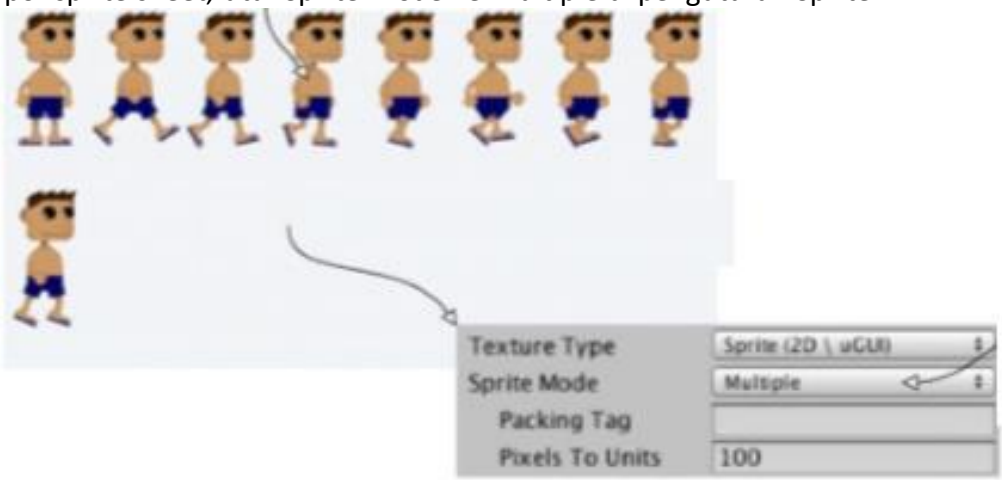
Drag semua file gambar ke tampilan Proyek untuk mengimpornya; pastikan gambar diimpor sebagai sprite dan bukan tekstur. (Ini otomatis jika editor diatur ke 2D. Pilih aset untuk melihat pengaturan impornya di Inspector.) Sekarang drag sprite table_top (gambar latar belakang kita) ke atas dari tampilan Project ke scene kosong, lalu simpan scene . Seperti objek mesh, di Inspector ada komponen Transform untuk sprite; ketik 0, 0, 5 untuk memposisikan gambar latar belakang.

Tip Pengaturan impor lain yang perlu diperhatikan adalah Pixels-To-Units. Karena Unity sebelumnya adalah mesin 3D yang baru-baru ini memiliki grafik 2D yang dicangkokkan, satu unit di Unity belum tentu satu piksel dalam gambar. Anda dapat mengatur pengaturan Pixels-To-Units ke 1:1 tetapi saya sarankan untuk membiarkannya pada default 100:1 (karena mesin fisika tidak bekerja dengan baik pada 1:1, dan default lebih baik untuk kompatibilitas dengan yang lain ' kode).

Sprite animasi

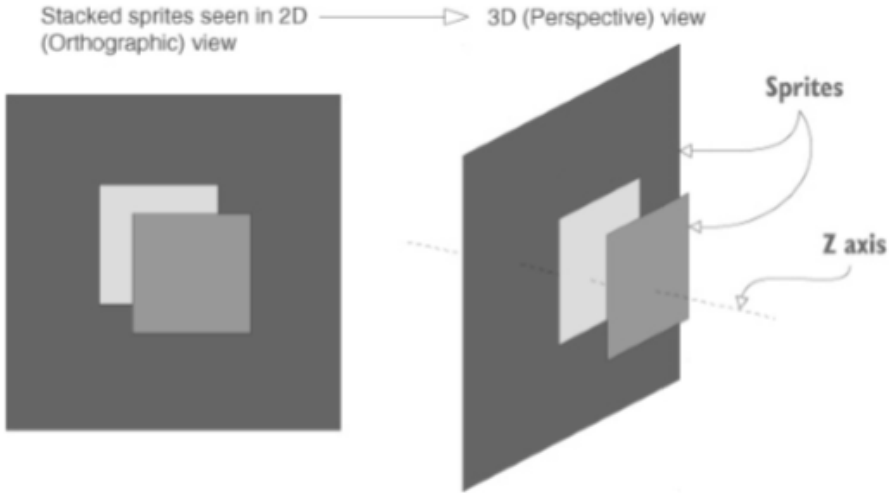
Meskipun kami hanya akan menggunakan gambar diam untuk proyek ini, game 2D biasanya memiliki sprite animasi. Sprite animasi dibuat dengan menggambar setiap frame animasi dan kemudian menampilkan frame secara berurutan dalam Unity. Beberapa frame dapat diimpor sebagai gambar terpisah, tetapi game biasanya memiliki semua frame animasi yang diletakkan pada satu gambar, yang disebut sprite sheet. Sprite sheet dapat dibuat secara

otomatis oleh Unity, atau dapat dibuat menggunakan alat seperti Texture Packer. Saat mengimpor sprite sheet, atur Sprite Mode ke Multiple di pengaturan Sprite.



Gambar 4.65 Sprite animasi

Lembar sprite akan tetap muncul dalam tampilan Proyek sebagai aset tunggal, tetapi jika Anda mengklik panah pada aset, lembar itu akan meluas dan menampilkan semua sprite individu. Alih-alih menyeret sprite ke dalam scene satu per satu, Anda dapat memilih sekelompok untuk didrag bersama. 0 untuk posisi X dan Y sangat mudah (sprite ini akan memenuhi seluruh layar, jadi Anda menginginkannya di tengah), tetapi 5 untuk posisi Z mungkin tampak aneh. Untuk grafik 2D, bukankah hanya X dan Y yang penting? Nah, X dan Y adalah satu-satunya koordinat yang penting untuk memposisikan objek pada layar 2D; Koordinat Z masih penting untuk menumpuk objek di atas satu sama lain. Nilai Z yang lebih rendah lebih dekat ke kamera, sehingga sprite dengan nilai Z yang lebih rendah ditampilkan di atas sprite lainnya. Dengan demikian, sprite latar belakang harus memiliki nilai Z tertinggi. Kami akan mengatur latar belakang kami ke posisi Z positif, dan kemudian memberikan yang lainnya posisi 0 atau Z negatif.



Gambar 4.66 Bagaimana sprite menumpuk di sepanjang sumbu Z

Sprite lain akan diposisikan dengan nilai hingga dua tempat desimal karena pengaturan Pixels-To-Units yang disebutkan sebelumnya. Rasio 100:1 berarti 100 piksel dalam gambar adalah 1 unit dalam Unity; dengan kata lain, 1 piksel adalah 0,01 unit. Tapi sebelum kita memasukkan sprite lagi ke dalam scene, mari kita siapkan kamera untuk game ini.

Membuat atlas menggunakan Sprite Packer

Seperti yang disebutkan di sidebar "Animated Sprite", Anda dapat memiliki beberapa sprite yang ditata dalam satu gambar. Gambar biasanya disebut sprite sheet ketika beberapa frame dari satu animasi 2D digabungkan menjadi satu, tetapi istilah yang lebih umum untuk beberapa gambar yang digabungkan menjadi satu adalah atlas.

Lembar sprite berguna untuk menyatukan bingkai animasi, tetapi atlas sprite juga sering digunakan untuk gambar diam. Itu karena atlas dapat mengoptimalkan kinerja sprite dalam dua cara: 1) dengan mengurangi jumlah ruang yang terbuang dalam gambar dengan mengemasnya dengan rapat, dan 2) dengan mengurangi panggilan undian dari kartu video (setiap gambar baru yang dimuat menyebabkan sedikit lebih banyak pekerjaan untuk kartu video).

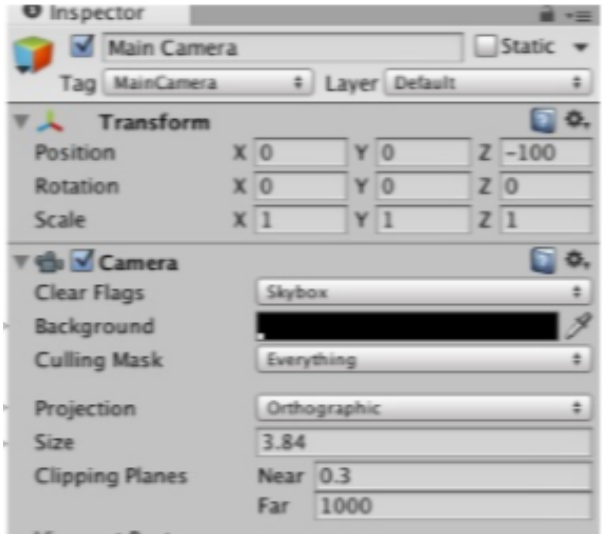
Atlas sprite dapat dibuat menggunakan alat eksternal (beralih ke Beberapa dalam pengaturan Sprite) dan pendekatan itu pasti akan berhasil. Tetapi Unity menyertakan Sprite Packer yang akan mengemas beberapa sprite secara otomatis. Untuk menggunakan fitur ini, aktifkan Sprite Packer di pengaturan Editor (ditemukan di bawah Edit > Pengaturan Proyek). Sekarang tulis nama di opsi Tag Pengemasan saat melihat pengaturan Impor gambar sprite; Unity akan mengemas sprite dengan label pengepakan yang sama menjadi satu atlas. Untuk informasi lebih lanjut, lihat dokumentasi Unity: http://docs.unity3d.com/Manual/SpritePacker.html

Mengalihkan kamera ke mode 2D

Sekarang mari kita sesuaikan pengaturan pada kamera utama di scene. Anda mungkin berpikir bahwa karena tampilan Scene diatur ke 2D, apa yang Anda lihat di Unity adalah apa yang akan Anda lihat di game. Agak non-intuitif, meskipun, bukan itu masalahnya.

Peringatan Apakah tampilan Scene disetel ke 2D atau tidak, tidak ada hubungannya dengan tampilan kamera dalam game yang sedang berjalan. Ternyata terlepas dari apakah tampilan Scene diatur ke mode 2D, kamera dalam game diatur secara independen. Ini bisa berguna dalam banyak situasi sehingga Anda dapat mengubah tampilan Scene kembali ke 3D untuk bekerja pada efek tertentu di dalam scene. Putusnya hubungan ini berarti bahwa apa yang Anda lihat di Unity belum tentu seperti yang Anda lihat di game, dan mudah bagi pemula untuk melupakannya.

Pengaturan kamera yang paling penting untuk disesuaikan adalah Proyeksi. Proyeksi kamera mungkin sudah benar karena Anda membuat new project dalam mode 2D, tetapi ini tetap penting untuk diketahui dan perlu diperiksa ulang. Pilih kamera di Hierarchy untuk menampilkan pengaturannya di Inspector, lalu cari pengaturan Projection (lihat gambar 4.67). Untuk grafik 3D, pengaturannya harus Perspektif, tetapi untuk grafik 2D, proyeksi kamera harus Ortografis.



Gambar 4.67 Pengaturan kamera untuk menyesuaikan grafik 2D

Definisi Ortografi adalah istilah untuk tampilan kamera datar yang tidak memiliki perspektif yang jelas. Ini adalah kebalikan dari kamera Perspektif, di mana objek yang lebih dekat tampak lebih besar dan garis semakin menjauh. Meskipun mode Proyeksi adalah pengaturan kamera yang paling penting untuk grafik 2D, ada beberapa pengaturan lain untuk kita sesuaikan juga. Selanjutnya kita akan melihat Ukuran; pengaturan itu di bawah Proyeksi. Ukuran ortografis kamera menentukan ukuran tampilan kamera dari bagian tengah layar hingga bagian atas layar. Dengan kata lain, atur Ukuran ke setengah dimensi piksel layar yang Anda inginkan. Jika nanti Anda menyetel resolusi game yang digunakan ke dimensi piksel yang sama, Anda akan mendapatkan grafis piksel yang sempurna.

Definisi Piksel-sempurna berarti satu piksel pada layar sesuai dengan satu piksel dalam gambar (jika tidak, kartu video akan membuat gambar agak buram saat diperbesar agar sesuai dengan layar). Misalnya, katakanlah Anda menginginkan layar 1024x768 piksel yang sempurna. Itu berarti tinggi kamera harus 384 piksel. Bagilah dengan 100 (karena scale piksel ke unit) dan Anda mendapatkan 3,84 untuk ukuran kamera. Sekali lagi, matematika itu adalah SCREEN_SIZE 2 100f (f seperti dalam float, bukan nilai int). Mengingat gambar latar belakang adalah 1024x768 (pilih aset untuk memeriksa dimensinya), maka jelas nilai 3,84 inilah yang kita inginkan untuk kamera kita.

Dua penyesuaian yang tersisa untuk dilakukan di Inspector adalah warna latar belakang kamera dan posisi Z. Seperti yang disebutkan sebelumnya untuk sprite, posisi Z yang lebih tinggi berada lebih jauh ke dalam scene. Jadi kamera harus memiliki posisi Z yang cukup rendah; atur posisi kamera ke 0, 0, -100. Warna latar belakang kamera mungkin harus hitam; warna defaultnya adalah biru, dan itu akan terlihat aneh ditampilkan di sepanjang sisi jika layar lebih lebar dari gambar latar belakang (yang mungkin). Klik contoh warna di sebelah Latar Belakang dan atur pemilih warna menjadi hitam.

Sekarang simpan scene sebagai Scene dan tekan Play; Anda akan melihat tampilan Game diisi dengan sprite meja kami. Seperti yang Anda lihat, sampai ke titik ini tidak sepenuhnya jelas (sekali lagi, itu karena Unity adalah game engine 3D yang baru-baru ini memiliki grafik 2D yang dicangkokkan). Tapi bagian atas meja benar-benar kosong, jadi langkah kita selanjutnya adalah meletakkan kartu di atas meja.

Membangun objek kartu dan membuatnya bereaksi terhadap klik

Sekarang setelah semua gambar diimpor dan siap digunakan, mari kita buat objek kartu yang membentuk inti dari game ini. Di Memori, semua kartu pada awalnya menghadap

ke bawah, dan hanya sementara menghadap ke atas saat Anda memilih sepasang kartu untuk dibalik. Untuk mengimplementasikan fungsi ini, kita akan membuat objek yang terdiri dari beberapa sprite yang ditumpuk di atas satu sama lain. Kemudian kita akan menulis kode yang membuat kartu muncul saat diklik dengan mouse.

Membangun objek dari sprite

Drag salah satu gambar kartu ke dalam scene. Gunakan salah satu bagian depan kartu, karena Anda akan menambahkan kartu di bagian atas untuk menyembunyikan gambar. Secara teknis posisi sekarang tidak masalah, tetapi pada akhirnya itu akan menjadi masalah sehingga Anda juga dapat menempatkan kartu pada -3, 1, 0. Sekarang drag sprite card_back ke TKP. Jadikan sprite baru ini sebagai anak dari sprite kartu sebelumnya (ingat, dalam Hierarki tarik objek anak ke objek induk) dan kemudian atur posisinya menjadi 0, 0, -.1 (Perlu diingat bahwa posisi ini relatif terhadap orang tua, jadi ini berarti "Letakkan di X Y yang sama tetapi pindahkan lebih dekat ke Z.")

Tip Alih-alih alat Pindah, Putar, dan Scale yang kami gunakan dalam 3D, dalam mode 2D kami menggunakan alat manipulasi tunggal yang disebut Rect Tool. Dalam mode 2D, alat ini dipilih secara otomatis, atau Anda dapat mengklik tombol navigasi paling kanan di sudut kiri atas Unity. Dengan alat ini aktif, klik dan drag objek untuk melakukan ketiga operasi (Move/Rotate/Scale) dalam dua dimensi.

Gambar 4.68 link Hierarchy dan pemposisian untuk bavk sprite

Kode Input Mouse

Untuk merespons saat player mengkliknya, sprite kartu harus memiliki komponen clasher. Sprite baru tidak memiliki penabrakan secara default, sehingga tidak dapat diklik. Kami akan menempelkan penumbuk ke objek kartu root, tetapi tidak ke bagian belakang kartu, sehingga hanya bagian depan kartu dan bukan bagian belakang kartu yang akan menerima klik mouse. Untuk melakukan ini, pilih objek kartu root di Hierarchy (jangan klik kartu di scene, karena bagian belakang kartu ada di atas dan Anda akan memilih bagian itu) lalu klik tombol Tambahkan Komponen di Inspector. Pilih Fisika 2D (bukan Fisika, karena sistem itu untuk fisika 3D dan ini adalah permainan 2D), lalu pilih penumbuk kotak.

Selain Collider, kartu membutuhkan skrip agar reaktif terhadap player yang mengkliknya, jadi mari kita tulis beberapa kode. Buat skrip baru bernama MemoryCard.cs dan lampirkan skrip ini ke objek kartu root (sekali lagi, bukan kartu belakang). Daftar berikut menunjukkan kode yang membuat kartu mengeluarkan pesan debug saat diklik.

Daftar Memancarkan pesan debug saat diklik

```
using UnityEngine;
using System.Collections;

public class MemoryCard : MonoBehaviour {
    public void OnMouseDown() {
        Debug.Log("testing 1 2 3");
    }
}
```

Jika Anda belum terbiasa, mengatur aset Anda ke dalam folder terpisah mungkin merupakan ide yang bagus; buat folder untuk skrip dan drag file dalam tampilan Proyek. Berhati-hatilah untuk menghindari nama folder khusus yang ditanggapi Unity: Sumber Daya, Plugin, Editor, dan Gizmos. Nanti di buku ini kita akan membahas apa yang dilakukan beberapa folder khusus ini, tetapi untuk saat ini hindari memberi nama folder apa pun dengan kata-kata itu.

Bagus, kita bisa klik kartunya sekarang! Sama seperti Update(), OnMouseDown() adalah fungsi lain yang disediakan oleh MonoBehaviour, kali ini merespons saat objek diklik. Mainkan game dan tonton pesan yang muncul di konsol. Tapi ini hanya mencetak ke konsol untuk pengujian; kami ingin kartu itu terungkap.

Mengungkap kartu saat diklik

Tulis ulang kode agar sesuai dengan apa yang ditampilkan di daftar berikutnya (kode belum berjalan cukup lama, tetapi jangan khawatir).

Daftar Script yang menyembunyikan bagian belakang ketika kartu diklik

```
using UnityEngine;
using System.Collections;

public class MemoryCard : MonoBehaviour {
    [SerializeField] private GameObject cardBack;

    public void OnMouseDown() {
        if (cardBack.activeSelf) {
            cardBack.SetActive(false);
        }
    }
}
```

Ada dua tambahan kunci pada skrip: referensi ke objek dalam scene, dan metode SetActive() yang menonaktifkan objek itu. Bagian pertama, referensi ke objek dalam scene, mirip dengan apa yang telah kita lakukan di bab sebelumnya: tandai variabel sebagai serial, lalu drag objek dari Hierarchy ke variabel di Inspector. Dengan set referensi objek, kode sekarang akan memengaruhi objek dalam scene.

Penambahan kunci kedua pada kode adalah perintah SetActive. Perintah itu akan menonaktifkan GameObject apa pun, membuat objek itu tidak terlihat. Jika sekarang kita menyeret card_back dalam scene ke variabel skrip ini di Inspector, saat Anda memainkan game, kartu kembali menghilang saat Anda mengklik kartu. Menyembunyikan bagian belakang kartu akan memperlihatkan bagian depan kartu; kami telah menyelesaikan tugas penting lainnya untuk game Memori! Tapi ini masih hanya satu kartu, jadi sekarang mari kita buat banyak kartu.

Menampilkan berbagai gambar kartu

Kami telah memprogram objek kartu yang awalnya menunjukkan kartu kembali tetapi muncul sendiri saat diklik. Itu adalah satu kartu, tetapi gim ini membutuhkan seluruh kotak

kartu, dengan gambar berbeda di sebagian besar kartu. Kami akan menerapkan kisi-kisi kartu menggunakan beberapa konsep yang terlihat di bab sebelumnya, bersama dengan beberapa konsep baru yang belum pernah Anda lihat sebelumnya. Bab 3 mencakup pengertian 1) menggunakan komponen SceneController yang tidak terlihat dan 2) membuat klon objek. Kali ini SceneController akan menerapkan gambar yang berbeda ke kartu yang berbeda.

Memuat gambar secara terprogram

Ada empat gambar kartu dalam game yang kami buat. Kedelapan kartu di atas meja (dua untuk setiap simbol) akan dibuat dengan mengkloning kartu asli yang sama, jadi awalnya semua kartu akan memiliki simbol yang sama. Kita harus mengubah gambar pada kartu dalam skrip, memuat gambar yang berbeda secara terprogram.

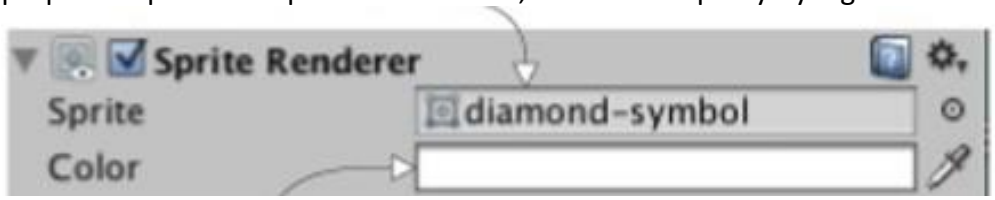
Untuk memeriksa bagaimana gambar dapat ditetapkan secara terprogram, mari kita tulis beberapa kode pengujian sederhana (yang akan diganti nanti) untuk mendemonstrasikan tekniknya. Pertama tambahkan kode dari daftar berikut ke skrip MemoryCard.

Daftar Kode uji untuk mendemonstrasikan mengubah gambar sprite

```
...
[SerializeField] private Sprite image;
void Start() {
    GetComponent<SpriteRenderer>().sprite = image;
}
...
```

Setelah Anda menyimpan skrip ini, variabel gambar baru akan muncul di Inspector karena telah disetel sebagai serial. Drag sprite ke atas dari tampilan Proyek (pilih salah satu gambar kartu, dan tidak sama dengan gambar yang sudah ada di scene) dan letakkan di slot Gambar. Sekarang jalankan scenenya, dan Anda akan melihat gambar baru di kartu.

Kunci untuk memahami kode ini adalah mengetahui tentang komponen SpriteRenderer. Anda akan melihat pada gambar 5.7 bahwa objek kartu belakang hanya memiliki dua komponen, komponen Transform standar pada semua objek dalam scene, dan komponen baru yang disebut Sprite Renderer. Komponen ini menjadikannya sebagai objek sprite dan menentukan aset sprite mana yang akan ditampilkan. Perhatikan bahwa properti pertama dalam komponen disebut Sprite dan menautkan ke salah satu sprite dalam tampilan Project; properti dapat dimanipulasi dalam kode, dan itulah tepatnya yang dilakukan skrip ini.



Gambar 4.69 Objek sprite dalam scene memiliki komponen SpriteRenderer yang melekat padanya.

Seperti yang dilakukan dengan CharacterController dan skrip kustom di bab sebelumnya, metode GetComponent() mengembalikan komponen lain pada objek yang sama, jadi kami menggunakannya untuk mereferensikan objek SpriteRenderer. Properti sprite dari SpriteRenderer dapat disetel ke aset sprite apa pun, jadi kode ini menyetel properti itu ke variabel Sprite yang dideklarasikan di bagian atas (yang kita isi dengan aset sprite di editor). Yah, itu tidak terlalu sulit! Tapi itu hanya satu gambar; kami memiliki empat gambar berbeda untuk digunakan, jadi sekarang hapus kode baru dari daftar 5.3 (itu hanya demonstrasi singkat tentang cara kerja teknik ini) untuk mempersiapkan bagian berikutnya.

Mengatur gambar dari SceneController yang tidak terlihat

Ingat di bab 3 bagaimana kita membuat objek tak terlihat dalam scene untuk mengontrol objek spawning. Kami akan mengambil pendekatan itu di sini juga, menggunakan objek tak terlihat untuk mengontrol lebih banyak fitur abstrak yang tidak terikat pada objek tertentu di scene. Pertama buat GameObject kosong (ingat, pilih menu GameObject > Create Empty). Kemudian buat skrip baru SceneController.cs di tampilan Project, dan drag aset skrip ini ke controller GameObject. Sebelum menulis kode di skrip baru, pertama-tama tambahkan konten daftar berikutnya ke skrip MemoryCard alih-alih apa yang Anda lihat di daftar 5.3.

Daftar Metode publik baru di MemoryCard.cs

```
...
[SerializeField] private SceneController controller;

private int _id;
public int id {
    get {return _id;}
}

public void SetCard(int id, Sprite image) {
    _id = id;
    GetComponent<SpriteRenderer>().sprite = image;
}
...
```

Perubahan utama dari daftar sebelumnya adalah kita sekarang mengatur gambar sprite di SetCard() alih-alih Start(). Karena itu adalah metode publik yang menggunakan sprite sebagai parameter, Anda dapat memanggil fungsi ini dari skrip lain dan mengatur gambar pada objek ini. Perhatikan bahwa SetCard() juga mengambil nomor ID sebagai parameter, dan kode menyimpan nomor itu. Meskipun kami belum membutuhkan ID, kami akan segera menulis kode yang membandingkan kartu untuk kecocokan, dan perbandingan itu akan bergantung pada ID kartu.

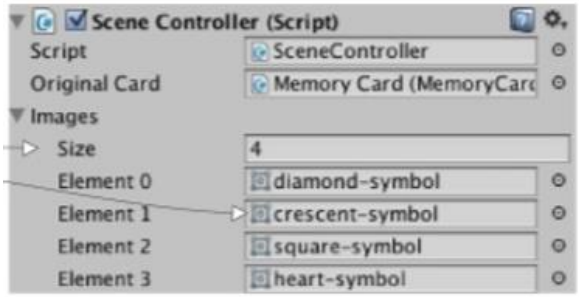
Catatan Tergantung pada bahasa pemrograman yang Anda gunakan sebelumnya, Anda mungkin tidak familiar dengan konsep “getter” dan “setter”. Singkat cerita, itu adalah fungsi yang dijalankan saat Anda mencoba mengakses properti yang terkait dengannya (misalnya, mengambil nilai card.id). Ada beberapa alasan untuk menggunakan getter dan setter, tetapi dalam kasus ini properti id bersifat read-only karena hanya ada fungsi untuk mendapatkan nilai dan bukan menyetelnya.

Terakhir, perhatikan bahwa kode memiliki variabel untuk controller; bahkan ketika SceneController mulai mengkloning objek kartu untuk mengisi scene, objek kartu juga memerlukan referensi kembali ke controller untuk memanggil metode publiknya. Seperti biasa, ketika kode mereferensikan objek dalam scene, drag objek controller di editor Unity ke slot variabel di Inspector. Lakukan ini sekali untuk kartu tunggal ini dan semua salinan yang akan datang akan memiliki referensi juga. Dengan kode tambahan itu sekarang di MemoryCard, tulis kode dari daftar berikutnya di SceneController.

Daftar Pass pertama di SceneController untuk game Memory

```
using UnityEngine;
using System.Collections;

public class SceneController : MonoBehaviour {
    [SerializeField] private MemoryCard originalCard;
    [SerializeField] private Sprite[] images;

    void Start() {
        int id = Random.Range(0, images.Length);
        originalCard.SetCard(id, images[id]);
    }
}
```

Untuk saat ini, ini adalah cuplikan singkat untuk mendemonstrasikan konsep memanipulasi kartu dari SceneController. Sebagian besar dari ini seharusnya sudah Anda kenal (misalnya, di editor Unity, drag objek kartu ke slot variabel di Inspector), tetapi susunan gambarnya baru. Seperti yang ditunjukkan pada gambar 4.70, di Inspector Anda dapat mengatur jumlah elemen. Ketik 4 untuk panjang larik, lalu drag sprite untuk gambar kartu ke slot larik. Sekarang sprite ini dapat diakses dalam array, seperti referensi objek lainnya.



Gambar 4.70 Array sprite yang terisi

Kebetulan, kami menggunakan metode Random.Range() di bab 3, jadi semoga Anda mengingatnya. Nilai batas yang tepat tidak menjadi masalah di sana, tetapi kali ini penting untuk dicatat bahwa nilai minimum inklusif dan dapat dikembalikan, sedangkan nilai pengembalian selalu di bawah maksimum. Tekan Mainkan untuk menjalankan kode baru ini. Anda akan melihat gambar berbeda diterapkan ke kartu yang terungkap setiap kali Anda menjalankan scene. Langkah selanjutnya adalah membuat seluruh kotak kartu, bukan hanya satu.

Membuat instance grid kartu

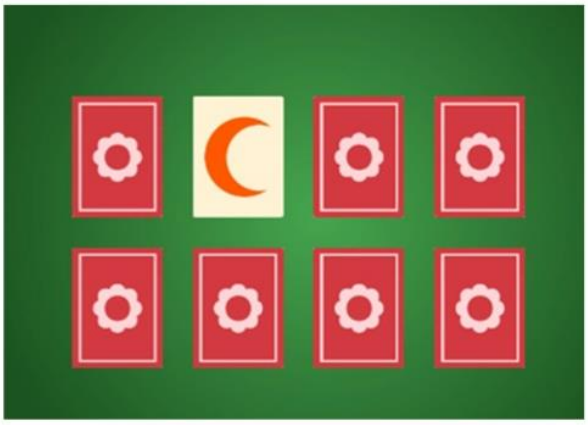
SceneController sudah memiliki referensi ke objek kartu, jadi sekarang Anda akan menggunakan metode Instantiate() (lihat daftar berikutnya) untuk mengkloning objek berkali-kali, seperti memunculkan objek di bab 3.

Daftar Mengkloning kartu delapan kali dan memposisikan dalam kotak

```
using UnityEngine;
using System.Collections;

public class SceneController : MonoBehaviour {
  public const int gridRows = 2;
public const int gridCols = 4;
public const float offsetX = 2f;
public const float offsetY = 2.5f;

[SerializeField] private MemoryCard originalCard;
[SerializeField] private Sprite[] images;


void Start() {
  Vector3 startPos = originalCard.transform.position;

  for (int i = 0; i < gridCols; i++) {
    for (int j = 0; j < gridRows; j++) {
      MemoryCard card;
      if (i == 0 && j == 0) {
        card = originalCard;
      } else {
        card = Instantiate(originalCard) as MemoryCard;
      }

      int id = Random.Range(0, images.Length);
      card.SetCard(id, images[id]);

      float posX = (offsetX * i) + startPos.x;
      float posY = -(offsetY * j) + startPos.y;
      card.transform.position = new Vector3(posX, posY, startPos.z);
    }
  }
}
```

Meskipun skrip ini lebih panjang dari daftar sebelumnya, tidak banyak yang bisa dijelaskan karena sebagian besar penambahan adalah deklarasi variabel dan matematika langsung. Bagian paling aneh dari kode ini mungkin adalah pernyataan if/else yang dimulai if (i == 0 && j == 0). Apa yang dilakukan kondisional itu adalah memilih objek kartu asli untuk slot kisi pertama atau mengkloning objek kartu untuk semua slot kisi lainnya. Karena kartu asli sudah ada di scene, jika Anda menyalin kartu di setiap iterasi loop, Anda akan mendapatkan satu terlalu banyak kartu di scene. Kartu-kartu tersebut kemudian diposisikan dengan mengimbanginya sesuai dengan jumlah iterasi melalui loop.

Sama seperti saat memindahkan objek 3D, objek 2D dapat dipindahkan dengan memanipulasi transform.position ke berbagai titik di layar, dan posisi ini dapat dinaikkan berulang kali di Update(). Tapi seperti yang Anda lihat saat menggerakkan player orang pertama, deteksi collision tidak diterapkan saat menyesuaikan transform.position secara langsung. Untuk memindahkan objek 2D dengan deteksi collision, Anda mungkin ingin menyesuaikan rigidbody2D.velocity setelah menetapkan komponen Physics2D.



Gambar 4.71 Kisi delapan kartu yang terungkap saat Anda mengkliknya

Mengacak kartu

Alih-alih membuat setiap kartu acak, kami akan mendefinisikan larik semua ID kartu (nomor 0 hingga 3 dua kali, untuk sepasang setiap kartu) dan kemudian mengacak larik itu. Kami kemudian akan menggunakan rangkaian ID kartu ini saat mengatur kartu, daripada membuat masing-masing secara acak. Daftar berikut menunjukkan kode.

Daftar Menempatkan kartu dari daftar yang diacak

```
...
void Start() {
   Vector3 startPos = originalCard.transform.position;

   int[] numbers = {0, 0, 1, 1, 2, 2, 3, 3};
   numbers = ShuffleArray(numbers);

   for (int i = 0; i < gridCols; i++) {
      for (int j = 0; j < gridRows; j++) {
        MemoryCard card;
        if (i == 0 && j == 0) {
          card = originalCard;
        } else {
          card = Instantiate(originalCard) as MemoryCard;
        }

        int index = j * gridCols + i;
        int id = numbers[index];
        card.SetCard(id, images[id]);


        float posX = (offsetX * i) + startPos.x;
        float posY = -(offsetY * j) + startPos.y;
        card.transform.position = new Vector3(posX, posY, startPos.z);
      }
   }
}

private int[] ShuffleArray(int[] numbers) {
   int[] newArray = numbers.Clone() as int[];
   for (int i = 0; i < newArray.Length; i++ ) {
      int tmp = newArray[i];
      int r = Random.Range(i, newArray.Length);
      newArray[i] = newArray[r];
      newArray[r] = tmp;
   }
   return newArray;
}
...
```

Sekarang ketika Anda menekan Mainkan kotak kartu akan menjadi kumpulan acak yang mengungkapkan tepat dua dari setiap gambar kartu. Array kartu dijalankan melalui algoritma shuffle Knuth (juga dikenal sebagai Fisher-Yates), cara sederhana namun efektif untuk mengacak elemen array. Algoritma ini mengulang array dan menukar setiap elemen array dengan posisi array lain yang dipilih secara acak. Anda dapat mengklik semua kartu untuk mengungkapkannya, tetapi permainan Memori seharusnya berjalan berpasangan; sedikit lebih banyak kode diperlukan.

Membuat dan mencetak pertandingan

Langkah terakhir dalam membuat game Memori yang berfungsi penuh adalah memeriksa kecocokan. Meskipun kami sekarang memiliki kotak kartu yang terungkap saat diklik, berbagai kartu tidak saling memengaruhi dengan cara apa pun. Dalam permainan Memori, setiap kali sepasang kartu terungkap, kita harus memeriksa untuk melihat apakah kartu yang terungkap cocok. Logika abstrak ini—memeriksa kecocokan dan merespons dengan tepat—memerlukan kartu yang memberi tahu SceneController saat diklik. Itu membutuhkan tambahan untuk SceneController.cs yang ditunjukkan pada daftar berikutnya.

Daftar SceneController, yang harus melacak kartu yang terungkap

```
...
private MemoryCard _firstRevealed;
private MemoryCard _secondRevealed;

public bool canReveal {
    get {return _secondRevealed == null;}
}
...
public void CardRevealed(MemoryCard card) {
    // initially empty
}
...
```

Metode CardRevealed() akan diisi sebentar lagi; kami membutuhkan perancah kosong untuk saat ini untuk merujuk ke MemoryCard.cs tanpa kesalahan kompiler. Perhatikan bahwa ada getter read-only lagi, kali ini digunakan untuk menentukan apakah kartu lain dapat dibuka; player hanya dapat mengungkapkan kartu lain ketika belum ada dua kartu yang terungkap.
Kita juga perlu memodifikasi MemoryCard.cs untuk memanggil metode (yang saat ini kosong) untuk menginformasikan SceneController ketika sebuah kartu diklik. Ubah kode di MemoryCard.cs sesuai dengan daftar berikut.

Daftar Modifikasi MemoryCard.cs untuk mengungkapkan kartu

```
...
public void OnMouseDown() {
    if (cardBack.activeSelf && controller.canReveal) {
        cardBack.SetActive(false);
        controller.CardRevealed(this);
    }
}

public void Unreveal() {
    cardBack.SetActive(true);
}
```

Jika Anda meletakkan pernyataan debug di dalam CardRevealed() untuk menguji komunikasi antar objek, Anda akan melihat pesan pengujian muncul setiap kali Anda mengklik kartu. Pertama-tama mari kita tangani satu kartu yang terungkap.

Menyimpan dan membandingkan kartu yang terungkap

Objek kartu diteruskan ke CardRevealed(), jadi mari kita mulai melacak kartu yang terungkap. Tulis kode dari daftar berikut.

Daftar Melacak kartu yang terungkap di SceneController

```
...
public void CardRevealed(MemoryCard card) {
  if (_firstRevealed == null) {
    _firstRevealed = card;
  } else {
    _secondRevealed = card;
    Debug.Log("Match? " + (_firstRevealed.id == _secondRevealed.id));
  }
}
...
```

Daftar tersebut menyimpan kartu yang terungkap di salah satu dari dua variabel kartu, tergantung pada apakah variabel pertama sudah terisi. Jika variabel pertama kosong, maka isi; jika sudah terisi, isi variabel kedua dan periksa ID kartu untuk kecocokan. Pernyataan debug mencetak benar atau salah di konsol. Saat ini kode tidak menanggapi kecocokan—itu hanya memeriksanya. Sekarang mari kita programkan responnya.

Menyembunyikan kartu yang tidak cocok

Kami akan menggunakan coroutine lagi karena reaksi terhadap kartu yang tidak cocok harus berhenti untuk memungkinkan player melihat kartu. Lihat kembali bab 3 untuk penjelasan lengkap tentang coroutine; singkat cerita, menggunakan coroutine akan memungkinkan kita untuk berhenti sejenak saat memeriksa kecocokan. Daftar berikutnya menunjukkan lebih banyak kode untuk Anda tambahkan ke SceneController.

Daftar SceneController, yang memberi skor kecocokan atau menyembunyikan kecocokan yang terlewat

```
...
private int _score = 0;
...
public void CardRevealed(MemoryCard card) {
  if (_firstRevealed == null) {
    _firstRevealed = card;
  } else {
    _secondRevealed = card;
    StartCoroutine(CheckMatch());
  }
}


private IEnumerator CheckMatch() {
  if (_firstRevealed.id == _secondRevealed.id) {
    _score++;
    Debug.Log("Score: " + _score);
  }
  else {
    yield return new WaitForSeconds(.5f);

    _firstRevealed.Unreveal();
    _secondRevealed.Unreveal();
  }

  _firstRevealed = null;
  _secondRevealed = null;
}
...
```

Pertama tambahkan nilai _score untuk dilacak; lalu luncurkan coroutine ke CheckMatch() saat kartu kedua dibuka. Di coroutine itu ada dua jalur kode, tergantung pada apakah kartunya cocok. Jika cocok, coroutine tidak berhenti; perintah hasil akan dilewati. Tetapi jika kartu tidak cocok, coroutine berhenti selama setengah detik sebelum memanggil Unreveal() pada kedua kartu, menyembunyikannya lagi. Akhirnya, apakah kecocokan dibuat atau tidak, variabel untuk menyimpan kartu keduanya dibatalkan, membuka jalan untuk mengungkapkan lebih banyak kartu. Saat Anda memainkan permainan, kartu yang tidak cocok akan ditampilkan sebentar sebelum bersembunyi lagi. Ada pesan debug saat Anda mencetak skor, tapi kami ingin skor ditampilkan sebagai label di layar.

Tampilan teks untuk skor

Menampilkan informasi kepada player adalah setengah dari alasan UI dalam game (separuh lainnya menerima input dari player; tombol UI dibahas di bagian berikutnya). Definisi UI berdiri untuk antarmuka pengguna. Istilah lain yang terkait erat adalah GUI (antarmuka pengguna grafis), yang mengacu pada bagian visual antarmuka, seperti teks dan tombol, dan itulah yang banyak orang maksudkan ketika mereka mengatakan UI.

Unity memiliki banyak cara untuk membuat tampilan teks. Salah satu caranya adalah dengan membuat objek teks 3D dalam scene. Ini adalah komponen mesh khusus, jadi pertama

buat objek kosong untuk melampirkan komponen ini. Dari menu GameObject, pilih Create Empty. Kemudian klik tombol Add Component dan pilih Mesh > Text Mesh.

Perhatikan Nama itu, teks 3D, mungkin terdengar tidak cocok dengan game 2D, tetapi jangan lupa bahwa ini secara teknis adalah scene 3D yang terlihat datar karena dilihat melalui kamera ortografis. Itu berarti kita bisa memasukkan objek 3D ke dalam game 2D jika kita mau—mereka hanya ditampilkan dalam perspektif datar.

Posisikan objek ini di -4,75, 3,65, -10; itu 475 piksel ke kiri dan 365 piksel ke atas, meletakkannya di sudut kiri atas, dan lebih dekat ke kamera sehingga akan muncul di atas objek game lainnya. Di Inspector, cari pengaturan Font di bagian bawah; klik tombol lingkaran kecil untuk membuka pemilih file, lalu pilih font Arial yang tersedia. Masukkan Skor: sebagai pengaturan Teks. Pemosisian yang benar juga memerlukan Kiri Atas untuk pengaturan Jangkar (ini mengontrol bagaimana huruf melebar saat diketik), jadi ubah ini jika perlu. Secara default, teks tampak buram, tapi itu mudah diperbaiki dengan menyesuaikan pengaturan.

Gambar 4.72 Pengaturan pemeriksa untuk objek teks untuk membuat teks tajam dan jelas

Jika kami mengimpor font TrueType baru ke dalam proyek, kami dapat menggunakannya sebagai gantinya, tetapi untuk tujuan kami, font default baik-baik saja. Anehnya, sedikit penyesuaian ukuran diperlukan untuk membuat teks default tajam dan jelas. Pertama-tama atur pengaturan Ukuran Font komponen TextMesh ke nilai yang sangat besar (saya menggunakan 80). Sekarang scale objek menjadi sangat kecil (seperti .1, .1, 1). Meningkatkan Ukuran Font menambahkan banyak piksel ke teks yang ditampilkan, dan penscalean objek mengompresi piksel tersebut ke dalam ruang yang lebih kecil.

Memanipulasi objek teks ini hanya memerlukan beberapa penyesuaian dalam kode penilaian (lihat daftar berikutnya).

Daftar Menampilkan skor pada objek teks

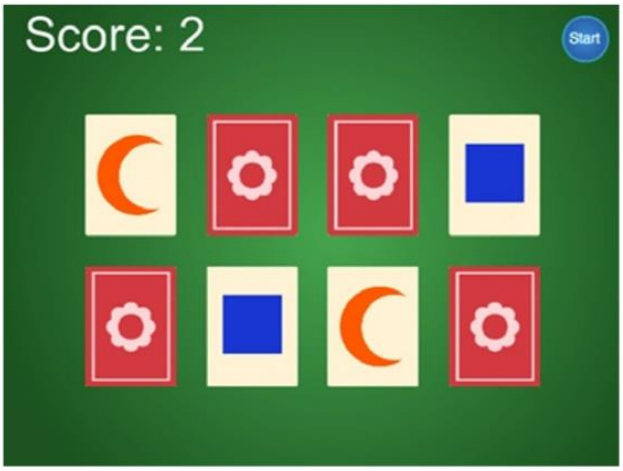
```
...
[SerializeField] private TextMesh scoreLabel;
...
private IEnumerator CheckMatch() {
  if (_firstRevealed.id == _secondRevealed.id) {
     _score++;
     scoreLabel.text = "Score: " + _score;
  }
...
```

Seperti yang Anda lihat, teks adalah properti dari objek yang dapat Anda atur ke string baru. Drag teks dalam scene ke variabel yang baru saja Anda tambahkan ke SceneController, lalu tekan Play. Sekarang Anda akan melihat skor yang ditampilkan saat Anda bermain game dan membuat pertandingan. Huzzah, permainannya berhasil!

Tombol mulai ulang

Pada titik ini, permainan Memori berfungsi penuh. Anda dapat memainkan game ini, dan semua fitur penting sudah tersedia. Tetapi inti yang dapat dimainkan ini masih kekurangan fungsionalitas menyeluruh yang diharapkan atau dibutuhkan player dalam permainan yang sudah selesai. Misalnya, saat ini Anda hanya dapat memainkan game sekali; Anda harus keluar dan memulai kembali untuk bermain lagi. Mari tambahkan kontrol ke layar sehingga player dapat memulai permainan dari awal tanpa harus berhenti.

Fungsionalitas ini dipecah menjadi dua tugas: membuat tombol UI, dan menyetel ulang game saat tombol itu diklik. Gambar 5.11 menunjukkan tampilan permainan dengan tombol restart.



Gambar 4.73 Layar permainan Memori Lengkap, termasuk tombol Mulai

Omong-omong, tidak ada tugas khusus untuk game 2D; semua game membutuhkan tombol UI, dan semua game membutuhkan kemampuan untuk mengatur ulang. Kami akan membahas kedua topik untuk melengkapi bab ini.

Memprogram komponen UIButton menggunakan SendMessage

Pertama tempatkan sprite tombol di scene; drag ke atas dari tampilan Proyek. Berikan posisi seperti 4,5, 3,25, -10; yang akan menempatkan tombol di sudut kanan atas (yaitu 450 piksel ke kanan dan 325 piksel ke atas) dan memindahkannya lebih dekat ke kamera sehingga akan muncul di atas objek game lainnya. Karena kita ingin bisa mengklik objek ini, beri dia Collider (sama seperti objek kartu, pilih Add Component > Physics 2D > Box Collider).

Catatan Seperti yang disinggung di bagian sebelumnya, Unity menyediakan beberapa cara untuk membuat tampilan UI, termasuk sistem UI tingkat lanjut yang diperkenalkan di

versi Unity terbaru. Untuk saat ini kita akan membuat tombol tunggal dari objek tampilan standar. Bab berikutnya akan mengajarkan Anda tentang fungsionalitas UI tingkat lanjut; UI untuk game 2D dan 3D idealnya dibangun dengan sistem itu. Sekarang buat skrip baru bernama UIButton.cs dan tetapkan skrip itu (ditunjukkan dalam daftar berikut) ke objek tombol.

Daftar Kode untuk membuat tombol UI generik dan dapat digunakan kembali

```
using UnityEngine;
using System.Collections;

public class UIButton : MonoBehaviour {
  [SerializeField] private GameObject targetObject;
  [SerializeField] private string targetMessage;
  public Color highlightColor = Color.cyan;

  public void OnMouseOver() {
    SpriteRenderer sprite = GetComponent<SpriteRenderer>();
    if (sprite != null) {
      sprite.color = highlightColor;
    }
  }
  public void OnMouseExit() {
    SpriteRenderer sprite = GetComponent<SpriteRenderer>();
    if (sprite != null) {

      sprite.color = Color.white;
    }
  }

  public void OnMouseDown() {
    transform.localScale = new Vector3(1.1f, 1.1f, 1.1f);
  }
  public void OnMouseUp() {
    transform.localScale = Vector3.one;
    if (targetObject != null) {
      targetObject.SendMessage(targetMessage);
    }
  }
}
```

Sebagian besar kode ini terjadi di dalam serangkaian fungsi OnMouseSomething; seperti Start() dan Update(), ini adalah serangkaian fungsi yang tersedia secara otomatis untuk semua komponen skrip di Unity. MouseDown telah disebutkan kembali di bagian 5.2.2, tetapi semua fungsi ini merespons interaksi mouse jika objek memiliki Collider; MouseOver dan MouseExit adalah sepasang peristiwa yang digunakan untuk mengarahkan kursor mouse ke atas suatu objek: MouseOver adalah saat ketika kursor mouse pertama kali bergerak di atas suatu objek, dan MouseExit adalah saat kursor mouse bergerak menjauh. Demikian pula, MouseDown dan MouseUp adalah sepasang event untuk mengklik mouse. MouseDown adalah momen saat tombol mouse ditekan secara fisik, dan MouseUp adalah momen saat tombol mouse dilepas.

Anda dapat melihat bahwa kode ini mewarnai sprite ketika mouse melayang di atasnya dan scaling sprite ketika diklik. Dalam kedua kasus, Anda dapat melihat bahwa perubahan (dalam warna atau scale) terjadi saat interaksi mouse dimulai, dan kemudian properti kembali

ke default (putih atau scale 1) saat interaksi mouse berakhir. Untuk penscalean, kode menggunakan komponen transformasi standar yang dimiliki semua GameObjects. Namun, untuk tint, kode menggunakan komponen SpriteRenderer yang dimiliki objek sprite; sprite diatur ke warna yang ditentukan di editor Unity melalui variabel publik.

Selain mengembalikan scale ke 1, SendMessage() dipanggil saat mouse dilepaskan. SendMessage() memanggil fungsi dari nama yang diberikan di semua komponen GameObject itu. Di sini objek target untuk pesan, serta pesan yang akan dikirim, keduanya ditentukan oleh variabel serial. Dengan cara ini, komponen UIButton yang sama dapat digunakan untuk semua jenis tombol, dengan target tombol berbeda disetel ke objek berbeda di Inspector.

Biasanya ketika melakukan pemrograman berorientasi objek dalam bahasa yang sangat diketik seperti C#, Anda perlu mengetahui jenis objek target untuk berkomunikasi dengan objek itu (misalnya, untuk memanggil metode publik objek, seperti memanggil target-Object .SendMessage() itu sendiri). Tetapi skrip untuk elemen UI mungkin memiliki banyak jenis target yang berbeda, jadi Unity menyediakan metode SendMessage() untuk mengomunikasikan pesan tertentu dengan objek target meskipun Anda tidak tahu persis jenis objeknya.

Peringatan Menggunakan SendMessage() kurang efisien untuk CPU daripada memanggil metode publik pada tipe yang dikenal (yaitu, menggunakan object.SendMessage("Method") versus component.Method()) jadi hanya gunakan SendMessage() ketika itu adalah kemenangan besar dalam hal membuat kode lebih mudah dipahami dan digunakan. Sebagai aturan umum, itu hanya akan terjadi jika ada banyak jenis objek yang menerima pesan; dalam situasi seperti itu, ketidakfleksibelan pewarisan atau bahkan antarmuka akan menghambat proses developeran game dan menghambat eksperimen.

Dengan kode ini ditulis, pasang variabel publik di Inspector tombol. Warna sorotan dapat diatur ke apa pun yang Anda inginkan (walaupun cyan default terlihat cukup bagus pada tombol biru). Sementara itu, letakkan objek SceneController di slot objek target, lalu ketik Restart sebagai pesannya.

Jika Anda memainkan game sekarang, ada tombol Reset di sudut kanan atas yang berubah warna sebagai respons terhadap mouse, dan itu membuat sedikit "pop" visual saat diklik. Tetapi pesan kesalahan muncul saat Anda mengklik tombol; di konsol Anda akan melihat kesalahan tentang tidak ada penerima untuk pesan Mulai Ulang. Itu karena kita belum menulis metode Restart() di SceneController, jadi mari kita tambahkan berikutnya.

Memanggil LoadLevel dari SceneController

SendMessage() dari tombol mencoba memanggil Restart() di SceneController, jadi mari tambahkan itu (lihat daftar berikutnya).

Daftar Kode SceneController yang memuat ulang level

```
...
public void Restart() {
   Application.LoadLevel("Scene");
}
...
```

Anda dapat melihat satu hal yang dilakukan Restart() adalah memanggil Application.LoadLevel(). Perintah itu memuat aset scene yang disimpan (yaitu, file yang dibuat saat Anda mengklik Simpan Scene di Unity). Lewati metode nama scene yang ingin Anda muat; dalam kasus saya scene itu disimpan dengan nama Scene, tetapi jika Anda menggunakan nama yang berbeda, berikan itu ke metode sebagai gantinya. Tekan Mainkan untuk melihat apa yang terjadi. Ungkapkan beberapa kartu dan buat beberapa kecocokan; jika Anda kemudian

mengklik tombol Reset, permainan dimulai kembali, dengan semua kartu tersembunyi dan skor 0. Hebat, seperti yang kami inginkan!

Seperti namanya LoadLevel() menyiratkan, metode ini dapat memuat level yang berbeda. Tapi apa sebenarnya yang terjadi ketika level dimuat, dan mengapa ini mengatur ulang permainan? Apa yang terjadi adalah bahwa segala sesuatu dari level saat ini (semua objek dalam scene, dan dengan demikian semua skrip yang dilampirkan ke objek tersebut) dihapus dari memori, dan kemudian semuanya dari scene baru dimuat. Karena scene "baru" dalam hal ini adalah aset yang disimpan dari scene saat ini, semuanya dihapus dari memori dan kemudian dimuat ulang dari awal. Anda dapat menandai objek tertentu untuk dikecualikan dari flush memori default saat level dimuat. Unity menyediakan metode DontDestroyOnLoad() untuk menyimpan objek dalam beberapa scene; kita akan menggunakan metode ini pada bagian arsitektur kode di bab selanjutnya.

Game lain berhasil diselesaikan! Nah, "selesai" adalah istilah yang relatif; Anda selalu dapat menerapkan lebih banyak fitur, tetapi semuanya dari rencana awal sudah selesai. Banyak konsep dari game 2D ini juga berlaku untuk game 3D, terutama pemeriksaan status game dan level pemuatan. Saatnya beralih persneling lagi dan menjauh dari game Memori ini dan ke new project.

Menempatkan GUI 2D dalam game 3D

Tampilan interaksi abstrak ini disebut sebagai UI, atau lebih khusus GUI. GUI mengacu pada bagian visual dari antarmuka, seperti teks dan tombol (lihat gambar 4.74). Secara teknis, UI menyertakan kontrol nongrafis, seperti keyboard atau gamepad, tetapi orang cenderung merujuk ke bagian grafis saat mereka mengatakan "antarmuka pengguna."

Gambar 4.74 GUI (tampilan pendahuluan, atau HUD) yang akan Anda buat untuk game

Meskipun software apa pun memerlukan semacam UI agar pengguna software itu dapat mengontrolnya, game sering kali menggunakan GUI mereka dengan cara yang sedikit berbeda dari software lain. Dalam sebuah situs web, misalnya, GUI pada dasarnya adalah situs web (dalam hal representasi visual). Namun, dalam gim, teks dan tombol sering kali menjadi hamparan tambahan di atas tampilan gim, semacam tampilan yang disebut HUD.

Definisi Head-up display (HUD) menampilkan grafis di atas tampilan dunia. Konsep HUD berasal dari jet militer sehingga pilot dapat melihat informasi penting tanpa harus melihat ke bawah. Demikian pula, GUI yang ditumpangkan pada tampilan game disebut sebagai HUD. Untuk mempelajari tentang alat UI di Unity, Anda akan membangun di atas proyek first-person shooter (FPS) dari bab 3. Proyek dalam bab ini akan melibatkan langkah-langkah berikut:

1. Merencanakan antarmuka
2. Menempatkan elemen UI di tampilan
3. Memprogram interaksi dengan elemen UI
4. Membuat GUI merespons kejadian di scene

5. Membuat scene merespons tindakan di GUI

Catatan Bab ini sebagian besar tidak tergantung pada proyek yang Anda bangun—bab ini hanya menambahkan antarmuka grafis di atas demo game yang ada. Semua contoh dalam bab ini dibangun di atas FPS yang dibuat di bab 3, dan Anda dapat mengunduh proyek sampel itu, tetapi Anda bebas menggunakan demo game apa pun yang Anda inginkan. Salin proyek dari bab 3 dan buka salinannya untuk mulai mengerjakan bab ini. Seperti biasa, aset seni yang Anda butuhkan ada di unduhan sampel. Dengan file-file itu diatur, Anda siap untuk mulai membangun UI game.

Sebelum Anda mulai menulis kode...

Untuk mulai membangun HUD, Anda harus terlebih dahulu memahami cara kerja sistem UI. Unity menyediakan banyak pendekatan untuk membangun HUD game, jadi kita perlu membahas cara kerja sistem tersebut. Kemudian kita dapat merencanakan UI secara singkat dan menyiapkan aset seni yang kita perlukan.

GUI mode langsung atau antarmuka 2D lanjutan?

Dari versi pertamanya, Unity hadir dengan sistem GUI mode langsung, dan sistem itu memudahkan untuk meletakkan tombol yang dapat diklik di layar. Daftar 6.1 menunjukkan kode untuk melakukan itu; cukup lampirkan skrip ini ke objek apa pun di scene. Untuk contoh lain dari UI mode langsung, ingat kursor target yang ditampilkan di bab 3. Sistem GUI ini sepenuhnya didasarkan pada kode, tanpa bekerja di editor Unity.

Definisi Mode segera mengacu pada mengeluarkan perintah menggambar secara eksplisit setiap bingkai, versus sistem di mana Anda mendefinisikan semua visual sekali dan kemudian untuk setiap bingkai sistem tahu apa yang harus digambar tanpa Anda harus mengatakannya lagi. Pendekatan terakhir disebut mode yang dipertahankan.

Daftar Contoh tombol menggunakan GUI mode langsung

```
using UnityEngine;
using System.Collections;

public class BasicUI : MonoBehaviour {
   void OnGUI() {
      if (GUI.Button(new Rect(10, 10, 40, 20), "Test")) {
         Debug.Log("Test button");
      }
   }
}
```

Inti dari kode dalam daftar ini adalah metode OnGUI(). Sama seperti Start() dan Update(), setiap MonoBehaviour secara otomatis merespons OnGUI(). Fungsi itu menjalankan setiap frame setelah scene 3D dirender, menyediakan tempat untuk meletakkan perintah menggambar GUI. Kode ini menggambar sebuah tombol; perhatikan bahwa perintah untuk tombol dijalankan setiap frame (yaitu, dalam gaya mode langsung). Perintah tombol digunakan dalam kondisi yang merespons ketika tombol diklik.

Karena GUI mode langsung memudahkan untuk mendapatkan beberapa tombol di layar dengan sedikit usaha, kami akan menggunakannya sebagai contoh di bab-bab selanjutnya (terutama bab 8). Tetapi membuat tombol default adalah satu-satunya hal yang mudah dibuat dengan sistem itu, jadi versi terbaru Unity sekarang memiliki sistem antarmuka baru berdasarkan grafik 2D yang diletakkan di editor. Dibutuhkan sedikit lebih banyak upaya untuk menyiapkan, tetapi Anda mungkin ingin menggunakan sistem antarmuka yang lebih baru di game yang sudah selesai karena menghasilkan hasil yang lebih halus.

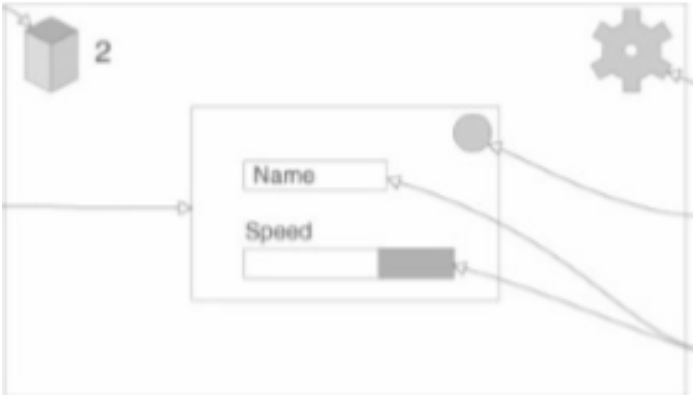
Sistem UI baru bekerja dalam mode dipertahankan, sehingga grafik diletakkan satu kali dan kemudian digambar setiap frame tanpa perlu terus-menerus didefinisikan ulang. Dalam sistem ini, grafik untuk UI ditempatkan di editor Unity. Ini memberikan dua keuntungan

dibandingkan UI mode langsung: 1) Anda dapat melihat tampilan UI saat menempatkan elemen UI, dan 2) sistem ini memudahkan penyesuaian UI dengan gambar Anda sendiri.

Untuk menggunakan sistem ini, Anda akan mengimpor gambar dan kemudian menyeret objek ke dalam scene. Selanjutnya mari kita rencanakan bagaimana tampilan UI ini.

Merencanakan tata letak

HUD untuk sebagian besar game hanyalah beberapa kontrol UI berbeda yang berulang-ulang. Itu berarti proyek ini tidak perlu menjadi UI yang sangat rumit agar Anda dapat mempelajari cara membuat UI game. Anda akan menempatkan tampilan skor dan tombol pengaturan di sudut layar di atas tampilan permainan utama. Tombol pengaturan akan memunculkan jendela pop-up, dan jendela itu akan memiliki bidang teks dan penggeser.



Gambar 4.75 GUI yang direncanakan

Untuk contoh ini, kontrol input tersebut akan digunakan untuk menyetel nama player dan kecepatan gerakan, tetapi pada akhirnya elemen UI tersebut dapat mengontrol setelan apa pun yang relevan dengan game Anda.

Nah, rencana itu cukup sederhana! Langkah selanjutnya adalah memasukkan gambar-gambar yang dibutuhkan.

Mengimpor gambar UI

UI ini memerlukan beberapa gambar untuk ditampilkan untuk hal-hal seperti tombol. UI dibangun dari gambar 2D seperti grafik di bab 5, jadi Anda akan mengikuti dua langkah yang sama:

1. Impor gambar (jika perlu, atur ke Sprite).
2. Drag sprite ke dalam scene. Untuk menyelesaikan langkah-langkah ini, pertama-tama drag gambar ke tampilan Proyek untuk mengimpornya, lalu di Inspector ubah pengaturan Jenis Teksturnya ke Sprite (2D Dan UI).

Pengaturan Jenis Tekstur default ke Tekstur dalam proyek 3D dan Sprite dalam proyek 2D. Jika Anda ingin sprite dalam proyek 3D, Anda perlu menyesuaikan pengaturan ini secara manual. Dapatkan semua gambar yang diperlukan dari unduhan sampel (lihat gambar 4.76) dan kemudian impor gambar ke dalam proyek Anda. Pastikan semua aset yang diimpor diatur ke Sprite; Anda mungkin perlu menyesuaikan Jenis Tekstur dalam pengaturan yang ditampilkan setelah mengimpor.

Gambar 4.76 Gambar yang diperlukan untuk proyek bab ini

Sprite ini terdiri dari tombol, tampilan skor, dan pop-up yang akan Anda buat. Sekarang setelah gambar diimpor, mari letakkan grafik ini ke layar.

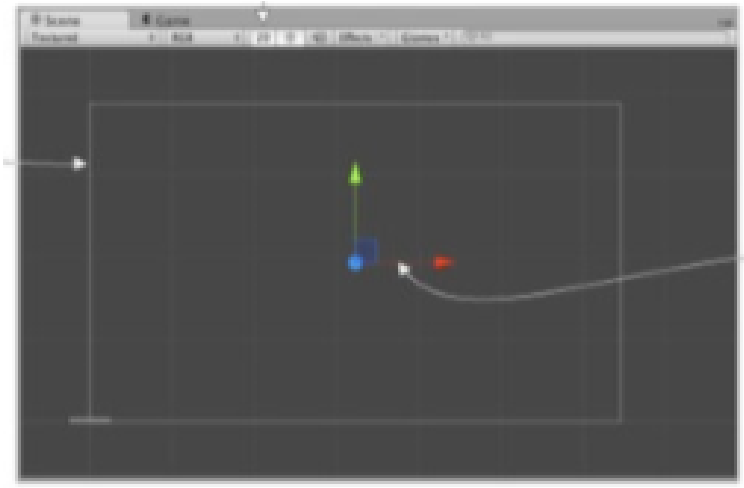
Menyiapkan tampilan GUI

Aset seni adalah jenis sprite 2D yang sama yang kita gunakan di bab 5, tetapi penggunaan aset tersebut dalam scene sedikit berbeda. Unity menyediakan alat khusus untuk membuat gambar menjadi HUD yang ditampilkan di atas scene 3D, daripada menampilkan gambar sebagai bagian dari scene. Penempatan elemen UI juga memiliki beberapa trik khusus, karena kebutuhan tampilan yang dapat berubah pada layar yang berbeda.

Membuat kanvas untuk antarmuka

Salah satu aspek paling mendasar dan tidak jelas tentang cara kerja sistem UI adalah bahwa semua gambar harus dilampirkan ke objek kanvas. *Tip* Canvas adalah jenis objek khusus yang dirender oleh Unity sebagai UI untuk sebuah game. Buka menu GameObject untuk melihat berbagai macam objek yang dapat Anda buat; di kategori UI, pilih Canvas. Sebuah objek kanvas akan muncul di scene (itu mungkin lebih jelas untuk mengubah nama objek HUD Canvas). Objek ini mewakili seluruh luas layar, dan sangat besar jika dibandingkan dengan scene 3D karena scaling satu piksel layar ke satu unit dalam scene.

Peringatan Saat Anda membuat objek kanvas, objek EventSystem juga dibuat secara otomatis. Objek itu diperlukan untuk interaksi UI tetapi Anda dapat mengabaikannya. Beralih ke mode tampilan 2D (lihat gambar 4.77) dan klik dua kali kanvas di Hierarchy untuk memperkecil dan melihatnya sepenuhnya. Mode tampilan 2D otomatis ketika seluruh proyek adalah 2D, tetapi dalam proyek 3D, sakelar ini harus diklik untuk beralih antara UI dan scene utama. Untuk kembali melihat scene 3D, matikan mode tampilan 2D lalu klik dua kali bangunan untuk memperbesar objek itu.



Gambar 4.77 Objek kanvas kosong dalam tampilan Scene

Jangan lupa tip ini dari bab 4: di bagian atas panel Scene viewer terdapat tombol yang mengontrol apa yang terlihat, jadi cari tombol Effects untuk mematikan skybox. Kanvas memiliki sejumlah pengaturan yang dapat Anda sesuaikan. Pertama adalah opsi Render Mode; biarkan ini pada pengaturan default, tetapi Anda harus tahu apa arti dari tiga kemungkinan pengaturan:

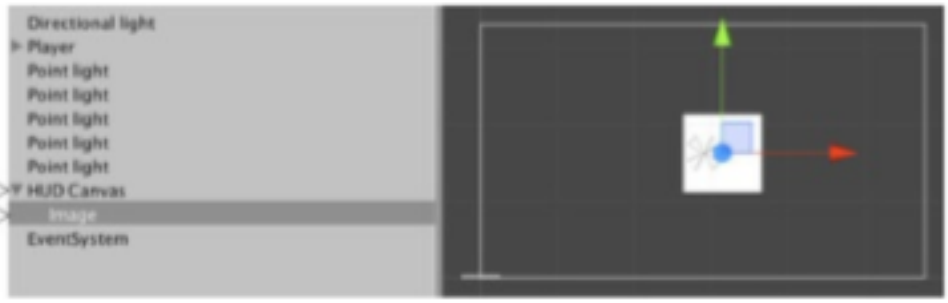
- ***Screen Space—Overlay*** —Menampilkan UI sebagai grafik 2D di atas tampilan kamera (ini adalah pengaturan default).
- ***Screen PAce—Camera*** —Juga merender UI di atas tampilan kamera, tetapi elemen UI dapat berputar untuk efek perspektif.

- ***World Space*** —Menempatkan objek kanvas di dalam scene, seolah-olah UI adalah bagian dari scene 3D.

Dua mode selain default awal terkadang dapat berguna untuk efek tertentu tetapi sedikit lebih rumit. Pengaturan penting lainnya adalah Pixel Perfect. Pengaturan ini menyebabkan rendering menyesuaikan posisi gambar secara halus sehingga selalu benar-benar tajam dan tajam (sebagai lawan dari memburamkannya saat diposisikan di antara piksel). Silakan dan pilih kotak centang itu. Sekarang kanvas HUD sudah diatur, tetapi masih kosong dan membutuhkan sprite.

Tombol, gambar, dan label teks

Objek kanvas mendefinisikan area untuk ditampilkan sebagai UI, tetapi masih membutuhkan sprite untuk ditampilkan. Jika Anda merujuk kembali ke mockup UI pada gambar 6.2, ada gambar blok/musuh di sudut kiri atas, teks yang menampilkan skor di sebelahnya, dan tombol berbentuk roda gigi di sudut kanan atas. Oleh karena itu, di bagian UI menu GameObject terdapat opsi untuk membuat gambar, teks, atau tombol. Buat satu dari masing-masing. Elemen UI harus menjadi anak dari objek kanvas agar dapat ditampilkan dengan benar. Unity melakukan ini secara otomatis, tetapi ingat bahwa seperti biasa Anda dapat menyeret objek di sekitar tampilan Hierarki (lihat gambar 4.78) untuk membuat hubungan induk-anak.



Gambar 4.78 Kanvas dengan gambar yang ditautkan dalam tampilan Hierarki

Objek di dalam kanvas dapat dikandung bersama untuk tujuan pemosisian, sama seperti objek lain dalam scene. Misalnya, Anda mungkin ingin menyeret objek teks ke gambar sehingga teks akan bergerak bersama gambar. Demikian pula, objek tombol default memiliki objek teks sebagai anaknya; tombol ini tidak memerlukan label teks, jadi hapus objek teks. Posisikan elemen UI secara kasar ke sudutnya. Di bagian selanjutnya kita akan membuat posisinya tepat; untuk saat ini, cukup drag objek sampai posisinya cukup banyak. Klik dan drag objek gambar ke kiri atas kanvas; tombol masuk di kanan atas.

Tip Seperti disebutkan dalam bab 5, Anda menggunakan alat Rect dalam mode 2D. Saya menggambarkannya sebagai alat manipulasi tunggal yang mencakup ketiga transformasi: Pindahkan, Putar, dan Scale. Operasi ini harus menjadi alat terpisah dalam 3D tetapi digabungkan dalam 2D karena itu adalah satu dimensi yang tidak perlu dikhawatirkan. Dalam mode 2D, alat ini dipilih secara otomatis, atau Anda dapat mengklik tombol di sudut kiri atas Unity.

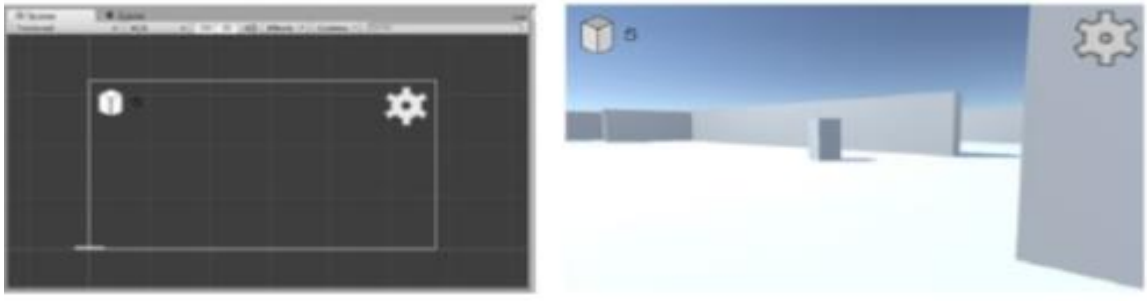
Saat ini gambar keduanya kosong. Jika Anda memilih objek UI dan melihat Inspector, Anda akan melihat slot Gambar Sumber di dekat bagian atas komponen gambar. Seperti yang ditunjukkan pada gambar 6.6, drag sprite (ingat, bukan tekstur!) dari tampilan Project untuk menetapkan gambar ke objek. Tetapkan sprite musuh ke objek gambar, dan sprite roda gigi ke objek tombol (klik Set Native Size setelah menetapkan sprite untuk mengukur objek gambar dengan benar).

Gambar 4.79 Tetapkan sprite 2D ke properti Gambar elemen UI.

Itu menjaga penampilan gambar musuh dan tombol persneling. Adapun objek teks, ada banyak pengaturan di Inspector. Pertama, ketik satu nomor di kotak teks besar; teks ini akan ditimpa nanti, tetapi ini berguna karena terlihat seperti tampilan skor di dalam editor. Teksnya kecil, jadi tingkatkan Ukuran Font menjadi 24 dan buat gaya Bold. Anda juga ingin mengatur label ini ke perataan horizontal kiri (lihat gambar 4.80) dan perataan vertikal tengah. Untuk saat ini pengaturan yang tersisa dapat dibiarkan pada nilai defaultnya.

Gambar 4.80 Pengaturan untuk objek teks UI.

Catatan Selain kotak teks dan perataan, properti yang paling umum untuk disesuaikan adalah font. Anda dapat mengimpor font TrueType ke Unity, lalu memasukkan font tersebut ke dalam Inspector. Sekarang sprite telah ditetapkan ke gambar UI, dan teks skor sudah diatur, Anda dapat menekan Play untuk melihat HUD di atas game 3D.

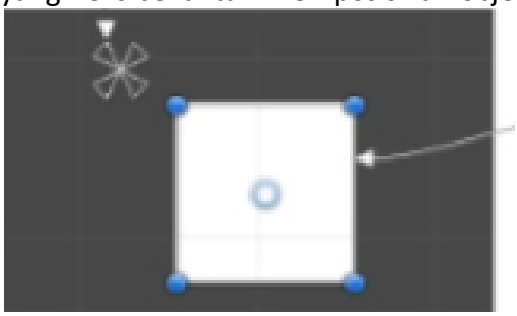


Gambar 4.81 GUI seperti yang terlihat di editor (kiri) dan saat memainkan game (kanan)

Hebat, Anda membuat HUD dengan gambar 2D yang ditampilkan di atas game 3D! Satu lagi pengaturan visual yang kompleks tetap: memposisikan elemen UI relatif terhadap kanvas.

Mengontrol posisi elemen UI

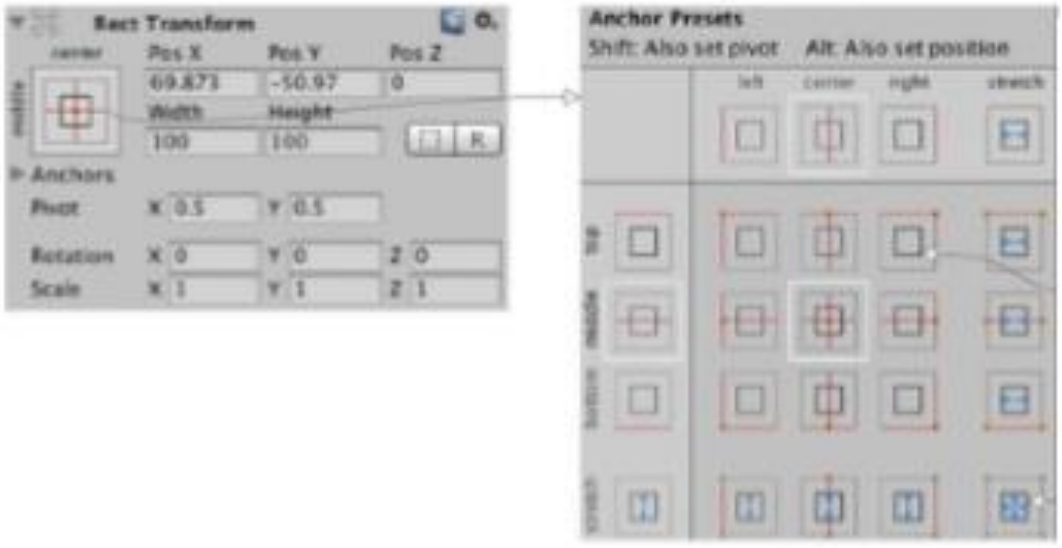
Semua objek UI memiliki jangkar, ditampilkan di editor sebagai target X (lihat gambar 4.82). Jangkar adalah cara yang fleksibel untuk memposisikan objek di UI.



Gambar 4.82 Titik jangkar objek gambar

Definisi Jangkar suatu objek adalah titik di mana suatu objek menempel pada kanvas atau layar. Ini menentukan posisi objek yang diukur relatif terhadap Posisi adalah nilai seperti "50 piksel pada sumbu X". Tapi itu menyisakan pertanyaan: 50 piksel dari apa? Di sinilah jangkar masuk. Tujuan jangkar adalah bahwa sementara objek tetap di tempat relatif terhadap titik jangkar, jangkar bergerak relatif terhadap kanvas. Jangkar didefinisikan sebagai sesuatu seperti "tengah layar", dan kemudian jangkar akan tetap berada di tengah saat layar berubah ukuran. Demikian pula, menyetel jangkar ke sisi kanan layar akan membuat objek tetap berada di sisi kanan meskipun ukuran layar berubah (misalnya, jika game dimainkan di monitor yang berbeda).

Cara termudah untuk memahami apa yang saya bicarakan adalah dengan melihatnya beraksi. Pilih objek gambar dan lihat ke Inspector. Pengaturan jangkar (lihat gambar 4.83) akan muncul tepat di bawah komponen transformasi. Secara default, elemen UI memiliki jangkar yang disetel ke Tengah, tetapi Anda ingin mengatur jangkar ke Kiri Atas untuk gambar ini.



Gambar 4.83 cara menyesuaikan pengaturan nchor

Ubah jangkar tombol roda gigi juga. Atur ke Kanan Atas untuk objek ini; klik Preset Jangkar kanan atas. Sekarang coba scalekan jendela ke kiri dan kanan; klik dan drag di sisi tampilan Scene. Berkat jangkar, objek UI akan tetap berada di sudutnya saat kanvas berubah

ukuran. Seperti yang ditunjukkan gambar 6.11, elemen UI ini sekarang berakar di tempatnya saat layar bergerak.

Gambar 4.84 Jangkar tetap di tempatnya saat layar berubah

Jangkar poin dapat menyesuaikan scale serta posisi. Kami tidak akan menjelajahi fungsionalitas itu dalam bab ini, tetapi setiap sudut gambar dapat di-root ke sudut layar yang berbeda. Pada Gambar 6.11 gambar tidak berubah ukuran, tetapi kita dapat menyesuaikan jangkar sehingga ketika layar berubah ukuran, gambar akan melebar dengannya. Semua pengaturan visual sudah selesai, jadi saatnya untuk memprogram interaktivitas.

Interaktivitas pemrograman di UI

Sebelum Anda dapat berinteraksi dengan UI, Anda harus memiliki kursor mouse. Jika Anda ingat, game ini menyesuaikan pengaturan Cursor di metode Start() dari kode RayShooter. Pengaturan tersebut mengunci dan menyembunyikan kursor mouse, perilaku yang berfungsi untuk kontrol dalam game FPS tetapi mengganggu penggunaan UI. Hapus baris tersebut dari RayShooter.cs sehingga Anda dapat mengklik HUD. Selama Anda membuka RayShooter.cs, Anda juga dapat memastikan untuk tidak memotret saat berinteraksi dengan GUI. Daftar berikut menunjukkan kode untuk itu.

Daftar Menambahkan pemeriksaan GUI ke kode di RayShooter.cs

```
using UnityEngine.EventSystems;
...
void Update() {
   if (Input.GetMouseButtonDown(0) &&
!EventSystem.current.IsPointerOverGameObject()) {
   Vector3 point = new Vector3(
      camera.pixelWidth/2, camera.pixelHeight/2, 0);
   ...
```

Sekarang Anda dapat memainkan game dan mengklik tombol, meskipun belum melakukan apa-apa. Anda dapat melihat warna tombol berubah saat Anda mengarahkan mouse ke atasnya dan mengkliknya. Perilaku mouseover dan klik ini adalah warna default yang dapat diubah untuk setiap tombol, tetapi defaultnya terlihat baik untuk saat ini. Anda dapat mempercepat perilaku fading default; Durasi Fade adalah pengaturan di komponen tombol, jadi coba turunkan ke .01 untuk melihat bagaimana tombol berubah

Terkadang kontrol interaksi default UI juga mengganggu permainan. Ingat objek EventSystem yang dibuat secara otomatis bersama dengan kanvas? Objek itu mengontrol kontrol interaksi UI, dan secara default menggunakan tombol panah untuk berinteraksi dengan GUI. Anda mungkin perlu mematikan tombol panah di EventSystem: di pengaturan EventSystem, hapus centang pada kotak Send Navigation Event. Tetapi tidak ada hal lain yang terjadi ketika Anda mengklik tombol karena Anda belum menautkannya ke kode apa pun. Mari kita urus itu selanjutnya.

Memprogram UIController yang tidak terlihat

Secara umum, interaksi UI diprogram dengan serangkaian langkah standar yang sama untuk semua elemen UI:

1. Buat objek UI dalam scene (tombol yang dibuat di bagian sebelumnya).
2. Tulis skrip untuk dipanggil saat UI dioperasikan.
3. Lampirkan skrip itu ke objek di scene.
4. Tautkan elemen UI (seperti tombol) ke objek dengan skrip tersebut.

Untuk mengikuti langkah-langkah ini, pertama-tama kita perlu membuat objek controller untuk ditautkan ke tombol. Buat skrip yang disebut UIController (ditampilkan dalam daftar berikut) dan drag skrip itu ke objek controller di scene.

Daftar Skrip UIController digunakan untuk memprogram tombol

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class UIController : MonoBehaviour {
   [SerializeField] private Text scoreLabel;

   void Update() {
      scoreLabel.text = Time.realtimeSinceStartup.ToString();
   }

   public void OnOpenSettings() {
      Debug.Log("open settings");
   }
}
```

Anda mungkin bertanya-tanya mengapa kita membutuhkan objek terpisah untuk SceneController dan UIController. Memang, scene ini sangat sederhana sehingga Anda dapat memiliki satu controller yang menangani scene 3D dan UI. Namun, karena permainan menjadi lebih kompleks, itu akan menjadi semakin berguna untuk scene 3D dan UI untuk menjadi modul terpisah, berkomunikasi secara tidak langsung. Gagasan ini jauh melampaui game hingga software secara umum; insinyur software mengacu pada prinsip ini sebagai pemisahan masalah.

Sekarang drag objek ke slot komponen untuk menghubungkannya. Drag label skor (objek teks yang kita buat sebelumnya) ke slot teks UIController. Kode di UIController mengatur teks yang ditampilkan pada label itu. Saat ini kode menampilkan timer untuk menguji tampilan teks; yang akan diubah menjadi skor nanti.

Selanjutnya, tambahkan entri OnClick ke tombol untuk menyeret objek controller. Pilih tombol untuk melihat pengaturannya di Inspector. Di bagian bawah Anda akan melihat panel OnClick; awalnya panel itu kosong, tetapi (seperti yang Anda lihat pada gambar 6.12) Anda dapat mengklik tombol + untuk menambahkan entri ke panel itu. Setiap entri mendefinisikan satu fungsi yang dipanggil saat tombol itu diklik; daftar memiliki slot untuk objek dan menu untuk fungsi yang dipanggil. Drag objek controller ke slot objek, lalu cari UIController di menu; pilih OnOpenSettings() di bagian itu.

Gambar 4.85 Panel OnClick menuju bagian bawah pengaturan tombol

Menanggapi event mouse lainnya

OnClick adalah satu-satunya peristiwa yang ditampilkan oleh komponen tombol, tetapi elemen UI dapat merespons sejumlah interaksi yang berbeda. Untuk melampaui interaksi default, gunakan komponen EventTrigger. Tambahkan komponen baru ke objek tombol dan cari bagian Event dari menu komponen. Pilih EventTrigger dari menu itu. Meskipun OnClick tombol hanya merespons satu klik penuh (tombol mouse ditekan ke bawah lalu dilepaskan), mari kita coba merespons tombol mouse yang ditekan tetapi tidak dilepaskan. Lakukan langkah yang sama seperti untuk OnClick, hanya menanggapi peristiwa yang berbeda. Pertama tambahkan metode lain ke UIController:

```
...
public void OnPointerDown() {
Debug.Log("pointer down");
}
...
```

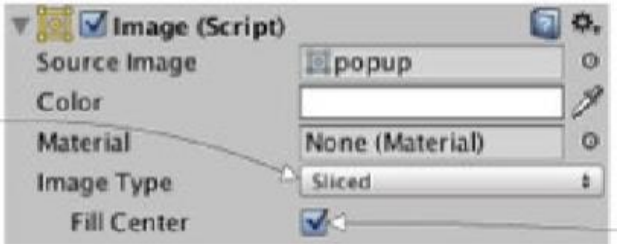
Sekarang klik Add New Event Type untuk menambahkan tipe baru ke komponen EventTrigger. Pilih Pointer Down untuk event tersebut. Ini akan membuat panel kosong untuk event itu, seperti yang dimiliki OnClick. Klik tombol + untuk menambahkan daftar event, drag objek controller ke entri ini, dan pilih OnPointerDown() di menu. Ini dia!

Mainkan game dan klik tombol untuk menampilkan pesan debug di konsol. Sekali lagi, kode saat ini adalah keluaran acak untuk menguji fungsionalitas tombol. Apa yang ingin kita lakukan adalah membuka pop-up pengaturan, jadi mari kita buat jendela pop-up berikutnya.

Membuat jendela pop-up

UI memiliki tombol untuk membuka jendela pop-up, tetapi belum ada pop-up. Itu akan menjadi objek gambar baru, bersama dengan beberapa kontrol (seperti tombol dan penggeser) yang dilampirkan ke objek itu. Langkah pertama adalah membuat gambar baru, jadi pilih GameObject > UI > Image. Sama seperti sebelumnya, gambar baru memiliki slot di Inspector yang disebut Gambar Sumber. Drag sprite ke slot itu untuk mengatur gambar ini. Kali ini menggunakan sprite bernama popup.

Biasanya, sprite direntangkan ke seluruh objek gambar; ini adalah bagaimana skor dan gambar roda gigi bekerja, dan Anda mengklik tombol Setel Ukuran Asli untuk mengubah ukuran objek ke ukuran gambar. Perilaku ini adalah default untuk objek gambar, tetapi pop-up akan melakukan sesuatu yang berbeda. Seperti yang Anda lihat pada gambar 6.13, komponen gambar memiliki pengaturan Jenis Gambar. Pengaturan ini default ke Sederhana, yang merupakan jenis gambar yang benar sebelumnya. Namun, untuk pop-up, atur Jenis Gambar ke Irisan.

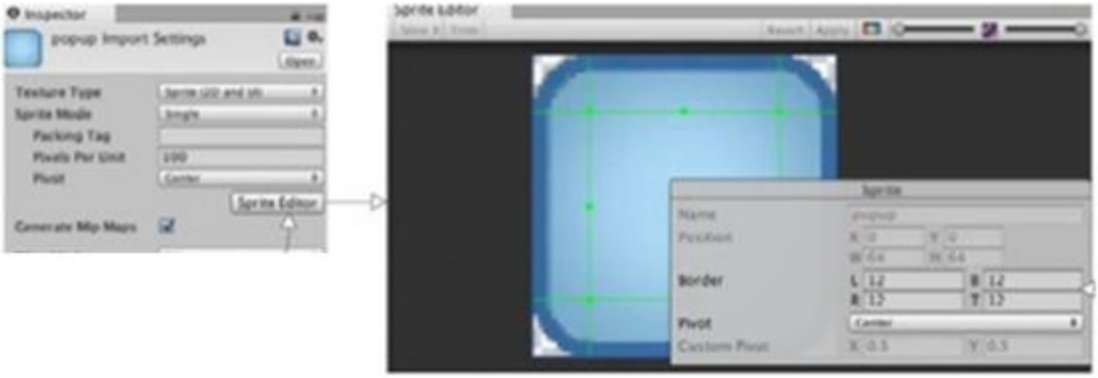


Gambar 4.86 Pengaturan untuk komponen gambar, termasuk Jenis Gambar

Definisi Gambar yang diiris dibagi menjadi sembilan bagian yang scalenya berbeda satu sama lain. Dengan scaling tepi gambar secara terpisah dari tengah, Anda memastikan bahwa gambar dapat discalekan ke ukuran apa pun yang Anda inginkan sambil mempertahankan

tepinya yang tajam dan tajam. Dalam alat developeran lainnya, jenis gambar ini sering memiliki "9" di suatu tempat di namanya (seperti 9-slice, 9-patch, scale-9) untuk menunjukkan 9 bagian gambar.

Setelah Anda beralih ke gambar irisan, Unity mungkin menampilkan kesalahan dalam pengaturan komponen, mengeluh bahwa gambar tidak memiliki batas. Itu karena sprite popup belum memiliki sembilan bagian yang ditentukan. Untuk mengaturnya, pertama-tama pilih sprite popup di tampilan Project. Di Inspector Anda akan melihat tombol Sprite Editor; klik tombol itu dan jendela Editor Sprite akan muncul.



Gambar 4.87 Tombol Editor Sprite di Inspector dan jendela pop-up

Di Editor Sprite Anda dapat melihat garis hijau yang menunjukkan bagaimana gambar akan dipotong. Awalnya gambar tidak akan memiliki batas (yaitu, semua pengaturan Perbatasan adalah 0). Tingkatkan lebar batas keempat sisinya, yang akan menghasilkan batas yang ditunjukkan pada gambar diatas Karena keempat sisi (Kiri, Kanan, Bawah, dan Atas) memiliki batas yang diatur ke lebar 12 piksel, garis batas akan tumpang tindih menjadi sembilan bagian. Tutup jendela editor dan terapkan perubahan.

Sekarang sprite memiliki sembilan bagian yang ditentukan, gambar yang diiris akan bekerja dengan benar (dan pengaturan komponen Gambar akan menampilkan Pusat Isi; pastikan pengaturannya aktif). Klik dan drag indikator biru di sudut gambar untuk scalingnya (beralih ke alat Rect yang dijelaskan di bab 5 jika Anda tidak melihat indikator scale apa pun). Bagian perbatasan akan mempertahankan ukurannya sementara bagian tengahnya menscale. Karena bagian perbatasan mempertahankan ukurannya, gambar irisan dapat discalekan ke ukuran apa pun dan tetap memiliki tepi yang tajam. Ini sempurna untuk elemen UI—jendela yang berbeda mungkin memiliki ukuran yang berbeda tetapi harus tetap terlihat sama. Untuk pop-up ini, masukkan lebar 250 dan tinggi 200 agar terlihat seperti gambar dibawah ini (juga, pusatkan pada posisi 0, 0, 0).

Gambar 4.88 Gambar irisan discalekan ke dimensi pop-up

Tip Bagaimana gambar UI menumpuk di atas satu sama lain ditentukan oleh urutannya dalam tampilan Hierarki. Dalam daftar Hierarchy, drag objek pop-up di atas objek UI lainnya (tentu saja tetap melekat pada kanvas). Sekarang pindahkan pop-up di dalam tampilan Scene; Anda dapat melihat bagaimana gambar tumpang tindih dengan jendela pop-up. Terakhir, drag pop-up ke bagian bawah hierarki kanvas sehingga akan ditampilkan di atas segalanya.

Objek pop-up sudah diatur sekarang, jadi tulis beberapa kode untuk itu. Buat skrip bernama SettingsPopup (lihat daftar berikutnya) dan drag skrip tersebut ke objek pop-up.

Daftar Pengaturan Skrip popup untuk objek pop-up

```
using UnityEngine;
using System.Collections;

public class SettingsPopup : MonoBehaviour {
    public void Open() {
        gameObject.SetActive(true);
    }
    public void Close() {
        gameObject.SetActive(false);
    }
}
```

Selanjutnya, buka UIController.cs untuk melakukan beberapa penyesuaian, seperti yang ditunjukkan pada daftar berikut.

Daftar Menyesuaikan UIController untuk menangani pop-up

```
...
[SerializeField] private SettingsPopup settingsPopup;
void Start() {
    settingsPopup.Close();
}
...
public void OnOpenSettings() {
    settingsPopup.Open();
}
...
```

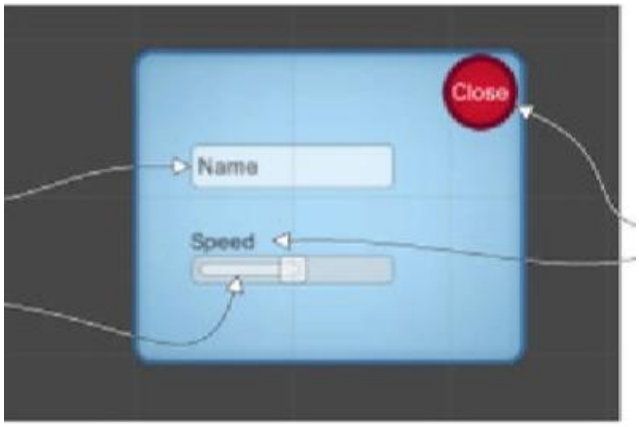
Kode ini menambahkan slot untuk objek pop-up, jadi drag pop-up ke UIController. Sekarang pop-up akan ditutup pada awalnya ketika Anda memainkan game, dan itu akan terbuka ketika Anda mengklik tombol pengaturan.

Saat ini tidak ada cara untuk menutupnya lagi, jadi tambahkan tombol tutup ke popup. Langkah-langkahnya hampir sama dengan tombol yang dibuat sebelumnya: pilih GameObject > UI> Button, posisikan tombol baru di pojok kanan atas pop-up, drag close sprite ke properti Source Image elemen UI ini, dan lalu klik Setel Ukuran Asli untuk mengubah ukuran gambar dengan benar. Berbeda dengan tombol sebelumnya, kami sebenarnya menginginkan label teks ini, jadi pilih teks dan ketik Tutup di bidang teks, dan atur Warna menjadi putih. Dalam tampilan Hierarchy, drag tombol ini ke objek pop-up sehingga akan menjadi anak dari jendela pop-up. Dan sebagai sentuhan akhir polesan, sesuaikan transisi tombol ke nilai Fade Duration .01 dan pengaturan Normal Color yang lebih gelap dari 110, 110, 110, 255. Untuk membuat tombol menutup pop-up, diperlukan entri OnClick; klik tombol + pada panel OnClick tombol, drag jendela pop-up ke dalam slot objek, dan pilih Close() dari daftar fungsi. Sekarang mainkan gamenya dan tombol ini akan menutup jendela pop-up. Jendela pop-up telah

ditambahkan ke HUD. Jendela saat ini kosong, jadi mari kita tambahkan beberapa kontrol berikutnya.

Menetapkan nilai menggunakan bilah geser dan bidang input

Menambahkan beberapa kontrol ke pop-up pengaturan melibatkan dua langkah utama, seperti tombol yang kami buat sebelumnya. Anda membuat elemen UI yang dilampirkan ke kanvas, dan menautkan objek tersebut ke skrip. Kontrol input yang kita butuhkan adalah penggeser dan bidang teks, dan akan ada label teks statis untuk mengidentifikasi penggeser. Pilih GameObject > UI > Text untuk membuat objek teks, GameObject > UI > InputField untuk membuat bidang teks, dan GameObject > UI > Slider untuk membuat objek slider.



Gambar 4.89 Kontrol input ditambahkan ke jendela pop-up

Jadikan ketiga objek anak dari pop-up dengan menyeretnya dalam tampilan Hierarchy dan kemudian posisikan seperti yang ditunjukkan pada gambar, berbaris di tengah pop-up. Atur teks ke Kecepatan sehingga bisa menjadi label untuk penggeser. Bidang input untuk mengetik teks, dan Teks ditampilkan di dalam kotak sebelum player mengetik sesuatu yang lain; atur nilai ini ke Nama. Anda dapat membiarkan opsi Content Type dan Line Type pada defaultnya; jika diinginkan, Anda dapat menggunakan Jenis Konten untuk membatasi pengetikan pada hal-hal seperti hanya huruf atau angka saja, sedangkan Anda dapat menggunakan Jenis Garis untuk beralih dari satu baris ke teks multibaris. *Peringatan* Anda tidak akan dapat mengklik penggeser jika label teks menutupinya. Pastikan objek teks muncul di bawah slider dengan menempatkannya di atas slider di Hierarchy.

Adapun penggeser itu sendiri, beberapa pengaturan muncul di bagian bawah inspector komponen. Nilai Min diatur ke 0 secara default; tinggalkan itu. Nilai Maks default ke 1, tetapi buat 2 untuk contoh ini. Demikian pula, Nilai dan Bilangan Utuh dapat dibiarkan secara default; Nilai mengontrol nilai awal penggeser, dan Bilangan Utuh membatasinya ke 0 1 2 daripada nilai desimal (batasan yang tidak kita inginkan). Jadikan ketiga objek anak dari pop-up dengan menyeretnya dalam tampilan Hierarchy dan kemudian posisikan seperti yang ditunjukkan pada gambar, berbaris di tengah pop-up. Atur teks ke Kecepatan sehingga bisa menjadi label untuk penggeser. Bidang input untuk mengetik teks, dan Teks ditampilkan di dalam kotak sebelum player mengetik sesuatu yang lain; atur nilai ini ke Nama. Anda dapat membiarkan opsi Content Type dan Line Type pada defaultnya; jika diinginkan, Anda dapat menggunakan Jenis Konten untuk membatasi pengetikan pada hal-hal seperti hanya huruf atau angka saja, sedangkan Anda dapat menggunakan Jenis Garis untuk beralih dari satu baris ke teks multibaris. Peringatan Anda tidak akan dapat mengklik penggeser jika label teks menutupinya. Pastikan objek teks muncul di bawah slider dengan menempatkannya di atas slider di Hierarchy.

Adapun penggeser itu sendiri, beberapa pengaturan muncul di bagian bawah inspector komponen. Nilai Min diatur ke 0 secara default; tinggalkan itu. Nilai Maks default ke 1, tetapi buat 2 untuk contoh ini. Demikian pula, Nilai dan Bilangan Utuh dapat dibiarkan secara default; Nilai mengontrol nilai awal penggeser, dan Bilangan Utuh membatasinya ke 0 1 2 daripada nilai desimal (batasan yang tidak kita inginkan).

```
...
public void OnSubmitName(string name) {
   Debug.Log(name);
}
public void OnSpeedValue(float speed) {
   Debug.Log("Speed: " + speed);
}
```

Bagus, ada metode untuk kontrol yang digunakan. Dimulai dengan bidang input, dalam pengaturan Anda akan melihat panel End Edit; peristiwa yang tercantum di sini dipicu saat pengguna selesai mengetik. Tambahkan entri ke panel ini, drag pop-up ke slot objek, dan pilih OnSubmitName() dalam daftar fungsi.

Peringatan Pastikan untuk memilih fungsi di bagian atas panel Edit Akhir, String Dinamis, dan bukan bagian bawah, Parameter Statis. Fungsi OnSubmitName() muncul di kedua bagian, tetapi memilihnya di bawah Parameter Statis hanya akan mengirim satu string yang ditentukan sebelumnya; string dinamis mengacu pada nilai apa pun yang diketik di bidang input. Ikuti langkah-langkah yang sama untuk penggeser: cari panel event di akhir pengaturan komponen (dalam hal ini, panel adalah OnValueChanged), klik + untuk menambahkan entri, drag pop-up pengaturan, dan pilih OnSpeedValue( ) dalam daftar fungsi nilai dinamis. Sekarang kedua kontrol input terhubung ke kode dalam skrip pop-up. Mainkan game, dan tonton konsol saat Anda menggerakkan penggeser atau tekan Enter setelah mengetik input.

Menyimpan pengaturan di antara pemutaran menggunakan PlayerPrefs

Beberapa metode berbeda tersedia untuk menyimpan data persisten di Unity, dan salah satu yang paling sederhana disebut PlayerPrefs. Unity menyediakan cara abstrak (yaitu, Anda tidak perlu khawatir tentang detailnya) untuk menyimpan sejumlah kecil informasi yang berfungsi di semua platform (dengan sistem file yang berbeda). PlayerPrefs tidak terlalu berguna untuk data dalam jumlah besar (di bab 11 kita akan menggunakan metode lain untuk menyimpan kemajuan game), tetapi sangat cocok untuk menyimpan pengaturan.

PlayerPrefs menyediakan perintah sederhana untuk mendapatkan dan menetapkan nilai bernama (berfungsi seperti tabel hash atau kamus). Misalnya, Anda dapat menyimpan pengaturan kecepatan dengan menambahkan baris PlayerPrefs.SetFloat("speed", speed); di dalam metode OnSpeedValue() dari skrip SettingsPopup. Metode itu akan menyimpan float dalam nilai yang disebut kecepatan. Demikian pula, Anda ingin menginisialisasi penggeser ke nilai yang disimpan. Tambahkan kode berikut ke SettingsPopup:

```
using UnityEngine.UI;
...
[SerializeField] private SliderspeedSlider; void Start() {
speedSlider.value = PlayerPrefs.GetFloat("speed", 1);
}
...
```

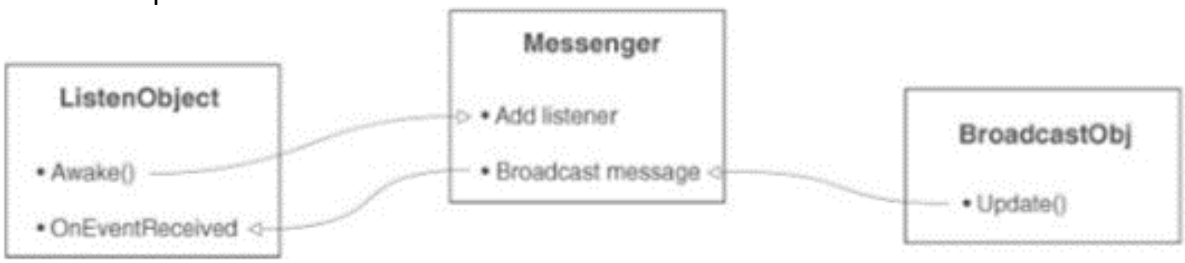
Perhatikan bahwa perintah get memiliki nilai untuk mendapatkan serta nilai default jika kecepatan tidak disimpan sebelumnya. Meskipun kontrol menghasilkan keluaran debug,

mereka tetap tidak benar-benar memengaruhi permainan. Membuat HUD mempengaruhi permainan (dan sebaliknya) adalah topik bagian terakhir dari bab ini.

Memperbarui game dengan menanggapi event

Sampai sekarang, HUD dan permainan utama telah mengabaikan satu sama lain, tetapi mereka harus berkomunikasi bolak-balik. Itu dapat dicapai melalui referensi skrip seperti yang telah kami lakukan untuk jenis komunikasi antar objek lainnya, tetapi pendekatan itu akan memiliki kelemahan besar. Secara khusus, melakukan hal itu akan menyatukan scene dan HUD dengan erat; Anda ingin membuatnya cukup independen satu sama lain sehingga Anda dapat dengan bebas mengedit game tanpa khawatir Anda telah merusak HUD.

Untuk mengingatkan UI tindakan di scene, kami akan menggunakan sistem utusan siaran. Gambar 6.17 mengilustrasikan bagaimana sistem pesan event ini bekerja: skrip dapat mendaftar untuk mendengarkan suatu event, kode lain dapat menyiarkan suatu event, dan pendengar akan diberitahu tentang pesan siaran. Mari kita membahas sistem perpesanan untuk mencapai itu.



Gambar 4.90 Diagram sistem event siaran yang akan kami terapkan

C# memang memiliki sistem bawaan untuk menangani event, jadi Anda mungkin bertanya-tanya mengapa kami tidak menggunakannya. Nah, sistem event bawaan memberlakukan pesan yang ditargetkan, sedangkan kami menginginkan sistem pesan siaran. Sistem yang ditargetkan memerlukan kode untuk mengetahui dengan tepat dari mana pesan berasal; siaran dapat berasal dari mana saja.

Mengintegrasikan sistem event

Untuk mengingatkan UI tindakan di scene, kami akan menggunakan sistem utusan siaran. Meskipun Unity tidak memiliki fitur ini di dalamnya, skrip yang bagus untuk tujuan ini ada secara online. Di antara sumber daya yang tercantum dalam lampiran D adalah wiki komunitas Unify; ini adalah gudang kode gratis yang disumbangkan oleh developer lain. Sistem messenger mereka sangat bagus untuk menyediakan cara terpisah untuk mengkomunikasikan event ke seluruh program. Ketika beberapa kode menyiarkan pesan, kode itu tidak perlu tahu apa-apa tentang pendengar, memungkinkan banyak fleksibilitas dalam beralih atau menambahkan objek. Buat skrip bernama Messenger dan rekatkan kode dari halaman ini di Unify: http://wiki.unity3d.com/index.php/CSharpMessenger_Extended
Kemudian Anda juga perlu membuat skrip yang disebut GameEvent (lihat daftar berikut).

Daftar Skrip GameEvent untuk digunakan dengan Messenger

```
public static class GameEvent {
public const string ENEMY_HIT = "ENEMY_HIT";
public const string SPEED_CHANGED = "SPEED_CHANGED";
}
```

Script dalam daftar mendefinisikan konstanta untuk beberapa pesan event; pesan lebih terorganisir dengan cara ini, dan Anda tidak perlu mengingat dan mengetik string pesan di semua tempat. Sekarang sistem pembawa pesan event siap digunakan, jadi mari kita mulai

menggunakannya. Pertama kita akan berkomunikasi dari scene ke HUD, dan kemudian kita akan pergi ke arah lain.

Menyiarkan dan mendengarkan event dari scene

Hingga kini tampilan skor telah menampilkan pengatur waktu sebagai uji fungsionalitas tampilan teks. Tapi kami ingin menampilkan jumlah musuh yang terkena, jadi mari kita ubah kode di UIController. Pertama-tama hapus seluruh metode Update(), karena itu adalah kode pengujian. Ketika musuh mati, itu akan memancarkan suatu peristiwa, jadi daftar berikut membuat UIController mendengarkan peristiwa itu.

Daftar Menambahkan pendengar event ke UIController

```
...
private int _score;

void Awake() {
   Messenger.AddListener(GameEvent.ENEMY_HIT, OnEnemyHit);
}
void OnDestroy() {
   Messenger.RemoveListener(GameEvent.ENEMY_HIT, OnEnemyHit);
}

void Start() {
   _score = 0;
   scoreLabel.text = _score.ToString();

   settingsPopup.Close();
}

private void OnEnemyHit() {
   _score += 1;
   scoreLabel.text = _score.ToString();
}
...
```

Pertama-tama perhatikan metode Awake() dan OnDestroy(). Sama seperti Start() dan Update(), setiap MonoBehaviour secara otomatis merespons saat objek bangun atau dihapus. Listener ditambahkan dan dihapus di Awake()/OnDestroy(). Listener ini adalah bagian dari sistem pesan siaran, dan memanggil OnEnemyHit() saat pesan itu diterima. OnEnemyHit() menambah skor dan kemudian menempatkan nilai itu di tampilan skor. Pendengar event diatur dalam kode UI, jadi sekarang kita perlu menyiarkan pesan itu setiap kali musuh terkena. Kode untuk menanggapi hit ada di RayShooter.cs, jadi pancarkan pesan seperti yang ditunjukkan pada daftar berikut.

Daftar Siarkan pesan event dari RayShooter

```
...
if (target != null) {
   target.ReactToHit();
   Messenger.Broadcast(GameEvent.ENEMY_HIT);
} else {
...
```

Mainkan game setelah menambahkan pesan itu dan lihat tampilan skor saat Anda menembak musuh. Anda akan melihat hitungan naik setiap kali Anda membuat hit. Itu mencakup pengiriman pesan dari game 3D ke antarmuka 2D, tetapi kami juga ingin contoh yang mengarah ke arah lain.

```
...
public const float baseSpeed = 3.0f;
...
void Awake() {
   Messenger<float>.AddListener(GameEvent.SPEED_CHANGED, OnSpeedChanged);
}
void OnDestroy() {
   Messenger<float>.RemoveListener(GameEvent.SPEED_CHANGED, OnSpeedChanged);
}
...
private void OnSpeedChanged(float value) {
   speed = baseSpeed * value;
}
...
```

Awake() dan OnDestroy() masing-masing menambah dan menghapus, pendengar event di sini, tetapi metode ini memiliki nilai kali ini. Nilai itu digunakan untuk mengatur kecepatan AI yang berkeliaran. Tip Kode di bagian sebelumnya hanya menggunakan peristiwa umum, tetapi sistem pesan ini dapat memberikan nilai bersama dengan pesan. Mendukung sebuah nilai di listener semudah menambahkan definisi tipe; perhatikan <float> ditambahkan ke perintah listener. Sekarang buat perubahan yang sama di FPSInput.cs untuk memengaruhi kecepatan pemutar. Kode di daftar berikutnya hampir sama persis dengan yang ada di daftar 6.10, kecuali bahwa player memiliki nomor yang berbeda untuk baseSpeed.

Daftar Pendengar event ditambahkan ke FPSInput

```
...
public const float baseSpeed = 6.0f;
...
void Awake() {
   Messenger<float>.AddListener(GameEvent.SPEED_CHANGED, OnSpeedChanged);
}
void OnDestroy() {
   Messenger<float>.RemoveListener(GameEvent.SPEED_CHANGED, OnSpeedChanged);
}
...
private void OnSpeedChanged(float value) {
   speed = baseSpeed * value;
}
...
```

Awake() dan OnDestroy() masing-masing menambah dan menghapus, pendengar event di sini, tetapi metode ini memiliki nilai kali ini. Nilai itu digunakan untuk mengatur kecepatan AI yang berkeliaran. *Tip* Kode di bagian sebelumnya hanya menggunakan peristiwa umum, tetapi sistem pesan ini dapat memberikan nilai bersama dengan pesan. Mendukung sebuah nilai di listener semudah menambahkan definisi tipe; perhatikan <float> ditambahkan ke perintah listener.

Sekarang buat perubahan yang sama di FPSInput.cs untuk memengaruhi kecepatan pemutar. Kode di daftar berikutnya hampir sama persis dengan yang ada di daftar 6.10, kecuali bahwa player memiliki nomor yang berbeda untuk baseSpeed.

Terakhir, siarkan nilai kecepatan dari SettingsPopup sebagai respons terhadap penggeser, seperti yang ditunjukkan pada daftar berikut.

Daftar Pesan siaran dari SettingsPopup

```
public void OnSpeedValue(float speed) {
   Messenger<float>.Broadcast(GameEvent.SPEED_CHANGED, speed);
   ...
```

Sekarang kecepatan musuh dan player berubah saat Anda menyesuaikan slider. Tekan Mainkan dan cobalah!

Latihan: Mengubah kecepatan musuh yang muncul

Saat ini nilai kecepatan hanya diperbarui untuk musuh yang sudah ada di scene dan bukan untuk musuh yang baru muncul; musuh baru tidak dibuat pada pengaturan kecepatan yang benar. Saya akan membiarkannya sebagai latihan bagi Anda untuk mengetahui cara mengatur kecepatan pada musuh yang muncul. Berikut petunjuknya: tambahkan pendengar SPEED_CHANGED ke SceneController, karena skrip itu adalah tempat musuh muncul.

Anda sekarang tahu cara membangun antarmuka grafis menggunakan alat UI baru yang ditawarkan oleh Unity. Pengetahuan ini akan berguna di semua proyek masa depan, bahkan saat kami menjelajahi genre game yang berbeda.

BAB 5
MEMBUAT GAME 3D ORANG KETIGA: GERAKAN DAN ANIMASI PLAYER

5.1 OBJEK GAME

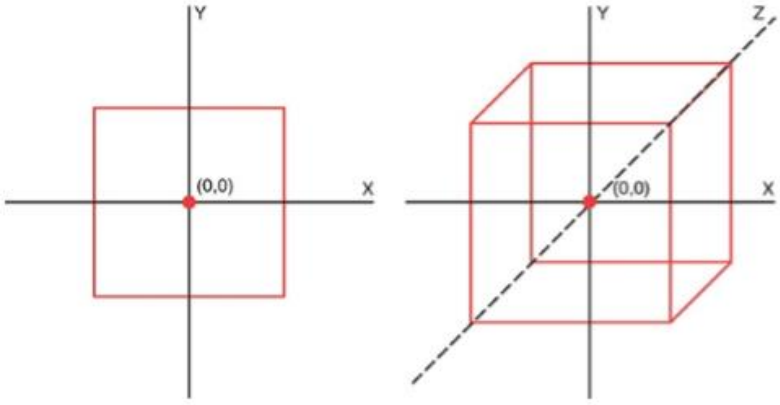
Objek game adalah komponen dasar dari proyek game Unity. Setiap item yang ada dalam sebuah scene adalah, atau didasarkan pada, objek permainan. Dalam jam ini, Anda akan belajar tentang objek game di Unity. Namun, sebelum Anda dapat mulai bekerja dengan objek di Unity, Anda harus mempelajari tentang sistem koordinat 2D dan 3D. Setelah Anda mempelajari sistem tersebut dalam jam ini, Anda akan mulai bekerja dengan objek game Unity bawaan, dan Anda akan mengakhiri jam dengan mempelajari berbagai transformasi objek game. Informasi yang diperoleh pada jam ini adalah dasar dari semua hal lain dalam buku ini. Pastikan untuk meluangkan waktu Anda dan mempelajarinya dengan baik.

Dimensi dan Sistem Koordinat

Untuk semua kemewahan dan keglamorannya, video game adalah konstruksi matematis. Semua properti, gerakan, dan interaksi dapat diringkas menjadi angka. Beruntung bagi Anda, banyak landasan telah diletakkan. Matematikawan telah bekerja keras selama berabad-abad untuk menemukan, menemukan, dan menyederhanakan berbagai proses, dan semua pekerjaan mereka berarti Anda dapat lebih mudah membuat game dengan software modern. Anda mungkin berpikir bahwa objek dalam game hanya ada di luar angkasa secara acak, tetapi sebenarnya setiap ruang game memiliki dimensi, dan setiap objek ditempatkan dalam sistem koordinat (atau kisi).

Menempatkan 2D dalam 3D

Seperti disebutkan sebelumnya, setiap game menggunakan beberapa level dimensi. Sistem dimensi yang paling umum, yang kemungkinan besar Anda kenal, adalah 2D dan 3D (kependekan dari dua dimensi dan tiga dimensi). Sistem 2D adalah sistem datar. Dalam sistem 2D, Anda hanya berurusan dengan elemen vertikal dan horizontal (atau, dengan kata lain: atas, bawah, kiri, dan kanan). Game seperti Tetris, Pong, dan Pac Man adalah contoh game 2D yang bagus. Sistem 3D seperti sistem 2D, tetapi jelas memiliki satu dimensi lagi. Dalam sistem 3D, Anda tidak hanya memiliki horizontal dan vertikal (atas, bawah, kiri, dan kanan) tetapi juga kedalaman (masuk dan keluar). Gambar di bawah ini menggambarkan dengan baik perbedaan antara persegi 2D dan persegi 3D, atau dikenal sebagai kubus. Perhatikan bagaimana penyertaan sumbu kedalaman dalam kubus 3D membuatnya tampak "muncul".



Gambar 5.1 Kotak 2D versus kubus 3D.

Belajar Tentang 2D dan 3D

Unity adalah mesin 3D. Oleh karena itu, semua proyek yang dibuat dengannya secara inheren menggunakan ketiga dimensi. Rahasia sebenarnya adalah bahwa tidak ada lagi yang namanya game 2D sejati. Hardware modern menjadikan semuanya sebagai 3D. Game 2D masih memiliki sumbu z tetapi tidak menggunakannya. Anda mungkin bertanya-tanya mengapa buku ini repot-repot membahas sistem 2D sama sekali. Yang benar adalah bahwa bahkan dalam proyek 3D, masih ada banyak elemen 2D. Tekstur, elemen layar, dan teknik pemetaan semuanya menggunakan sistem 2D. Unity memiliki seperangkat alat yang kuat untuk bekerja dengan game 2D, dan sistem 2D tidak akan hilang dalam waktu dekat.

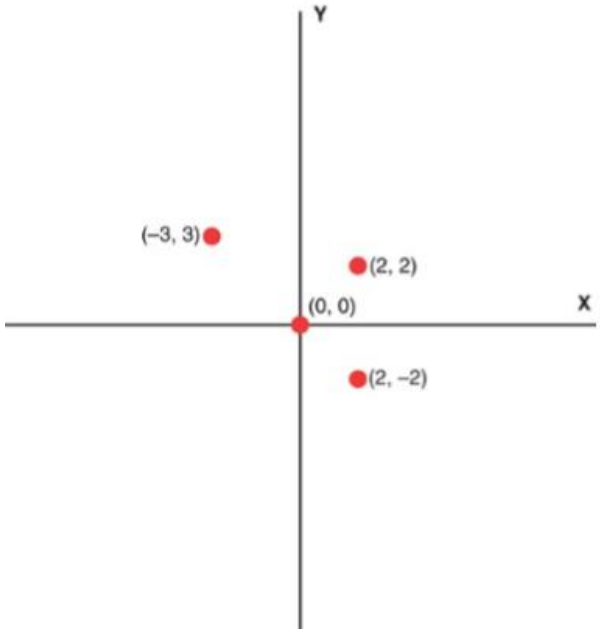
Menggunakan Sistem Koordinat

Persamaan matematis dari sistem dimensi adalah sistem koordinat. Sistem koordinat menggunakan serangkaian garis, yang disebut sumbu (bentuk jamak dari sumbu), dan lokasi, yang disebut titik. Sumbu ini berhubungan langsung dengan dimensi yang mereka tiru. Misalnya, sistem koordinat 2D memiliki sumbu x dan sumbu y, yang masing-masing mewakili arah horizontal dan vertikal. Jika sebuah benda bergerak horizontal, kita katakan benda itu bergerak "sepanjang sumbu x". Demikian pula, sistem koordinat 3D menggunakan sumbu x, sumbu y, dan sumbu z untuk gerakan dan pemosisian horizontal, vertikal, dan kedalaman.

Sintaks Koordinat Umum

Saat mengacu pada posisi objek, Anda biasanya mencantumkan koordinatnya. Mengatakan bahwa sebuah objek adalah 2 pada sumbu x dan 4 pada sumbu y bisa menjadi sedikit rumit. Untungnya, ada cara singkat untuk menulis koordinat. Dalam sistem 2D, Anda menulis koordinat seperti (x, y), dan dalam sistem 3D, Anda menulisnya seperti (x, y, z). Oleh karena itu, untuk suatu benda yang berada 2 pada sumbu x dan 4 pada sumbu y, cukup tuliskan (2, 4). Jika objek itu juga 10 pada sumbu z, Anda akan menulis (2, 4, 10).

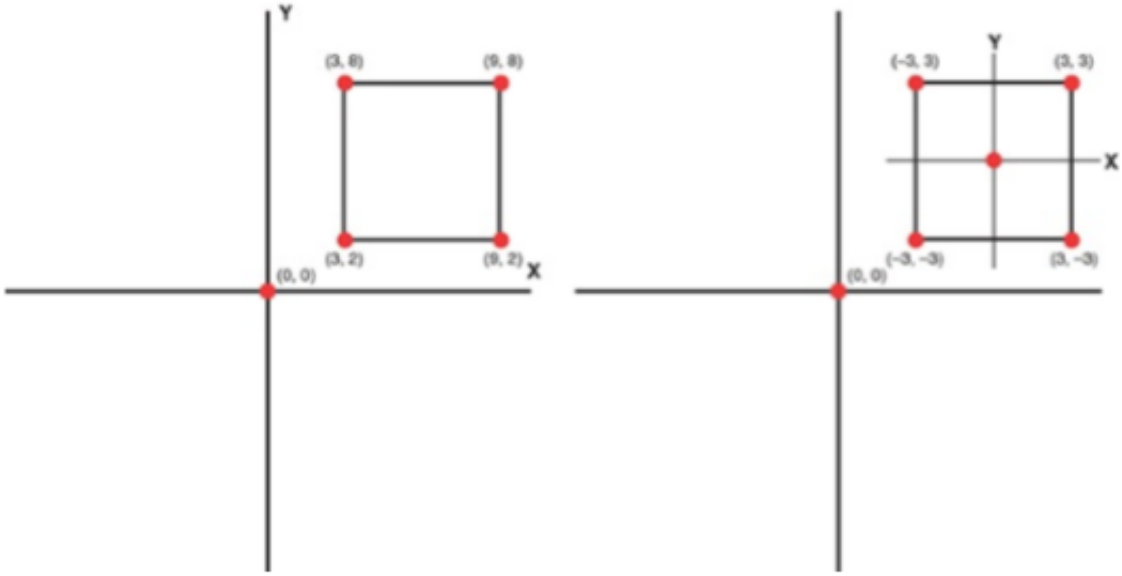
Setiap sistem koordinat memiliki titik di mana semua sumbu berpotongan. Titik ini disebut titik asal, dan koordinat titik asal selalu (0, 0) dalam sistem 2D dan (0, 0, 0) dalam sistem 3D. Titik asal ini sangat penting karena merupakan dasar dari mana semua titik lainnya diturunkan. Koordinat untuk titik lain hanyalah jarak titik itu dari titik asal sepanjang setiap sumbu. Koordinat titik menjadi lebih besar saat bergerak menjauh dari titik asal. Misalnya, saat sebuah titik bergerak ke kanan, nilai sumbu x-nya semakin besar. Ketika bergerak ke kiri, nilai sumbu x semakin kecil hingga melewati titik asal. Pada saat itu, nilai x dari titik tersebut mulai membesar lagi, tetapi juga menjadi negatif. Perhatikan Gambar 2.2. Sistem koordinat 2D ini memiliki tiga titik yang ditentukan. Titik (2, 2) berjarak 2 satuan dari titik asal baik dalam arah x maupun y. Titik (–3, 3) adalah 3 satuan di sebelah kiri titik asal dan 3 satuan di atas titik asal. Titik (2, –2) adalah 2 satuan di sebelah kanan titik asal dan 2 satuan di bawah titik asal.



Gambar 5.2 Poin dalam kaitannya dengan asal.

5.2 KOORDINAT DUNIA VERSUS LOKAL

Anda sekarang telah belajar tentang dimensi dunia game dan tentang sistem koordinat yang menyusunnya. Apa yang telah Anda kerjakan sejauh ini dianggap sebagai sistem koordinat dunia. Pada waktu tertentu, hanya ada satu sumbu x, satu sumbu y, dan satu sumbu z dalam sistem koordinat dunia. Demikian juga, hanya ada satu asal yang dimiliki semua objek. Apa yang Anda mungkin tidak tahu adalah bahwa ada juga sesuatu yang disebut sistem koordinat lokal. Sistem ini unik untuk setiap objek, dan benar-benar terpisah dari sistem koordinat lokal objek lain. Sistem lokal ini memiliki sumbu dan asalnya sendiri yang tidak digunakan objek lain. Gambar 2.3 mengilustrasikan sistem koordinat dunia versus lokal dengan menunjukkan empat titik yang membentuk persegi untuk masing-masing titik.



Gambar 5.3 Koordinat dunia (kiri) versus koordinat lokal (kanan).

Anda mungkin bertanya-tanya untuk apa sistem koordinat lokal jika sistem koordinat dunia digunakan untuk posisi objek. Nanti di jam ini, Anda akan melihat mengubah objek game dan mengasuh objek game. Keduanya membutuhkan sistem koordinat lokal.

Objek Game

Semua bentuk, model, lampu, kamera, sistem partikel, dan sebagainya dalam game Unity memiliki satu kesamaan: Semuanya adalah objek game. Objek permainan adalah unit dasar dari setiap scene. Meskipun sederhana, objek game sangat kuat. Pada dasarnya, objek game tidak lebih dari sekadar transformasi (dibahas lebih detail nanti) dan container. Bagian wadah dari objek permainan ada untuk menampung berbagai komponen yang membuat objek lebih dinamis dan bermakna. Apa yang Anda tambahkan ke objek game terserah Anda. Ada banyak komponen, dan mereka menambahkan sejumlah besar variasi. Sepanjang kursus buku ini, Anda akan belajar menggunakan banyak komponen ini

Objek bawaan

Tidak setiap objek game yang Anda gunakan akan dimulai sebagai objek kosong. Unity memiliki beberapa objek game bawaan yang tersedia untuk digunakan langsung. Anda dapat melihat banyak item yang tersedia dengan mengklik item menu GameObject di bagian atas editor Unity. Sebagian besar pembelajaran menggunakan Unity adalah belajar bekerja dengan objek game bawaan dan kustom.

Membuat Beberapa Objek Game

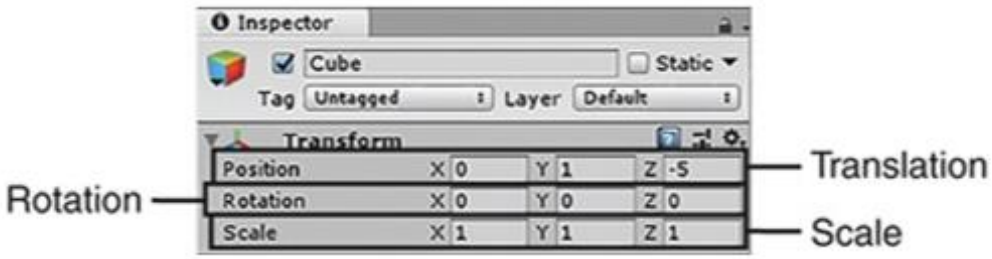
Mari kita luangkan waktu sekarang untuk bekerja dengan objek game. Ikuti langkah-langkah ini untuk membuat beberapa objek dasar dan memeriksa komponennya yang berbeda:

1. Buat new project atau buat scene baru di proyek yang sudah ada.
2. Tambahkan objek game kosong dengan mengklik item menu GameObject dan memilih Create Empty. (Perhatikan bahwa Anda juga dapat membuat objek game kosong dengan menekan Ctrl+Shift+N di PC atau Command+Shift+N di Mac.)
3. Lihat di tampilan Inspector dan perhatikan bahwa objek game yang baru saja Anda buat tidak memiliki komponen selain transformasi. (Setiap objek game memiliki transformasi.) Klik tombol Add Component di Inspector untuk melihat semua komponen yang dapat Anda tambahkan ke objek. Jangan pilih komponen apa pun saat ini.
4. Tambahkan kubus ke proyek Anda dengan mengklik item menu GameObject, arahkan kursor ke Objek 3D, dan pilih Cube dari daftar.
5. Perhatikan berbagai komponen kubus yang tidak dimiliki objek permainan kosong. Komponen mesh membuat kubus terlihat, dan collider membuatnya dapat berinteraksi dengan objek lain pada tingkat fisik.
6. Tambahkan lampu titik ke proyek Anda dengan mengklik drop-down Create di tampilan Hierarchy dan memilih Light > Point Light dari daftar.
7. Perhatikan bahwa lampu titik hanya berbagi komponen transformasi dengan kubus. Ini difokuskan sepenuhnya pada memancarkan cahaya. Cahaya yang dipancarkan oleh titik cahaya ini menambah cahaya terarah yang sudah ada di scene.

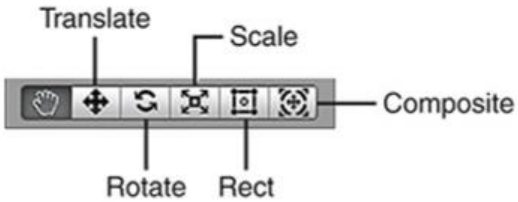
5.3 TRANSFORM

Sejauh ini dalam satu jam ini, Anda telah mempelajari dan menjelajahi berbagai sistem koordinat dan bereksperimen dengan beberapa objek permainan. Saatnya menyatukan

keduanya. Ketika berhadapan dengan objek 3D, Anda sering mendengar istilah transform. Bergantung pada konteksnya, transform adalah kata benda atau kata kerja. Semua objek dalam ruang 3D memiliki posisi, rotasi, dan scale. Jika Anda menggabungkannya, Anda mendapatkan transformasi objek (kata benda). Atau, transformasi dapat menjadi kata kerja jika mengacu pada perubahan posisi objek, rotasi, atau scale. Unity menggabungkan dua arti kata dengan komponen transform. Ingatlah bahwa komponen transform adalah satu-satunya komponen yang harus dimiliki setiap objek game. Bahkan objek game kosong pun memiliki transformasi. Dengan menggunakan komponen ini, Anda berdua dapat melihat transformasi objek saat ini dan mengubah (atau mengubah) transformasi objek. Mungkin terdengar membingungkan sekarang, tetapi ini cukup sederhana. Anda akan menguasainya dalam waktu singkat. Karena transformasi terdiri dari posisi, rotasi, dan scale, maka ada tiga metode terpisah (disebut transformasi) untuk mengubah transformasi: translasi, rotasi, dan penscalean. Transformasi ini dapat dicapai dengan menggunakan Inspector atau alat transformasi. Gambar 2.4 dan 2.5 mengilustrasikan komponen dan alat Inspector mana yang berkorelasi dengan berbagai transformasi.



Gambar 5.4 Ospi Transform di Inspector



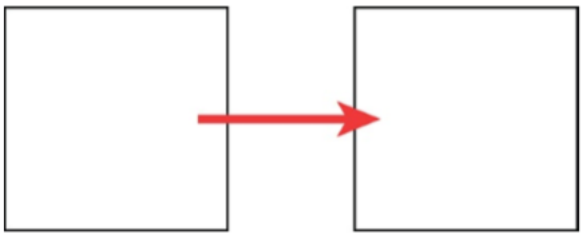
Gambar 5.5 Transform Tool

Periksa Diri Anda Sebelum Anda Memperbaiki Diri

Istilah Rect (kependekan dari persegi panjang) sering dapat dilihat dalam hubungannya dengan objek game 2D. Misalnya, sementara objek 2D seperti sprite memiliki jenis transformasi yang sama dengan objek 3D, Anda dapat menggunakan alat Rect yang lebih sederhana untuk mengontrol posisi, rotasi, dan scalenya sekaligus. (Lihat Gambar 5.6 untuk lokasi alat Rect.) Selanjutnya, objek antarmuka pengguna (UI) di Unity memiliki tipe transformasi yang berbeda yang disebut transformasi Rect. Jam 14, "Antarmuka Pengguna," membahas ini secara lebih rinci; untuk saat ini, pahami saja bahwa mereka adalah setara 2D dari transformasi normal yang digunakan secara eksklusif untuk UI.

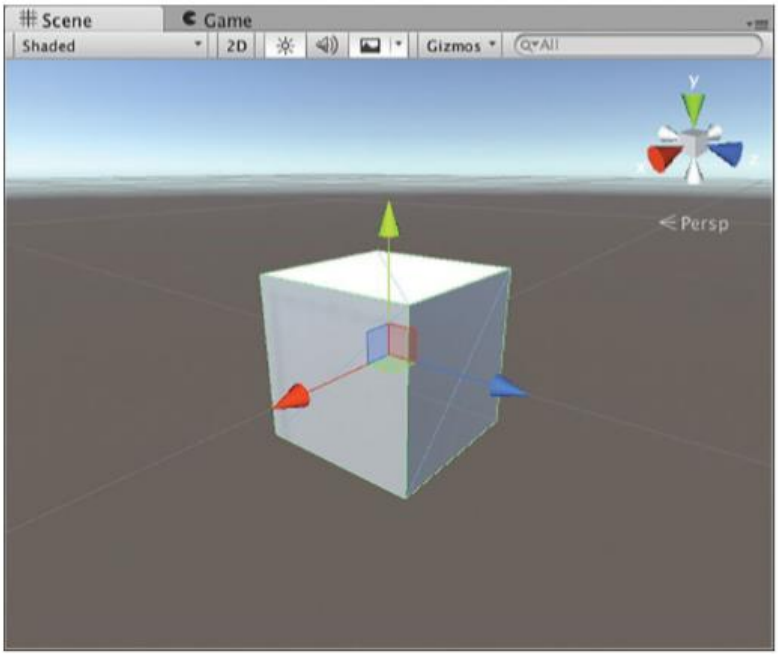
Translasi

Proses mengubah posisi koordinat suatu objek dalam sistem 3D disebut translasi, dan ini adalah transformasi paling sederhana yang dapat Anda terapkan pada suatu objek. Saat Anda translate suatu objek, objek tersebut digeser sepanjang sumbu. Gambar 5.6 menunjukkan persegi yang sedang ditranslate sepanjang sumbu x.



Gambar 5.6 Contoh terjemahan.

Saat Anda memilih alat Translate (tombol pintas: W), objek apa pun yang telah Anda pilih akan sedikit berubah dalam tampilan Scene. Lebih khusus lagi, Anda melihat tiga panah muncul, menunjuk menjauh dari pusat objek di sepanjang tiga sumbu. Ini adalah alat terjemahan, dan mereka membantu Anda memindahkan objek Anda di sekitar scene. Mengklik dan menahan salah satu panah sumbu ini menyebabkannya menjadi kuning. Kemudian, jika Anda menggerakkan mouse, objek bergerak di sepanjang sumbu itu.



Gambar 5.7 Translation Gimoz

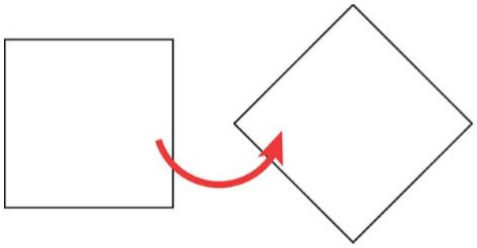
Transform Component dan Transform Tools

Unity menyediakan dua cara untuk mengelola transformasi objek Anda. Mengetahui kapan harus menggunakan masing-masing adalah penting. Saat Anda mengubah transformasi objek dalam tampilan Scene dengan alat transformasi, data transformasi juga berubah dalam tampilan Inspector. Seringkali lebih mudah untuk membuat perubahan besar pada transformasi objek menggunakan tampilan Inspector karena Anda dapat mengubah nilai sesuai kebutuhan. Alat transformasi, bagaimanapun, lebih berguna untuk perubahan kecil yang cepat. Belajar menggunakan keduanya secara bersamaan akan sangat meningkatkan alur kerja Anda.

5.4 ROTASI

Memutar objek tidak memindahkannya di ruang angkasa. Sebaliknya, itu mengubah hubungan objek ke ruang itu. Lebih sederhana lagi, rotasi memungkinkan Anda untuk

mendefinisikan kembali arah mana sumbu x, y, dan z menunjuk untuk objek tertentu. Ketika sebuah benda berputar pada suatu sumbu, dikatakan bahwa benda tersebut berputar terhadap sumbu tersebut. Gambar 2.8 menunjukkan sebuah persegi yang diputar terhadap sumbu z

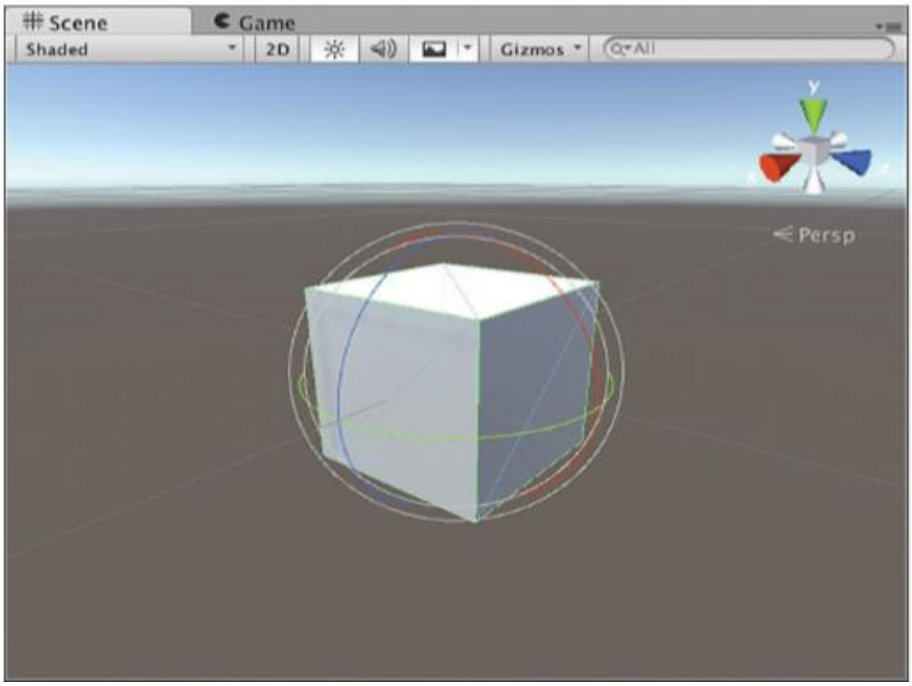


Gambar 5.8 Rotasi terhadap sumbu z.

Menentukan Sumbu Rotasi

Jika Anda tidak yakin sumbu mana yang Anda perlukan untuk memutar objek untuk mendapatkan efek yang diinginkan, Anda dapat menggunakan metode mental sederhana. Satu sumbu pada satu waktu, berpura-puralah bahwa benda itu dijepit di tempatnya oleh peniti yang sejajar dengan sumbu itu. Benda itu hanya bisa berputar di sekitar pin yang tertancap di dalamnya. Sekarang, tentukan pin mana yang memungkinkan objek berputar seperti yang Anda inginkan. Itu adalah sumbu yang Anda butuhkan untuk memutar objek.

Sama seperti alat Terjemahan, memilih alat rotate (tombol pintas: E) menyebabkan alat rotasi muncul di sekitar objek Anda. Gizmos ini adalah lingkaran yang mewakili jalur rotasi objek di sekitar sumbu. Mengklik dan menyeret salah satu lingkaran ini akan mengubahnya menjadi kuning dan memungkinkan Anda untuk memutar objek pada sumbu itu

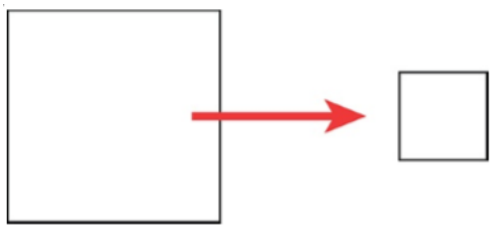


Gambar 5.9 Tool Rotate gimoz

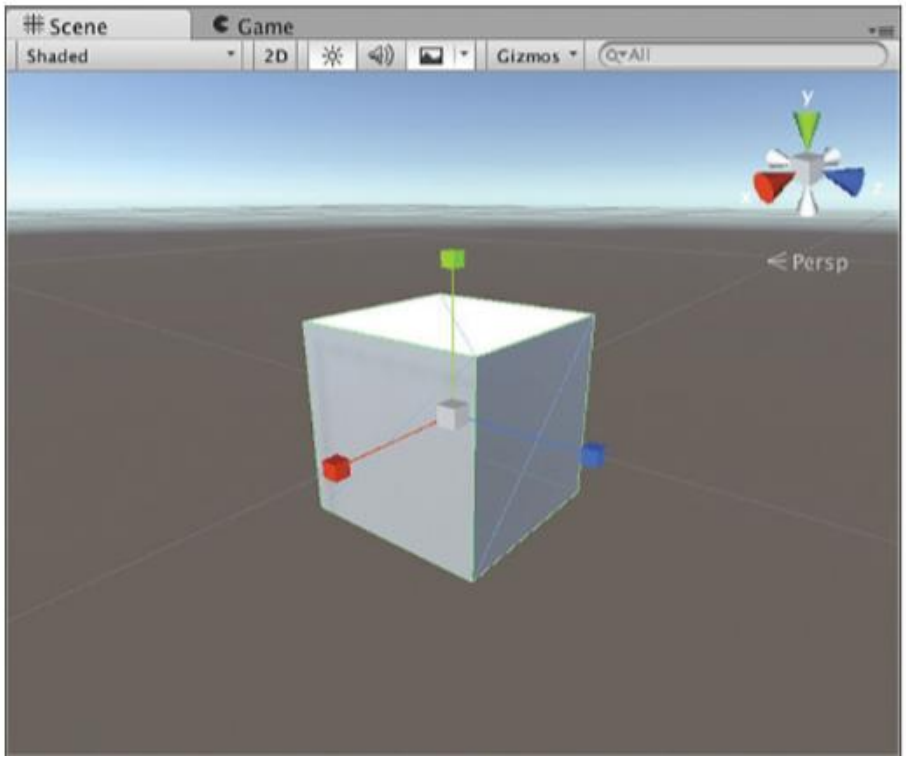
5.5 SCALLING

Penscalean menyebabkan objek tumbuh atau menyusut dalam ruang 3D. Transformasi ini sangat mudah dan mudah digunakan. Scaling objek pada sumbu apa pun menyebabkan

ukurannya berubah pada sumbu itu. Gambar 2.10 menunjukkan persegi yang diperkecil pada sumbu x dan y. Gambar dibawah ini menunjukkan seperti apa alat penscalean saat Anda memilih alat Penscalean (tombol pintas: r).



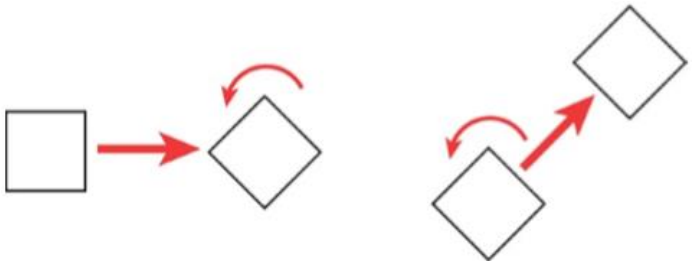
Gambar 5.10 Sclaing sumbu x dan y



Gambar 5.11 Scaling gimoz

Bahaya Transformasi

Seperti disebutkan sebelumnya dalam jam ini, transformasi menggunakan sistem koordinat lokal. Oleh karena itu, perubahan yang dilakukan berpotensi berdampak pada transformasi di masa depan. Misalnya, pada Gambar dibawah ini, perhatikan bagaimana dua transformasi yang sama, ketika diterapkan dalam urutan terbalik, memiliki efek yang sangat berbeda.



Gambar 5.12 Efek dari urutan transformasi.

Seperti yang Anda lihat, tidak memperhatikan urutan transformasi dapat memiliki konsecookiesnsi yang tidak terduga. Untungnya, transformasi memiliki efek konsisten yang dapat direncanakan:

- **Translation**: Terjemahan adalah transformasi yang cukup inert. Ini berarti bahwa setiap perubahan yang diterapkan setelahnya umumnya tidak terpengaruh.
- **Rotate**: Rotasi mengubah orientasi sumbu sistem koordinat lokal. Setiap terjemahan yang diterapkan setelah rotasi menyebabkan objek bergerak sepanjang sumbu baru. Jika Anda memutar objek 180 derajat terhadap sumbu z, misalnya, dan kemudian bergerak ke arah y positif, objek akan tampak bergerak ke bawah bukannya ke atas.
- **Scaling**: Penscalean secara efektif mengubah ukuran kisi koordinat lokal. Pada dasarnya, ketika Anda scaling objek menjadi lebih besar, Anda benar-benar scaling sistem koordinat lokal menjadi lebih besar. Hal ini menyebabkan objek tampak membesar. Perubahan ini bersifat multiplikasi. Misalnya, jika sebuah objek discalekan ke 1 (alamiah, ukuran default) dan kemudian ditranslate 5 unit sepanjang sumbu x, objek tampak bergerak 5 unit ke kanan. Namun, jika benda yang sama diperbesar menjadi 2, maka translate 5 satuan pada sumbu x akan mengakibatkan benda tampak bergerak 10 satuan ke kanan. Ini karena sistem koordinat lokal sekarang berukuran dua kali lipat, dan 5 kali 2 sama dengan 10. Kebalikannya, jika objek diperbesar menjadi .5 dan kemudian dipindahkan, itu akan tampak hanya bergerak 2,5 unit (0,5 × 5 = 2,5) .

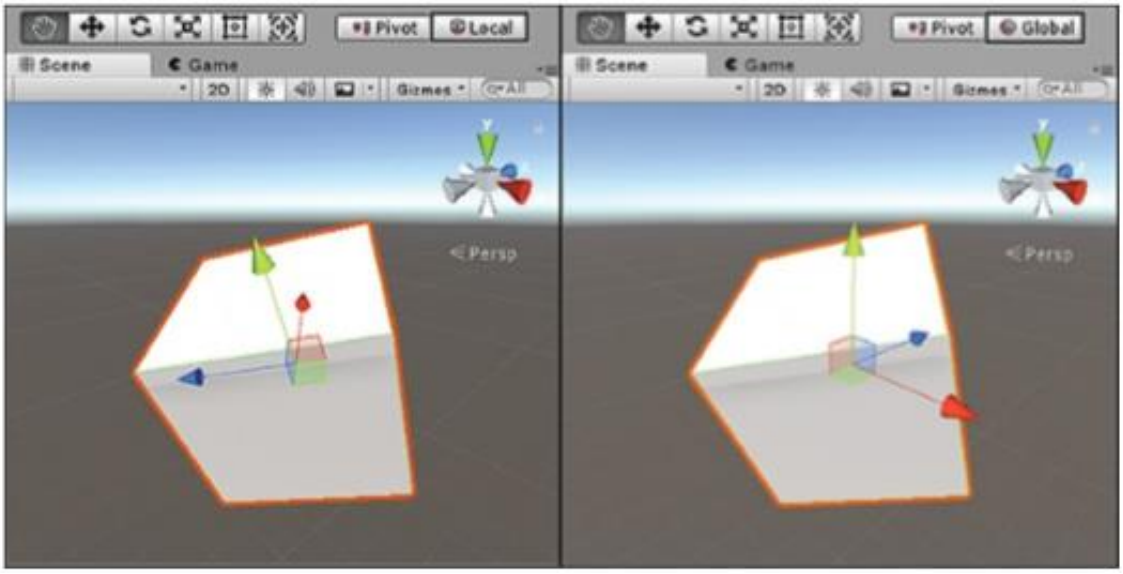
Setelah Anda memahami aturan ini, menentukan bagaimana suatu objek akan berubah dengan serangkaian transformasi menjadi mudah. Efek ini bergantung pada apakah Anda memilih Lokal atau Global, dan mereka perlu membiasakan diri, jadi ada baiknya bereksperimen.

Semua Tool Sekaligus

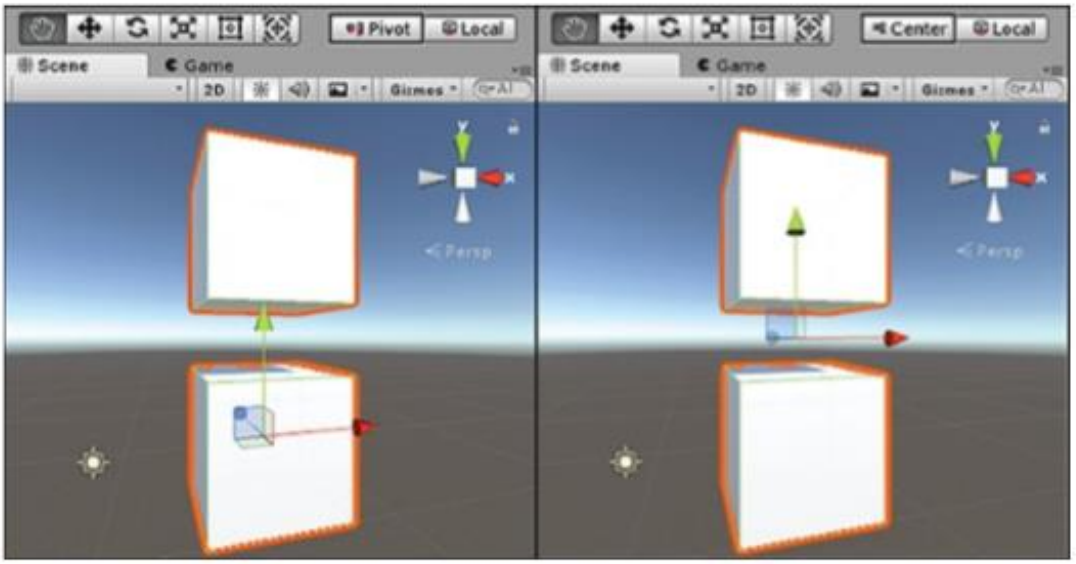
Alat scene terakhir pada toolbar adalah alat Komposit baru (tombol pintas: y). Alat ini memungkinkan Anda translate, memutar, dan scaling objek 3D sekaligus.

Posisi Gizmo

Opsi penempatan alat bisa sangat membingungkan bagi pengguna baru yang tidak tahu apa yang mereka lakukan. Secara efektif, opsi ini mengontrol posisi dan penyelarasan alat transformasi dalam tampilan scene. Lihat Gambar 5.13 untuk melihat bagaimana penyelarasan alat terjemahan berubah tergantung pada apakah Anda memiliki opsi yang disetel ke ruang koordinat Lokal atau Global. Selain itu, lihat Gambar 5.14 untuk mempelajari bagaimana penempatan gizmos berubah ketika Anda memiliki beberapa objek yang dipilih. Dengan opsi Pivot dipilih, fungsi gizmos berpusat pada objek pertama yang dipilih. Dengan opsi Center dipilih, gizmos ditempatkan dan berfungsi di tengah semua objek yang dipilih.



Gambar 5.13 Penyelarasan gizmos untuk Lokal versus Global.



Gambar 5.14 Posisi gizmos untuk Pivot versus Center.

Secara umum, Anda harus mengatur gizmos Anda ke Pivot dan Lokal untuk menghindari kebingungan dan membuat pekerjaan sehari-hari Anda di Unity lebih mudah.

Transformasi dan Objek Bersarang

Di Jam 1, "Pengantar Kesatuan," Anda telah mempelajari cara menyusun objek game dalam tampilan Hierarki (dengan menyeret satu objek ke objek lainnya). Ingatlah bahwa ketika Anda memiliki objek yang bersarang di dalam objek lain, objek tingkat atas adalah induknya, dan objek lainnya adalah anak. Transformasi yang diterapkan ke objek induk berfungsi seperti biasa. Objek dapat dipindahkan, discalekan, dan diputar. Yang istimewa adalah bagaimana objek anak berperilaku. Setelah bersarang, transformasi objek anak relatif terhadap objek induk—bukan relatif terhadap dunia. Oleh karena itu, posisi objek anak tidak didasarkan pada jaraknya dari titik asal tetapi jarak dari objek induk. Jika objek induk diputar, objek anak bergerak dengannya. Jika Anda melihat rotasi anak, bagaimanapun, itu tidak akan menunjukkan bahwa itu telah diputar sama sekali. Hal yang sama berlaku untuk penscalean.

Jika Anda scaling objek induk, ukurannya juga berubah. Scale objek anak tetap tidak berubah. Anda mungkin bingung dengan ini. Ingat bahwa ketika transformasi diterapkan, itu tidak diterapkan pada objek tetapi pada sistem koordinat objek. Sebuah objek tidak diputar; sistem koordinatnya diputar. Efeknya adalah objek tampak berputar. Ketika sistem koordinat objek anak didasarkan pada sistem koordinat lokal orang tua, setiap perubahan pada sistem induk secara langsung mengubah anak (tanpa anak mengetahuinya).

5.6 MODEL, MATERIAL DAN TEKSTUR

Dasar-dasar Model Video game tidak akan menjadi sangat video tanpa komponen grafis. Dalam game 2D, grafiknya terdiri dari gambar datar yang disebut sprite. Dalam game 2D, yang perlu Anda lakukan hanyalah mengubah posisi x dan y dari sprite dan membalik beberapa sprite secara berurutan, dan mata pemirsa tertipu untuk percaya bahwa ia melihat gerakan dan animasi yang sebenarnya. Namun, dalam game 3D, semuanya tidak sesederhana itu. Di dunia dengan sumbu ketiga, objek harus memiliki volume untuk menipu mata. Game menggunakan sejumlah besar objek; oleh karena itu, mereka perlu memproses sesuatu dengan cepat. Jerat memungkinkan itu. Jaring, yang paling sederhana, adalah serangkaian segitiga yang saling berhubungan. Segitiga ini membangun satu sama lain dalam strip untuk membentuk objek dasar hingga sangat kompleks. Strip ini memberikan definisi 3D dari model dan dapat diproses dengan sangat cepat. Namun, jangan khawatir: Unity menangani semua ini untuk Anda sehingga Anda tidak perlu mengelolanya sendiri. Nanti di jam ini, Anda akan melihat bagaimana segitiga dapat membentuk berbagai bentuk dalam tampilan Unity Scene.

Mengapa Segitiga?

Anda mungkin bertanya-tanya mengapa objek 3D seluruhnya terdiri dari segitiga. Jawabannya sederhana: Komputer memproses grafik sebagai rangkaian titik, atau dikenal sebagai simpul. Semakin sedikit simpul yang dimiliki suatu objek, semakin cepat objek tersebut dapat ditarik. Segitiga memiliki dua sifat yang membuatnya diinginkan. Yang pertama adalah bahwa setiap kali Anda memiliki segitiga tunggal, Anda hanya perlu satu simpul lagi untuk membuat yang lain. Jadi untuk membuat satu segitiga, Anda membutuhkan tiga simpul, untuk membuat dua segitiga Anda hanya perlu empat, dan untuk membuat tiga segitiga Anda hanya perlu lima. Ini membuat segitiga sangat efisien. Properti kedua yang membuat segitiga diinginkan adalah bahwa dengan membuat strip segitiga, Anda dapat memodelkan objek 3D apa pun. Tidak ada bentuk lain yang memberi Anda tingkat fleksibilitas dan kinerja itu.

Terminologi: Model atau Mesh?

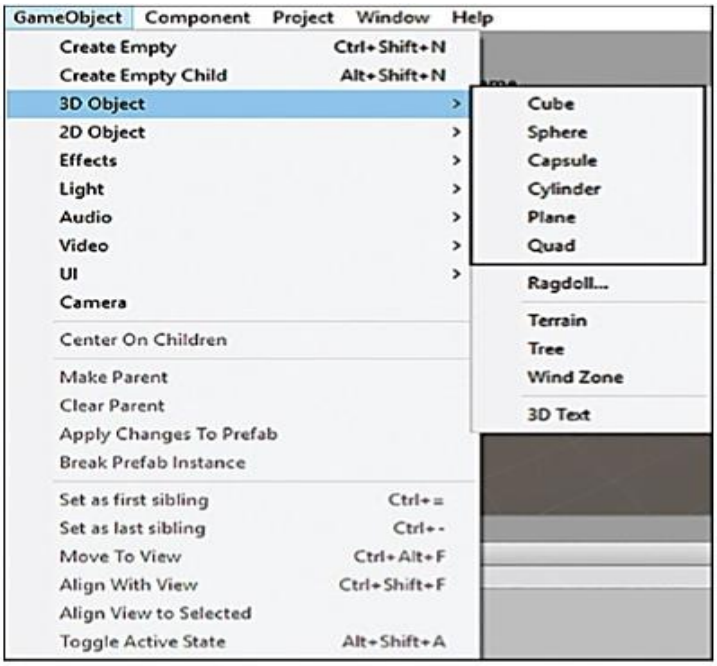
Istilah model dan mesh serupa, dan Anda sering dapat menggunakannya secara bergantian. Namun, ada perbedaan. Sebuah mesh berisi semua titik dan garis yang mendefinisikan bentuk 3D dari suatu objek. Ketika Anda mengacu pada bentuk atau bentuk model, Anda benar-benar mengacu pada mesh. mesh juga bisa disebut geografi model, atau geo-nya. Model, oleh karena itu, adalah objek yang berisi mesh. Sebuah model memiliki mesh untuk menentukan dimensinya, tetapi juga dapat berisi animasi, tekstur, material, shader, dan mesh lainnya. Aturan umum yang baik adalah ini: Jika item tersebut berisi apa pun selain informasi titik, itu adalah model; jika tidak, itu adalah mesh.

Bagaimana dengan 2D?

Jam ini mencakup banyak informasi tentang rendering objek 3D. Meskipun itu berguna, bagaimana jika Anda hanya ingin membuat game 2D? Apakah Anda masih harus repot dengan pelajaran ini (atau pelajaran lain yang tidak mencakup 2D)? Tentu saja! Seperti yang

disebutkan di Jam 2, "Objek Game," sebenarnya tidak ada game 2D lagi. Sprite hanyalah tekstur yang diterapkan pada objek 3D datar. Pencahayaan dapat diterapkan pada objek 2D. Bahkan kamera yang digunakan untuk game 2D dan 3D pun identik. Semua konsep yang dipelajari di sini dapat diterapkan langsung ke game 2D Anda karena game 2D adalah benar-benar game 3D. Saat Anda lebih banyak berlatih dengan Unity, Anda akan segera melihat bahwa garis antara 2D dan 3D memang sangat kabur!

Built-in 3D Objects Unity hadir dengan beberapa mesh built-in dasar (atau primitif) untuk Anda gunakan. Mereka cenderung menjadi bentuk sederhana yang melayani utilitas sederhana atau dapat digabungkan untuk membuat objek yang lebih kompleks. Gambar 3.1 menunjukkan mesh built-in yang tersedia. (Anda bekerja dengan kubus dan bola di jam-jam sebelumnya.)



Gambar 5.15 Mesh bawaan di Unity.

5.7 PEMODELAN DENGAN SIMPLE MESH

Apakah Anda memerlukan objek kompleks dalam permainan Anda tetapi tidak dapat menemukan jenis model yang tepat untuk digunakan? Dengan menumpuk objek di Unity, Anda dapat dengan mudah membuat model sederhana menggunakan mesh bawaan. Tempatkan mata mesh di dekat satu sama lain sehingga membentuk tampilan kasar yang Anda inginkan. Kemudian sarangkan semua objek di bawah satu objek pusat. Dengan cara ini, ketika Anda memindahkan orang tua, semua anak juga bergerak. Ini mungkin bukan cara tercantik untuk membuat model untuk sebuah game, tetapi ini akan berguna dalam keadaan darurat dan berguna untuk membuat prototipe!

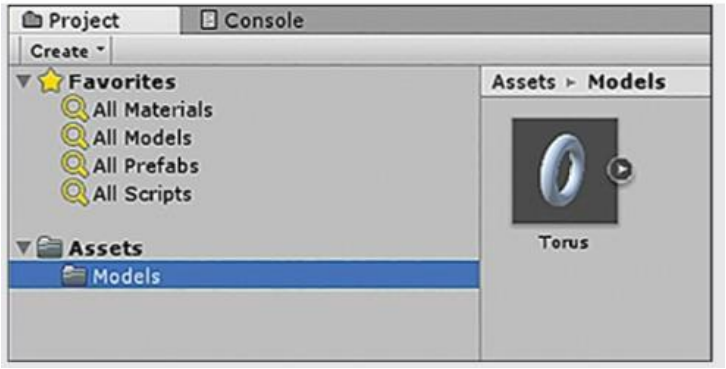
Mengimpor Model

Memiliki model bawaan memang bagus, tetapi sebagian besar waktu gim Anda akan membutuhkan aset seni yang sedikit lebih kompleks. Untungnya, Unity membuatnya lebih mudah untuk membawa model 3D Anda sendiri ke dalam proyek Anda. Hanya menempatkan file yang berisi model 3D di folder Aset Anda sudah cukup untuk membawanya ke dalam proyek. Dari sana, Anda dapat menyeretnya ke dalam scene atau hierarki untuk membangun

objek game di sekitarnya. Secara native, Unity mendukung file .fbx, .dae, .3ds, .dxf, dan .obj. Ini memungkinkan Anda untuk bekerja dengan hampir semua alat pemodelan 3D.

Mengimpor Model 3D Anda Sendiri Ikuti langkah-langkah berikut untuk membawa mode 3D kustom ke dalam proyek Unity:

1. Buat proyek atau scene Unity baru. 2.
2. Dalam tampilan Proyek, buat folder baru bernama Model di bawah folder Aset dengan mengklik kanan folder Aset dan memilih Buat > Folder.
3. Temukan file Torus.fbx yang disediakan untuk Anda di folder Hour 3 dari file buku.
4. Dengan browser file sistem operasi dan editor Unity terbuka dan berdampingan, klik dan drag file Torus.fbx dari browser file ke folder Models yang Anda buat di langkah 2. Di Unity, klik folder Models untuk melihat file Torus baru. Jika Anda melakukannya dengan benar, tampilan Proyek Anda akan terlihat seperti Gambar 3.2. (Catatan: Jika Anda menggunakan versi Unity yang lebih lama atau telah mengubah pengaturan editor Anda, Anda mungkin juga memiliki folder Material yang dibuat secara otomatis. Jika ini masalahnya, itu tidak menjadi masalah. Material akan dibahas lebih rinci nanti di bagian ini. jam.)



Gambar 5.16 Tampilan Proyek setelah model Torus ditambahkan.

5. Drag aset Torus dari folder Models ke tampilan Scene. Perhatikan bagaimana objek game Torus ditambahkan ke scene yang berisi filter mesh dan penyaji mesh. Ini memungkinkan Torus untuk ditarik ke layar.
6. Saat ini torus di scene sangat kecil, jadi pilih aset Torus di project view dan lihat di Inspector view. Ubah nilai faktor scale dari 1 menjadi 100 dan klik tombol Terapkan di kanan bawah Inspector.

5.8 MODEL DAN ASSET STORE

Anda tidak perlu menjadi ahli modeler untuk membuat game dengan Unity. Asset Store menyediakan cara sederhana dan efektif untuk menemukan model yang sudah jadi dan mengimpornya ke dalam proyek Anda. Secara umum, model di Asset Store bisa gratis atau berbayar dan datang sendiri atau dalam kumpulan model serupa. Beberapa model hadir dengan teksturnya sendiri, dan beberapa di antaranya hanyalah data mesh.

Mengunduh Model dari Asset Store

Mari kita lihat cara menemukan dan mengunduh model dari Unity's Asset Store. Ikuti langkah-langkah ini untuk mendapatkan model bernama Robot Kyle dan mengimpornya ke scene Anda:

1. Buat scene baru (dengan memilih File > New Scene). Dalam tampilan Proyek, ketik Robot Kyle t: Model di bilah pencarian

Gambar 5.17 Menemukan aset model.

2. Di bagian filter pencarian, klik tombol Asset Store (lihat Gambar 5.18). Jika kata-kata ini tidak terlihat, Anda mungkin perlu mengubah ukuran jendela editor atau jendela tampilan Proyek. Anda juga harus terhubung ke Internet.
3. Cari Robot Kyle, diterbitkan oleh Unity Technologies. Anda dapat melihat penerbit dengan mengklik aset dan memeriksa tampilan Inspector (lihat Gambar 5.19).

Gambar 5.18 Tampilan Inspector aset.

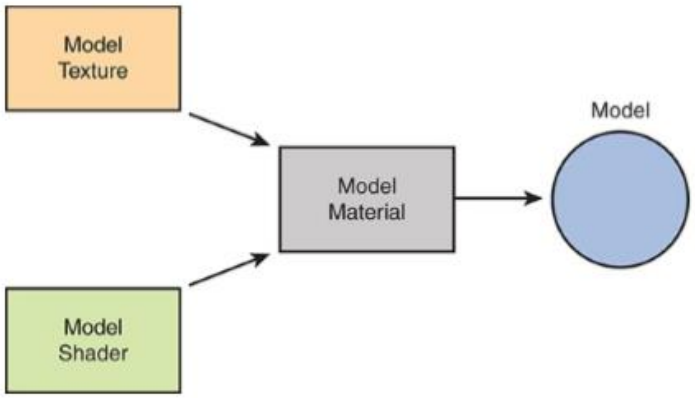
4. Pada tampilan Inspector, klik Import Package. Pada titik ini, Anda mungkin diminta untuk memberikan kredensial akun Unity Anda. Jika Anda mengalami masalah dengan

ini, saya juga telah menyediakan file dalam file buku jam ini sebagai paket Unity; cukup klik dua kali untuk mengimpornya.

5. Ketika dialog Importing Package terbuka, biarkan semuanya tercentang dan klik Import.
6. Temukan model robot di bawah Assets\Robot Kyle\Model dan drag ke tampilan Scene. Perhatikan bahwa model akan cukup kecil dalam tampilan Scene; Anda mungkin perlu bergerak lebih dekat untuk melihatnya.

5.9 TEKSTUR, SHADER, DAN MATERIAL

Menerapkan aset grafis ke model 3D dapat menjadi hal yang menakutkan bagi pemula yang tidak terbiasa dengan prosesnya. Unity menggunakan alur kerja yang sederhana dan spesifik yang memberi Anda banyak kekuatan dalam menentukan dengan tepat bagaimana Anda menginginkan sesuatu terlihat. Aset grafis dipecah menjadi tekstur, shader, dan material. Masing-masing dibahas secara individual di bagiannya sendiri, dan Gambar 5.19 menunjukkan bagaimana mereka cocok satu sama lain. Perhatikan bahwa tekstur tidak diterapkan langsung ke model. Sebagai gantinya, tekstur dan shader diterapkan pada material. Bahan-bahan tersebut pada gilirannya diterapkan pada model. Dengan cara ini, tampilan model dapat ditukar atau dimodifikasi dengan cepat dan bersih tanpa banyak pekerjaan.



Gambar 5.19 Alur kerja Asset Model

Texture

Tekstur adalah gambar datar yang diterapkan pada objek 3D. Mereka bertanggung jawab atas model yang penuh warna dan menarik, bukannya kosong dan membosankan. Mungkin aneh untuk berpikir bahwa gambar 2D dapat diterapkan ke model 3D, tetapi ini adalah proses yang cukup mudah setelah Anda terbiasa dengannya. Pikirkan tentang kaleng sup sejenak. Jika Anda melepas label dari kaleng, Anda akan melihat bahwa itu adalah selembar kertas datar. Label itu seperti tekstur. Setelah label dicetak, label dibungkus dengan kaleng 3D untuk memberikan tampilan yang lebih menyenangkan. Sama seperti semua aset lainnya, menambahkan tekstur ke proyek Unity itu mudah. Mulailah dengan membuat folder untuk tekstur Anda; nama yang bagus adalah Tekstur. Kemudian drag tekstur apa pun yang Anda inginkan dalam proyek Anda ke dalam folder Tekstur yang baru saja Anda buat. Itu dia! Membayangkan bagaimana tekstur membungkus kaleng itu baik-baik saja, tetapi bagaimana dengan objek yang lebih kompleks? Saat membuat model yang rumit, biasanya menghasilkan sesuatu yang disebut unwrap. Pembukaannya agak mirip dengan peta yang menunjukkan dengan tepat bagaimana tekstur datar akan membungkus kembali model. Jika Anda melihat folder Tekstur di bawah folder Robot Kyle dari awal jam ini, Anda melihat tekstur Robot_Color.

Kelihatannya aneh, tapi itulah tekstur yang tidak terbungkus untuk modelnya. Pembuatan bungkus, model, dan tekstur adalah bentuk seni dan tidak tercakup dalam teks ini. Pengetahuan awal tentang cara kerjanya sudah cukup pada tingkat ini.

Tekstur Aneh

Nanti di jam ini, Anda akan menerapkan beberapa tekstur ke model. Anda mungkin memperhatikan bahwa teksturnya sedikit melengkung atau terbalik ke arah yang salah. Ketahuilah bahwa ini bukan kesalahan atau kesalahan. Ini terjadi saat Anda mengambil tekstur 2D persegi panjang dasar dan menerapkannya ke model. Model tidak tahu arah mana yang benar, jadi ia menerapkan tekstur semampunya. Jika Anda ingin menghindari masalah ini, gunakan tekstur yang dirancang khusus untuk (yaitu, dibuka untuk) model yang Anda gunakan.

Shader

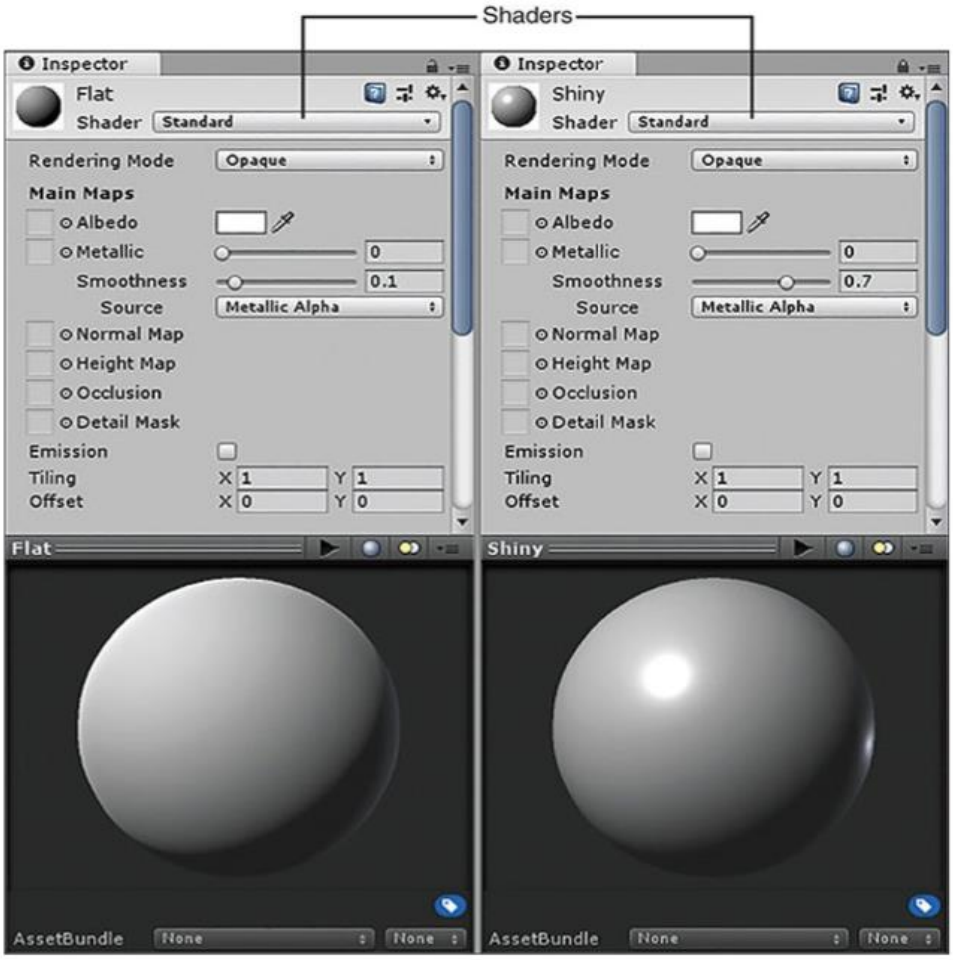
Sementara tekstur model menentukan apa yang digambar di permukaannya, shader menentukan cara menggambarnya. Berikut cara lain untuk melihatnya: Material adalah antarmuka antara Anda dan shader. Materi memberi tahu Anda apa yang dibutuhkan shader untuk membuat objek, dan Anda menyediakan item tersebut agar terlihat seperti yang Anda inginkan. Ini mungkin tampak tidak masuk akal sekarang, tetapi nanti, saat Anda membuat materi, Anda akan mulai memahami cara kerjanya. Sebagian besar informasi tentang shader dibahas nanti jam ini karena Anda tidak dapat menggunakan shader tanpa bahan. Faktanya, banyak informasi yang harus dipelajari tentang materi sebenarnya tentang shader materi.

Latihan Pikiran

Jika Anda kesulitan memahami cara kerja shader, pertimbangkan skenario ini: Bayangkan Anda memiliki sepotong kayu. Sifat fisik kayu adalah jaringnya; warna, tekstur, dan elemen yang terlihat adalah teksturnya. Sekarang ambil potongan kayu itu dan tuangkan air di atasnya. Kayunya masih memiliki mesh yang sama. Masih terbuat dari bahan yang sama (kayu). Ini terlihat berbeda. Warnanya sedikit lebih gelap dan mengkilat. Dalam contoh ini, Anda memiliki dua "shader": kayu kering dan kayu basah. "Shader" kayu basah memiliki sesuatu yang ditambahkan yang membuatnya terlihat sedikit berbeda tanpa benar-benar mengubahnya.

Material

Seperti disebutkan sebelumnya, bahan tidak lebih dari wadah untuk shader dan tekstur yang dapat diterapkan pada model. Sebagian besar penyesuaian bahan tergantung pada shader yang dipilih untuk itu, meskipun semua shader memiliki beberapa fungsi umum. Untuk membuat material baru, mulailah dengan membuat folder Materials. Kemudian klik kanan folder tersebut dan pilih Create > Material. Berikan materi Anda beberapa nama deskriptif, dan Anda selesai. Gambar 3.6 menunjukkan dua bahan dengan pengaturan shader yang berbeda. Perhatikan bagaimana keduanya menggunakan Standard shader yang sama. Masing-masing memiliki warna dasar albedo putih (lebih lanjut tentang albedo nanti), tetapi mereka memiliki pengaturan Smoothness yang berbeda. Bahan Flat memiliki Smoothness yang rendah, sehingga pencahayaan terlihat sangat datar karena cahaya memantul ke banyak arah. Bahan Mengkilap memiliki Kehalusan yang lebih tinggi, yang menciptakan pantulan cahaya yang lebih terfokus. Untuk kedua bahan tersebut juga terdapat preview dari bahan tersebut, sehingga Anda dapat melihat seperti apa bentuknya setelah menjadi model.



Gambar 5.20 Dua material dengan pengaturan shader yang berbeda

Shaders Revisited

Kembali Sekarang Anda telah diperkenalkan dengan tekstur, model, dan shader, sekarang saatnya untuk melihat bagaimana semuanya menyatu. Unity memiliki shader Standar yang sangat kuat, yang menjadi fokus buku ini. Tabel 5.1 menjelaskan sifat umum shader. Selain daftar pengaturan di Tabel 5.1, ada banyak pengaturan lain dari Standard shader; namun, buku ini berfokus pada penguasaan yang tercantum dalam tabel.

Tabel 5.1 Sifat umum shader

Properti	Deskripsi
Albedo	Properti Albedo mendefinisikan warna dasar objek. Dengan sistem rendering berbasis fisik (PBR) Unity yang kuat, warna ini berinteraksi dengan cahaya seperti halnya objek nyata. Misalnya, albedo kuning akan terlihat kuning dalam cahaya putih tetapi hijau di bawah cahaya biru. Di sinilah Anda akan menerapkan tekstur yang berisi informasi warna tentang model Anda.
Metallic	Pengaturan ini melakukan persis seperti yang terdengar: Ini mengubah tampilan material metalik. Pengaturan ini juga dapat mengambil tekstur sebagai "peta" tentang bagaimana bagian-bagian berbeda dari model yang metalik muncul. Untuk hasil yang realistis, setel properti ini ke 0 atau 1.

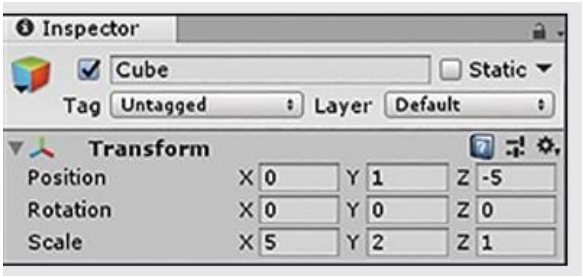
Smoothness	Kehalusan adalah elemen kunci dalam PBR karena mewakili berbagai ketidaksempurnaan mikro, detail, tanda, dan usia yang mengontrol seberapa halus (atau kasar) suatu permukaan. Hasilnya membuat model terlihat kurang lebih berkilau. Properti ini menggunakan peta tekstur yang sama dengan properti Metallic. Untuk hasil yang realistis, hindari nilai ekstrim seperti 0 dan 1.
Normal Map	Properti Normal Map berisi peta normal yang akan diterapkan ke model. Peta normal dapat digunakan untuk menerapkan relief, atau gundukan, pada model. Ini berguna saat menghitung pencahayaan untuk memberikan model lebih detail daripada yang seharusnya.
Tilling	Properti Tiling menentukan seberapa sering tekstur dapat diulang pada model. Itu bisa berulang di sumbu x dan y. (Ingat bahwa teksturnya datar, itulah sebabnya tidak ada sumbu ubin z.)
Offset	Properti Offset mendefinisikan di mana dalam sumbu x dan y tekstur untuk mulai diterapkan.

Ini mungkin tampak seperti banyak informasi untuk diambil, tetapi begitu Anda menjadi lebih akrab dengan beberapa dasar tekstur, shader, dan bahan, Anda akan menemukan bahwa nilai Kelancaran pemahaman Anda menjadi 1 (komedi klasik). Unity juga memiliki beberapa shader lain yang tidak dicakup oleh buku ini. Shader Standar sangat fleksibel dan dapat digunakan untuk sebagian besar kebutuhan dasar Anda.

Coba latih diri Anda dengan ini.

Menerapkan Tekstur, Shader, dan Material ke Model Ikuti langkah-langkah berikut untuk menggabungkan semua pengetahuan Anda tentang tekstur, shader, dan material untuk menciptakan dinding bata yang tampak layak:

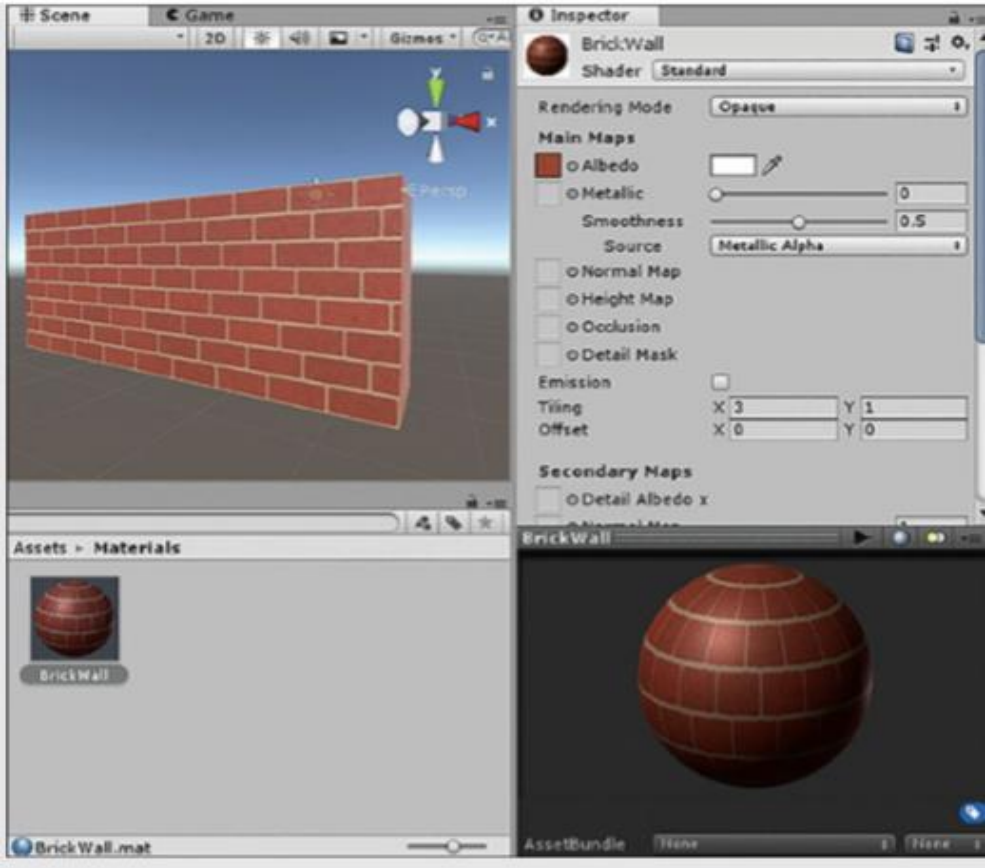
1. Mulai proyek atau scene baru. Perhatikan bahwa membuat new project menyebabkan editor menutup dan membuka kembali.
2. Buat folder Tekstur dan folder Bahan.
3. Cari file Brick_Texture.png di file buku dan drag ke folder Tekstur yang dibuat pada langkah 2.
4. Tambahkan kubus ke scene. Posisikan di (0, 1, -5). Berikan scale (5, 2, 1). Gambar 3.7 menunjukkan sifat-sifat kubus.



Gambar 5.21 Sifat-sifat kubus.

5. Buat material baru (dengan mengklik kanan folder Material dan pilih Create > Material) dan beri nama BrickWall. 6. Biarkan shader sebagai Standar, dan di bawah Peta Utama klik pemilih lingkaran (ikon lingkaran kecil) di sebelah kiri kata Albedo. Pilih

Brick_Texture dari jendela pop-up. 7. Klik dan drag bahan dinding bata dari tampilan Proyek ke kubus dalam tampilan Scene. 8. Perhatikan bahwa teksturnya terlalu melebar di dinding. Dengan bahan yang dipilih, ubah nilai ubin x menjadi 3. Pastikan Anda melakukannya di bagian Peta Utama, bukan bagian Peta Sekunder. Sekarang dinding terlihat jauh lebih baik. Anda sekarang memiliki dinding bata bertekstur di tempat Anda. Gambar 5.22 menunjukkan produk akhir.



Gambar 5.22 Produk akhir

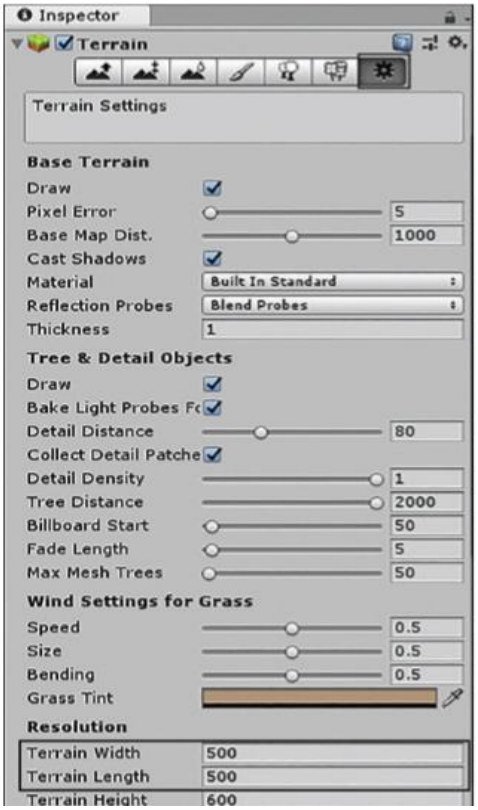
5.10 GENERASI MEDAN

Semua level game 3D ada dalam beberapa bentuk dunia. Dunia ini bisa sangat abstrak atau realistis. Seringkali, game dengan level "outdoor" yang luas dikatakan memiliki medan. Istilah medan mengacu pada setiap bagian tanah yang mensimulasikan lanskap eksternal dunia. Pegunungan tinggi, dataran jauh, dan rawa lembap adalah contoh medan permainan yang memungkinkan. Di Unity, medan adalah mesh datar yang dapat dipahat menjadi berbagai bentuk. Mungkin membantu untuk menganggap medan sebagai pasir di kotak pasir. Anda dapat menggali pasir atau mengangkat bagian-bagiannya. Satu-satunya hal yang tidak bisa dilakukan oleh medan dasar adalah tumpang tindih. Ini berarti Anda tidak dapat membuat hal-hal seperti gua atau overhang. Item tersebut harus dimodelkan secara terpisah. Juga, seperti halnya objek lain di Unity, medan memiliki posisi, rotasi, dan scale (walaupun biasanya tidak berubah selama bermain game).

Menambahkan Medan ke Proyek

Membuat medan datar dalam sebuah scene adalah tugas yang mudah dengan beberapa parameter dasar. Untuk menambahkan medan ke scene, cukup klik GameObject > 3D Object > Terrain. Aset bernama Medan Baru muncul di tampilan proyek, dan objek bernama Medan ditambahkan ke scene Anda. Jika Anda bernavigasi dalam tampilan Scene,

Anda mungkin juga memperhatikan bahwa bagian medan sangat besar. Faktanya, potongan itu jauh lebih besar daripada yang mungkin Anda butuhkan saat ini. Oleh karena itu, Anda perlu memodifikasi beberapa properti dari medan ini. Untuk membuat medan ini lebih mudah dikelola, Anda perlu mengubah resolusi medan. Dengan memodifikasi resolusi, Anda dapat mengubah panjang, lebar, dan tinggi maksimum bidang Anda. Alasan penggunaan istilah resolusi akan menjadi lebih jelas nanti dalam pelajaran ini, ketika Anda mempelajari tentang heighmap. Untuk mengubah resolusi potongan medan, ikuti langkah-langkah berikut: 1. Pilih medan Anda dalam tampilan Hierarki. Pada tampilan Inspector, cari dan klik tombol Terrain Settings (lihat Gambar 5.23). 2. Di bagian Resolution, perhatikan bahwa saat ini, Terrain Width dan Terrain Length diatur ke 500. Ubah keduanya menjadi 50, yang merupakan ukuran yang lebih mudah diatur.



Gambar 5.23 Pengaturan Resolusi.

Opsi lain dalam pengaturan Resolusi mengubah cara tekstur digambar dan kinerja medan Anda. Tinggalkan ini sendiri untuk saat ini. Setelah Anda mengubah lebar dan panjangnya, Anda akan melihat bahwa medannya jauh lebih kecil dan lebih mudah diatur. Sekarang saatnya untuk mulai memahat.

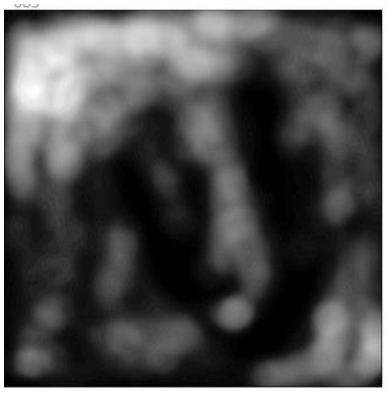
Ukuran Medan

Saat ini, Anda bekerja dengan medan yang panjang dan lebarnya 50 unit. Ini murni untuk pengelolaan saat Anda belajar menggunakan berbagai alat. Dalam gim nyata, medannya mungkin berukuran lebih besar agar sesuai dengan kebutuhan Anda. Perlu juga dicatat bahwa jika Anda sudah memiliki heighmap (dibahas di bagian berikutnya), Anda akan menginginkan rasio medan (rasio panjang dan lebar) agar sesuai dengan rasio heighmap.

Memahat Heighmap

Secara tradisional, 256 warna abu-abu tersedia dalam gambar 8-bit. Nuansa ini berkisar dari 0 (hitam) hingga 255 (putih). Mengetahui hal ini, Anda dapat menggunakan gambar hitam-putih, yang sering disebut gambar scale abu-abu, sebagai sesuatu yang disebut

heighmap. Heighmap adalah gambar scale abu-abu yang berisi informasi ketinggian yang mirip dengan peta topografi. Nuansa yang lebih gelap dapat dianggap sebagai titik rendah dan warna yang lebih terang sebagai titik tinggi. Gambar 5.24 menunjukkan contoh heighmap. Ini mungkin tidak terlihat banyak, tetapi gambar sederhana seperti ini dapat menghasilkan beberapa scene yang dinamis.



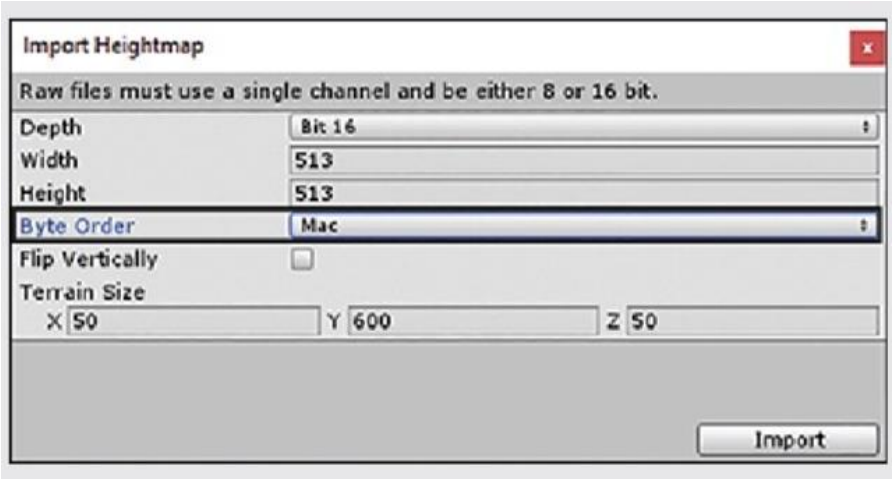
Gambar 5.24 Heighmap sederhana

Menerapkan heighmap ke medan datar Anda saat ini sangatlah sederhana. Anda cukup memulai dengan medan datar dan mengimpor heighmap ke atasnya, seperti yang ditunjukkan berikut ini. Coba Sendiri.

Mari Berlatih -Menerapkan Heighmap ke Medan

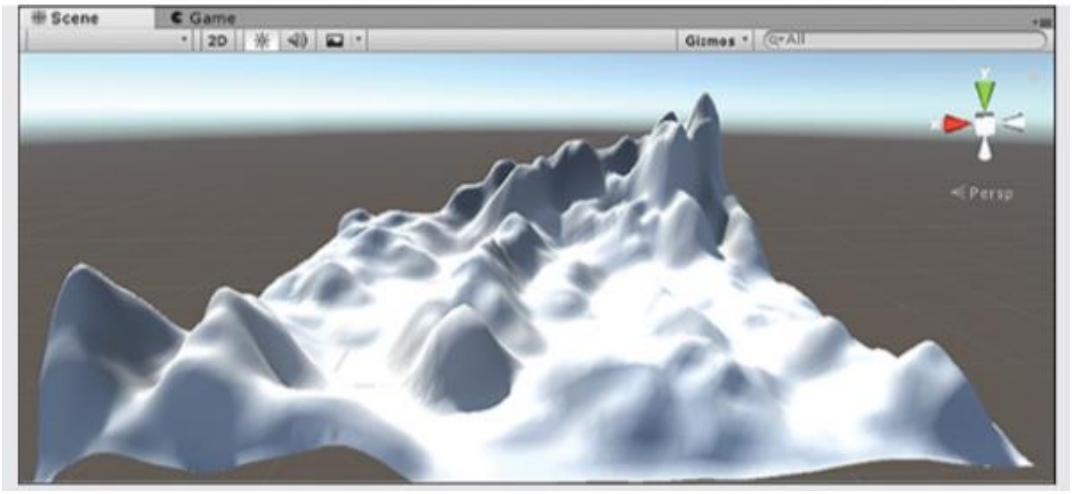
Ikuti langkah-langkah ini untuk mengimpor dan menerapkan heighmap:

1. Buat proyek atau scene Unity baru. Temukan file terrain.raw di file Hour 4 dan letakkan di tempat yang mudah Anda temukan.
2. Buat medan baru menggunakan langkah-langkah yang dijelaskan sebelumnya dalam jam ini. Pastikan untuk mengatur lebar dan tingginya menjadi 50.
3. Dengan medan Anda dipilih dalam tampilan Hierarki, klik tombol Pengaturan Medan di tampilan Inspector. (Lihat Gambar 5.25 jika diperlukan.) Di bagian Heighmap, klik Impor Mentah.
4. Dalam dialog Import Raw Heightmap, cari file terrain.raw dari langkah 1 dan klik Open.
5. Dalam dialog Import Heightmap, atur opsi seperti yang muncul pada Gambar 5.25 (Catatan: Properti Urutan Byte tidak terkait dengan sistem operasi yang dijalankan komputer Anda. Sebaliknya, properti itu terkait dengan sistem operasi tempat heighmap dibuat.)



Gambar 5.25 Impor dialog Heighmap.

6. Klik Impor. Saat ini, medanmu terlihat aneh. Masalahnya adalah ketika Anda mengatur panjang dan lebar medan Anda agar lebih mudah diatur, Anda meninggalkan ketinggian di 600. Ini jelas terlalu tinggi untuk kebutuhan Anda saat ini.
7. Ubah resolusi medan dengan kembali ke bagian Resolusi di bawah Pengaturan Medan di tampilan Inspector. Kali ini, ubah nilai ketinggian menjadi 60. Hasilnya akan menjadi sesuatu yang jauh lebih menyenangkan, seperti yang ditunjukkan pada Gambar 5.26.



Gambar 5.26 Penampakan medan yang Anda import dari heighmap

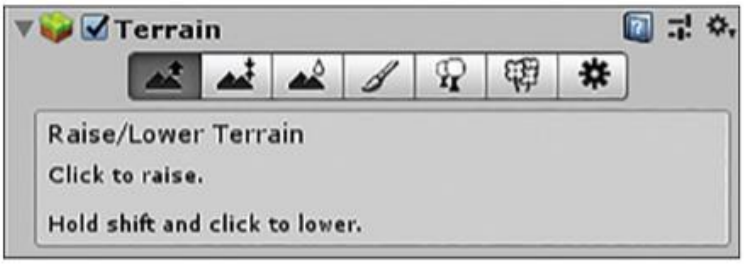
Menghitung Tinggi Sejauh ini, heighmap mungkin tampak acak, tetapi sebenarnya cukup mudah untuk diketahui. Semuanya didasarkan pada persentase 255 dan ketinggian maksimum medan. Ketinggian maksimum medan default ke 600 tetapi mudah diubah. Jika Anda menerapkan rumus (Abu-abu Naungan)/255 × (Tinggi Maks), Anda dapat dengan mudah menghitung titik mana pun di medan. Misalnya, hitam memiliki nilai 0, jadi setiap titik yang berwarna hitam adalah tinggi 0 unit (0/255 × 600 = 0). Putih memiliki nilai 255; oleh karena itu, menghasilkan titik-titik setinggi 600 unit (255/255 × 600 = 600). Jika Anda memiliki abu-abu sedang dengan nilai 125, titik mana pun dari warna itu menghasilkan medan yang tingginya sekitar 294 unit (125/255 × 600 = 294).

Format Heighmap

Di Unity, heighmap harus berupa gambar scale abu-abu dalam format .raw. Ada banyak cara untuk menghasilkan jenis gambar ini; Anda dapat menggunakan editor gambar sederhana atau bahkan Unity itu sendiri. Jika Anda membuat heighmap menggunakan editor gambar, coba buat peta dengan rasio panjang dan lebar yang sama dengan medan Anda. Jika tidak, beberapa distorsi akan terlihat. Jika Anda memahat beberapa medan menggunakan alat pahat Unity dan ingin membuat heighmap untuk itu, Anda dapat melakukannya dengan membuka bagian Heighmap di bawah Pengaturan Medan di tampilan Inspector dan mengklik Ekspor Mentah. Umumnya untuk medan besar, atau di mana kinerja penting, Anda harus mengekspor heighmap Anda dan mengonversi medan Anda menjadi mesh di program lain. Dengan mesh 3D, Anda juga dapat menambahkan gua, overhang, dan sebagainya sebelum mengimpor mesh untuk digunakan di Unity. Namun, perhatikan bahwa jika Anda mengimpor mesh untuk digunakan sebagai medan, Anda tidak akan dapat menggunakan alat tekstur dan lukisan medan Unity dengannya. (Namun, Anda dapat menemukan aset pihak ketiga di Asset Store yang dapat menyediakan fungsi ini.)

Unity Terrain Sculpting Tools

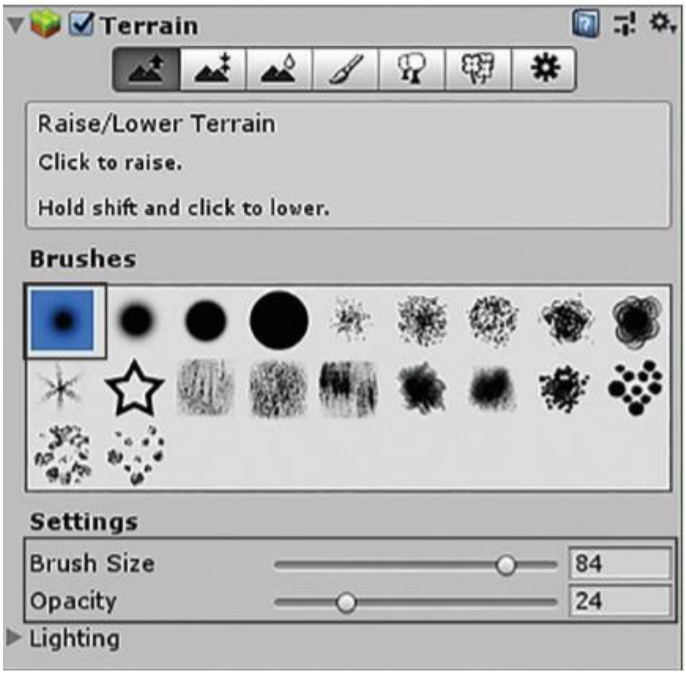
Unity memberi Anda banyak alat untuk memahat medan Anda dengan tangan. Anda dapat melihat alat ini dalam tampilan Inspector di bawah komponen Medan. Semua alat ini pada dasarnya bekerja dengan cara yang sama: Anda menggunakan kuas dengan ukuran dan opasitas tertentu untuk "melukis" medan. Akibatnya, apa yang Anda lakukan di belakang layar adalah melukis heighmap yang ditranslate ke dalam perubahan untuk medan 3D. Efek lukisan bersifat kumulatif, yang berarti semakin banyak Anda mengecat suatu area, semakin kuat efeknya pada area tersebut. Gambar 4.5 menunjukkan alat-alat ini, yang dapat Anda gunakan untuk menghasilkan hampir semua lanskap yang dapat Anda bayangkan.



Gambar 5.27 Sculpting tool medan.

Alat pertama yang akan Anda pelajari adalah alat Naikkan/Turunkan. Alat ini, seperti kedengarannya, memungkinkan Anda menaikkan atau menurunkan medan di mana pun Anda melukis. Untuk memahat dengan alat ini, ikuti langkah-langkah berikut:

1. Pilih kuas. (Kuas menentukan ukuran dan bentuk efek pahatan.)
2. Pilih ukuran kuas dan opasitas. (Opacity menentukan seberapa kuat efek pahatannya.)
3. Klik dan drag ke atas medan dalam tampilan Scene untuk menaikkan medan. Menahan Shift saat Anda mengklik dan menyeret malah menurunkan medan. Gambar 4.6 mengilustrasikan beberapa opsi awal yang baik untuk memahat dengan mempertimbangkan ukuran medan 50 × 50 dengan ketinggian 60.



Gambar 5.28 Sifat awal yang baik untuk memahat.

Alat selanjutnya adalah alat Paint Height. Alat ini bekerja dengan cara yang hampir sama dengan alat Naikkan/Turunkan kecuali alat ini mengecat medan Anda hingga ketinggian tertentu. Jika ketinggian yang ditentukan lebih tinggi dari medan saat ini, pengecatan akan menaikkan medan. Namun, jika ketinggian yang ditentukan lebih rendah dari medan saat ini, medan diturunkan. Ini berguna untuk membuat mesa dan struktur datar lainnya di lanskap Anda. Silakan dan mencobanya!

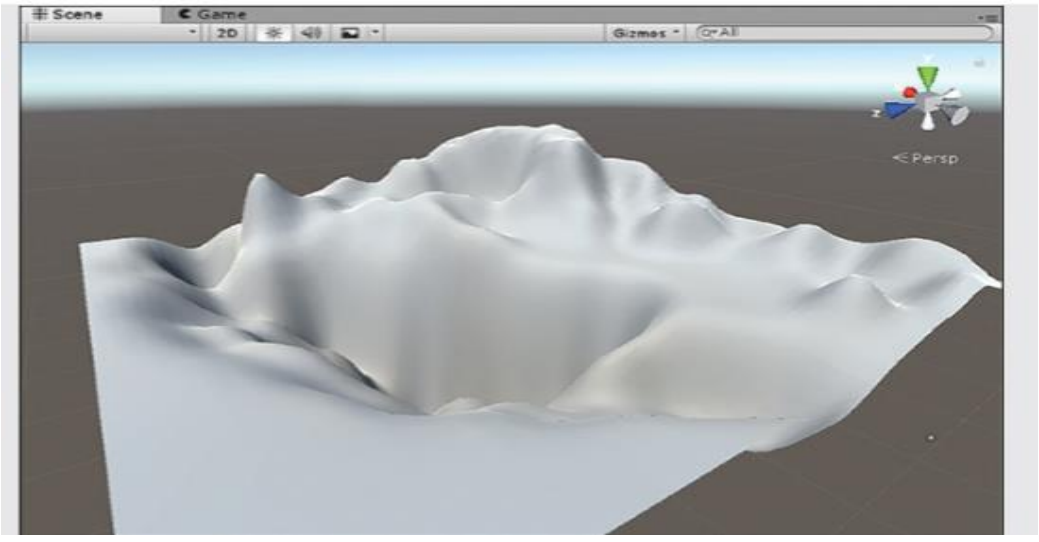
Meratakan Medan

Jika, kapan saja, Anda ingin menyetel ulang medan Anda kembali menjadi datar, Anda dapat melakukannya dengan membuka alat Paint Height dan mengklik Flatten. Salah satu keuntungan melakukan ini adalah Anda dapat meratakan medan ke ketinggian selain 0 defaultnya. Jika tinggi maksimum Anda adalah 60 dan Anda meratakan heighmap menjadi 30, Anda memiliki kemampuan untuk menaikkan medan sebanyak 30 unit, tetapi Anda bisa juga menurunkannya sebanyak 30 unit. Ini membuatnya mudah untuk memahat lembah ke medan datar Anda. Alat terakhir yang akan Anda gunakan adalah alat Smooth Height. Alat ini tidak mengubah medan dengan cara yang sangat mencolok. Sebaliknya, itu menghilangkan banyak garis bergerigi yang muncul saat memahat medan. Pikirkan alat ini sebagai penggosok. Anda akan menggunakannya hanya untuk membuat perubahan kecil setelah pemahatan besar Anda selesai.

Memahat Medan

Sekarang setelah Anda mempelajari tentang alat pahat, saatnya untuk berlatih menggunakannya. Dalam latihan ini, Anda mencoba untuk memahat bagian tertentu dari medan:

1. Buat proyek atau scene baru dan tambahkan medan. Atur resolusi medan menjadi 50 × 50 dan berikan ketinggian 60.
2. Ratakan medan hingga ketinggian 20 dengan mengklik alat Paint Height, ubah ketinggian menjadi 20, dan klik Ratakan. (Catatan: Jika medan tampak menghilang, itu hanya naik 20 unit.)
3. Menggunakan alat pahat, coba buat lanskap yang mirip dengan yang ditunjukkan pada Gambar 4.7. (Catatan: Pencahayaan pada gambar telah diubah agar lebih mudah dilihat.)
4. Lanjutkan bereksperimen dengan alat dan coba tambahkan fitur unik ke medan Anda.



Gambar 5.29 Sebuah sampel medan.

Latihan, Latihan, dan Latihan

Mengembangkan level yang kuat dan menarik adalah bentuk seni itu sendiri. Banyak pemikiran harus diberikan pada penempatan bukit, lembah, gunung, dan danau dalam permainan. Tidak hanya elemen yang harus memuaskan secara visual, mereka juga perlu ditempatkan sedemikian rupa untuk membuat level dapat dimainkan. Keterampilan membangun level tidak berkembang dalam semalam. Pastikan untuk melatih dan menyempurnakan keterampilan membangun level Anda untuk membuat level yang menarik dan berkesan

Tekstur Medan

Anda sekarang tahu cara membuat dimensi fisik dunia 3D. Meskipun mungkin ada banyak fitur pada lanskap Anda, lanskap Anda tetap hambar, berkilau (karena bahan default), dan sulit dinavigasi. Saatnya untuk menambahkan beberapa karakter ke level Anda. Di bagian ini, Anda akan mempelajari cara membuat tekstur medan Anda untuk memberikan tampilan yang menarik. Seperti bidang pahatan, bidang tekstur sangat mirip dengan lukisan. Anda memilih kuas dan tekstur dan melukisnya ke dunia Anda.

Mengimpor Aset Medan

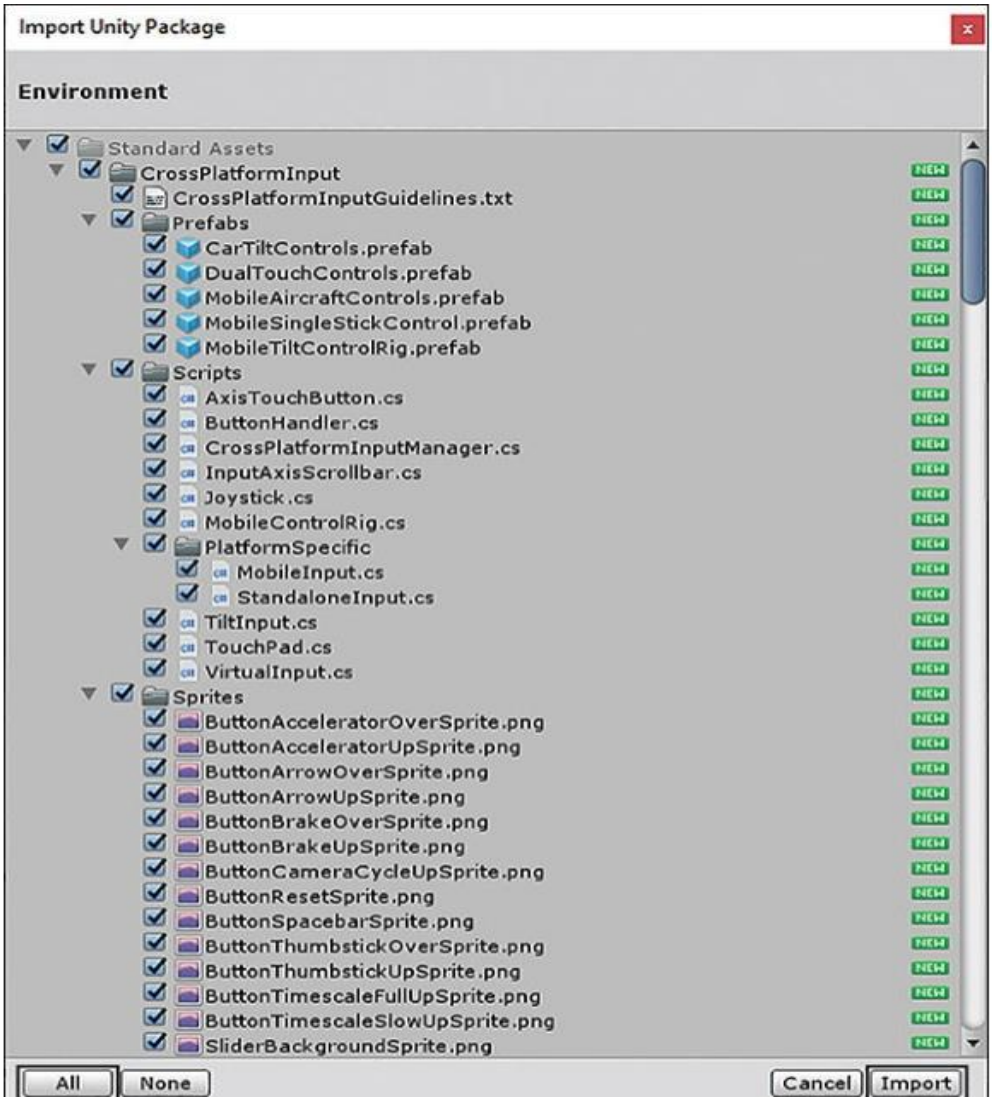
Sebelum Anda dapat mulai melukis dunia dengan tekstur, Anda memerlukan beberapa tekstur untuk dikerjakan. Unity memiliki beberapa aset medan yang tersedia untuk Anda, tetapi Anda perlu mengimpornya untuk menggunakannya. Untuk memuat aset ini, pilih Aset > Paket Impor > Lingkungan. Dialog Import Unity Package muncul (lihat Gambar 5.30). Anda menggunakan dialog ini untuk menentukan aset mana yang ingin Anda impor. Membatalkan pilihan item yang tidak dibutuhkan adalah ide yang bagus jika Anda ingin memperkecil ukuran proyek Anda. Untuk saat ini, biarkan saja semua opsi dicentang dan klik Impor. Anda sekarang harus memiliki folder baru di bawah Aset dalam tampilan Proyek yang disebut Aset Standar. Folder ini berisi semua aset medan yang akan Anda gunakan di sisa jam ini.

Paket Hilang

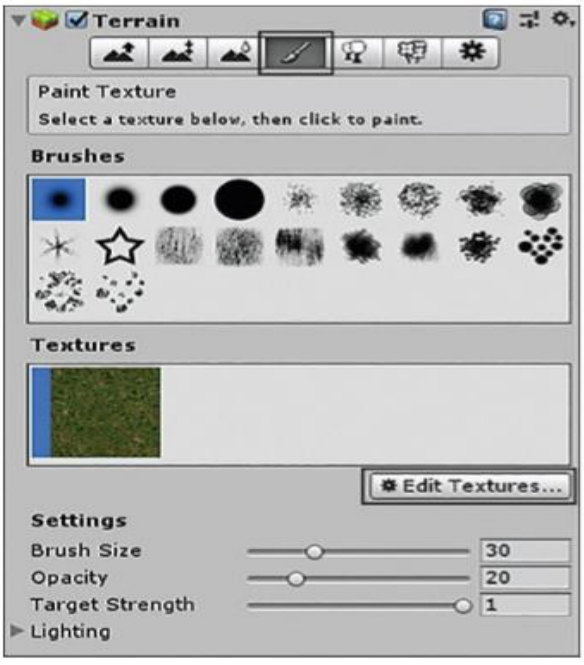
Jika Anda kehilangan paket aset Lingkungan saat membuka Aset > Paket Impor, itu berarti Anda tidak memilih opsi Aset Standar saat menginstal Unity. Jika Anda menginginkan aset ini (dan banyak lagi yang akan digunakan di seluruh buku ini), Anda dapat menjalankan penginstal lagi dan memilih untuk menginstal aset standar.

Tekstur Medan

Untuk mulai melukis medan, Anda perlu memuat tekstur. Gambar 5.31 mengilustrasikan alat Tekstur Cat di Inspector, yang Anda akses setelah memilih medan di Hierarki Anda. Perhatikan tiga properti numerik: ukuran kuas, opacity, dan kekuatan target. Anda harus terbiasa dengan dua properti pertama, tetapi yang terakhir baru. Kekuatan target adalah opacity maksimum yang dapat dicapai melalui pengecatan konstan. Nilainya adalah persentase, dengan 1 menjadi 100%. Gunakan ini sebagai kontrol untuk menghindari pengecatan tekstur Anda terlalu kuat.



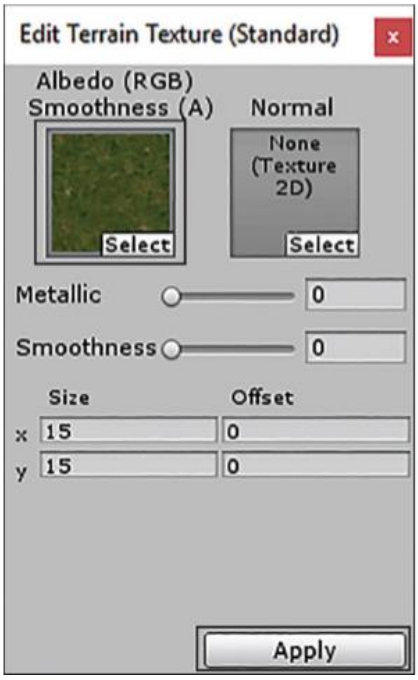
Gambar 5.30 Dialog Impor Paket Unity.



Gambar 5.31 Alat dan properti Tekstur Cat.

Untuk memuat tekstur, ikuti langkah berikut:

1. Pilih Edit Textures > Add Texture di Inspector (bukan dari menu Unity).
2. Dalam dialog Edit Terrain Texture, klik Select di kotak tekstur dan pilih tekstur GrassHillAlbedo.
3. Klik Tambah. Tidak perlu menambahkan peta normal, tetapi Anda bisa melakukannya jika Anda memiliki tekstur dengan sedikit ketidaksesuaian.

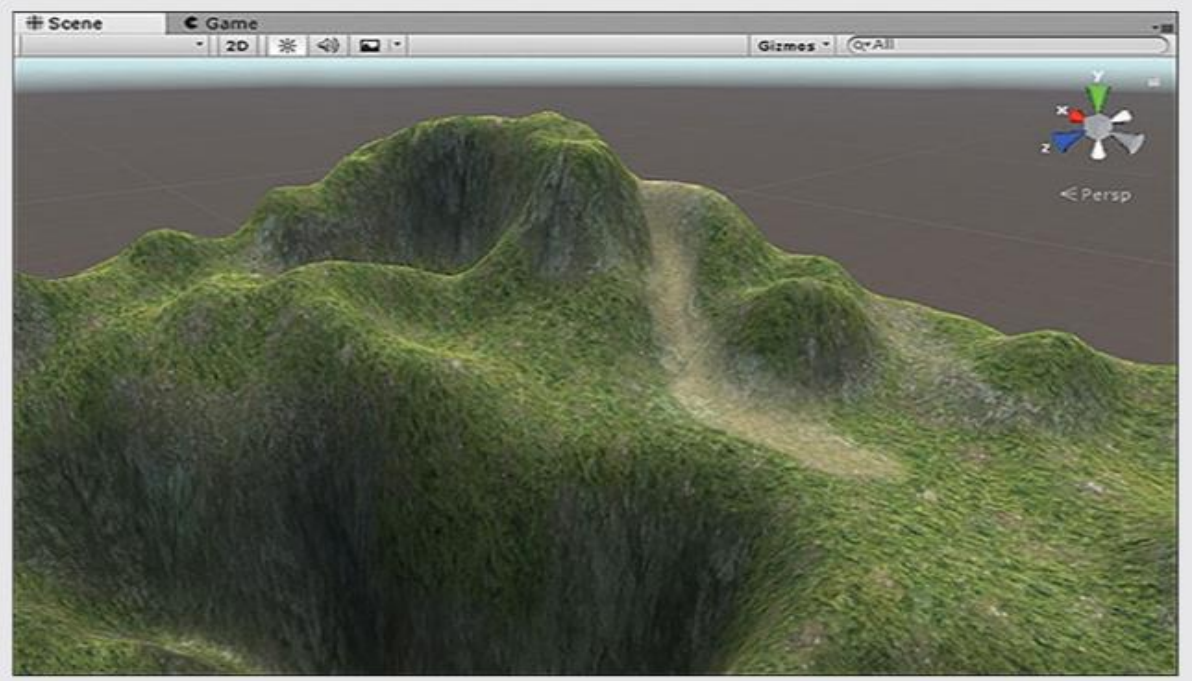


Gambar 5.32 Dialog Edit Tekstur Medan.

Pada titik ini, seluruh medan Anda harus tertutup rumput yang tidak rata. Ini terlihat lebih baik daripada medan putih yang Anda miliki sebelumnya, tetapi masih jauh dari realistis. Selanjutnya, Anda benar-benar akan mulai melukis dan membuat medan Anda terlihat lebih baik.

Melukis Tekstur di Medan Ikuti langkah-langkah ini untuk menerapkan tekstur baru ke medan Anda untuk memberikan efek dua nada yang lebih realistis:

1. Menggunakan langkah-langkah yang tercantum di awal jam ini, tambahkan tekstur baru. Kali ini, muat tekstur GrassRockyAlbedo. Setelah Anda memuatnya, pastikan untuk memilihnya dengan mengkliknya. (Catatan: Bilah biru muncul di sebelahnya jika dipilih.)
2. Atur ukuran kuas ke 30, opacity ke 20, dan kekuatan target ke 0,6.
3. Hemat cat (dengan mengklik dan menyeret) pada bagian curam dan celah-celah medan Anda. Ini memberi kesan bahwa rumput tidak tumbuh di sisi lereng curam dan di antara perbukitan.
4. Lanjutkan bereksperimen dengan lukisan tekstur. Coba muat tekstur CliffAlbedoSpecular dan terapkan ke bagian yang lebih curam atau tekstur SandAlbedo dan buat jalur.



Gambar 5.33 Contoh tebing bertekstur dua warna dengan jalur pasir.

Anda dapat memuat tekstur sebanyak yang Anda inginkan dengan cara ini dan mencapai beberapa efek realistis. Pastikan untuk berlatih tekstur untuk menentukan pola yang paling bagus.

Membuat Tekstur Medan

Dunia game sering kali unik dan memerlukan tekstur khusus agar sesuai dengan konteks game tempat mereka dibuat. Anda dapat mengikuti beberapa panduan umum saat membuat tekstur sendiri untuk medan. Yang pertama adalah selalu mencoba membuat polanya bisa diulang. Ini berarti tekstur dapat ditata dengan mulus. Semakin besar teksturnya, semakin tidak jelas pola berulangnya. Pedoman kedua adalah membuat tekstur persegi. Terakhir, coba buat dimensi tekstur menjadi pangkat 2 (32, 64, 128, 512, dan seterusnya). Dua pedoman terakhir mempengaruhi kompresi tekstur dan efisiensi tekstur. Dengan sedikit latihan, Anda akan membuat tekstur medan yang cemerlang dalam waktu singkat.

Kehalusan Adalah Kebijakan Terbaik

Saat membuat tekstur, ingatlah untuk menjaga agar efek Anda tetap halus. Di alam, satu elemen cenderung memudar ke elemen lain tanpa banyak transisi yang keras. Upaya tekstur Anda harus mencerminkan hal itu. Jika Anda dapat memperkecil jarak dari bagian medan dan memberi tahu titik yang tepat di mana satu tekstur dimulai, efek Anda tidak cukup halus. Seringkali lebih baik bekerja di banyak aplikasi tekstur yang kecil dan halus daripada dengan satu aplikasi yang luas.

Kesalahan TerrainData…Apa?

Bergantung pada pengaturan proyek Anda dan versi Unity yang Anda gunakan, Anda mungkin mendapatkan kesalahan saat menjalankan scene Anda. Satu kesalahan mengatakan sesuatu seperti "TerrainData kehilangan tekstur percikan ... pastikan itu ditandai untuk baca/tulis di importir." Kesalahan ini memberi tahu Anda bahwa Anda mengalami masalah akses tekstur runtime. Untungnya, perbaikannya sangat sederhana. Yang perlu Anda lakukan hanyalah mengeklik tekstur yang menyinggung di tampilan Proyek, yang menyebabkan

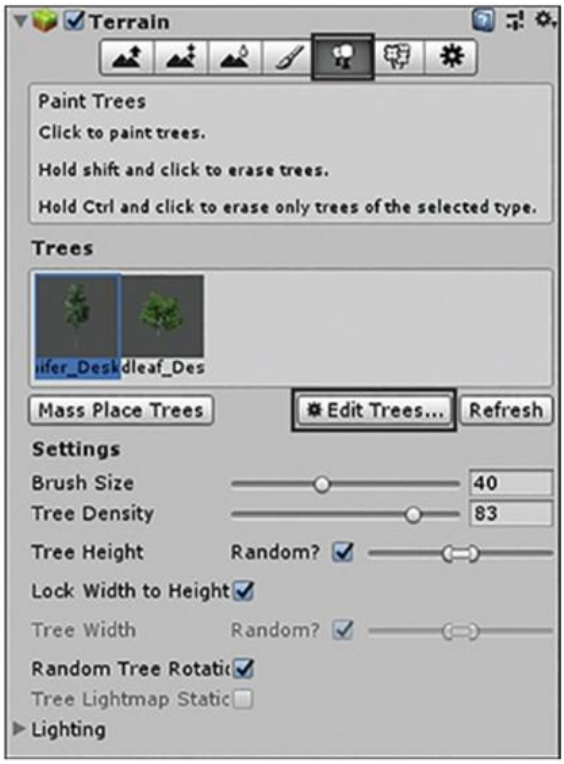
Inspector menampilkan pengaturan Impornya. Kemudian, dalam tampilan Inspector, perluas Lanjutan dan centang kotak di samping Baca/Tulis Diaktifkan. Masalah terpecahkan!

Menghasilkan Pohon dan Rumput

Dunia dengan tekstur datar saja akan membosankan. Hampir setiap lanskap alam memiliki beberapa bentuk kehidupan tanaman. Di bagian ini, Anda akan mempelajari cara menambahkan dan menyesuaikan pohon dan rumput untuk memberikan tampilan dan nuansa organik pada medan Anda.

Lukisan Pohon

Menambahkan pohon ke medan Anda bekerja sangat mirip dengan memahat dan memberi tekstur; seluruh proses sangat mirip dengan lukisan. Premis dasarnya adalah memuat model pohon, mengatur properti untuk pohon, dan kemudian mengecat area di mana Anda ingin pohon muncul. Berdasarkan opsi yang Anda pilih, Unity akan menyebarkan pepohonan dan memvariasikannya untuk memberikan tampilan yang lebih alami dan organik. Anda menggunakan alat Paint Trees untuk menyebarkan pohon di atas medan. Setelah medan dipilih dalam scene, pilih alat Paint Trees di tampilan Inspector untuk komponen Terrain (Script). Gambar 5.34 menunjukkan alat Paint Trees dan properti standarnya.



Gambar 5.34 Alat Pohon Cat.

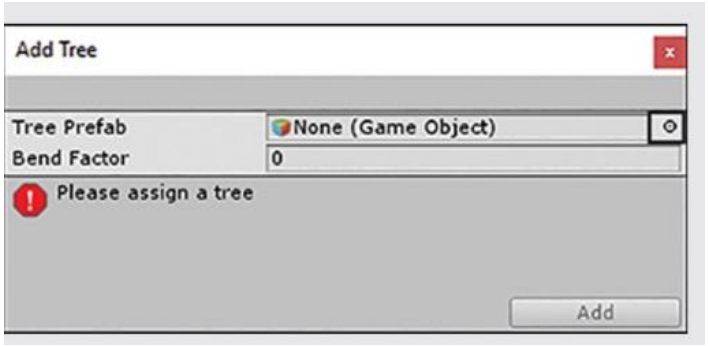
Tabel 5.1 Properti Paint Trees Tool

Properti	**Deskripsi**
Brush Size	Ukuran area tempat pohon ditambahkan saat melukis.
Tree Density	Betapa rapatnya pohon-pohon itu dapat dipadatkan.
Tee Height/Width, Rotation, and so on	Bagaimana semua pohon berbeda satu sama lain. Menggunakan properti ini memungkinkan Anda untuk memberikan kesan banyak pohon yang berbeda, bukan pohon yang sama berulang.

Menempatkan Pohon di Medan

Mari kita telusuri langkah-langkah yang terlibat dalam menempatkan pohon ke medan dengan menggunakan alat Paint Trees. Latihan ini mengasumsikan bahwa Anda telah membuat scene baru dan telah menambahkan medan. Medan harus diatur ke panjang dan lebar 100. Ini akan terlihat lebih baik jika medan sudah memiliki beberapa pahatan dan tekstur. Ikuti langkah-langkah berikut:

1. Klik Edit Trees > Add Tree untuk membuka dialog Add Tree (lihat Gambar 5.35).
2. Mengklik ikon lingkaran di sebelah kanan kotak teks Tree Prefab pada dialog Add Tree akan memunculkan dialog Tree Selector.

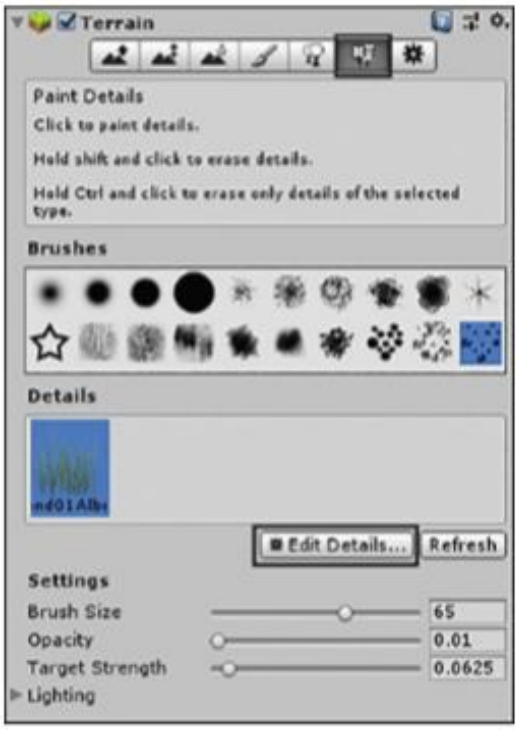


Gambar 5.35 Dialog Tambahkan Pohon.

3. Pilih Conifer_Desktop dan klik Tambah.
4. Atur ukuran kuas Anda menjadi 2 dan kerapatan pohon Anda menjadi 10. Biarkan tinggi/lebar pohon diatur ke Acak tetapi kurangi ukuran keseluruhannya.
5. Lukis pohon di medan dengan mengklik dan menyeret area di mana Anda ingin pohon. Tahan tombol Shift sambil klik-drag untuk menghapus pohon. Jika Anda tidak bisa melukis, buka Terrain Settings > Tree & Detail Objects dan pastikan bahwa kotak centang Draw dipilih.
6. Lanjutkan bereksperimen dengan berbagai ukuran kuas, kerapatan, dan pohon.

Lukisan Rumput

Sekarang setelah Anda belajar cara melukis pohon, Anda akan belajar cara menerapkan rumput dan kehidupan tanaman kecil lainnya ke dunia Anda. Rumput dan tanaman kecil lainnya disebut detail di Unity. Oleh karena itu, alat yang digunakan untuk mengecat rumput adalah alat Paint Details. Tidak seperti pohon, yang merupakan model 3D, detailnya adalah billboard (lihat catatan "Billboard"). Seperti yang telah Anda lakukan berulang kali dalam satu jam ini, Anda menerapkan detail ke medan dengan menggunakan kuas dan gerakan melukis.



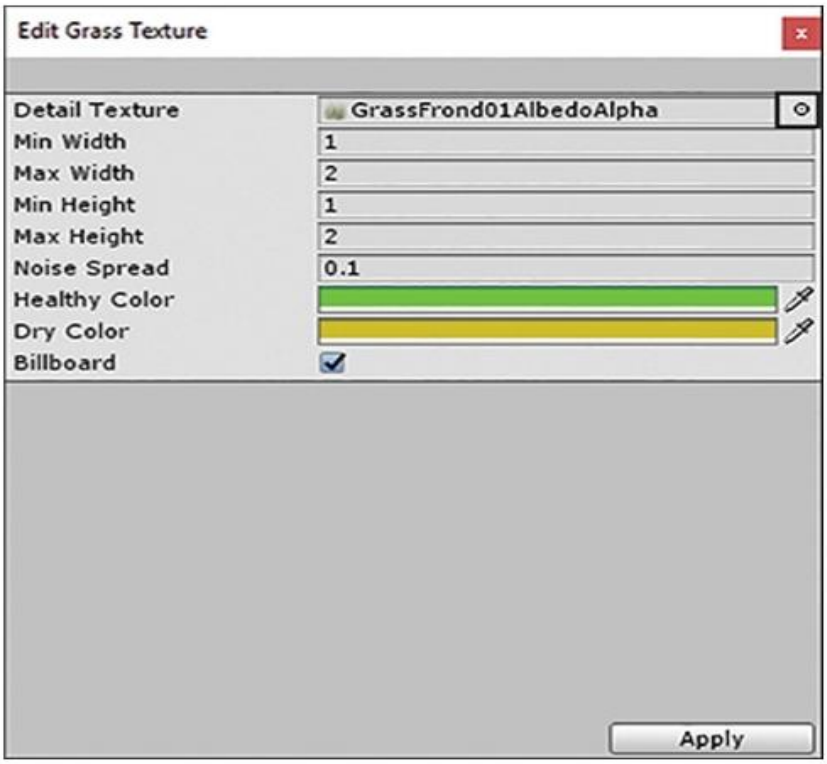
Gambar 5.36 Alat rumput

Billboard

Billboard adalah jenis komponen visual khusus dalam dunia 3D yang memberikan efek model 3D tanpa benar-benar menjadi model 3D. Model ada di ketiga dimensi. Oleh karena itu, ketika bergerak di sekitar satu, Anda dapat melihat sisi yang berbeda. Billboard, bagaimanapun, adalah gambar datar yang selalu menghadap kamera. Saat Anda mencoba mengitari billboard, billboard itu menghadap ke posisi baru Anda. Penggunaan umum untuk billboard adalah detail rumput, partikel, dan efek pada layar.

Menerapkan rumput ke medan Anda adalah proses yang cukup mudah. Anda harus terlebih dahulu menambahkan tekstur rumput:

1. Klik Edit Details pada tampilan Inspector dan pilih Add Grass Texture.
2. Dalam dialog Edit Grass Texture, klik ikon lingkaran di sebelah kotak teks Texture (lihat Gambar 5.37). Pilih tekstur GrassFrond01AlbedoAlpha. Anda dapat mencari "rumput" untuk membantu menemukannya.



Gambar 5.37 Dialog Edit Tekstur Rumput.

3. Atur properti tekstur Anda ke nilai apa pun yang Anda inginkan. Berikan perhatian khusus pada properti warna karena mereka membentuk rentang warna alami untuk rumput Anda.
4. Setelah selesai mengubah pengaturan, klik Terapkan. Setelah rumput Anda dimuat, Anda hanya perlu memilih kuas dan properti kuas, dan Anda siap untuk mulai mengecat rumput.

Rumput Realistis

Anda mungkin memperhatikan bahwa ketika Anda mulai mengecat rumput, itu tidak terlihat realistis. Anda perlu fokus pada beberapa hal saat menambahkan rumput ke medan Anda. Yang pertama adalah memperhatikan warna yang Anda atur untuk tekstur rumput. Cobalah untuk membuatnya lebih gelap dan warna bumi. Hal berikutnya yang perlu Anda lakukan adalah memilih bentuk kuas nongeometris untuk membantu memecah tepi yang keras. Terakhir, pertahankan opacity dan properti kekuatan target sangat rendah. Pengaturan yang baik untuk memulai adalah 0,01 untuk opacity dan 0,0625 untuk kekuatan target. Jika Anda membutuhkan lebih banyak rumput, Anda bisa terus mengecat di area yang sama. Anda juga dapat kembali ke Edit Details dan mengubah properti tekstur rumput.

5.11 VEGETASI DAN KINERJA

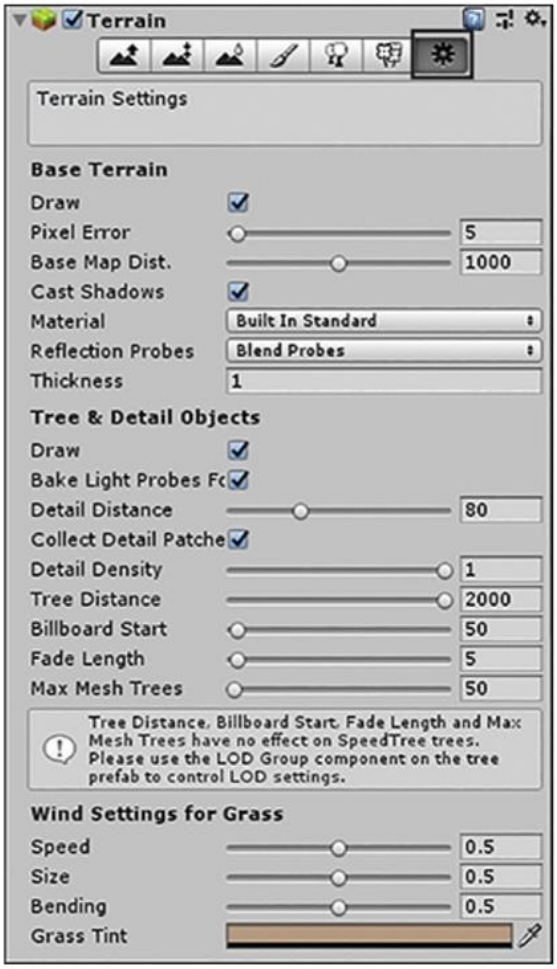
Semakin banyak pohon dan rumput yang Anda miliki dalam sebuah scene, semakin banyak pemrosesan yang diperlukan untuk membuatnya. Jika Anda khawatir tentang kinerja, jaga agar jumlah vegetasi tetap rendah. Beberapa properti yang akan Anda lihat nanti di jam ini dapat membantu Anda mengelolanya, tetapi sebagai aturan yang mudah, coba tambahkan pohon dan rumput hanya ke area yang benar-benar dibutuhkan.

Rumput yang Menghilang

Seperti halnya pohon, rumput dipengaruhi oleh jaraknya dari penonton. Sementara pepohonan kembali ke kualitas yang lebih rendah saat penonton berada jauh, rumput tidak direnderer. Hasilnya adalah lingkaran di sekitar penampil yang di luarnya tidak ada rumput yang terlihat. Sekali lagi, Anda dapat mengubah jarak ini dengan menggunakan properti yang dibahas nanti di jam ini.

Pengaturan Medan

Alat medan terakhir dalam tampilan Inspector adalah alat Pengaturan Medan. Pengaturan alat ini mengontrol bagaimana medan, tekstur, pepohonan, dan detail terlihat dan berfungsi secara keseluruhan.



Gambar 5.38 Alat Pengaturan Medan.

Kelompok pengaturan pertama adalah untuk keseluruhan medan. Tabel 5.2 menjelaskan beberapa pengaturan ini.

Tabel 5.2 Pengaturan Medan Dasar

Setting	Deskripsi
Draw	Menentukan apakah medan harus digambar.
Pixel Error	Menentukan jumlah kesalahan yang diizinkan saat menampilkan geometri medan. Semakin tinggi nilainya, semakin rendah detail medannya.

Base Map Dist.	Menentukan jarak maksimum dari mana tekstur resolusi tinggi ditampilkan. Saat pemirsa lebih jauh dari jarak yang ditetapkan, tekstur menurun ke resolusi yang lebih rendah.
Cast Shadow	Menentukan apakah geometri medan menghasilkan bayangan.
Material	Slot ini untuk menetapkan material khusus yang mampu merender medan. Materi harus mengandung shader yang dapat membuat medan.
Reflection Probes	Menentukan bagaimana probe refleksi digunakan di medan. Ini hanya efektif bila menggunakan bahan standar bawaan atau bahan khusus yang mendukung rendering dengan refleksi. Ini adalah pengaturan lanjutan yang tidak tercakup dalam teks ini.
Thickness	Menentukan seberapa besar volume tumbukan medan harus meluas sepanjang sumbu y negatif. Benda dianggap bercollision dengan medan dari permukaan hingga kedalaman yang sama dengan ketebalannya. Ini membantu mencegah objek bergerak berkecepatan tinggi menembus medan tanpa menggunakan deteksi collision berkelanjutan yang mahal.

Selain itu, beberapa pengaturan secara langsung memengaruhi cara pohon dan detail (seperti rumput) berperilaku di medan.

Tabel 5.3 Tree dan Detail Object Setting

Setting	**Deskripsi**
Draw	Menentukan apakah pohon dan detail ditampilkan dalam scene.
Bake Light Probes For	Membuat pencahayaan real-time lebih realistis dan efisien. Ini adalah pengaturan kinerja tingkat lanjut.
Detail Distance	Menentukan jarak dari kamera di mana detail tidak lagi digambar ke layar.
Collect Details Patches	Pramuat detail medan untuk mencegah cegukan saat bergerak di sekitar medan, dengan mengorbankan penggunaan memori.
Detail Desity	Menentukan jumlah detail/objek rumput dalam satuan luas tertentu. Nilai dapat diatur lebih rendah untuk mengurangi overhead rendering.
Tee Distance	Menentukan jarak dari kamera di mana pohon tidak lagi digambar ke layar.
Billboard Start	Menentukan jarak dari kamera tempat model pohon 3D mulai bertransisi ke papan iklan berkualitas lebih rendah.
Fade Length	Menentukan jarak transisi pohon antara billboard ke model 3D berkualitas lebih tinggi. Semakin tinggi pengaturannya, semakin halus transisinya.
Max Mesh Trees	Menentukan jumlah total pohon yang dapat digambar secara bersamaan sebagai mesh 3D dan bukan billboard.

Pengaturan selanjutnya adalah untuk angin. Karena Anda belum memiliki kesempatan untuk benar-benar berlarian di dalam dunia Anda (walaupun Anda akan melakukannya nanti di jam ini), Anda mungkin bertanya-tanya bagaimana cara kerja angin. Pada dasarnya, Unity

mensimulasikan angin sepoi-sepoi di atas medan Anda, dan angin sepoi-sepoi ini menyebabkan rumput menekuk dan bergoyang dan menghidupkan dunia.

Tabel 5.4 Pengaturan Angin

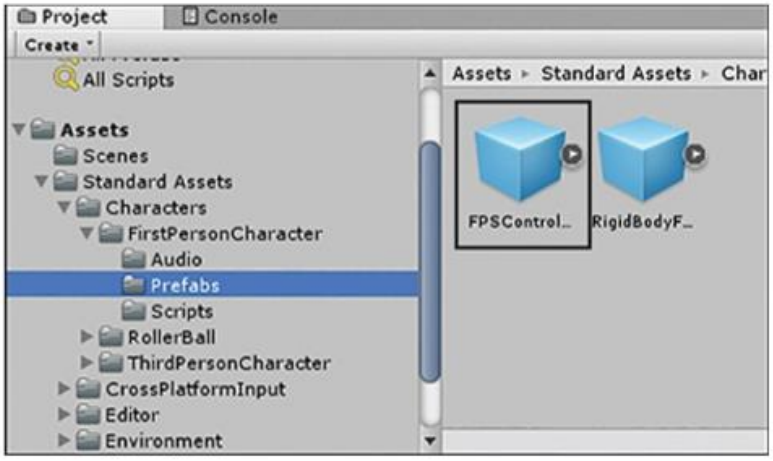
Setting	Deskripsi
Speed	Menentukan kecepatan, dan karena itu kekuatan, dari efek angin.
Size	Menentukan ukuran area rumput yang terkena angin pada satu waktu.
Bending	Menentukan jumlah goyangan rumput karena angin.
Grass Tint	Mengontrol keseluruhan warna semua rumput dalam satu level. (Ini sebenarnya bukan pengaturan angin, tetapi ini terkait.

5.12 CONTROLLER KARAKTER

Pada titik ini, Anda telah menyelesaikan medan Anda. Anda telah memahatnya, membuatnya bertekstur, dan menutupinya dengan pepohonan dan rumput. Sekarang saatnya untuk masuk ke level Anda dan lihat bagaimana rasanya memainkannya. Unity menyediakan dua controller karakter dasar yang memungkinkan Anda untuk dengan mudah masuk ke scene Anda tanpa banyak pekerjaan di pihak Anda. Pada dasarnya, Anda menjatuhkan controller ke dalam scene dan kemudian bergerak dengan skema kontrol yang umum untuk sebagian besar game orang pertama.

Menambahkan Controller Karakter

Untuk menambahkan controller karakter ke scene Anda, pertama-tama Anda harus mengimpor aset dengan memilih Aset > Impor Paket > Karakter. Dalam dialog Impor Paket, biarkan semuanya dicentang dan klik Impor. Folder baru bernama Karakter ditambahkan ke tampilan Proyek Anda, di bawah folder Aset Standar. Karena Anda tidak memiliki model 3D untuk digunakan sebagai pemutar, dalam hal ini Anda akan menggunakan controller orang pertama. Temukan aset FPSController di folder Character Controllers (lihat Gambar 5.39) dan drag ke medan Anda dalam tampilan Scene.



Gambar 5.39 Controller karakter FPSController

Sekarang controller karakter telah ditambahkan ke scene Anda, Anda dapat bergerak di medan yang Anda buat. Saat Anda memainkan scene Anda, perhatikan bahwa Anda sekarang dapat melihat dari mana controller ditempatkan. Anda dapat menggunakan tombol

WASD untuk bergerak, mouse untuk melihat-lihat, dan spasi untuk melompat. Bermain-main dengan kontrol jika mereka merasa agak tidak biasa bagi Anda, dan nikmati pengalaman dunia Anda!

Pesan "2 Pendengar Audio"

Saat Anda menambahkan controller karakter ke scene, Anda mungkin melihat pesan di bagian bawah editor yang mengatakan, "Ada 2 pendengar audio di scene." Ini karena Kamera Utama (kamera yang ada secara default) memiliki komponen pendengar audio, dan begitu juga controller karakter yang Anda tambahkan. Karena kamera mewakili perspektif player, hanya satu dari mereka yang dapat mendengarkan audio. Anda dapat mengubahnya dengan menghapus komponen pendengar audio dari Kamera Utama. Anda bahkan dapat menghapus objek permainan Kamera Utama sama sekali jika Anda mau, karena FPSController memiliki kameranya sendiri.

Jatuh Melalui Dunia

Jika Anda menemukan kamera jatuh ke seluruh dunia setiap kali Anda menjalankan scene Anda, kemungkinan controller karakter Anda macet sebagian di tanah. Coba naikkan controller karakter Anda sedikit di atas tanah. Saat scene dimulai, kamera harus jatuh sedikit hingga menyentuh tanah dan berhenti.

Selanjutnya, seperti biasa, kita akan fokus pada langkah terakhir di peta jalan: memprogram objek di scene. Berikut rekap rencana aksi kami:

1. Impor model karakter ke dalam scene.
2. Menerapkan kontrol kamera untuk melihat karakter.
3. Tulis skrip yang memungkinkan player untuk berlarian di tanah.
4. Tambahkan kemampuan melompat ke skrip gerakan. 5. Memutar animasi pada model berdasarkan gerakannya.

Salin proyek dari bab sebelumnya untuk memodifikasinya, atau buat proyek Unity baru (pastikan itu disetel ke 3D, bukan proyek 2D dari bab 5) dan salin file scene dari proyek bab 2; apa pun itu, ambil juga folder awal dari unduhan bab ini untuk mendapatkan model karakter yang akan kita gunakan. Catatan Kami akan membangun proyek bab ini di area bertembok dari bab 2. Kami akan mempertahankan dinding dan lampu tetapi mengganti player dan semua skrip. Jika Anda membutuhkannya, unduh file sampel dari bab itu.

Dengan asumsi Anda memulai proyek yang sudah selesai dari bab 2 (demo gerakan, bukan proyek selanjutnya), mari hapus semua yang tidak kita perlukan untuk bab ini. Pertama-tama putuskan sambungan kamera dari pemutar dalam daftar Hierarchy (drag objek kamera dari objek pemutar). Sekarang hapus objek player; jika Anda tidak melepaskan kamera terlebih dahulu maka itu akan dihapus juga, tetapi yang Anda inginkan adalah menghapus hanya kapsul player dan meninggalkan kamera. Atau, jika Anda telah menghapus kamera secara tidak sengaja, buat objek kamera baru dengan memilih GameObject > Camera. Hapus semua skrip juga (yang melibatkan penghapusan komponen skrip dari kamera serta menghapus file dalam tampilan Proyek), hanya menyisakan dinding, lantai, dan lampu.

Menyesuaikan tampilan kamera untuk orang ketiga

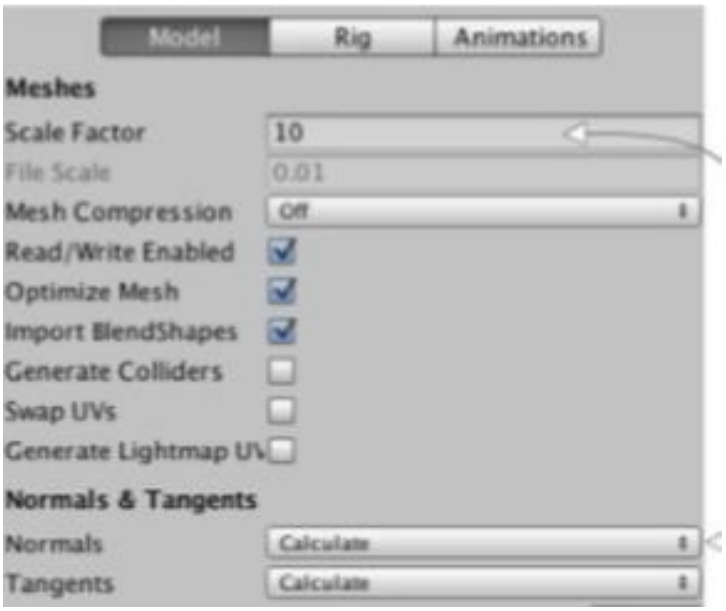
Sebelum kita dapat menulis kode untuk membuat player bergerak, kita perlu menempatkan karakter dalam scene dan menyiapkan kamera untuk melihat karakter tersebut. Kami akan mengimpor model humanoid tanpa wajah untuk digunakan sebagai karakter player, dan kemudian menempatkan kamera di atas pada sudut untuk melihat player secara miring. Gambar 5.40 membandingkan seperti apa scene dalam pandangan orang pertama dengan seperti apa scene itu dalam pandangan orang ketiga (ditunjukkan dengan beberapa blok besar yang akan kita tambahkan dalam bab ini).

Gambar 5.40 Perbandingan pandangan orang pertama dan orang ketiga secara berdampingan.

Kami sudah menyiapkan scenenya, jadi sekarang mari kita masukkan model karakter ke dalam scene

5.13 MENGIMPOR KARAKTER UNTUK DILIHAT

Folder awal untuk unduhan bab ini mencakup model dan tekstur; seperti yang akan Anda ingat dari bab 4, FBX adalah modelnya dan TGA adalah teksturnya. Impor file FBX ke dalam proyek; drag file ke tampilan Proyek, atau klik kanan di tampilan Proyek dan pilih Impor Aset Baru. Kemudian lihat di Inspector untuk menyesuaikan pengaturan impor untuk model. Nanti di bab ini Anda akan menyesuaikan animasi yang diimpor, tetapi untuk saat ini Anda hanya perlu membuat beberapa penyesuaian di tab Model. Pertama-tama ubah nilai Scale Factor menjadi 10 (untuk menetralkan sebagian nilai File Scale dari .01) sehingga model akan menjadi ukuran yang benar. Sedikit lebih jauh ke bawah Anda akan menemukan opsi Normals (lihat gambar 5.41). Pengaturan ini mengontrol bagaimana pencahayaan dan bayangan muncul pada model, menggunakan konsep matematika 3D yang dikenal sebagai normal.



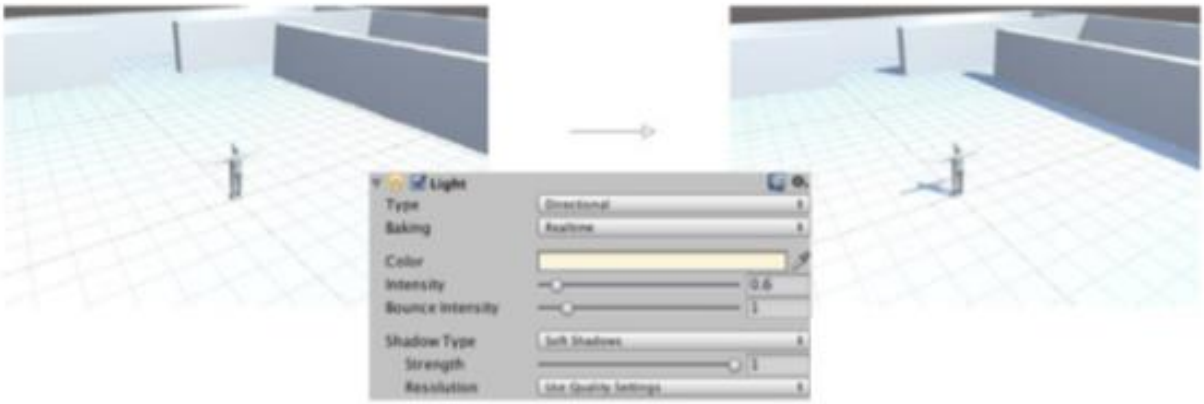
Gambar 5.41 Impor pengaturan untuk model karakter

Dengan tekstur yang diterapkan, drag model pemutar dari tampilan Proyek ke atas ke dalam scene. Posisikan karakter pada 0, 1.1, 0 sehingga berada di tengah ruangan dan terangkat untuk berdiri di lantai. Hebat, kami memiliki karakter orang ketiga di scene! Karakter yang diimpor memiliki lengan terjulur lurus ke samping, bukan pose lengan ke bawah yang

lebih alami. Itu karena animasi belum diterapkan; bahwa posisi lengan-keluar disebut sebagai T-pose dan standar untuk karakter animasi default ke T-pose sebelum mereka dianimasikan.

Menambahkan bayangan ke scene

Sebelum kita melanjutkan, saya ingin menjelaskan sedikit tentang bayangan yang dilemparkan oleh karakter. Kami menerima bayangan begitu saja di dunia nyata, tetapi bayangan tidak dijamin di dunia virtual game. Untungnya Unity dapat menangani detail ini, dan bayangan diaktifkan untuk cahaya default yang datang dengan scene baru. Pilih cahaya arah dalam scene Anda dan kemudian lihat di Inspector untuk opsi Jenis Bayangan. Pengaturan itu (ditunjukkan pada gambar 7.5) sudah pada Soft Shadows untuk lampu default, tetapi perhatikan menu juga memiliki opsi No Shadows.



Gambar 5.42 Sebelum dan sesudah mengeluarkan bayangan dari cahaya arah

Itu saja yang perlu Anda lakukan untuk mengatur bayangan dalam proyek ini, tetapi masih banyak lagi yang harus Anda ketahui tentang bayangan dalam game. Menghitung bayangan dalam sebuah scene adalah bagian yang sangat memakan waktu dari grafik komputer, sehingga permainan sering mengambil jalan pintas dan memalsukan hal-hal dengan berbagai cara untuk mencapai tampilan visual yang diinginkan. Jenis bayangan yang dilemparkan dari karakter disebut sebagai bayangan waktu nyata karena bayangan dihitung saat permainan berjalan dan bergerak dengan objek yang bergerak. Pengaturan pencahayaan yang sangat realistis akan membuat semua objek memancarkan dan menerima bayangan dalam waktu nyata, tetapi agar perhitungan bayangan berjalan cukup cepat, bayangan waktu nyata dibatasi dalam bagaimana bayangan terlihat atau lampu mana yang bahkan dapat mengeluarkan bayangan. Perhatikan bahwa hanya cahaya arah yang memberikan bayangan dalam scene ini.

Cara umum lainnya untuk menangani bayangan dalam game adalah dengan teknik yang disebut lightmapping. *Definisi,* Lightmaps adalah tekstur yang diterapkan pada geometri level, dengan gambar bayangan dimasukkan ke dalam gambar tekstur. Definisi Menggambar bayangan pada tekstur model disebut sebagai baking bayangan. Karena gambar-gambar ini dibuat sebelumnya (bukan saat game sedang berjalan), mereka bisa sangat rumit dan realistis. Pada sisi negatifnya, karena bayangan dihasilkan sebelumnya, mereka tidak akan bergerak. Dengan demikian, lightmap sangat bagus digunakan untuk geometri level statis, tetapi tidak berguna untuk objek dinamis seperti karakter. Lightmaps dihasilkan secara otomatis daripada dilukis dengan tangan. Komputer menghitung bagaimana lampu di scene akan menerangi level sementara kegelapan halus menumpuk di sudut-sudut. Di Unity, sistem untuk merender lightmaps disebut Enlighten, jadi Anda bisa mencari kata kunci itu di manual Unity.

Apakah menggunakan bayangan atau peta cahaya real-time atau tidak bukanlah pilihan semua-atau-tidak sama sekali. Anda dapat mengatur properti Culling Mask pada

cahaya sehingga bayangan waktu nyata hanya digunakan untuk objek tertentu, memungkinkan Anda untuk menggunakan peta cahaya berkualitas lebih tinggi untuk objek lain dalam scene. Demikian pula, meskipun Anda hampir selalu ingin karakter utama memberikan bayangan, terkadang Anda tidak ingin karakter tersebut menerima bayangan; semua objek mesh memiliki pengaturan untuk melemparkan dan menerima bayangan.

Gambar 5.43 pengaturan Cast Shadows dan Receive Shadows di Inspector

Culling adalah istilah umum untuk menghilangkan hal-hal yang tidak diinginkan. Kata itu banyak muncul dalam grafik komputer dalam banyak konteks berbeda, tetapi dalam kasus ini culling mask adalah kumpulan objek yang ingin Anda hapus dari shadow casting. Baiklah, sekarang Anda memahami dasar-dasar bagaimana menerapkan bayangan pada scene Anda. Pencahayaan dan bayangan level bisa menjadi topik besar tersendiri (buku tentang pengeditan level sering menghabiskan banyak bab tentang pemetaan cahaya), tetapi di sini kita membatasi diri untuk menyalakan bayangan waktu nyata pada satu lampu. Dan dengan itu, mari kita alihkan perhatian kita ke kamera.

Mengorbit kamera di sekitar karakter player

Dalam demo orang pertama, kamera ditautkan ke objek pemutar dalam tampilan Hierarki sehingga mereka dapat berputar bersama. Namun, dalam gerakan orang ketiga, karakter player akan menghadap ke arah yang berbeda secara independen dari kamera. Oleh karena itu, Anda tidak ingin menyeret kamera ke karakter player dalam tampilan Hierarki kali ini. Sebagai gantinya, kode kamera akan bergerak posisinya bersama dengan karakter tetapi akan berputar secara independen dari karakter.

Pertama, tempatkan kamera di tempat yang Anda inginkan agar relatif terhadap pemutar; Saya menggunakan posisi 0, 3.5, -3.75 untuk meletakkan kamera di atas dan di belakang karakter (reset rotasi ke 0, 0, 0 jika diperlukan). Kemudian buat skrip bernama OrbitCamera. Pasang komponen skrip ke kamera lalu drag karakter player ke slot Target skrip. Sekarang Anda dapat memutar scene untuk melihat kode kamera beraksi.

Daftar Skrip kamera untuk memutar target sambil melihatnya

```
using UnityEngine;
using System.Collections;

public class OrbitCamera : MonoBehaviour {
    [SerializeField] private Transform target;

    public float rotSpeed = 1.5f;

    private float _rotY;
    private Vector3 _offset;

    void Start() {
        _rotY = transform.eulerAngles.y;
        _offset = target.position - transform.position;
    }


    void LateUpdate() {
        float horInput = Input.GetAxis("Horizontal");
        if (horInput != 0) {
            _rotY += horInput * rotSpeed;
        } else {
            _rotY += Input.GetAxis("Mouse X") * rotSpeed * 3;
        }

        Quaternion rotation = Quaternion.Euler(0, _rotY, 0);
        transform.position = target.position - (rotation * _offset);
        transform.LookAt(target);
    }
}
```

Saat Anda membaca daftar, perhatikan variabel serial untuk target. Kode perlu mengetahui objek apa yang mengorbit kamera, jadi variabel ini diserialkan agar muncul di dalam editor Unity dan memiliki karakter player yang terhubung dengannya. Beberapa variabel berikutnya adalah nilai rotasi yang digunakan dengan cara yang sama seperti pada kode kontrol kamera dari bab 2. Dan ada nilai _offset yang dideklarasikan; _offset diatur dalam Start() untuk menyimpan perbedaan posisi antara kamera dan target. Dengan cara ini, posisi relatif kamera dapat dipertahankan saat skrip berjalan. Dengan kata lain, kamera akan tetap berada pada jarak awal dari karakter terlepas ke arah mana ia berputar. Sisa kode ada di dalam fungsi LateUpdate().

LateUpdate() adalah metode lain yang disediakan oleh MonoBehaviour dan sangat mirip dengan Update(); ini adalah metode yang menjalankan setiap frame. Perbedaannya, seperti namanya, adalah bahwa LateUpdate() dipanggil pada semua objek setelah Update() dijalankan pada semua objek. Dengan cara ini, kami dapat memastikan bahwa kamera diperbarui setelah target dipindahkan. Pertama, kode menambah nilai rotasi berdasarkan kontrol input. Kode ini melihat dua kontrol input yang berbeda—tombol panah horizontal dan gerakan mouse horizontal—jadi sebuah kondisi digunakan untuk beralih di antara keduanya. Kode memeriksa apakah tombol panah horizontal sedang ditekan; jika ya, maka ia menggunakan input itu, tetapi jika tidak, ia akan memeriksa mouse. Dengan memeriksa kedua input secara terpisah, kode dapat berputar pada kecepatan yang berbeda untuk setiap jenis input.

Selanjutnya kode memposisikan kamera berdasarkan posisi target dan nilai rotasi. Baris transform.position mungkin adalah "aha!" terbesar. dalam kode ini, karena menyediakan beberapa matematika penting yang belum pernah Anda lihat sebelumnya di bab-bab sebelumnya. Mengalikan vektor posisi dengan quaternion (perhatikan bahwa sudut rotasi diubah menjadi quaternion menggunakan Quaternion.Euler) menghasilkan posisi yang

digeser sesuai dengan rotasi itu. Vektor posisi yang diputar ini kemudian ditambahkan sebagai offset dari posisi karakter untuk menghitung posisi kamera.

Terakhir, kode menggunakan metode LookAt() untuk mengarahkan kamera ke target; fungsi ini mengarahkan satu objek (bukan hanya kamera) ke objek lain. Nilai rotasi yang dihitung sebelumnya digunakan untuk memposisikan kamera pada sudut yang benar di sekitar target, tetapi pada langkah itu kamera hanya diposisikan dan tidak diputar. Jadi tanpa garis LookAt akhir, posisi kamera akan mengorbit di sekitar karakter tetapi tidak selalu melihatnya. Silakan dan komentari baris itu untuk melihat apa yang terjadi. Kamera memiliki skrip untuk mengorbit di sekitar karakter player; selanjutnya adalah kode yang menggerakkan karakter.

Memprogram kontrol gerakan relatif kamera

Sekarang model karakter diimpor ke Unity dan kami telah menulis kode untuk mengontrol tampilan kamera, saatnya untuk memprogram kontrol untuk bergerak di sekitar scene. Mari programkan kontrol relatif kamera yang akan menggerakkan karakter ke berbagai arah saat tombol panah ditekan, serta memutar karakter untuk menghadap ke arah yang berbeda tersebut.

Apa yang dimaksud dengan "kamera-relatif"?

Seluruh gagasan "kamera-relatif" agak tidak jelas tetapi sangat penting untuk dipahami. Ini mirip dengan perbedaan lokal versus global yang disebutkan dalam bab-bab sebelumnya: "kiri" menunjuk ke arah yang berbeda ketika Anda berarti "kiri dari objek lokal" atau "kiri dari seluruh dunia." Dengan cara yang sama, ketika Anda "memindahkan karakter ke kiri", maksud Anda ke kiri karakter, atau sisi kiri layar? Kamera dalam game orang pertama ditempatkan di dalam karakter dan bergerak dengannya, jadi tidak ada perbedaan antara kiri karakter dan kiri kamera. Namun, pandangan orang ketiga menempatkan kamera di luar karakter, dan dengan demikian kiri kamera dapat diarahkan ke arah yang berbeda dari kiri karakter. Misalnya, mereka benar-benar berlawanan arah jika kamera melihat ke depan karakter. Jadi kami harus memutuskan apa yang kami inginkan terjadi dalam permainan khusus kami dan pengaturan kontrol.

Meskipun terkadang game melakukannya dengan cara lain, sebagian besar game orang ketiga membuat kontrolnya relatif terhadap kamera. Saat player menekan tombol kiri, karakter bergerak ke kiri layar, bukan kiri karakter. Seiring waktu dan melalui eksperimen dengan mencoba skema kontrol yang berbeda, desainer game telah menemukan bahwa player menemukan kontrol yang lebih intuitif dan lebih mudah dipahami ketika "kiri" berarti "sisi kiri layar" (yang, tidak secara kebetulan, juga merupakan milik player). Menerapkan kontrol kamera-relatif melibatkan dua langkah utama: pertama-tama rotate karakter player untuk menghadap ke arah kontrol, dan kemudian gerakkan karakter ke depan. Mari kita tulis kode untuk kedua langkah ini selanjutnya.

Memutar karakter untuk menghadapi arah gerakan

Pertama kita akan menulis kode untuk membuat karakter menghadap ke arah tombol panah. Buat skrip C# bernama RelativeMovement (lihat daftar 7.2). Drag skrip itu ke karakter player, lalu tautkan kamera ke properti target komponen skrip (sama seperti Anda menautkan karakter ke target skrip kamera). Sekarang karakter akan menghadap ke arah yang berbeda saat Anda menekan kontrol, menghadap ke arah relatif ke kamera, atau berdiri diam saat Anda tidak menekan tombol panah apa pun (yaitu, saat memutar menggunakan mouse).

Daftar Memutar karakter relatif terhadap kamera

```
using UnityEngine;
using System.Collections;

public class RelativeMovement : MonoBehaviour {
    [SerializeField] private Transform target;

    void Update() {
        Vector3 movement = Vector3.zero;

        float horInput = Input.GetAxis("Horizontal");
        float vertInput = Input.GetAxis("Vertical");
        if (horInput != 0 || vertInput != 0) {
            movement.x = horInput;
            movement.z = vertInput;

            Quaternion tmp = target.rotation;
            target.eulerAngles = new Vector3(0, target.eulerAngles.y, 0);
            movement = target.TransformDirection(movement);
            target.rotation = tmp;

            transform.rotation = Quaternion.LookRotation(movement);
        }
    }
}
```

Kode dalam daftar ini dimulai dengan cara yang sama seperti yang dilakukan daftar 7.1, dengan variabel serial untuk target. Sama seperti skrip sebelumnya yang membutuhkan referensi ke objek yang akan diorbitnya, skrip ini membutuhkan referensi ke objek yang akan dipindahkan relatif. Kemudian kita masuk ke fungsi Update(). Baris pertama fungsi mendeklarasikan nilai Vector3 dari 0, 0, 0. Penting untuk membuat vektor nol dan mengisi nilainya nanti daripada hanya membuat vektor nanti dengan nilai pergerakan yang dihitung, karena nilai pergerakan vertikal dan horizontal akan dihitung dalam langkah yang berbeda namun semuanya harus menjadi bagian dari vektor yang sama.

Selanjutnya kita periksa kontrol input, seperti yang kita miliki di skrip sebelumnya. Di sinilah nilai X dan Z ditetapkan dalam vektor gerakan, untuk gerakan horizontal di sekitar scene. Ingat bahwa Input.GetAxis() mengembalikan 0 jika tidak ada tombol yang ditekan, dan bervariasi antara 1 dan -1 saat tombol tersebut ditekan; menempatkan nilai itu dalam vektor gerakan mengatur gerakan ke arah positif atau negatif dari sumbu itu (sumbu X kiri-kanan, dan sumbu Z maju mundur).

Beberapa baris berikutnya adalah tempat vektor gerakan disesuaikan menjadi kamera relatif. Secara khusus, TransformDirection() digunakan untuk mengubah dari koordinat Lokal ke Global. Ini adalah hal yang sama yang kita lakukan dengan TransformDirection() di bab 2, kecuali kali ini kita mengubah dari sistem koordinat target, bukan dari sistem koordinat player. Sementara itu, kode sebelum dan sesudah baris TransformDirection() adalah menyelaraskan sistem koordinat untuk kebutuhan kita: pertama simpan rotasi target untuk memulihkan nanti, lalu sesuaikan rotasi sehingga hanya di sekitar sumbu Y dan tidak ketiga sumbu . Terakhir lakukan transformasi dan kembalikan rotasi target. Semua kode itu untuk menghitung arah gerakan sebagai vektor. Baris terakhir kode menerapkan arah gerakan itu ke karakter dengan mengubah Vector3 menjadi Quaternion menggunakan Quaternion.LookDirection() dan menetapkan nilai itu. Coba jalankan gamenya sekarang untuk melihat apa yang terjadi!

Memutar dengan lancar (interpolasi) dengan menggunakan Lerp

Saat ini, rotasi karakter langsung terkunci ke wajah yang berbeda, tetapi akan terlihat lebih baik jika karakter diputar dengan mulus ke wajah yang berbeda. Kita dapat

melakukannya dengan menggunakan operasi matematika yang disebut Lerp. Pertama tambahkan variabel ini ke skrip:
Public float rotSpee = 15.0f;
Kemudian ganti baris transform.rotation... yang ada di akhir listing 7.2 dengan kode berikut:

```
...
                Quaternion direction = Quaternion.LookRotation(movement);
                transform.rotation = Quaternion.Lerp(transform.rotation,
                direction, rotSpeed * Time.deltaTime);
                }
        }
}
```

Sekarang, alih-alih mengambil langsung ke nilai LookRotation(), nilai tersebut digunakan secara tidak langsung sebagai arah target untuk diputar. Metode Quaternion.Lerp() berputar dengan mulus antara rotasi saat ini dan target (dengan parameter ketiga mengontrol seberapa cepat untuk memutar). Kebetulan, istilah untuk perubahan mulus antar nilai adalah interpolasi; Anda dapat menginterpolasi antara dua jenis nilai apa pun, bukan hanya nilai rotasi. Lerp adalah akronim kuasi untuk "interpolasi linier," dan Unity menyediakan metode Lerp untuk vektor dan nilai float juga (untuk menginterpolasi posisi, warna, atau apa pun). Quaternions juga memiliki metode alternatif terkait erat untuk interpolasi yang disebut Slerp (untuk interpolasi linier bola). Untuk putaran yang lebih lambat, rotasi Slerp mungkin terlihat lebih baik daripada Lerp.

Saat ini karakter berputar di tempat tanpa bergerak; di bagian selanjutnya kita akan menambahkan kode untuk memindahkan karakter. Catatan Karena menghadap ke samping menggunakan kontrol keyboard yang sama seperti mengorbit kamera, karakter akan berputar perlahan sementara arah gerakan menunjuk ke samping. Penggandaan kontrol ini adalah perilaku yang diinginkan dalam proyek ini.

Bergerak maju ke arah itu

Seperti yang Anda ingat dari bab 2, untuk memindahkan player di sekitar scene, kita perlu menambahkan komponen controller karakter ke objek player. Pilih karakter lalu pilih Components > Physics > Character Controller. Di Inspector Anda harus sedikit mengurangi radius controller menjadi .4, tetapi jika tidak, pengaturan default semuanya baik-baik saja untuk model karakter ini. Daftar berikutnya menunjukkan apa yang perlu Anda tambahkan dalam skrip RelativeMovement.

Daftar Menambahkan kode untuk mengubah posisi player

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(CharacterController))]
public class RelativeMovement : MonoBehaviour {
...
public float moveSpeed = 6.0f;

private CharacterController _charController;

void Start() {
    _charController = GetComponent<CharacterController>();
}

    void Update() {
        ...
          movement.x = horInput * moveSpeed;
          movement.z = vertInput * moveSpeed;
          movement = Vector3.ClampMagnitude(movement, moveSpeed);
          ...
        }

        movement *= Time.deltaTime;
        _charController.Move(movement);
    }
}
```

Jika Anda memainkan game ini sekarang, Anda dapat melihat karakter (terjebak dalam pose-T) bergerak di sekitar scene. Hampir keseluruhan daftar ini adalah kode yang sudah Anda lihat sebelumnya, jadi saya hanya akan meninjau semuanya secara singkat. Pertama, ada metode RequireComponent() di bagian atas kode. Seperti yang dijelaskan di bab 2, RequireComponent() akan memaksa Unity untuk memastikan GameObject memiliki komponen dari tipe yang diteruskan ke perintah. Baris ini opsional; Anda tidak perlu memintanya, tetapi tanpa komponen ini skrip akan mengalami kesalahan.

Selanjutnya ada nilai gerakan yang dideklarasikan, diikuti dengan membuat skrip ini referensi ke controller karakter. Seperti yang Anda ingat dari bab-bab sebelumnya, GetComponent() mengembalikan komponen lain yang dilampirkan ke objek yang diberikan, dan jika objek yang dicari tidak didefinisikan secara eksplisit, maka diasumsikan sebagai this.GetComponent() (yaitu, sama objek sebagai skrip ini).

Nilai gerakan ditetapkan berdasarkan kontrol input. Ini juga ada di daftar sebelumnya; perubahan di sini adalah kami juga memperhitungkan kecepatan gerakan. Kalikan kedua sumbu gerakan dengan kecepatan gerakan, lalu gunakan Vector3.Clamp-Magnitude() untuk membatasi besaran vektor ke kecepatan gerakan; penjepit diperlukan karena jika tidak, gerakan diagonal akan memiliki magnitudo yang lebih besar daripada gerakan langsung di sepanjang sumbu (gambarkan sisi dan sisi miring segitiga siku-siku).

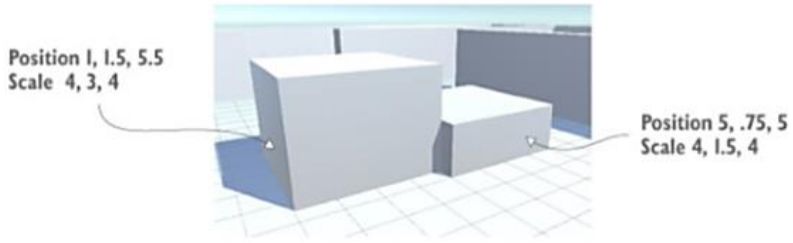
Akhirnya, pada akhirnya kita mengalikan nilai pergerakan dengan deltaTime untuk mendapatkan pergerakan frame rate-independen (ingat bahwa "frame rate-independen" berarti karakter bergerak pada kecepatan yang sama pada komputer yang berbeda dengan frame rate yang berbeda). Berikan nilai gerakan ke CharacterController.Move() untuk membuat gerakan. Ini menangani semua gerakan horizontal; selanjutnya mari kita jaga gerakan vertikal.

Menerapkan aksi lompat

Di bagian sebelumnya kami menulis kode untuk membuat karakter berlarian di tanah. Namun, dalam pengantar bab, saya juga menyebutkan membuat karakter melompat, jadi mari kita lakukan sekarang. Sebagian besar game orang ketiga memang memiliki kontrol untuk melompat. Dan bahkan jika tidak, mereka hampir selalu memiliki gerakan vertikal dari

karakter yang jatuh dari tepian. Kode kami akan menangani lompatan dan jatuh. Secara khusus, kode ini akan memiliki gravitasi yang menarik player ke bawah setiap saat, tetapi kadang-kadang sentakan ke atas akan diterapkan saat player melompat.

Sebelum kita menulis kode ini, mari tambahkan beberapa platform yang dinaikkan ke TKP. Saat ini tidak ada yang bisa dilompati atau jatuh! Buat beberapa objek kubus lagi, lalu ubah posisi dan scalenya untuk memberikan platform player untuk melompat. Dalam proyek sampel, saya menambahkan dua kubus dan menggunakan pengaturan ini: Posisi 5, .75, 5 dan Scale 4, 1,5, 4; Posisi 1, 1.5, 5.5 dan Scale 4, 3, 4. Gambar 7.8 menunjukkan platform yang ditinggikan.



Gambar 5.44 Beberapa platform terangkat ditambahkan ke scene yang jarang

Menerapkan kecepatan dan akselerasi vertikal

Seperti yang disebutkan ketika kami pertama kali mulai menulis skrip RelativeMovement dalam daftar 7.2, nilai gerakan dihitung dalam langkah terpisah dan ditambahkan ke vektor gerakan secara bertahap. Daftar berikutnya menambahkan gerakan vertikal ke vektor yang ada.

Daftar Menambahkan gerakan vertikal ke skrip RelativeMovement

```
...
public float jumpSpeed = 15.0f;
public float gravity = -9.8f;
public float terminalVelocity = -10.0f;
public float minFall = -1.5f;

private float _vertSpeed;
...
void Start() {
    _vertSpeed = minFall;
    ...
}


    void Update() {
        ...
        if (_charController.isGrounded) {
            if (Input.GetButtonDown("Jump")) {
                _vertSpeed = jumpSpeed;
            } else {
                _vertSpeed = minFall;
            }
        } else {
            _vertSpeed += gravity * 5 * Time.deltaTime;
            if (_vertSpeed < terminalVelocity) {
                _vertSpeed = terminalVelocity;
            }
        }
        movement.y = _vertSpeed;

        movement *= Time.deltaTime;
        _charController.Move(movement);
    }
}
```

Seperti biasa kita mulai dengan menambahkan beberapa variabel baru ke atas skrip untuk berbagai nilai gerakan, dan menginisialisasi nilai dengan benar. Kemudian kita melompat ke bawah tepat setelah pernyataan if besar untuk gerakan horizontal, di mana kita akan menambahkan pernyataan if besar lainnya untuk gerakan vertikal. Secara khusus, kode akan memeriksa apakah karakter ada di tanah, karena kecepatan vertikal akan disesuaikan secara berbeda tergantung pada apakah karakter ada di tanah. CharacterController menyertakan isGrounded untuk memeriksa apakah karakter ada di tanah; nilai ini benar jika bagian bawah controller karakter bercollision dengan apa pun di bingkai terakhir. Jika karakter berada di tanah, maka nilai kecepatan vertikal (variabel personal _vertSpeed) harus disetel ulang ke nol. Karakter tidak jatuh saat di tanah, jadi jelas kecepatan vertikalnya adalah 0; jika karakter kemudian melangkah dari langkan, kita akan mendapatkan gerakan yang bagus dan tampak alami karena kecepatan jatuh akan dipercepat dari nol.

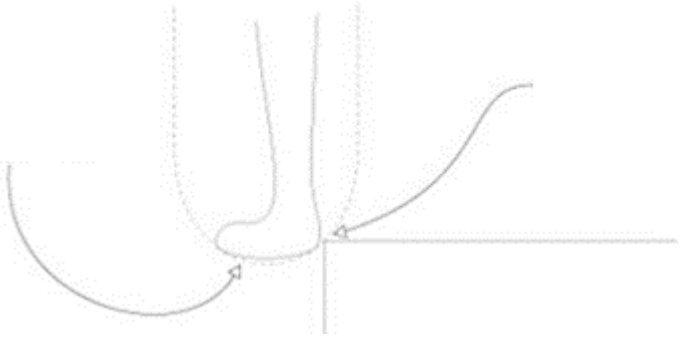
Yah, tidak persis 0; kami sebenarnya mengatur kecepatan vertikal ke minFall, gerakan sedikit ke bawah, sehingga karakter akan selalu menekan tanah sambil berlari secara horizontal. Perlu ada kekuatan ke bawah untuk berlari naik turun di medan yang tidak rata. Pengecualian untuk nilai kecepatan ground ini adalah jika tombol lompat diklik. Dalam hal ini, kecepatan vertikal harus disetel ke angka tinggi. Pernyataan if memeriksa GetButtonDown(), fungsi input baru yang bekerja seperti GetAxis(), mengembalikan status kontrol input yang ditunjukkan. Dan seperti sumbu input Horizontal dan Vertikal, kunci persis yang ditetapkan untuk Jump ditentukan dengan masuk ke Pengaturan input di bawah Edit > Pengaturan Proyek (penetapan kunci default adalah Spasi—yaitu, bilah spasi).

Kembali ke kondisi if yang lebih besar, jika karakter tidak di tanah, maka kecepatan vertikal harus terus-menerus dikurangi oleh gravitasi. Perhatikan bahwa kode ini tidak hanya mengatur nilai kecepatan tetapi juga menguranginya; dengan cara ini, itu bukan kecepatan konstan melainkan akselerasi ke bawah, menghasilkan gerakan jatuh yang realistis. Melompat akan terjadi secara alami, karena kecepatan karakter ke atas secara bertahap berkurang menjadi 0 dan malah mulai jatuh.

Terakhir, kode memastikan kecepatan ke bawah tidak melebihi kecepatan terminal. Perhatikan bahwa operator "kurang dari" dan bukan "lebih besar dari", karena ke bawah adalah nilai kecepatan negatif. Kemudian setelah pernyataan if besar, tetapkan kecepatan vertikal yang dihitung ke sumbu Y dari vektor gerakan. Dan hanya itu yang Anda butuhkan untuk gerakan vertikal yang realistis! Dengan menerapkan akselerasi ke bawah yang konstan saat karakter tidak di tanah, dan menyesuaikan kecepatan dengan tepat saat karakter berada di tanah, kode ini menciptakan perilaku jatuh yang bagus. Tapi ini semua tergantung pada pendeteksian tanah dengan benar, dan ada kesalahan halus yang perlu kita perbaiki.

Memodifikasi deteksi tanah untuk menangani tepi dan lereng

Seperti yang dijelaskan di bagian sebelumnya, properti isGrounded dari CharacterController menunjukkan apakah bagian bawah controller karakter bercollision dengan apa pun di bingkai terakhir. Meskipun pendekatan untuk mendeteksi tanah ini berhasil sebagian besar waktu, Anda mungkin akan memperhatikan bahwa karakter tersebut tampaknya melayang di udara saat melangkah keluar. Itu karena area tumbukan karakter adalah kapsul di sekitarnya (Anda bisa melihatnya saat memilih objek karakter) dan bagian bawah kapsul ini akan tetap bersentuhan dengan tanah saat player keluar dari tepi platform. Gambar 5.45 mengilustrasikan masalah tersebut. Ini tidak akan berhasil sama sekali!



Gambar 5.45 Diagram yang menunjukkan kapsul controller karakter menyentuh tepi platform

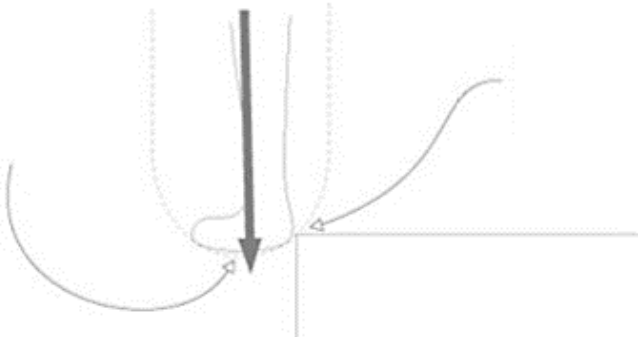
Demikian pula, jika karakter berdiri di lereng, deteksi tanah saat ini akan menyebabkan perilaku bermasalah. Cobalah sekarang dengan membuat balok miring pada platform yang ditinggikan. Buat objek kubus baru dan atur nilai transformasinya ke Posisi -1.5, 1.5, 5 Rotasi 0, 0, -25 Scale 1, 4, 4.

Jika Anda melompat ke lereng dari tanah, Anda akan menemukan bahwa Anda dapat melompat lagi dari tengah lereng dan dengan demikian naik ke atas. Itu karena kemiringan menyentuh bagian bawah kapsul secara miring dan kode saat ini menganggap setiap collision di bagian bawah sebagai pijakan yang kokoh. Sekali lagi, ini tidak akan berhasil; karakter harus meluncur kembali ke bawah, tidak memiliki pijakan yang kokoh untuk melompat.

Meluncur kembali hanya diinginkan di lereng yang curam. Di lereng yang dangkal, seperti tanah yang tidak rata, kami ingin player berlari tanpa terpengaruh. Jika Anda ingin mengujinya, buat tanjakan dangkal dengan membuat kubus dan atur ke Posisi 5.25, .25, .25 Rotasi 0, 90, 75 Scale 1, 6, 3.

Semua masalah ini memiliki akar penyebab yang sama: memeriksa collision di bagian bawah karakter bukanlah cara yang bagus untuk menentukan apakah karakter ada di tanah. Sebagai gantinya, mari gunakan raycasting untuk mendeteksi tanah. Dalam bab 3 AI menggunakan raycasting untuk mendeteksi rintangan di depannya; mari gunakan pendekatan yang sama untuk mendeteksi permukaan di bawah karakter. Keluarkan sinar lurus ke bawah dari posisi player. Jika mencatat pukulan tepat di bawah kaki karakter, itu berarti player berdiri di tanah.

Ini memang memperkenalkan situasi baru untuk ditangani: ketika raycast tidak mendeteksi tanah di bawah karakter tetapi controller karakter bercollision dengan tanah. Seperti pada gambar 5.45, kapsul masih bercollision dengan platform saat karakter berjalan dari tepi. Gambar 5.46 menambahkan pancaran sinar ke diagram untuk menunjukkan apa yang akan terjadi sekarang: sinar tidak mengenai platform, tetapi kapsul menyentuh tepi. Kode perlu menangani situasi khusus ini.



Gambar 5.46 Diagram raycasting ke bawah saat melangkah dari langkan

Dalam hal ini, kode harus membuat karakter meluncur dari langkan. Karakter akan tetap jatuh (karena tidak berdiri di tanah), tetapi juga akan menjauh dari titik tumbukan (karena perlu memindahkan kapsul dari platform yang dipukul). Dengan demikian kode akan mendeteksi collision dengan controller karakter dan merespons collision tersebut dengan menyenggol. Daftar berikut menyesuaikan gerakan vertikal dengan semua yang baru saja kita bahas.

Daftar Menggunakan raycasting untuk mendeteksi tanah

```
...
private ControllerColliderHit _contact;
...
        bool hitGround = false;
        RaycastHit hit;
        if (_vertSpeed < 0 &&
            Physics.Raycast(transform.position, Vector3.down, out hit)) {
            float check =
                (_charController.height + _charController.radius) / 1.9f;
            hitGround = hit.distance <= check;
        }

        if (hitGround) {
            if (Input.GetButtonDown("Jump")) {
                _vertSpeed = jumpSpeed;
            } else {
                _vertSpeed = minFall;
            }
        } else {

            _vertSpeed += gravity * 5 * Time.deltaTime;
            if (_vertSpeed < terminalVelocity) {
                _vertSpeed = terminalVelocity;
            }

            if (_charController.isGrounded) {
                if (Vector3.Dot(movement, _contact.normal) < 0) {
                    movement = _contact.normal * moveSpeed;
                } else {
                    movement += _contact.normal * moveSpeed;
                }
            }
        }
        movement.y = _vertSpeed;

        movement *= Time.deltaTime;
        _charController.Move(movement);
    }

    void OnControllerColliderHit(ControllerColliderHit hit) {
        _contact = hit;
    }
}
```

Daftar ini berisi banyak kode yang sama seperti daftar sebelumnya; kode baru diselingi di seluruh skrip gerakan yang ada dan daftar ini membutuhkan kode yang ada untuk konteksnya. Baris pertama menambahkan variabel baru ke atas skrip RelativeMovement. Variabel ini digunakan untuk menyimpan data tentang collision antar fungsi. Beberapa baris berikutnya melakukan raycasting. Kode ini juga berada di bawah gerakan horizontal tetapi sebelum pernyataan if untuk gerakan vertikal. Panggilan Fisika.Raycast() yang sebenarnya seharusnya sudah tidak asing lagi di bab-bab sebelumnya, tetapi parameter spesifik kali ini berbeda. Meskipun posisi untuk mengeluarkan sinar adalah sama (posisi karakter), kali ini arahnya akan ke bawah dan bukan ke depan. Kemudian kami memeriksa seberapa jauh

pancaran sinar itu ketika mengenai sesuatu; jika jarak pukulan berada pada jarak kaki karakter, maka karakter berdiri di tanah, jadi set hitGround ke true.

Agak tidak jelas bagaimana jarak pemeriksaan dihitung, jadi mari kita bahas secara detail. Pertama ambil tinggi controller karakter (yang merupakan tinggi tanpa ujung yang membulat) dan kemudian tambahkan ujung yang membulat. Bagilah nilai ini menjadi dua karena sinar dipancarkan dari tengah karakter (yaitu, sudah setengah jalan) untuk mendapatkan jarak ke bagian bawah karakter. Tapi kami benar-benar ingin memeriksa sedikit di luar bagian bawah karakter untuk memperhitungkan ketidakakuratan kecil dalam raycasting, jadi bagilah dengan 1,9 alih-alih 2 untuk mendapatkan jarak yang sedikit terlalu jauh.

Setelah melakukan raycasting ini, gunakan hitGround alih-alih isGrounded dalam pernyataan if untuk gerakan vertikal. Sebagian besar kode gerakan vertikal akan tetap sama, tetapi tambahkan kode untuk ditangani ketika controller karakter bercollision dengan tanah meskipun player tidak berada di atas tanah (yaitu, ketika player berjalan keluar dari tepi platform). Ada isGrounded conditional baru yang ditambahkan, tetapi perhatikan bahwa itu bersarang di dalam conditional hitGround sehingga isGrounded hanya dicentang ketika hitGround tidak mendeteksi ground.

Data tumbukan mencakup properti normal (sekali lagi, vektor normal mengatakan ke arah mana sesuatu menghadap) yang memberi tahu kita arah untuk menjauh dari titik tumbukan. Tapi satu hal yang rumit adalah kita ingin nudge menjauh dari titik kontak ditangani secara berbeda tergantung ke arah mana player sudah bergerak: ketika gerakan horizontal sebelumnya menuju platform, kita ingin mengganti gerakan itu sehingga karakter menang. 't terus bergerak ke arah yang salah; tetapi ketika menghadap jauh dari tepi, kami ingin menambahkan gerakan horizontal sebelumnya untuk menjaga momentum maju menjauh dari tepi. Vektor gerakan yang menghadap relatif terhadap titik tumbukan dapat ditentukan dengan menggunakan produk titik.

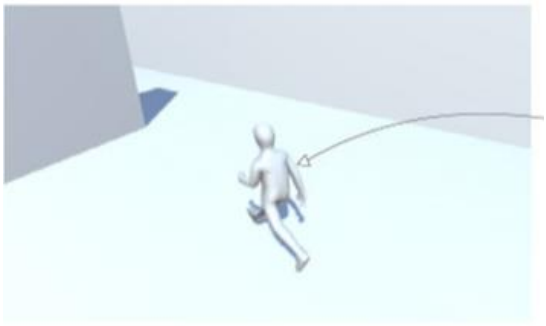
Perkalian titik adalah salah satu jenis operasi matematika yang dapat dilakukan pada dua buah vektor. Singkat cerita, hasil kali titik dari dua vektor berkisar antara -1 dan 1, dengan 1 artinya keduanya menunjuk ke arah yang sama persis, dan -1 ketika keduanya menunjuk ke arah yang berlawanan. Jangan bingung antara "produk titik" dan "produk silang"; perkalian silang adalah operasi matematika vektor yang berbeda tetapi juga sering terlihat. Vector3 menyertakan fungsi Dot() untuk menghitung produk titik dari dua vektor yang diberikan. Jika kita menghitung hasil kali titik antara vektor gerakan dan normal tumbukan, itu akan mengembalikan angka negatif ketika kedua arah saling berhadapan dan angka positif ketika gerakan dan tumbukan menghadap ke arah yang sama.

Akhirnya, bagian paling akhir dari daftar 7.5 menambahkan metode baru ke skrip. Dalam kode sebelumnya kami memeriksa collision normal, tetapi dari mana informasi itu berasal? Ternyata collision dengan controller karakter dilaporkan melalui fungsi callback yang disebut OnControllerColliderHit() yang disediakan MonoBehaviour; untuk menanggapi data collision di tempat lain dalam skrip, data itu harus disimpan dalam variabel eksternal. Itu saja yang dilakukan metode di sini: menyimpan data collision di _contact sehingga data ini dapat digunakan dalam metode Update().

Sekarang kesalahan diperbaiki di sekitar tepi platform dan di lereng. Silakan dan mainkan untuk mengujinya dengan melangkahi tepian dan melompat ke lereng yang curam. Demo gerakan ini hampir selesai. Karakter bergerak di sekitar scene dengan benar, jadi hanya satu hal yang tersisa: menganimasikan karakter keluar dari T-pose.

Menyiapkan animasi pada karakter player

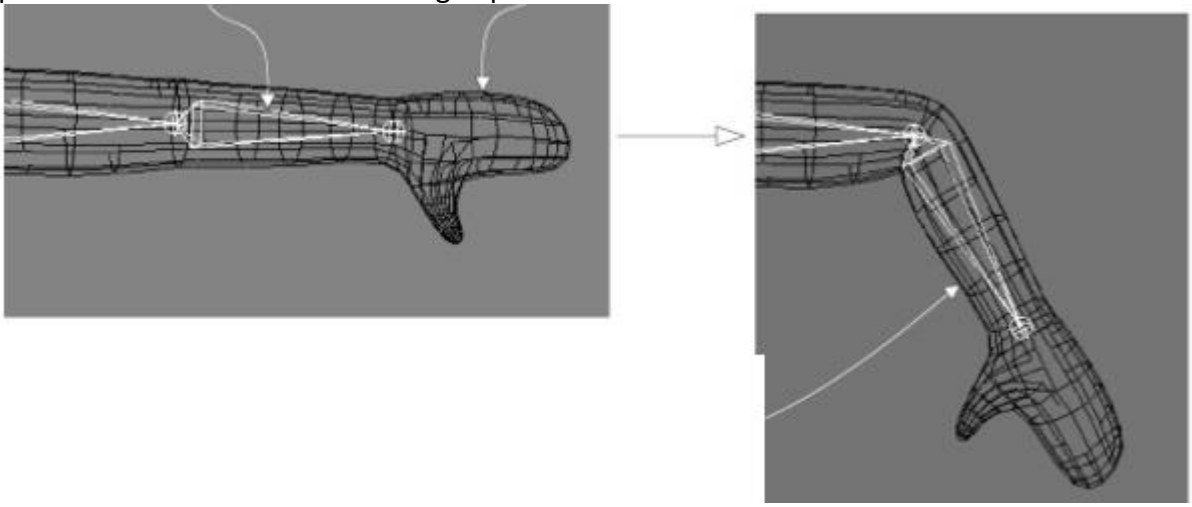
Selain bentuk yang lebih kompleks yang didefinisikan oleh geometri mesh, karakter humanoid membutuhkan animasi. Dalam bab 4 Anda telah mempelajari bahwa animasi adalah paket informasi yang mendefinisikan pergerakan objek 3D terkait. Contoh konkret yang saya berikan adalah karakter yang berjalan-jalan, dan situasi itu persis seperti yang akan Anda lakukan sekarang! Karakter akan berlari di sekitar scene, jadi Anda akan menetapkan animasi yang membuat lengan dan kaki berayun maju mundur. Gambar 7.11 menunjukkan seperti apa tampilannya saat karakter memiliki animasi yang diputar saat bergerak di sekitar scene.



Gambar 5.47 Karakter bergerak dengan animasi berlari

Analogi yang baik untuk memahami animasi 3D adalah dengan berpikir tentang dalang: model 3D adalah boneka, animator adalah dalang, dan animasi adalah rekaman gerakan boneka. Animasi dapat dibuat dengan beberapa pendekatan berbeda; sebagian besar animasi karakter dalam game modern (tentu semua animasi pada karakter bab ini) menggunakan teknik yang disebut animasi kerangka. Animasi kerangka adalah jenis animasi di mana serangkaian tulang diatur di dalam model, dan kemudian tulang dipindahkan selama animasi. Saat tulang bergerak, permukaan model yang terkait dengan tulang itu ikut bergerak.

Sesuai namanya, animasi kerangka paling masuk akal ketika mensimulasikan kerangka di dalam karakter tetapi "kerangka" adalah abstraksi yang berguna setiap kali Anda ingin model menekuk dan melenturkan saat masih memiliki struktur yang pasti tentang bagaimana ia bergerak (misalnya, tentakel yang melambai). Meskipun tulang bergerak dengan kaku, permukaan model di sekitar tulang dapat ditekuk dan dilenturkan.



Gambar 5.48 Animasi kerangka karakter humanoid

Mencapai hasil yang diilustrasikan pada gambar 7.11 melibatkan beberapa langkah: pertama-tama tentukan klip animasi dalam file yang diimpor, kemudian atur controller untuk

memutar klip animasi tersebut, dan terakhir gabungkan controller animasi itu dalam kode Anda. Animasi pada model karakter akan diputar ulang sesuai dengan skrip gerakan yang akan Anda tulis.

Tentu saja hal pertama yang perlu Anda lakukan, sebelum melakukan langkah-langkah tersebut, adalah mengaktifkan sistem animasi. Pilih model pemutar dalam tampilan Proyek untuk melihat pengaturan Impornya di Inspector. Pilih tab Animations dan pastikan Import Animation dicentang. Lalu buka tab Rig dan alihkan Animation Type dari Generic ke Humanoid (ini adalah karakter humanoid, tentu saja). Perhatikan bahwa menu terakhir ini juga memiliki pengaturan Legacy; Generik dan Humanoid keduanya pengaturan dalam istilah umum Mecanim.

Menjelaskan sistem animasi Mecanim Unity

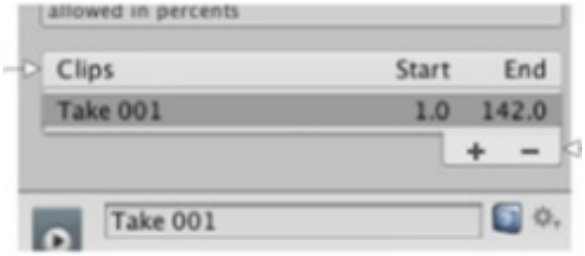
Unity memiliki sistem canggih untuk mengelola animasi pada model, yang disebut Mecanim. Mecanim didasarkan pada animasi kerangka, gaya animasi yang didefinisikan dalam bab ini. Nama khusus Mecanim mengidentifikasi sistem animasi yang lebih baru dan lebih canggih yang baru-baru ini ditambahkan ke Unity sebagai pengganti sistem animasi yang lebih lama. Sistem yang lebih lama masih ada, diidentifikasi sebagai animasi Legacy, tetapi mungkin akan dihapus dalam versi Unity yang akan datang, di mana Mecanim hanya akan menjadi sistem animasi.

Meskipun animasi yang akan kita gunakan semuanya termasuk dalam file FBX yang sama dengan model karakter kita, salah satu keuntungan utama dari pendekatan Mecanim adalah Anda dapat menerapkan animasi dari file FBX lain ke karakter. Misalnya, semua musuh manusia dapat berbagi satu set animasi. Ini memiliki sejumlah keuntungan, termasuk menjaga semua data Anda tetap teratur (model bisa masuk dalam satu folder, sedangkan animasi masuk ke folder lain) serta menghemat waktu yang dihabiskan untuk menganimasikan setiap karakter terpisah. Klik tombol Terapkan di bagian bawah Inspector untuk mengunci pengaturan ini ke model yang diimpor dan kemudian lanjutkan mendefinisikan klip animasi. Anda mungkin melihat peringatan (bukan kesalahan) di konsol yang mengatakan "peringatan konversi: spine3 berada di antara transformasi humanoid." Peringatan khusus itu bukanlah alasan untuk khawatir; ini menunjukkan bahwa kerangka dalam model yang diimpor memiliki tulang ekstra di luar kerangka yang diharapkan Mecanim.\

Mendefinisikan klip animasi dalam model yang diimpor

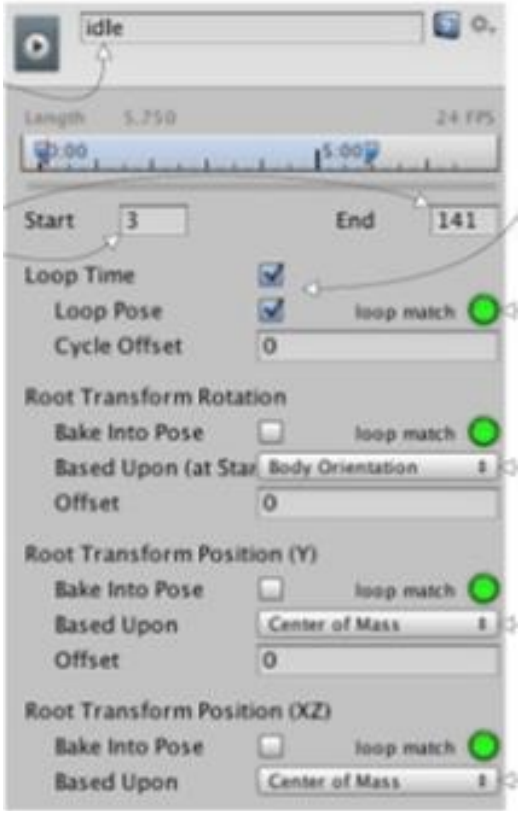
Langkah pertama dalam menyiapkan animasi untuk karakter kita adalah menentukan berbagai klip animasi yang akan dimainkan. Jika Anda berpikir tentang karakter yang hidup, gerakan yang berbeda dapat terjadi pada waktu yang berbeda: terkadang player berlarian, terkadang player melompat di platform, dan terkadang karakter hanya berdiri di sana dengan tangan ke bawah. Masing-masing gerakan ini adalah "klip" terpisah yang dapat dimainkan secara individual.

Seringkali animasi yang diimpor datang sebagai klip panjang tunggal yang dapat dipotong menjadi animasi individu yang lebih pendek. Untuk memisahkan klip animasi, pertama-tama pilih tab Animations di Inspector. Anda akan melihat panel Clips, yang ditunjukkan pada gambar 5.48; ini mencantumkan semua klip animasi yang ditentukan, yang awalnya merupakan satu klip yang diimpor. Anda akan melihat tombol + dan – di bagian bawah daftar; Anda menggunakan tombol ini untuk menambah dan menghapus klip pada daftar. Pada akhirnya kami membutuhkan empat klip untuk karakter ini, jadi tambahkan dan hapus klip seperlunya saat Anda bekerja.



Gambar 5.48 Daftar Klip dalam pengaturan Animasi

Saat Anda memilih klip, informasi tentang klip itu (ditunjukkan pada gambar 5.49) akan muncul di area di bawah daftar. Bagian atas area informasi ini menunjukkan nama klip ini, dan Anda dapat mengetikkan nama baru. Beri nama klip pertama kami yang menganggur. Tentukan bingkai Mulai dan Akhir untuk klip animasi ini; ini memungkinkan Anda untuk memotong sebagian dari animasi impor yang lebih panjang. Untuk animasi idle masukkan Start 3 dan End 141. Selanjutnya adalah pengaturan Loop.



Gambar 5.49 Informasi tentang klip animasi yang dipilih

Loop mengacu pada rekaman yang diputar berulang-ulang. Klip animasi perulangan adalah klip yang diputar lagi dari awal segera setelah pemutaran mencapai akhir. Loop animasi idle, jadi pilih Loop Time dan Loop Pose. Kebetulan, titik indikator hijau memberi tahu Anda saat pose di awal klip cocok dengan pose di akhir untuk pengulangan yang benar; indikator ini berubah menjadi kuning ketika posenya agak tidak aktif, dan berubah menjadi merah ketika pose awal dan akhir benar-benar berbeda.

Di bawah pengaturan Loop adalah serangkaian pengaturan yang terkait dengan transformasi root. Kata root memiliki arti yang sama untuk animasi kerangka seperti halnya untuk hierarki yang terhubung dalam Unity: objek root adalah objek dasar yang terhubung dengan segala sesuatu lainnya. Dengan demikian, akar animasi dapat dianggap sebagai basis

karakter, dan segala sesuatu yang lain bergerak relatif terhadap basis tersebut. Ada beberapa pengaturan berbeda di sini untuk menyiapkan basis itu, dan Anda mungkin ingin bereksperimen di sini saat bekerja dengan animasi Anda sendiri. Namun, untuk tujuan kita, pengaturannya harus Orientasi Tubuh, Pusat Massa, dan Pusat Massa, dalam urutan itu. Sekarang klik Apply dan Anda telah menambahkan klip animasi idle ke karakter Anda. Lakukan hal yang sama untuk dua klip lagi: berjalan dimulai pada bingkai 144 dan berakhir pada 169, dan lari dimulai pada 171 dan berakhir pada 190. Semua pengaturan lainnya harus sama seperti untuk diam karena mereka juga loop animasi.

Klip animasi keempat adalah lompat, dan pengaturan untuk klip itu sedikit berbeda. Pertama, ini bukan loop melainkan pose diam, jadi jangan pilih Loop Time. Atur Start dan End menjadi 190.5 dan 191; ini adalah pose satu bingkai, tetapi Unity mengharuskan Awal dan Akhir berbeda. Preview animasi di bawah ini tidak akan terlihat benar karena angka-angka yang rumit ini, tetapi pose ini akan terlihat bagus di dalam game.

Klik Terapkan untuk mengonfirmasi klip animasi baru, lalu lanjutkan ke langkah berikutnya: membuat controller animasi. Loop animasi idle, jadi pilih Loop Time dan Loop Pose. Kebetulan, titik indikator hijau memberi tahu Anda saat pose di awal klip cocok dengan pose di akhir untuk pengulangan yang benar; indikator ini berubah menjadi kuning ketika posenya agak tidak aktif, dan berubah menjadi merah ketika pose awal dan akhir benar-benar berbeda. Di bawah pengaturan Loop adalah serangkaian pengaturan yang terkait dengan transformasi root. Kata root memiliki arti yang sama untuk animasi kerangka seperti halnya untuk hierarki yang terhubung dalam Unity: objek root adalah objek dasar yang terhubung dengan segala sesuatu lainnya. Dengan demikian, akar animasi dapat dianggap sebagai basis karakter, dan segala sesuatu yang lain bergerak relatif terhadap basis tersebut. Ada beberapa pengaturan berbeda di sini untuk menyiapkan basis itu, dan Anda mungkin ingin bereksperimen di sini saat bekerja dengan animasi Anda sendiri. Namun, untuk tujuan kita, pengaturannya harus Orientasi Tubuh, Pusat Massa, dan Pusat Massa, dalam urutan itu.

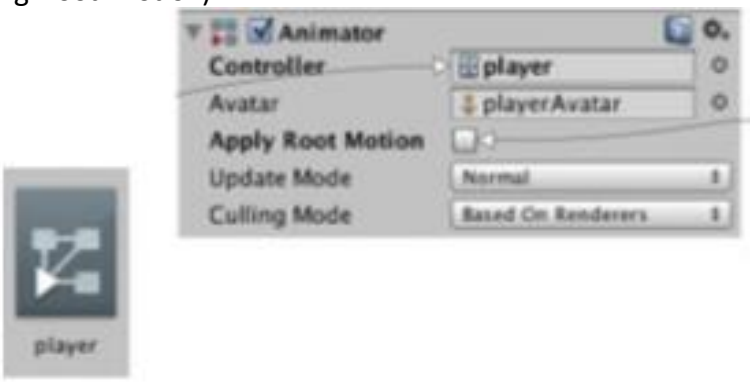
Sekarang klik Apply dan Anda telah menambahkan klip animasi idle ke karakter Anda. Lakukan hal yang sama untuk dua klip lagi: berjalan dimulai pada bingkai 144 dan berakhir pada 169, dan lari dimulai pada 171 dan berakhir pada 190. Semua pengaturan lainnya harus sama seperti untuk diam karena mereka juga loop animasi. Klip animasi keempat adalah lompat, dan pengaturan untuk klip itu sedikit berbeda. Pertama, ini bukan loop melainkan pose diam, jadi jangan pilih Loop Time. Atur Start dan End menjadi 190.5 dan 191; ini adalah pose satu bingkai, tetapi Unity mengharuskan Awal dan Akhir berbeda. Preview animasi di bawah ini tidak akan terlihat benar karena angka-angka yang rumit ini, tetapi pose ini akan terlihat bagus di dalam game. Klik Terapkan untuk mengonfirmasi klip animasi baru, lalu lanjutkan ke langkah berikutnya: membuat controller animasi.

Membuat controller animator untuk animasi ini

Langkah selanjutnya adalah membuat controller animator untuk karakter ini. Langkah ini memungkinkan kita untuk mengatur status animasi dan membuat transisi di antara status tersebut. Berbagai klip animasi diputar selama status animasi yang berbeda, dan kemudian skrip kami akan menyebabkan controller beralih di antara status animasi. Ini mungkin tampak seperti tipuan yang aneh—menempatkan abstraksi controller di antara kode kita dan permainan animasi yang sebenarnya. Anda mungkin akrab dengan sistem di mana Anda langsung memutar animasi dari kode Anda; memang, sistem animasi Legacy yang lama bekerja persis seperti itu, dengan panggilan seperti Play("idle"). Tapi tipuan ini memungkinkan kita untuk berbagi animasi antar model, daripada hanya bisa memainkan animasi yang ada di dalam model ini. Dalam bab ini kita tidak akan memanfaatkan kemampuan ini, tetapi perlu diingat bahwa ini dapat membantu ketika Anda mengerjakan proyek yang lebih besar. Anda

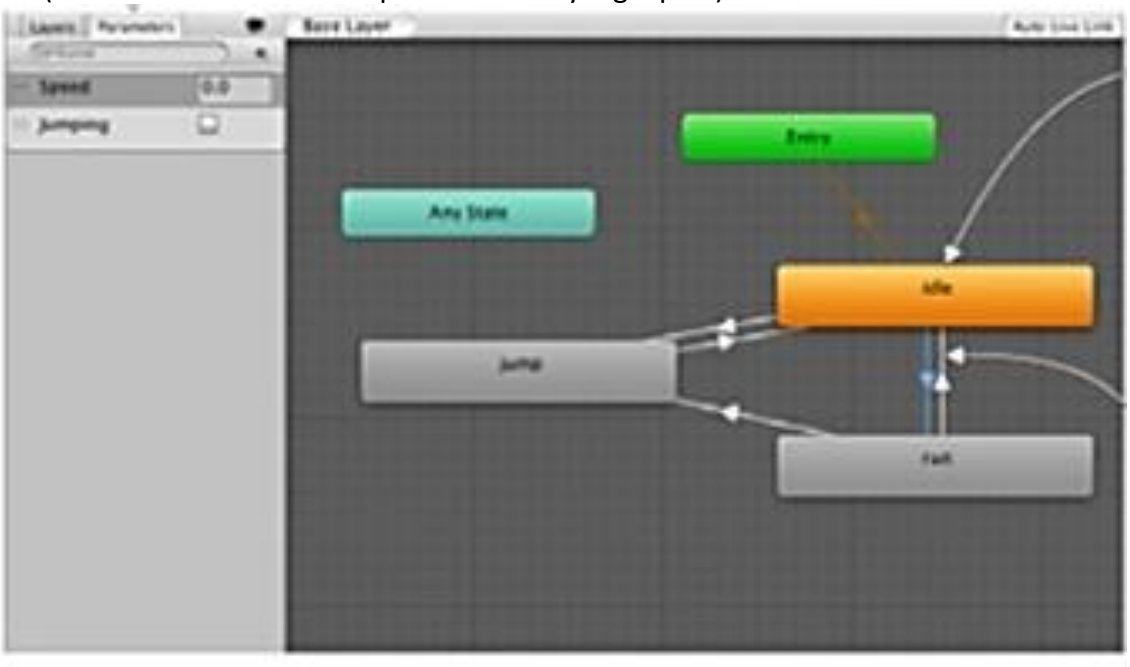
dapat memperoleh animasi dari beberapa sumber, termasuk beberapa animator, atau Anda dapat membeli animasi individual dari toko online (seperti Unity's Asset Store).

Mulailah dengan membuat aset controller animator baru (Aset > Buat> Controller Animator—bukan Animasi, jenis aset yang berbeda). Dalam tampilan Proyek Anda akan melihat ikon dengan jaringan garis yang tampak lucu di atasnya (lihat gambar 5.50); ganti nama aset ini menjadi player. Pilih karakter dalam scene dan Anda akan melihat objek ini memiliki komponen yang disebut Animator; model apa pun yang dapat dianimasikan memiliki komponen ini, selain komponen Transform dan apa pun yang telah Anda tambahkan. Komponen Animator memiliki slot Pengendali bagi Anda untuk menautkan controller animator tertentu, jadi drag dan lepas aset controller baru Anda (dan pastikan untuk menghapus centang Root Motion).



Gambar 5.50 Controller animator dan komponen Animator

Controller animator adalah pohon simpul yang terhubung (dengan demikian ikon pada aset itu) yang dapat Anda lihat dan manipulasi dengan membuka tampilan Animator. Ini adalah tampilan lain seperti Scene atau Project (ditunjukkan pada gambar 7.16) kecuali tampilan ini tidak terbuka secara default. Pilih Animator dari menu Window (berhati-hatilah agar tidak bingung dengan jendela Animation; itu adalah pilihan terpisah dari Animator). Jaringan node yang ditampilkan di sini adalah controller animator mana pun yang saat ini dipilih (atau controller animator pada karakter yang dipilih).



Gambar 5.51 Tampilan Animator dengan controller animator kami yang lengkap

Ingatlah bahwa Anda dapat memindahkan tab di Unity dan memasangnya di mana pun Anda suka untuk mengatur antarmuka. Saya suka memasang Animator tepat di sebelah jendela Scene dan Game. Controller animator adalah pohon simpul yang terhubung (dengan demikian ikon pada aset itu) yang dapat Anda lihat dan buka dengan membuka tampilan Animator. Ini adalah tampilan lain seperti Scene atau Project (ditunjukkan pada gambar 7.16) kecuali tampilan ini tidak terbuka secara default. Pilih Animator dari menu Window (berhati-hatilah agar tidak bingung dengan jendela Animation; itu adalah pilihan terpisah dari Animator). Node jaringan yang ditampilkan di sini adalah controller animator mana pun yang saat ini dipilih (atau controller animator pada karakter yang dipilih).

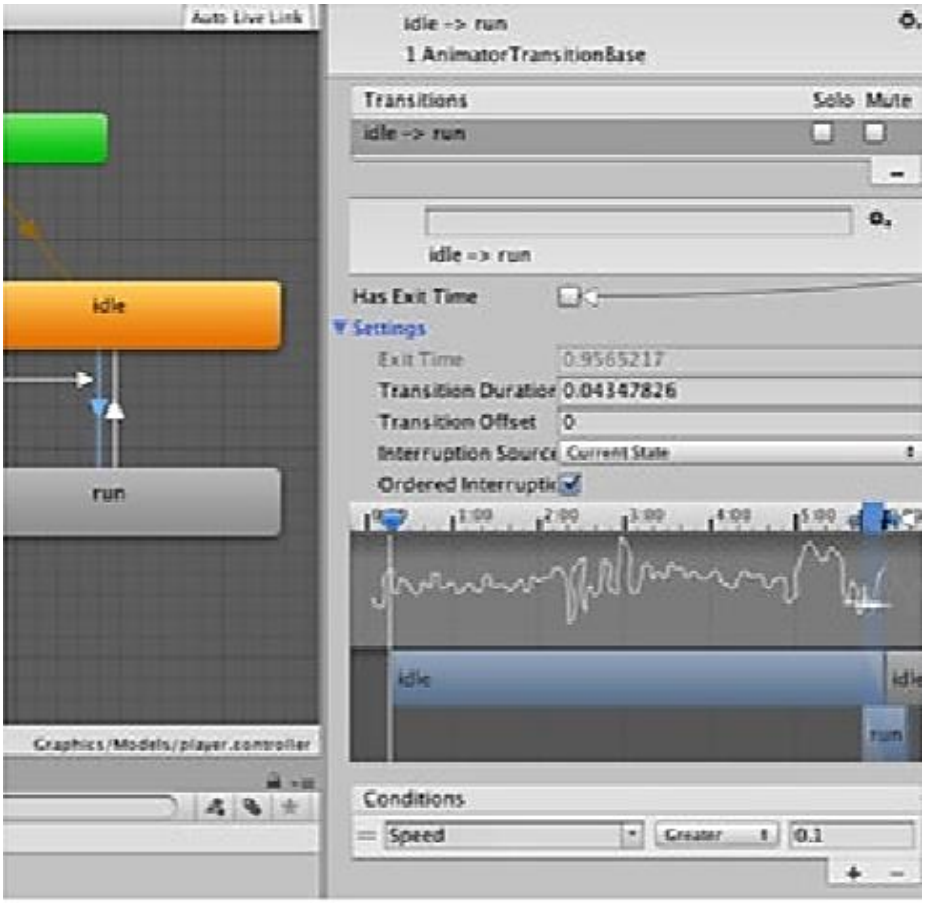
Gambar 5.52 Tampilan Animator dengan controller animator kami yang lengkap

Ingatlah bahwa Anda dapat memindahkan tab di Unity dan memasangnya di mana pun Anda suka untuk mengatur antarmuka. Saya suka memasang Animator tepat di sebelah jendela Scene dan Game.

Klik kanan pada node Idle dan pilih Set As Layer Default State. Node itu akan berubah menjadi oranye sementara node lainnya tetap abu-abu; status animasi default adalah tempat jaringan node dimulai sebelum game membuat perubahan apa pun. Anda harus menautkan node bersama dengan garis yang menunjukkan transisi antara status animasi; klik kanan pada node dan pilih Make Transition untuk mulai menarik panah yang dapat Anda klik pada node lain untuk dihubungkan. Hubungkan node dalam pola (pastikan untuk membuat transisi di kedua arah untuk sebagian besar node, tetapi tidak dari lompat ke lari). Garis transisi ini menentukan bagaimana status animasi terhubung satu sama lain, dan mengontrol perubahan dari satu status ke status lainnya selama permainan.

Saat bekerja dalam tampilan Animator, Anda mungkin melihat kesalahan tentang AnimationStateMachine.TransitionEditionContext.BuildNames. Cukup mulai ulang Unity; ini tampaknya menjadi bug yang tidak berbahaya. Transisi bergantung pada sekumpulan nilai controller, jadi mari buat parameter tersebut. Di kiri atas gambar 7.16 adalah tab yang disebut Parameter; klik itu untuk melihat panel dengan tombol + untuk menambahkan parameter. Tambahkan pelampung yang disebut Kecepatan dan Boolean yang disebut Melompat. Nilai tersebut akan disesuaikan dengan kode kita, dan akan memicu transisi antar status animasi.

Klik pada garis transisi untuk melihat pengaturannya di Inspector. Di sinilah kita akan menyesuaikan bagaimana status animasi berubah ketika parameter berubah. Misalnya, klik transisi Idle-to-Run untuk menyesuaikan kondisi transisi tersebut. Di bawah Kondisi, pilih Kecepatan, Lebih Besar, dan 0.1. Matikan Has Exit Time (itu akan memaksa pemutaran animasi sepenuhnya, sebagai lawan dari memotong segera saat transisi terjadi). Kemudian klik panah di sebelah label Pengaturan untuk melihat seluruh menu itu; transisi lain harus dapat menginterupsi yang satu ini, jadi ubah menu Sumber Interupsi dari Tidak Ada ke Status Saat Ini.



Gambar 5.53 Pengaturan transisi di Inspector

Tabel 5.4 Ketentuan untuk semua transisi di controller animasi ini

Transition	Condition	Interruption
Idle-to-Run	Speed greater than .1	Current State
Run-to-Idle	Speed less than .1	None
Idle-to-Jump	Jumping is true	None
Run-to-Jump	Jumping is true	None
Jump-to-Idle	Jumping is false	None

Selain pengaturan berbasis menu ini, ada antarmuka visual kompleks yang ditunjukkan pada tepat di atas pengaturan Kondisi. Grafik ini memungkinkan Anda untuk menyesuaikan panjang waktu transisi secara visual. Waktu transisi default terlihat baik untuk kedua transisi antara Idle dan Run, tetapi semua transisi ke dan dari Jump harus lebih pendek sehingga karakter akan bergerak lebih cepat di antara animasi lompat. Area grafik yang diarsir menunjukkan berapa lama waktu yang dibutuhkan transisi; untuk melihat lebih detail, gunakan Alt+klik kiri untuk menggeser grafik dan Alt+klik kanan untuk scalingnya (ini adalah kontrol yang sama seperti menavigasi dalam tampilan Scene). Gunakan panah di atas area yang diarsir untuk mengecilkannya hingga di bawah 4 milidetik untuk ketiga transisi Lompat. Terakhir, Anda dapat menyempurnakan jaringan animasi dengan memilih node animasi satu per satu dan menyesuaikan urutan transisi. Inspector akan menampilkan daftar semua transisi ke dan dari node tersebut; Anda dapat menyeret item dalam daftar (gagang dragnya adalah ikon di sisi kiri) untuk menyusun ulang. Pastikan transisi Jump berada di atas untuk node Idle dan Run sehingga transisi Jump memiliki prioritas di atas transisi lainnya. Saat Anda melihat

pengaturan ini, Anda juga dapat mengubah kecepatan pemutaran jika animasi terlihat terlalu lambat (Jalankan terlihat lebih baik pada kecepatan 1,5). Kontroler animasi sudah diatur, jadi sekarang kita bisa mengoperasikan animasi dari skrip gerakan.

Menulis kode yang mengoperasikan animator

Terakhir, Anda akan menambahkan metode ke skrip RelativeMovement. Seperti yang dijelaskan sebelumnya, sebagian besar pekerjaan pengaturan status animasi dilakukan di controller animasi; hanya sedikit kode yang diperlukan untuk mengoperasikan sistem animasi yang kaya dan lancar (lihat daftar berikut).

Daftar Kode untuk mengatur nilai dalam komponen Animator

```
...
private Animator _animator;
...
_animator = GetComponent<Animator>();
...
        _animator.SetFloat("Speed", movement.sqrMagnitude);

        if (hitGround) {
            if (Input.GetButtonDown("Jump")) {
                _vertSpeed = jumpSpeed;
            } else {
                _vertSpeed = -0.1f;
                _animator.SetBool("Jumping", false);
            }
        } else {

            _vertSpeed += gravity * 5 * Time.deltaTime;
            if (_vertSpeed < terminalVelocity) {
                _vertSpeed = terminalVelocity;
            }
            if (_contact != null ) {
                _animator.SetBool("Jumping", true);
            }

            if (_charController.isGrounded) {
                if (Vector3.Dot(movement, _contact.normal) < 0) {
                    movement = _contact.normal * moveSpeed;
                } else {
                    movement += _contact.normal * moveSpeed;
                }
            }
        }
...
```

Sekali lagi, banyak dari daftar ini diulang dari daftar sebelumnya; kode animasi adalah beberapa baris yang diselingi di seluruh skrip gerakan yang ada. Pilih baris _animator untuk menemukan tambahan yang akan dibuat dalam kode Anda.

Script membutuhkan referensi ke komponen Animator, dan kemudian kode menetapkan nilai (baik float atau Boolean) pada animator. Satu-satunya bit kode yang agak tidak jelas adalah kondisi (_contact != null) sebelum menyetel Boolean Jumping. Kondisi tersebut membuat animator tidak bisa memainkan animasi jump sejak awal. Meskipun karakter secara teknis jatuh selama sepersekian detik, tidak akan ada data collision sampai karakter menyentuh tanah untuk pertama kalinya. Dan di sana Anda memilikinya! Sekarang kami memiliki demo gerakan orang ketiga yang bagus, dengan kontrol kamera-relatif dan pemutaran animasi karakter.

Menambahkan perangkat dan item interaktif di dalam game

Menerapkan item fungsional adalah topik berikutnya yang akan kita fokuskan. Bab-bab sebelumnya membahas sejumlah elemen berbeda dari game yang lengkap: gerakan, musuh, antarmuka pengguna, dan sebagainya. Tetapi proyek kami tidak memiliki apa pun untuk berinteraksi dengan selain musuh, mereka juga tidak memiliki banyak hal dalam keadaan permainan. Dalam bab ini, Anda akan mempelajari cara membuat perangkat fungsional seperti pintu. Kami juga akan membahas pengumpulan item, yang melibatkan interaksi dengan objek di level dan melacak status game. Game sering kali harus melacak status seperti statistik player saat ini, kemajuan melalui tujuan, dan sebagainya. Inventaris player adalah contoh dari keadaan semacam ini, jadi Anda akan membangun arsitektur kode yang dapat melacak item yang dikumpulkan oleh player. Di akhir bab ini, Anda akan membangun ruang dinamis yang benar-benar terasa seperti permainan!

Kami akan mulai dengan menjelajahi perangkat (seperti pintu) yang dioperasikan dengan penekanan tombol dari pemutar. Setelah itu, Anda akan menulis kode untuk mendeteksi ketika player bercollision dengan objek di level, memungkinkan interaksi seperti mendorong objek di sekitar atau mengumpulkan item inventaris. Kemudian Anda akan menyiapkan arsitektur kode gaya MVC (Model-View-Controller) yang kuat untuk mengelola data untuk inventaris yang dikumpulkan. Terakhir, Anda akan memprogram antarmuka untuk menggunakan inventaris untuk gameplay, seperti membutuhkan kunci untuk membuka pintu. Bab-bab sebelumnya relatif mandiri dan secara teknis tidak memerlukan proyek dari bab-bab sebelumnya, tetapi kali ini beberapa daftar kode mengedit skrip dari bab 7. Jika Anda langsung melompat ke bab ini, unduh proyek sampel untuk bab 7 di untuk membangun itu. Proyek contoh akan memiliki perangkat dan item ini berserakan di sekitar level secara acak. Gim yang dipoles akan memiliki banyak desain yang cermat di balik penempatan item, tetapi tidak perlu merencanakan level yang hanya menguji fungsionalitas dengan hati-hati. Meski begitu, meskipun penempatan objek akan serampangan, peluru pembuka bab menjelaskan urutan di mana kita akan mengimplementasikan sesuatu. Seperti biasa, penjelasan membangun kode langkah demi langkah, tetapi jika Anda ingin melihat semua kode yang sudah selesai di satu tempat, Anda dapat mengunduh proyek sampel.

5.14 MEMBUAT PINTU DAN PERANGKAT LAINNYA

Meskipun level dalam game sebagian besar terdiri dari dinding statis dan scene, mereka juga biasanya menggabungkan banyak perangkat fungsional juga. Saya berbicara tentang objek yang dapat berinteraksi dan dioperasikan oleh player—hal-hal seperti lampu yang menyala atau kipas yang mulai berputar. Perangkat tertentu dapat sangat bervariasi dan sebagian besar hanya dibatasi oleh imajinasi Anda, tetapi hampir semuanya menggunakan jenis kode yang sama agar player mengaktifkan perangkat. Kami akan menerapkan beberapa contoh dalam bab ini, dan kemudian Anda harus dapat mengadaptasi kode yang sama ini untuk bekerja dengan semua jenis perangkat lain.

Pintu yang membuka dan menutup dengan menekan tombol

Jenis perangkat pertama yang akan kami program adalah pintu yang membuka dan menutup, dan kami akan mulai mengoperasikan pintu dengan menekan tombol. Ada banyak jenis perangkat berbeda yang dapat Anda miliki dalam game, dan banyak cara berbeda untuk mengoperasikan perangkat tersebut. Kami akhirnya akan melihat beberapa variasi, tetapi pintu adalah perangkat interaktif paling umum yang ditemukan dalam permainan, dan menggunakan item dengan penekanan tombol adalah pendekatan paling mudah untuk memulai. Scene memiliki beberapa tempat di mana ada celah di antara dinding, jadi tempatkan objek baru yang menghalangi celah. Saya membuat objek kubus baru dan

kemudian mengatur transformasinya ke Posisi 2.5 1.5 17 dan Scale 5 3 .5, membuat pintu yang ditunjukkan pada gambar 8.1.

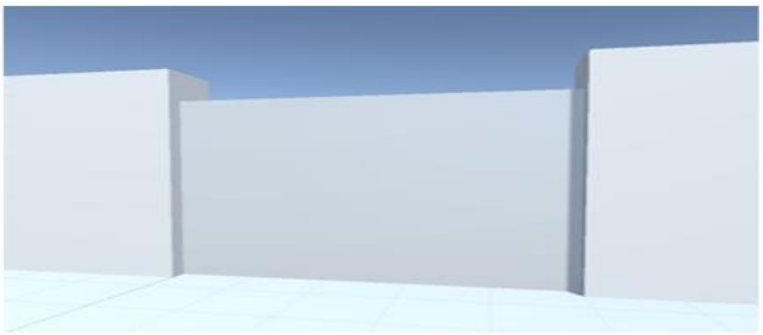
Membuat pintu dan perangkat lainnya

Meskipun level dalam game sebagian besar terdiri dari dinding statis dan scene, mereka juga biasanya menggabungkan banyak perangkat fungsional juga. Saya berbicara tentang objek yang dapat berinteraksi dan dioperasikan oleh player—hal-hal seperti lampu yang menyala atau kipas yang mulai berputar. Perangkat tertentu dapat sangat bervariasi dan sebagian besar hanya dibatasi oleh imajinasi Anda, tetapi hampir semuanya menggunakan jenis kode yang sama agar player mengaktifkan perangkat. Kami akan menerapkan beberapa contoh dalam bab ini, dan kemudian Anda harus dapat mengadaptasi kode yang sama ini untuk bekerja dengan semua jenis perangkat lain.

Pintu yang membuka dan menutup dengan menekan tombol

Jenis perangkat pertama yang akan kami program adalah pintu yang membuka dan menutup, dan kami akan mulai mengoperasikan pintu dengan menekan tombol. Ada banyak jenis perangkat berbeda yang dapat Anda miliki dalam game, dan banyak cara berbeda untuk mengoperasikan perangkat tersebut. Kami akhirnya akan melihat beberapa variasi, tetapi pintu adalah perangkat interaktif paling umum yang ditemukan dalam permainan, dan menggunakan item dengan penekanan tombol adalah pendekatan paling mudah untuk memulai.

Scene memiliki beberapa tempat di mana ada celah di antara dinding, jadi tempatkan objek baru yang menghalangi celah. Saya membuat objek kubus baru dan kemudian mengatur transformasinya ke Posisi 2.5 1.5 17 dan Scale 5 3 .5



Gambar 5.54 Scene celah di antara dinding

Buat skrip C#, beri nama DoorOpenDevice, dan letakkan skrip itu di objek pintu. Kode ini (ditunjukkan dalam daftar berikutnya) akan menyebabkan objek beroperasi sebagai pintu.

Daftar Script yang membuka dan menutup pintu atas perintah

```
using UnityEngine;
using System.Collections;

public class DoorOpenDevice : MonoBehaviour {
    [SerializeField] private Vector3 dPos;

    private bool _open;

    public void Operate() {
        if (_open) {
            Vector3 pos = transform.position - dPos;
            transform.position = pos;
        } else {
            Vector3 pos = transform.position + dPos;
            transform.position = pos;
        }
        _open = !_open;
    }
}
```

Variabel pertama mendefinisikan offset yang diterapkan saat pintu terbuka. Pintu akan memindahkan jumlah ini saat dibuka, dan kemudian akan mengurangi jumlah ini saat ditutup. Variabel kedua adalah Boolean personal untuk melacak apakah pintu terbuka atau tertutup. Dalam metode Operate(), transformasi objek diatur ke posisi baru, menambah atau mengurangi offset tergantung pada apakah pintu sudah terbuka; kemudian _open diaktifkan atau dinonaktifkan.

Seperti variabel serial lainnya, dPos muncul di Inspector. Tapi ini adalah nilai Vector3, jadi alih-alih satu kotak input ada tiga, semuanya di bawah satu nama variabel. Ketik posisi relatif pintu saat terbuka; Saya memutuskan untuk membuka pintu geser ke bawah, jadi offsetnya adalah 0 -2,9 0 (karena objek pintu memiliki ketinggian 3, bergerak ke bawah 2,9 hanya menyisakan sepotong kecil pintu yang mencuat dari lantai).

Transformasi diterapkan secara instan, tetapi Anda mungkin lebih suka melihat gerakan saat pintu terbuka. Seperti yang disebutkan kembali di bab 3, Anda dapat menggunakan tween untuk membuat objek bergerak dengan lancar seiring waktu. Kata tween berarti hal yang berbeda dalam konteks yang berbeda, tetapi dalam pemrograman game ini mengacu pada perintah kode yang menyebabkan objek bergerak; lampiran D menyebutkan iTween, salah satu sistem tweening yang bagus untuk Unity. Sekarang kode lain perlu memanggil Operate() untuk membuat pintu terbuka dan tertutup (panggilan fungsi tunggal menangani kedua kasus). Kami belum memiliki skrip lain di pemutar; menulis itu adalah langkah selanjutnya.

Memeriksa jarak dan menghadap sebelum membuka pintu

Buat skrip baru dan beri nama DeviceOperator. Daftar berikut mengimplementasikan kunci kontrol yang mengoperasikan perangkat di sekitar.

Daftar Kunci kontrol perangkat untuk player

```
using UnityEngine;
using System.Collections;

public class DeviceOperator : MonoBehaviour {
  public float radius = 1.5f;

  void Update() {
    if (Input.GetButtonDown("Fire3")) {
      Collider[] hitColliders =
            Physics.OverlapSphere(transform.position, radius);
      foreach (Collider hitCollider in hitColliders) {
        hitCollider.SendMessage("Operate",
            SendMessageOptions.DontRequireReceiver);
      }
    }
  }
}
```

Mayoritas skrip dalam daftar ini akan terlihat familier, tetapi metode baru yang penting ada di tengah kode ini. Pertama, tetapkan nilai seberapa jauh untuk mengoperasikan perangkat. Kemudian, di fungsi Update(), cari input keyboard; karena tombol Jump sudah digunakan oleh skrip RelativeMovement, kali ini kita akan merespons Fire3 (yang didefinisikan dalam pengaturan input proyek sebagai tombol Command kiri). Sekarang kita sampai pada metode baru yang penting: OverlapSphere(). Metode ini mengembalikan larik semua objek yang berada dalam jarak tertentu dari posisi tertentu. Dengan melewati posisi player dan variabel radius, ini mendeteksi semua objek di dekat player. Apa yang sebenarnya Anda lakukan dengan daftar ini dapat bervariasi (misalnya, mungkin Anda baru saja meledakkan

bom dan ingin menerapkan kekuatan ledakan), tetapi dalam situasi ini kami ingin mencoba memanggil Operate() pada semua objek terdekat.

Metode itu dipanggil melalui SendMessage() alih-alih notasi titik biasa, sebuah pendekatan yang juga Anda lihat dengan tombol UI di bab sebelumnya. Seperti yang terjadi di sana, alasan untuk menggunakan SendMessage() adalah karena kami tidak mengetahui jenis objek target yang tepat dan perintah itu berfungsi di semua GameObjects. Tapi kali ini kita akan memberikan opsi DontRequireReceiver ke metode. Ini karena sebagian besar objek yang dikembalikan oleh OverlapSphere() tidak akan memiliki metode Operate(); biasanya SendMessage() mencetak pesan kesalahan jika tidak ada objek yang menerima pesan tersebut, tetapi dalam kasus ini pesan kesalahan akan mengganggu karena kita sudah tahu sebagian besar objek akan mengabaikan pesan tersebut.

Setelah kode ditulis, Anda dapat melampirkan skrip ini ke objek pemutar. Sekarang Anda dapat membuka dan menutup pintu dengan berdiri di dekatnya dan menekan tombol. Ada satu detail kecil yang bisa kami perbaiki. Saat ini tidak masalah ke arah mana player menghadap, selama player cukup dekat. Tetapi kami juga dapat menyesuaikan skrip untuk hanya mengoperasikan perangkat yang dihadapi player, jadi ayo lakukan itu. Ingat kembali dari bab 7 bahwa Anda dapat menghitung perkalian titik untuk pemeriksaan hadap. Itu adalah operasi matematika yang dilakukan pada sepasang vektor yang mengembalikan rentang antara -1 dan 1, dengan 1 artinya mereka menunjuk ke arah yang sama persis dan -1 ketika mereka menunjuk ke arah yang berlawanan. Daftar berikutnya menunjukkan kode baru dalam skrip DeviceOperator.

Daftar Menyesuaikan DeviceOperator untuk hanya mengoperasikan perangkat yang dihadapi player

```
...
foreach (Collider hitCollider in hitColliders) {
   Vector3 direction = hitCollider.transform.position - transform.position;
   if (Vector3.Dot(transform.forward, direction) > .5f) {
      hitCollider.SendMessage("Operate",
            SendMessageOptions.DontRequireReceiver);
   }
}
...
```

Untuk menggunakan produk titik, pertama-tama kita menentukan arah yang akan diperiksa. Itu akan menjadi arah dari player ke objek; membuat vektor arah dengan mengurangkan posisi player dari posisi objek. Kemudian panggil Vector3.Dot() dengan vektor arah itu dan arah maju pemutar. Ketika produk titik mendekati 1 (khususnya, kode ini memeriksa lebih besar dari .5), itu berarti kedua vektor dekat dengan menunjuk ke arah yang sama.

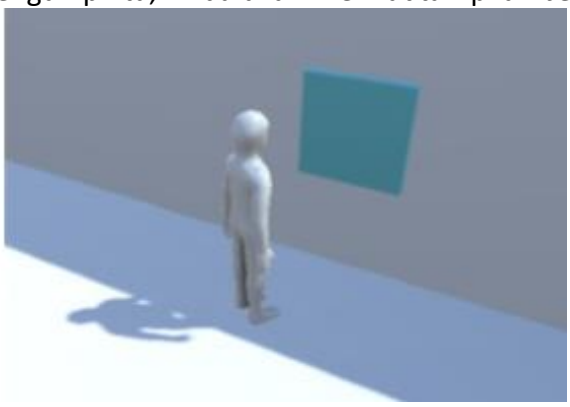
Dengan penyesuaian ini, pintu tidak akan terbuka dan tertutup ketika player menghadap jauh darinya, bahkan jika player dekat. Dan pendekatan yang sama untuk perangkat operasi ini dapat digunakan dengan perangkat apa pun. Untuk menunjukkan fleksibilitas itu, mari buat perangkat contoh lain.

Mengoperasikan monitor yang berubah warna

Kami telah membuat pintu yang membuka dan menutup, tetapi logika pengoperasian perangkat yang sama dapat digunakan dengan perangkat apa pun. Kami akan membuat perangkat lain yang dioperasikan dengan cara yang sama; kali ini, kita akan membuat tampilan yang berubah warna di dinding.

Buat kubus baru dan letakkan sehingga satu sisi hampir tidak mencuat dari dinding. Misalnya, saya memilih Posisi 10.9 1.5 -5. Sekarang buat skrip baru bernama ColorChangeDevice dan lampirkan skrip itu (ditunjukkan dalam daftar berikutnya) ke tampilan

dinding. Sekarang jalankan ke monitor dinding dan tekan tombol "operasikan" yang sama seperti yang digunakan dengan pintu; Anda akan melihat tampilan berubah warna



Gambar 5.55 Tampilan berubah warna tertanam di dinding

Untuk memulainya, nyatakan nama fungsi yang sama dengan skrip pintu yang digunakan. "Operate" adalah nama fungsi yang digunakan skrip operator perangkat, jadi kita perlu menggunakan nama itu agar dapat dipicu. Di dalam fungsi ini, kode memberikan warna acak pada material objek (ingat, warna bukanlah atribut dari objek itu sendiri, melainkan objek memiliki material dan material tersebut dapat memiliki warna). Meskipun warna didefinisikan dengan komponen Merah, Biru, dan Hijau seperti standar di sebagian besar grafik komputer, nilai dalam objek Warna Unity bervariasi antara 0 dan 1, bukan 0 dan 255, seperti yang umum di sebagian besar tempat (termasuk pemilih warna Unity UI). Baiklah, jadi kami telah membahas satu pendekatan untuk berinteraksi dengan perangkat dalam game dan bahkan telah menerapkan beberapa perangkat berbeda untuk didemonstrasikan. Cara lain untuk berinteraksi dengan item adalah dengan menabraknya, jadi mari kita bahas selanjutnya.

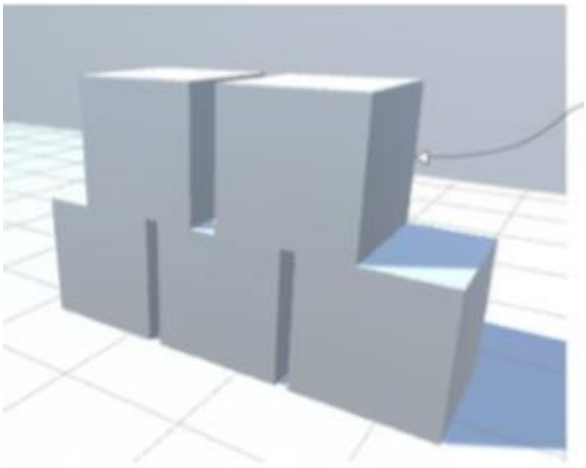
Berinteraksi dengan objek dengan menabraknya

Di bagian sebelumnya, perangkat dioperasikan oleh input keyboard dari player, tetapi itu bukan satu-satunya cara player dapat berinteraksi dengan item di level. Pendekatan lain yang sangat mudah adalah menanggapi collision dengan player. Unity menangani sebagian besar dari itu untuk Anda, dengan memiliki deteksi collision dan fisika yang dibangun ke dalam game engine. Unity akan mendeteksi collision untuk Anda, tetapi Anda masih perlu memprogram objek untuk merespons.

Bercollision dengan rintangan yang mendukung fisika

Untuk memulai, kita akan membuat tumpukan kotak dan kemudian menyebabkan tumpukan itu runtuh saat player menabraknya. Meskipun perhitungan fisika yang terlibat rumit, Unity memiliki semua yang ada di dalamnya dan akan menyebarkan kotak-kotak itu dengan cara yang realistis bagi kita. Secara default, Unity tidak menggunakan simulasi fisikanya untuk memindahkan objek. Itu dapat diaktifkan dengan menambahkan komponen Rigidbody ke objek. Konsep ini pertama kali dibahas kembali di chapter 3, karena bola api musuh juga membutuhkan komponen Rigidbody. Seperti yang saya jelaskan di bab itu, sistem fisika Unity hanya akan bekerja pada objek yang memiliki komponen Rigidbody. Klik Add Component dan cari Rigidbody di bawah menu Physics.

Buat objek kubus baru dan kemudian tambahkan komponen Rigidbody ke dalamnya. Buat beberapa kubus seperti itu dan posisikan dalam tumpukan yang rapi. Misalnya, dalam contoh unduhan saya membuat lima kotak dan menumpuknya menjadi dua tingkat (lihat gambar 5.56).



Gambar 5.56 Tumpukan lima kotak untuk bercollision

Kotak sekarang siap untuk bereaksi terhadap kekuatan fisika. Agar player menerapkan kekuatan ke kotak, buat tambahan kecil yang ditunjukkan dalam daftar berikut ke skrip RelativeMovement (ini adalah salah satu skrip yang ditulis di bab sebelumnya) yang ada di pemutar.

Daftar Menambahkan kekuatan fisika ke skrip RelativeMovement

```
...
public float pushForce = 3.0f;
...
void OnControllerColliderHit(ControllerColliderHit hit) {
   _contact = hit;

   Rigidbody body = hit.collider.attachedRigidbody;
   if (body != null && !body.isKinematic) {
      body.velocity = hit.moveDirection * pushForce;
   }
}
...
```

Tidak banyak yang bisa dijelaskan tentang kode ini: setiap kali player bercollision dengan sesuatu, periksa apakah objek yang bercollision memiliki komponen Rigidbody. Jika demikian, terapkan kecepatan pada Rigidbody itu.

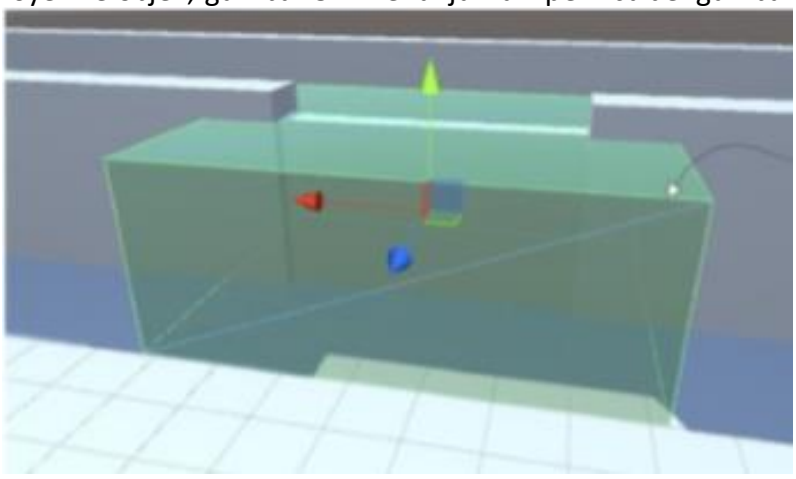
Mainkan gamenya lalu lari ke tumpukan kotak; Anda akan melihat mereka tersebar secara realistis. Dan hanya itu yang harus Anda lakukan untuk mengaktifkan simulasi fisika pada tumpukan kotak di scene! Unity memiliki simulasi fisika bawaan, jadi kami tidak perlu menulis banyak kode. Simulasi itu dapat menyebabkan objek bergerak sebagai respons terhadap collision, tetapi respons lain yang mungkin adalah memicu peristiwa pemicu, jadi mari gunakan peristiwa pemicu tersebut untuk mengontrol pintu.

Memicu pintu dengan pelat tekanan

Jika sebelumnya pintu dioperasikan dengan menekan tombol, kali ini pintu akan membuka dan menutup sebagai respons terhadap karakter yang bercollision dengan objek lain di scene. Buat pintu lain dan letakkan di celah dinding lain (saya menggandakan pintu sebelumnya dan memindahkan pintu baru ke -2,5 1,5 -17). Sekarang buat kubus baru untuk digunakan sebagai objek pemicu, dan pilih kotak centang Apakah Pemicu untuk bercollision (langkah ini diilustrasikan saat membuat bola api di bab 3). Selain itu, atur objek ke layer Ignore Raycast; sudut kanan atas Inspector memiliki menu Layer. Terakhir, Anda harus mematikan casting bayangan dari objek ini (ingat, pengaturan ini berada di bawah Mesh Renderer saat Anda memilih objek).

Langkah-langkah kecil ini mudah dilewatkan tetapi sangat penting: untuk menggunakan objek sebagai pemicu, pastikan untuk mengaktifkan Is Trigger. Di Inspector, cari kotak centang di komponen Collider. Juga, ubah layer menjadi Abaikan Raycast sehingga objek pemicu tidak muncul di raycasting. Saat objek pemicu pertama kali diperkenalkan di bab 3, objek tersebut perlu memiliki komponen Rigidbody yang ditambahkan. Rigidbody tidak diperlukan untuk pelatuk kali ini karena pelatuk akan merespons player (dibandingkan bercollision dengan dinding, situasi sebelumnya). Agar pemicu berfungsi, pemicu atau objek yang memasuki pemicu harus mengaktifkan sistem fisika Unity; komponen Rigidbody memenuhi persyaratan ini, tetapi begitu juga CharacterController player.

Posisikan dan scalekan objek pemicu sehingga keduanya mencakup pintu dan mengelilingi area di sekitar pintu; Saya menggunakan Posisi -2.5 1.5 -17 (sama seperti pintu) dan Scale 7.5 3 6. Selain itu, Anda mungkin ingin menetapkan bahan semitransparan ke objek sehingga Anda dapat membedakan volume pemicu dari objek padat secara visual. Buat materi baru menggunakan menu Aset, dan pilih materi baru di tampilan Proyek. Melihat Inspector, pengaturan teratas adalah Mode Rendering (saat ini disetel ke nilai default Buram); pilih Transparan di menu ini. Sekarang klik swatch warnanya untuk membuka jendela Color Picker. Pilih hijau di bagian utama jendela, dan turunkan alfa menggunakan penggeser bawah. Drag materi ini dari Proyek ke objek; gambar 8.4 menunjukkan pemicu dengan bahan ini.



Gambar 5.57 Volume pemicu di sekitar pintu yang akan dipicu

Pemicu sering disebut sebagai volume daripada objek untuk membedakan objek padat secara konseptual dari objek yang dapat Anda lewati

Mainkan gamenya sekarang dan Anda dapat dengan bebas bergerak melalui volume pemicu; Unity masih mencatat collision dengan objek, tetapi collision itu tidak lagi memengaruhi gerakan player. Untuk bereaksi terhadap collision, kita perlu menulis kode. Secara khusus, kami ingin pemicu ini mengontrol pintu. Buat skrip baru bernama DeviceTrigger (lihat daftar berikut).

Daftar Kode untuk pemicu yang mengontrol perangkat

```
using UnityEngine;
using System.Collections;

public class DeviceTrigger : MonoBehaviour {
   [SerializeField] private GameObject[] targets;

   void OnTriggerEnter(Collider other) {
      foreach (GameObject target in targets) {
         target.SendMessage("Activate");
      }
   }

   void OnTriggerExit(Collider other) {
      foreach (GameObject target in targets) {
         target.SendMessage("Deactivate");
      }
   }
}
```

Mainkan gamenya sekarang dan Anda dapat dengan bebas bergerak melalui volume pemicu; Unity masih mencatat collision dengan objek, tetapi collision itu tidak lagi memengaruhi gerakan player. Untuk bereaksi terhadap collision, kita perlu menulis kode. Secara khusus, kami ingin pemicu ini mengontrol pintu. Buat skrip baru bernama DeviceTrigger (lihat daftar berikut).

Daftar Kode untuk pemicu yang mengontrol perangkat

```
...
public void Activate() {
   if (!_open) {
      Vector3 pos = transform.position + dPos;
      transform.position = pos;
      _open = true;
   }
}
public void Deactivate() {
   if (_open) {
      Vector3 pos = transform.position - dPos;
      transform.position = pos;
      _open = false;
   }
}
...
```

Metode Activate() dan Deactivate() yang baru memiliki kode yang hampir sama dengan metode Operate() dari sebelumnya, kecuali sekarang ada fungsi terpisah untuk membuka dan menutup pintu alih-alih hanya satu fungsi yang menangani kedua kasus. Dengan semua kode yang diperlukan, Anda sekarang dapat menggunakan volume pemicu untuk membuka dan menutup pintu. Letakkan skrip DeviceTrigger pada volume pemicu dan kemudian tautkan pintu ke properti target skrip itu; di Inspector, pertama-tama atur ukuran larik lalu drag objek dari tampilan Hierarki ke slot dalam larik target. Karena kita hanya memiliki satu pintu yang ingin kita kendalikan dengan pemicu ini, ketik 1 di bidang Ukuran array dan kemudian drag pintu itu ke slot target.

Setelah semua ini selesai, mainkan permainan dan perhatikan apa yang terjadi pada pintu ketika player berjalan menuju dan menjauh darinya. Ini akan membuka dan menutup secara otomatis saat player masuk dan keluar dari volume pemicu.

Itu cara hebat lainnya untuk menempatkan interaktivitas ke level! Tetapi pendekatan volume pemicu ini tidak hanya berfungsi dengan perangkat seperti pintu; Anda juga dapat menggunakan pendekatan ini untuk membuat barang koleksi.

Mengumpulkan item yang tersebar di sekitar level

Banyak game menyertakan item yang dapat diambil oleh player. Item ini termasuk peralatan, paket kesehatan, dan power-up. Mekanisme dasar bercollision dengan item untuk mengambilnya sederhana; sebagian besar hal rumit terjadi setelah barang diambil, tetapi kita akan membahasnya nanti.

Buat objek bola dan letakkan melayang setinggi pinggang di area terbuka scene. Buat objeknya kecil, seperti Scale .5 .5 .5, tetapi siapkan seperti yang Anda lakukan dengan volume pemicu yang besar. Pilih pengaturan Is Trigger di Collinder, atur objek ke layer Ignore Raycast, lalu buat material baru untuk memberi objek warna yang berbeda. Karena objeknya kecil, Anda tidak ingin membuatnya semitransparan kali ini, jadi jangan turunkan penggeser alfa sama sekali. Juga, seperti yang disebutkan dalam bab 7, ada pengaturan untuk menghilangkan bayangan yang dilemparkan dari objek ini; apakah akan menggunakan bayangan atau tidak adalah keputusan penilaian, tetapi untuk item pickup kecil seperti ini saya lebih suka mematikannya. Sekarang objek dalam scene sudah siap, buat skrip baru untuk dilampirkan ke objek itu. Panggil skrip CollectibleItem (lihat daftar berikut).

Daftar Skrip yang membuat item menghapus dirinya sendiri saat bersentuhan dengan player

```
using UnityEngine;
using System.Collections;

public class CollectibleItem : MonoBehaviour {
   [SerializeField] private string itemName;

   void OnTriggerEnter(Collider other) {
      Debug.Log("Item collected: " + itemName);
      Destroy(this.gameObject);
   }
}
```

Script ini sangat pendek dan sederhana. Beri item tersebut nilai nama sehingga bisa ada item yang berbeda di scene. OnTriggerEnter() menghancurkan dirinya sendiri. Ada juga pesan debug yang dicetak ke konsol; akhirnya akan diganti dengan kode yang berguna. Pastikan untuk memanggil Destroy() di this.gameObject dan bukan ini! Jangan bingung antara keduanya; this hanya merujuk ke komponen skrip ini, sedangkan this.gameObject merujuk ke objek yang dilampirkan skrip.

Kembali ke Unity, variabel yang Anda tambahkan ke kode akan terlihat di Inspector. Ketik nama untuk mengidentifikasi item ini; Aku pergi dengan energi untuk item pertama saya. Kemudian duplikat item tersebut beberapa kali dan ubah nama salinannya; Saya juga membuat bijih, kesehatan, dan kunci (nama-nama ini harus tepat karena mereka akan digunakan dalam kode nanti). Juga buat bahan terpisah untuk setiap item untuk memberi mereka warna berbeda: Saya melakukan energi biru muda, bijih abu-abu gelap, kesehatan merah muda, dan kunci kuning. Daripada nama seperti yang telah kami lakukan di sini, item dalam game yang lebih kompleks sering kali memiliki pengenal yang digunakan untuk mencari data lebih lanjut. Misalnya, satu item mungkin diberi id 301, dan id 301 berkorelasi dengan nama tampilan, gambar, deskripsi, dan sebagainya.

Nama objek akan berubah menjadi biru di daftar Hierarchy; nama biru menunjukkan objek yang merupakan contoh dari prefab. Klik kanan instance prefab untuk memilih Select Prefab dan pilih prefab yang menjadi objeknya. Tarik keluar contoh prefab dan letakkan item

di area terbuka di level tersebut; bahkan menyeret beberapa salinan dari item yang sama untuk diuji. Mainkan game dan temukan item untuk "mengumpulkan" mereka. Itu cukup rapi, tetapi saat ini tidak ada yang terjadi ketika Anda mengumpulkan item. Kami akan mulai melacak item yang dikumpulkan; untuk melakukan itu, kita perlu menyiapkan struktur kode inventaris.

Mengelola data inventaris dan status game

Sekarang setelah kami memprogram fitur pengumpulan item, kami memerlukan pengelola data latar belakang (mirip dengan pola pengkodean web) untuk inventaris game. Kode yang akan kita tulis akan mirip dengan arsitektur MVC di balik banyak aplikasi web. Keuntungannya adalah memisahkan penyimpanan data dari objek yang ditampilkan di layar, memungkinkan eksperimen dan developeran berulang yang lebih mudah. Meskipun data dan/atau tampilannya kompleks, perubahan di satu bagian aplikasi tidak memengaruhi bagian aplikasi lainnya.

Yang mengatakan, struktur seperti itu sangat bervariasi di antara permainan yang berbeda. Tidak setiap game memiliki kebutuhan manajemen data yang sama, jadi tidak masuk akal bagi Unity untuk menerapkan aturan bahwa “Setiap game harus menggunakan pola desain ini dan itu.” Akan menjadi kontraproduktif untuk memperkenalkan konsep semacam itu terlalu cepat, karena orang akan disesatkan untuk berpikir bahwa mereka membutuhkannya sebelum mereka dapat membuat game apa pun.

Misalnya, permainan peran akan memiliki kebutuhan manajemen data yang sangat tinggi, jadi Anda mungkin ingin mengimplementasikan sesuatu seperti arsitektur MVC. Namun, permainan puzzle memiliki sedikit data untuk dikelola, jadi membangun struktur terpisah yang kompleks dari pengelola data akan berlebihan. Sebagai gantinya, status game dapat dilacak di objek controller khusus scene (memang, begitulah cara kami menangani status game di bab sebelumnya). Dalam proyek ini kita perlu mengelola inventaris player. Mari kita siapkan struktur kode untuk itu.

5.15 MENYIAPKAN PLAYER DAN MANAJER INVENTARIS

Ide umum di sini adalah untuk membagi semua manajemen data menjadi modul terpisah yang terdefinisi dengan baik yang masing-masing mengelola area tanggung jawabnya sendiri. Kami akan membuat modul terpisah untuk mempertahankan status player di PlayerManager (hal-hal seperti kesehatan player) dan memelihara daftar inventaris di InventoryManager. Manajer data ini akan berperilaku seperti Model di MVC; Controller adalah objek tak terlihat di sebagian besar scene (tidak diperlukan di sini, tapi ingat SceneController di bab sebelumnya), dan sisa scene analog dengan View.

Akan ada "manajer manajer" tingkat yang lebih tinggi yang melacak semua modul terpisah. Selain menyimpan daftar semua berbagai manajer, manajer tingkat tinggi ini akan mengontrol siklus hidup berbagai manajer, terutama menginisialisasi mereka di awal. Semua skrip lain dalam game akan dapat mengakses modul terpusat ini melalui manajer utama. Secara khusus, kode lain dapat menggunakan sejumlah properti statis di manajer utama untuk terhubung dengan modul tertentu yang diinginkan.

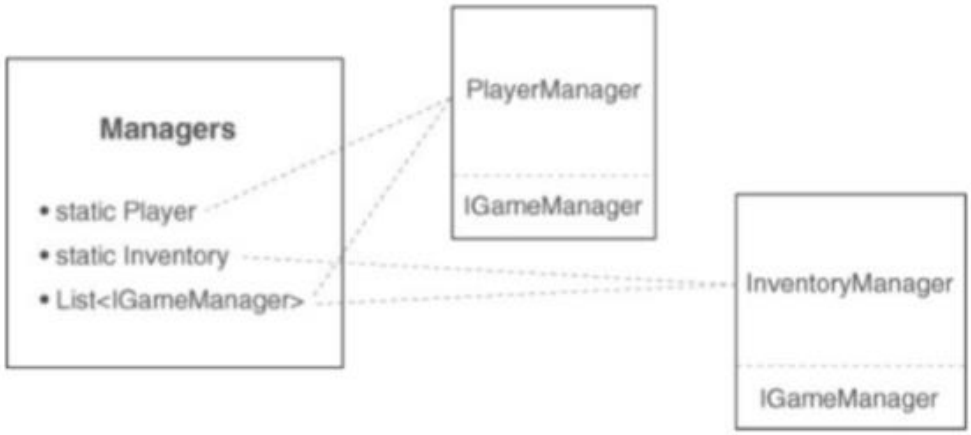
Pola desain untuk mengakses modul bersama yang terpusat

Selama bertahun-tahun berbagai pola desain telah muncul untuk memecahkan masalah menghubungkan bagian-bagian dari program ke modul terpusat yang dibagikan di seluruh program. Misalnya, pola Singleton diabadikan dalam buku asli “Gang of Four” tentang pola desain. Tetapi pola itu tidak disukai oleh banyak insinyur software, sehingga mereka menggunakan pola alternatif seperti pencari layanan dan injeksi ketergantungan. Dalam kode

saya, saya menggunakan kompromi antara kesederhanaan variabel statis dan fleksibilitas pencari layanan.

Desain ini membuat kode mudah digunakan sementara juga memungkinkan untuk bertukar modul yang berbeda. Misalnya, meminta InventoryManager menggunakan singleton akan selalu merujuk ke kelas yang sama persis dan dengan demikian akan memasangkan kode Anda dengan kelas itu secara erat; di sisi lain, meminta Inventaris dari pencari layanan meninggalkan opsi untuk mengembalikan Inventory Manager atau Different Inventory Manager. Terkadang berguna untuk dapat beralih di antara sejumlah versi yang sedikit berbeda dari modul yang sama (misalnya, menggunakan game pada platform yang berbeda).

Agar manajer utama dapat mereferensikan modul lain secara konsisten, semua modul ini harus mewarisi properti dari basis yang sama. Kami akan melakukannya dengan antarmuka; banyak bahasa pemrograman (termasuk C#) memungkinkan Anda untuk menentukan semacam cetak biru yang harus diikuti oleh kelas lain. Baik PlayerManager dan InventoryManager akan mengimplementasikan antarmuka umum (dalam kasus ini disebut IGameManager) dan kemudian objek Manajer utama dapat memperlakukan PlayerManager dan InventoryManager sebagai tipe IGameManager. Gambar 8.5 mengilustrasikan pengaturan yang saya jelaskan.



Gambar 5.58 Diagram dari berbagai modul dan bagaimana mereka terkait

Kebetulan, sementara semua arsitektur kode yang saya bicarakan terdiri dari modul tak terlihat yang ada di latar belakang, Unity masih membutuhkan skrip untuk ditautkan ke objek di scene untuk menjalankan kode itu. Seperti yang telah kita lakukan dengan controller khusus scene di proyek sebelumnya, kita akan membuat GameObject kosong untuk menautkan pengelola data ini.

5.16 MEMPROGRAM MANAJER GAME

Baiklah, jadi itu menjelaskan semua konsep di balik apa yang akan kita lakukan; saatnya menulis kode. Untuk memulainya, buat skrip baru bernama IGameManager (lihat daftar berikutnya).

Daftar Antarmuka dasar yang akan diterapkan oleh manajer data

```
public interface IGameManager {
    ManagerStatus status {get;}

    void Startup();
}
```

Hmm, hampir tidak ada kode dalam file ini. Perhatikan bahwa itu bahkan tidak mewarisi dari MonoBehaviour; sebuah antarmuka tidak melakukan apa pun sendiri dan hanya ada untuk memaksakan struktur pada kelas lain. Antarmuka ini mendeklarasikan satu properti (variabel yang memiliki fungsi pengambil) dan satu metode; keduanya perlu diimplementasikan di kelas mana pun yang mengimplementasikan antarmuka ini. Properti status memberi tahu sisa kode apakah modul ini telah menyelesaikan inisialisasinya. Tujuan Startup() adalah untuk menangani inisialisasi manajer, sehingga tugas inisialisasi terjadi di sana dan fungsi menetapkan status manajer. Perhatikan bahwa properti bertipe ManagerStatus; itu enum yang belum kami tulis, jadi buat skrip ManagerStatus.cs (lihat daftar berikutnya).

Daftar ManagerStatus: kemungkinan status untuk status IGameManager

```
public enum ManagerStatus {
Shutdown,
Initializing,
Started
}
```

Ini adalah file lain dengan hampir tidak ada kode di dalamnya. Kali ini kami membuat daftar kemungkinan status berbeda yang dapat diikuti oleh manajer, sehingga menegakkan bahwa properti status akan selalu menjadi salah satu dari nilai yang tercantum ini.

Listing InventoryManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class InventoryManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   public void Startup() {
      Debug.Log("Inventory manager starting...");
      status = ManagerStatus.Started;
   }
}
```

Mencantumkan PlayerManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class PlayerManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   public int health {get; private set;}
   public int maxHealth {get; private set;}

   public void Startup() {
      Debug.Log("Player manager starting...");

      health = 50;
      maxHealth = 100;


      status = ManagerStatus.Started;
   }

   public void ChangeHealth(int value) {
      health += value;
      if (health > maxHealth) {
         health = maxHealth;
      } else if (health < 0) {
         health = 0;
      }

      Debug.Log("Health: " + health + "/" + maxHealth);
   }
}
```

Untuk saat ini, InventoryManager adalah shell yang akan diisi nanti, sedangkan PlayerManager memiliki semua fungsi yang diperlukan untuk proyek ini. Manajer ini mewarisi dari kelas MonoBehaviour dan mengimplementasikan antarmuka IGameManager. Itu berarti kedua manajer mendapatkan semua fungsionalitas MonoBehaviour sementara juga perlu menerapkan struktur yang diberlakukan oleh IGameManager. Struktur di IGameManager adalah satu properti dan satu metode, jadi para manajer mendefinisikan dua hal itu.

Properti status didefinisikan sehingga status dapat dibaca dari mana saja (pengambil bersifat publik) tetapi hanya disetel dalam skrip ini (penyetel bersifat personal). Metode di antarmuka adalah Startup(), jadi kedua manajer mendefinisikan fungsi itu. Di kedua manajer, inisialisasi segera selesai (InventoryManager belum melakukan apa pun, sedangkan PlayerManager menetapkan beberapa nilai), jadi statusnya diatur ke Mulai. Tetapi modul data mungkin memiliki tugas yang berjalan lama sebagai bagian dari inisialisasinya (seperti memuat data yang disimpan), dalam hal ini Startup() akan meluncurkan tugas tersebut dan mengatur status manajer ke Inisialisasi. Ubah status menjadi Mulai setelah tugas tersebut selesai. Hebat—kami akhirnya siap untuk menyatukan semuanya dengan manajer-manajer utama! Buat satu skrip lagi dan beri nama Manajer (lihat daftar berikut).

Daftar Manajer-Manajer!

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

[RequireComponent(typeof(PlayerManager))]
[RequireComponent(typeof(InventoryManager))]

public class Managers : MonoBehaviour {
   public static PlayerManager Player {get; private set;}
   public static InventoryManager Inventory {get; private set;}

   private List<IGameManager> _startSequence;

   void Awake() {
      Player = GetComponent<PlayerManager>();
      Inventory = GetComponent<InventoryManager>();

      _startSequence = new List<IGameManager>();
      _startSequence.Add(Player);
      _startSequence.Add(Inventory);

      StartCoroutine(StartupManagers());
   }

    private IEnumerator StartupManagers() {
       foreach (IGameManager manager in _startSequence) {
          manager.Startup();
       }

       yield return null;

       int numModules = _startSequence.Count;
       int numReady = 0;

       while (numReady < numModules) {
          int lastReady = numReady;
          numReady = 0;

          foreach (IGameManager manager in _startSequence) {
             if (manager.status == ManagerStatus.Started) {
                numReady++;
             }
          }

          if (numReady > lastReady)
             Debug.Log("Progress: " + numReady + "/" + numModules);
          yield return null;
       }

       Debug.Log("All managers started up");
    }
}
```

Bagian terpenting dari pola ini adalah properti statis di bagian paling atas. Itu memungkinkan skrip lain untuk menggunakan sintaks seperti Managers.Player atau Managers.Inventory untuk mengakses berbagai modul. Properti tersebut awalnya kosong, tetapi langsung terisi saat kode dijalankan dalam metode Sedar().

Sama seperti Start() dan Update(), Awake() adalah metode lain yang secara otomatis disediakan oleh MonoBehaviour. Ini mirip dengan Start(), berjalan sekali ketika kode pertama kali mulai berjalan. Namun dalam urutan eksekusi kode Unity, Awake() bahkan lebih cepat dari Start(), memungkinkan tugas inisialisasi yang mutlak harus dijalankan sebelum modul kode lainnya. Metode Awake() juga mencantumkan urutan startup, dan kemudian

meluncurkan coroutine untuk memulai semua manajer. Secara khusus, fungsi membuat objek Daftar dan kemudian menggunakan List.Add() untuk menambahkan manajer.

Daftar adalah kumpulan struktur data yang disediakan oleh C#. Objek daftar mirip dengan array: mereka dideklarasikan dengan tipe tertentu dan menyimpan serangkaian entri secara berurutan. Tetapi Daftar dapat berubah ukuran setelah dibuat, sedangkan array dibuat dengan ukuran statis yang tidak dapat diubah nanti. Struktur data kumpulan terkandung dalam namespace baru yang harus Anda sertakan dalam skrip; perhatikan pernyataan penggunaan tambahan di bagian atas skrip. Jangan lupa detail ini di skrip Anda!

Karena semua manajer mengimplementasikan IGameManager, kode ini dapat mencantumkan semuanya sebagai tipe tersebut dan dapat memanggil metode Startup() yang ditentukan di masing-masing. Urutan startup dijalankan sebagai coroutine sehingga akan berjalan secara asinkron, dengan bagian lain dari permainan juga berjalan (misalnya, progress bar yang dianimasikan pada layar startup). Fungsi startup pertama-tama mengulang seluruh daftar manajer dan memanggil Startup() pada masing-masing manajer. Kemudian memasuki loop yang terus memeriksa apakah manajer telah memulai dan tidak akan melanjutkan sampai mereka semua melakukannya. Setelah semua manajer dimulai, fungsi startup akhirnya memberi tahu kita tentang fakta ini sebelum akhirnya selesai.

Manajer yang kami tulis sebelumnya memiliki inisialisasi sederhana sehingga tidak perlu menunggu, tetapi secara umum urutan startup berbasis coroutine ini dapat dengan elegan menangani tugas startup asinkron yang berjalan lama seperti memuat data yang disimpan. Sekarang semua struktur kode telah ditulis. Kembali ke Unity dan buat GameObject kosong baru; seperti biasa dengan objek kode kosong semacam ini, posisikan pada 0,0,0 dan beri objek nama deskriptif seperti Manajer Game. Lampirkan komponen skrip Managers, PlayerManager, dan InventoryManager ke objek baru ini.

Saat Anda memainkan game sekarang seharusnya tidak ada perubahan yang terlihat di scene, tetapi di konsol Anda akan melihat serangkaian pesan yang mencatat kemajuan urutan startup. Dengan asumsi manajer memulai dengan benar, saatnya untuk mulai memprogram manajer inventaris.

Menyimpan inventaris dalam objek koleksi: Daftar vs. Kamus

Daftar sebenarnya dari item yang dikumpulkan juga dapat disimpan sebagai objek Daftar. Daftar berikutnya menambahkan Daftar item ke InventoryManager.

Daftar Menambahkan item ke InventoryManager

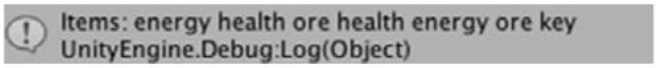
```
...
private List<string> _items;

public void Startup() {
    Debug.Log("Inventory manager starting...");

    _items = new List<string>();

    status = ManagerStatus.Started;
}

private void DisplayItems() {
    string itemDisplay = "Items: ";
    foreach (string item in _items) {
        itemDisplay += item + " ";
    }
    Debug.Log(itemDisplay);
}

public void AddItem(string name) {
    _items.Add(name);

    DisplayItems();
}
...
```

Dua tambahan kunci dibuat untuk InventoryManager. Pertama, kita menambahkan objek List untuk menyimpan item. Kedua, kita menambahkan metode publik, AddItem(), yang dapat dipanggil oleh kode lain. Fungsi ini menambahkan item ke daftar dan kemudian mencetak daftar ke konsol. Sekarang mari kita buat sedikit penyesuaian pada skrip CollectibleItem untuk memanggil metode AddItem() baru (lihat daftar berikut).

Daftar Menggunakan InventoryManager baru di CollectibleItem

```
...
void OnTriggerEnter(Collider other) {
    Managers.Inventory.AddItem(name);
    Destroy(this.gameObject);
}
...
```

Sekarang ketika Anda berkeliling mengumpulkan item, Anda akan melihat inventaris Anda bertambah di pesan konsol. Ini cukup keren, tetapi itu memperlihatkan satu batasan struktur data Daftar: saat Anda mengumpulkan banyak dari jenis item yang sama (seperti mengumpulkan item Kesehatan kedua), Anda akan melihat kedua salinan terdaftar, alih-alih menggabungkan semua item dari jenis yang sama (lihat gambar 5.59). Bergantung pada gim Anda, Anda mungkin ingin inventaris melacak setiap item secara terpisah, tetapi di sebagian besar gim, inventaris harus menggabungkan beberapa salinan dari item yang sama. Dimungkinkan untuk melakukannya menggunakan Daftar, tetapi ini dilakukan secara lebih alami dan efisien menggunakan Kamus.



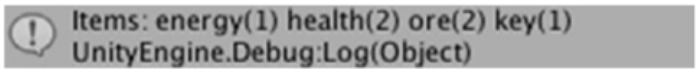
Gambar 5.59 Pesan konsol dengan kelipatan item yang sama terdaftar beberapa kali

Kamus adalah struktur data kumpulan lain yang disediakan oleh C#. Entri dalam kamus diakses oleh pengenal (atau kunci) daripada oleh posisinya dalam daftar. Ini mirip dengan tabel hash tetapi lebih fleksibel, karena kuncinya bisa berupa tipe apa saja (misalnya, "Kembalikan entri untuk GameObject ini"). Ubah kode di InventoryManager untuk menggunakan Kamus alih-alih Daftar. Ganti semuanya dari daftar 8.14 dengan kode dari daftar berikut.

Daftar Kamus item di InventoryManager

```
...
private Dictionary<string, int> _items;

public void Startup() {
    Debug.Log("Inventory manager starting...");

    _items = new Dictionary<string, int>();

    status = ManagerStatus.Started;
}

private void DisplayItems() {
    string itemDisplay = "Items: ";
    foreach (KeyValuePair<string, int> item in _items) {
        itemDisplay += item.Key + "(" + item.Value + ") ";
    }
    Debug.Log(itemDisplay);
}

public void AddItem(string name) {
    if (_items.ContainsKey(name)) {
        _items[name] += 1;
    } else {
        _items[name] = 1;
    }

    DisplayItems();
}
...
```

Secara keseluruhan kode ini terlihat sama seperti sebelumnya, tetapi ada beberapa perbedaan rumit. Jika Anda belum terbiasa dengan struktur data Kamus, perhatikan bahwa itu dideklarasikan dengan dua jenis. Sedangkan Daftar dideklarasikan dengan hanya satu tipe (tipe nilai yang akan dicantumkan), Kamus mendeklarasikan tipe kunci (yaitu, apa pengidentifikasinya) dan tipe nilai. Sedikit lebih banyak logika ada dalam metode AddItem(). Padahal sebelum setiap item ditambahkan ke Daftar, sekarang kita perlu memeriksa apakah Kamus sudah berisi item itu; untuk itulah metode BerisiKey() digunakan. Jika ini adalah entri baru, maka kami akan mulai menghitung dari 1, tetapi jika entri sudah ada, maka tingkatkan nilai yang disimpan. Mainkan dengan kode baru dan Anda akan melihat pesan inventaris memiliki jumlah agregat dari setiap item.



Gambar 5.60 Pesan konsol dengan kelipatan item yang sama digabungkan

Wah, akhirnya, item yang dikumpulkan dikelola dalam inventaris player! Ini mungkin tampak seperti banyak kode untuk menangani masalah yang relatif sederhana, dan jika ini adalah tujuan keseluruhannya, ya, itu direkayasa berlebihan. Namun, inti dari arsitektur kode yang rumit ini adalah untuk menyimpan semua data dalam modul fleksibel yang terpisah, pola yang berguna setelah permainan menjadi lebih kompleks. Misalnya, sekarang kita dapat menulis tampilan UI dan bagian kode yang terpisah akan lebih mudah ditangani.

UI inventaris untuk menggunakan dan melengkapi item

Koleksi item dalam inventaris Anda dapat digunakan dalam berbagai cara di dalam game, tetapi semua penggunaan tersebut pertama-tama bergantung pada semacam UI inventaris sehingga player dapat melihat item yang dikumpulkan. Kemudian, setelah inventaris ditampilkan kepada player, Anda dapat memprogram interaktivitas ke dalam UI dengan memungkinkan player mengklik item mereka. Sekali lagi, kami akan memprogram beberapa contoh spesifik (melengkapi kunci dan menggunakan paket kesehatan), dan kemudian Anda harus dapat menyesuaikan kode ini untuk bekerja dengan jenis item lainnya. Seperti disebutkan dalam bab 6, Unity memiliki GUI mode langsung yang lebih lama dan sistem UI berbasis sprite yang lebih baru. Kami akan menggunakan GUI mode langsung dalam bab ini karena sistem itu lebih cepat diimplementasikan dan membutuhkan lebih sedikit pengaturan; kurang setup sangat bagus untuk latihan latihan. Sistem UI berbasis sprite lebih dipoles, dan untuk gim yang sebenarnya Anda menginginkan antarmuka yang lebih halus.

Menampilkan item inventaris di UI

Untuk menampilkan item dalam tampilan UI, pertama-tama kita perlu menambahkan beberapa metode lagi ke InventoryManager. Saat ini daftar item bersifat personal dan hanya dapat diakses di dalam pengelola; untuk menampilkan daftar, informasi tersebut harus memiliki metode publik untuk mengakses data. Tambahkan dua metode yang ditunjukkan dalam daftar berikut ke InventoryManager.

Daftar Menambahkan metode akses data ke InventoryManager

```
...
public List<string> GetItemList() {
    List<string> list = new List<string>(_items.Keys);
    return list;
}

public int GetItemCount(string name) {
    if (_items.ContainsKey(name)) {
        return _items[name];
    }
    return 0;
}
...
```

Metode GetItemList() mengembalikan daftar item dalam inventaris. Anda mungkin berpikir, “Tunggu sebentar, bukankah kami baru saja menghabiskan banyak upaya untuk mengubah inventaris dari Daftar?” Bedanya sekarang adalah setiap jenis item hanya akan muncul satu kali dalam daftar. Jika inventaris berisi dua paket kesehatan, misalnya, nama "kesehatan" hanya akan muncul satu kali dalam daftar. Itu karena Daftar dibuat dari kunci dalam Kamus, bukan dari setiap item individual. Metode GetItemCount() mengembalikan hitungan berapa banyak item yang diberikan dalam inventaris. Misalnya, hubungi GetItemCount("health") untuk menanyakan "Berapa banyak paket kesehatan yang ada di inventaris?" Dengan cara ini, UI dapat menampilkan sejumlah setiap item bersama dengan menampilkan setiap item.

Dengan metode ini ditambahkan ke InventoryManager, kita dapat membuat tampilan UI. Mari kita tampilkan semua item dalam baris horizontal di bagian atas layar. Item akan ditampilkan menggunakan ikon, jadi kita perlu mengimpor gambar tersebut ke dalam proyek. Unity menangani aset dengan cara khusus jika aset tersebut berada di folder bernama Resources. Aset yang ditempatkan ke folder Resources dapat dimuat dalam kode menggunakan metode Resources.Load(). Jika tidak, aset hanya dapat ditempatkan dalam scene melalui editor Unity.

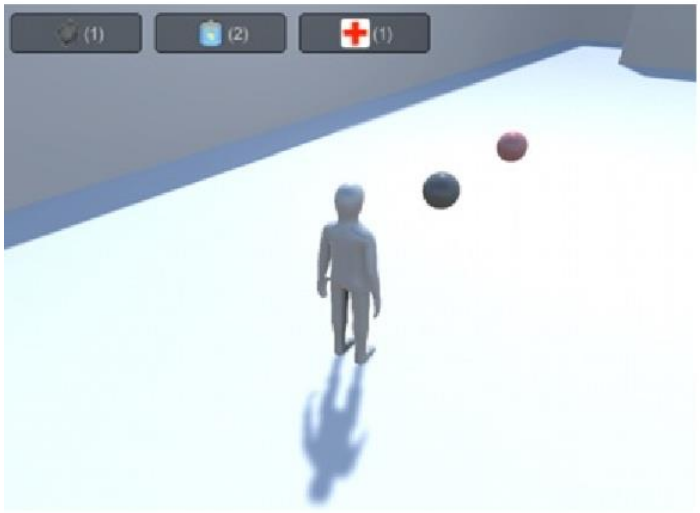
Gambar 5.61 Aset gambar untuk ikon peralatan ditempatkan di dalam folder Sumber Daya

Semua ikon sudah diatur, jadi buat GameObject kosong baru bernama Controller dan kemudian tetapkan skrip baru bernama BasicUI (lihat daftar berikutnya).

Daftar BasicUI menampilkan inventaris

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class BasicUI : MonoBehaviour {
   void OnGUI() {
      int posX = 10;
      int posY = 10;
      int width = 100;
      int height = 30;
      int buffer = 10;

      List<string> itemList = Managers.Inventory.GetItemList();
      if (itemList.Count == 0) {
         GUI.Box(new Rect(posX, posY, width, height), "No Items");
      }
      foreach (string item in itemList) {
         int count = Managers.Inventory.GetItemCount(item);
         Texture2D image = Resources.Load<Texture2D>("Icons/"+item);
         GUI.Box(new Rect(posX, posY, width, height),
                 new GUIContent("(" + count + ")", image));
         posX += width+buffer;
      }
   }
}
```

Daftar ini menampilkan item yang dikumpulkan dalam baris horizontal bersama dengan menampilkan nomor yang dikumpulkan. Seperti disebutkan dalam bab 3, setiap MonoBehaviour secara otomatis merespons metode OnGUI(). Fungsi itu menjalankan setiap frame tepat setelah scene 3D dirender.



Gambar 5.62 Tampilan UI inventaris

Di dalam OnGUI(), pertama-tama tentukan sekelompok nilai untuk memposisikan elemen UI. Nilai-nilai ini bertambah saat kita mengulang semua item untuk memposisikan elemen UI dalam satu baris. Elemen UI spesifik yang digambar adalah GUI.Box; itu adalah tampilan noninteraktif yang menampilkan teks dan gambar di dalam kotak.

Metode Resources.Load() digunakan untuk memuat aset dari folder Resources. Metode ini adalah cara praktis untuk memuat aset berdasarkan nama; perhatikan bahwa nama item dilewatkan sebagai parameter. Kita harus menentukan tipe yang akan dimuat; jika tidak, nilai kembalian untuk metode tersebut adalah objek generik. UI menunjukkan kepada kita item apa yang telah dikumpulkan. Sekarang kita benar-benar dapat menggunakan item.

Melengkapi kunci untuk digunakan pada pintu yang terkunci

Mari membahas beberapa contoh penggunaan item inventaris sehingga Anda dapat memperkirakan jenis item apa pun yang Anda inginkan. Contoh pertama melibatkan melengkapi kunci yang diperlukan untuk membuka pintu. Saat ini, skrip DeviceTrigger tidak memperhatikan item Anda (karena skrip itu ditulis sebelum kode inventaris). Daftar berikutnya menunjukkan cara menyesuaikan skrip itu.

Daftar Membutuhkan kunci di DeviceTrigger

```
...
public bool requireKey;
void OnTriggerEnter(Collider other) {
if (requireKey && Managers.Inventory.equippedItem != "key") { return;
}
...
```

Seperti yang Anda lihat, yang diperlukan hanyalah variabel publik baru dalam skrip dan kondisi yang mencari kunci yang dilengkapi. requireKey Boolean muncul sebagai kotak centang di Inspector sehingga Anda dapat meminta kunci dari beberapa pemicu tetapi tidak dari pemicu lainnya. Kondisi di awal OnTriggerEnter() memeriksa kunci yang dilengkapi di InventoryManager; yang mengharuskan Anda menambahkan kode dari daftar berikutnya ke InventoryManager.

Daftar dibawah ini Melengkapi kode untuk InventoryManager Di bagian atas tambahkan properti dilengkapiItem yang diperiksa oleh kode lain. Kemudian tambahkan metode publik EquipItem() untuk mengizinkan kode lain mengubah item mana yang dilengkapi. Metode itu melengkapi item jika belum dilengkapi, atau tidak melengkapi jika item itu sudah dilengkapi.

```
public string equippedItem {get; private set;}
...
public bool EquipItem(string name) {
   if (_items.ContainsKey(name) && equippedItem != name) {
      equippedItem = name;
      Debug.Log("Equipped " + name);
      return true;
   }

   equippedItem = null;
   Debug.Log("Unequipped");
   return false;
}
...
```

Terakhir, agar player melengkapi item, tambahkan fungsionalitas itu ke UI. Daftar berikut akan menambahkan deretan tombol untuk tujuan itu.

Daftar Melengkapi fungsionalitas ditambahkan ke BasicUI

```
...
      foreach (string item in itemList) {


         int count = Managers.Inventory.GetItemCount(item);
         GUI.Box(new Rect(posX, posY, width, height), item +
                  "(" + count + ")");
         posX += width+buffer;
      }

      string equipped = Managers.Inventory.equippedItem;
      if (equipped != null) {                          B
        posX = Screen.width - (width+buffer);
        Texture2D image = Resources.Load("Icons/"+equipped) as Texture2D;
        GUI.Box(new Rect(posX, posY, width, height),
                new GUIContent("Equipped", image));
      }



      posX = 10;
      posY += height+buffer;

      foreach (string item in itemList) {
         if (GUI.Button(new Rect(posX, posY, width, height),
                  "Equip "+item)) {
            Managers.Inventory.EquipItem(item);
         }
         posX += width+buffer;
      }
   }
}
```

GUI.Box() digunakan lagi untuk menampilkan item yang dilengkapi. Tapi elemen itu noninteraktif, jadi deretan tombol Equip digambar menggunakan GUI.Button() sebagai gantinya. Metode itu membuat tombol yang mengeksekusi kode di dalam pernyataan if saat diklik. Dengan semua kode yang diperlukan, pilih opsi requireKey di DeviceTrigger lalu mainkan gamenya. Coba jalankan ke volume pemicu sebelum melengkapi kunci; tidak ada

yang terjadi. Sekarang kumpulkan kunci dan klik tombol untuk melengkapinya; berlari ke volume pemicu membuka pintu.

Hanya untuk bersenang-senang, Anda dapat meletakkan kunci di Posisi -11 5 -14 untuk menambahkan tantangan gameplay sederhana untuk melihat apakah Anda dapat menemukan cara untuk mencapai kunci tersebut. Baik Anda mencobanya atau tidak, mari beralih menggunakan paket kesehatan.

Memulihkan kesehatan player dengan mengkonsumsi paket kesehatan

Menggunakan item untuk memulihkan kesehatan player adalah contoh lain yang umumnya berguna. Itu membutuhkan dua perubahan kode: metode baru di InventoryManager dan tombol baru di UI (lihat daftar 8.22 dan 8.23, masing-masing).

Daftar Metode baru di InventoryManager

```
...
public bool ConsumeItem(string name) {
   if (_items.ContainsKey(name)) {
      _items[name]--;
      if (_items[name] == 0) {
         _items.Remove(name);
      }
   } else {
      Debug.Log("cannot consume " + name);
      return false;
   }

   DisplayItems();
   return true;
}
...
```

Daftar Menambahkan item kesehatan ke UI Dasar

```
...
      foreach (string item in itemList) {
         if (GUI.Button(new Rect(posX, posY, width, height),
                  "Equip "+item)) {
            Managers.Inventory.EquipItem(item);
         }

         if (item == "health") {
            if (GUI.Button(new Rect(posX, posY + height+buffer, width,
                        height), "Use Health")) {
               Managers.Inventory.ConsumeItem("health");
               Managers.Player.ChangeHealth(25);
            }
         }

         posX += width+buffer;
      }
   }
}
```

Metode ConsumeItem() baru adalah kebalikan dari AddItem(); itu memeriksa item dalam inventaris dan mengurangi jika item ditemukan. Ini memiliki respons terhadap beberapa kasus rumit, seperti jika jumlah item berkurang menjadi 0. Kode UI memanggil metode inventaris baru ini, dan memanggil metode ChangeHealth() yang dimiliki PlayerManager sejak awal. Jika Anda mengumpulkan beberapa item kesehatan dan kemudian

menggunakannya, Anda akan melihat pesan kesehatan muncul di konsol. Dan begitulah—beberapa contoh cara menggunakan item inventaris!

BAB 6
MEMBUAT APLIKASI ANDROID

Game kita masih belum selesai pada saat ini, dan selama beberapa bab berikutnya Anda akan belajar cara menambahkan hal-hal seperti level, checkpoint, menu, lebih banyak elemen UI, dan sejumlah fitur lainnya. Tapi yang kami miliki adalah permainan dasar yang sekarang berfungsi sebagai permainan dan merupakan sesuatu yang pasti bisa dimainkan. Kanvas kita sudah terpasang—yang akan menjadi poin penting. Singkatnya, saya merasa Anda sudah menunggu cukup lama. Saatnya untuk menjalankan dan menjalankan hal ini di perangkat Android Anda. Dalam bab ini, Anda akan mempelajari cara membuat APK, cara menguji game di ponsel atau tablet, dan cara menambahkan kontrol sentuh ke game. Pada akhirnya, Anda akan dapat membawa game yang Anda buat ke mana saja dan memasukkannya ke dalam saku Anda. Secara teori, Anda bahkan bisa melepaskannya ke dunia. Tapi saya tidak akan merekomendasikan melakukan itu dulu.

6.1 USER INTERFACE (UI)

Antarmuka pengguna (UI) adalah seperangkat komponen khusus yang bertanggung jawab untuk mengirim informasi ke, dan membaca informasi dari, pengguna. Dalam jam ini, Anda akan mempelajari semua tentang menggunakan sistem UI bawaan Unity. Anda akan mulai dengan memeriksa dasar-dasar UI. Dari sana, Anda akan mencoba berbagai elemen UI, seperti teks, gambar, tombol, dan banyak lagi. Anda akan menyelesaikan pelajaran ini dengan membuat sistem menu yang sederhana namun lengkap untuk game Anda.

Prinsip UI Dasar

Antarmuka pengguna (biasa disebut sebagai UI) adalah layer khusus yang ada untuk memberikan informasi kepada pengguna dan untuk menerima input sederhana dari pengguna. Informasi dan input ini dapat berbentuk HUD (heads up display) yang digambar di atas game Anda atau beberapa objek yang sebenarnya terletak di dalam dunia 3D Anda. Di Unity, UI didasarkan pada kanvas tempat semua elemen UI dicat. Kanvas ini harus menjadi induk dari semua objek UI agar dapat berfungsi, dan kanvas ini adalah objek utama yang menggerakkan seluruh UI Anda.

Desain UI

Sebagai aturan umum, Anda harus membuat sketsa UI Anda sebelumnya. Sedikit pemikiran perlu masuk ke apa yang akan ditampilkan di layar, di mana itu akan ditampilkan, dan bagaimana caranya. Terlalu banyak informasi akan menyebabkan layar terasa berantakan. Informasi yang terlalu sedikit akan membuat para player bingung atau tidak yakin. Selalu mencari cara untuk memadatkan informasi dan membuat informasi lebih bermakna. Player Anda akan berterima kasih.

UI baru

Unity mendapat UI baru di versi 4.6. Di versi yang lebih lama, Anda harus menggunakan banyak baris kode yang membingungkan untuk membuat UI, tetapi sekarang jauh lebih mudah. Sistem UI lama masih ada. Jika Anda terbiasa dengan sistem warisan itu, Anda mungkin tergoda untuk menggunakannya. Tolong jangan. Sistem lama hanya ada untuk debugging, kompatibilitas mundur dengan proyek lama, dan ekstensi editor. Ini hampir tidak seefisien atau sekuat sistem baru!

6.2 KANVAS

Kanvas adalah blok bangunan dasar untuk UI, dan semua elemen UI terkandung di dalam kanvas. Semua elemen UI yang Anda tambahkan ke scene akan menjadi objek anak kanvas dalam tampilan Hierarki dan harus tetap sebagai anak; jika tidak, mereka akan menghilang dari scene. Menambahkan kanvas ke scene sangat mudah. Anda dapat menambahkannya hanya dengan memilih GameObject > UI > Canvas. Setelah kanvas ditambahkan ke scene, Anda siap untuk mulai membangun sisa UI.

Menambahkan Kanvas

Ikuti langkah-langkah ini untuk menambahkan kanvas ke scene dan menjelajahi fitur uniknya:

1. Buat new project (baik 2D atau 3D).
2. Tambahkan kanvas UI ke scene (dengan memilih GameObject > UI > Canvas).
3. Perkecil sehingga Anda dapat melihat seluruh kanvas (dengan mengklik dua kali dalam tampilan Hierarki). Perhatikan seberapa besar itu!
4. Catat di Inspector komponen odd transform yang dimiliki kanvas. Ini adalah transformasi Rect, dan kami akan segera membahasnya.

Sistem Event

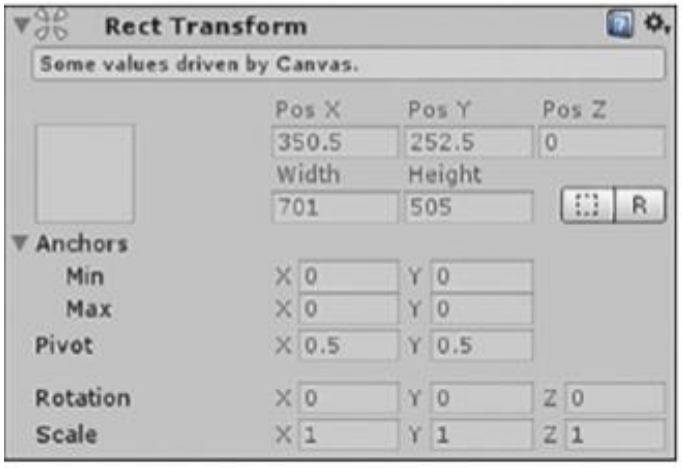
Anda mungkin telah memperhatikan bahwa ketika Anda menambahkan kanvas ke scene Anda, Anda juga mendapatkan objek game EventSystem. Objek ini selalu ditambahkan saat Anda menambahkan kanvas. Sistem event inilah yang memungkinkan pengguna berinteraksi dengan UI dengan menekan tombol atau menyeret elemen. Tanpa sistem event, UI tidak akan pernah tahu apakah itu sedang digunakan—jadi jangan hapus!

Kesengsaraan Kinerja

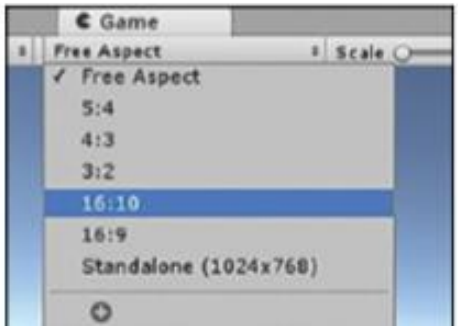
Kanvas sangat efisien karena mengubah elemen UI yang bersarang di atasnya menjadi satu objek statis di belakang layar. Ini memungkinkan mereka untuk diproses dengan sangat cepat. Kelemahannya adalah ketika satu bagian dari UI berubah, semuanya perlu dibangun kembali. Ini bisa menjadi proses yang sangat lambat dan tidak efisien dan dapat menyebabkan kegagapan yang nyata dalam permainan. Oleh karena itu, sebaiknya gunakan komponen kanvas untuk memisahkan objek yang banyak bergerak ke kanvasnya sendiri. Dengan begitu, gerakan mereka akan memaksa set UI Anda yang lebih kecil untuk dibangun kembali dan pada akhirnya menjadi jauh lebih cepat.

Transformasi Rect

Anda akan melihat bahwa kanvas (dan setiap elemen UI lainnya) memiliki transformasi Rect daripada transformasi 3D normal yang Anda kenal. Rect, kependekan dari rectangle, transforms memberi Anda kontrol fantastis atas pemosisian dan penscalean ulang elemen UI sambil tetap sangat fleksibel. Ini memungkinkan Anda untuk membuat antarmuka pengguna dan memastikannya berfungsi dengan baik di berbagai perangkat. Untuk kanvas yang Anda buat sebelumnya pada jam ini, transformasi Rect seluruhnya berwarna abu-abu (lihat Gambar 6.1). Ini karena, dalam bentuknya saat ini, kanvas memperoleh nilainya sepenuhnya dari tampilan Game (dan, dengan ekstensi, resolusi dan rasio aspek perangkat apa pun yang menjalankan game Anda). Ini berarti kanvas akan selalu memenuhi seluruh layar. Alur kerja yang baik adalah memastikan hal pertama yang Anda lakukan setiap kali membangun UI adalah memilih rasio aspek target untuk digunakan. Anda dapat melakukan ini dari drop-down Rasio Aspek di tampilan Game.



Gambar 6.1 Transformasi Rect dari kanvas.



Gambar 6.2 Mengatur rasio aspek game.

Transformasi rect bekerja sedikit berbeda dari transformasi tradisional. Dengan objek 2D dan 3D normal, transformasi berkaitan dengan menentukan seberapa jauh (dan dengan keselarasan apa) suatu objek dari asal dunia. UI, bagaimanapun, tidak peduli dengan asal dunia dan sebagai gantinya perlu tahu bagaimana itu selaras dalam kaitannya dengan titik jangkarnya. (Anda akan mempelajari lebih lanjut tentang transformasi Rect dan jangkar nanti, ketika Anda memiliki elemen UI yang benar-benar dapat menggunakannya.)

6.3 JANGKAR

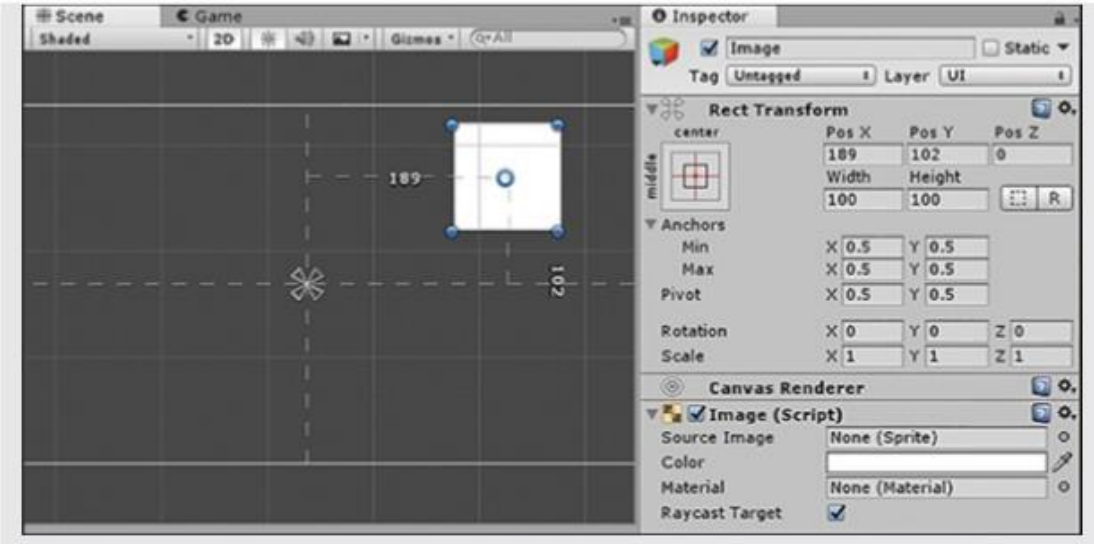
Konsep kunci dalam membuat elemen UI berfungsi adalah jangkar. Setiap elemen UI dilengkapi dengan jangkar dan menggunakan jangkar itu untuk menemukan tempatnya di dunia dalam kaitannya dengan transformasi Rect dari induknya. Jangkar menentukan bagaimana elemen diubah ukurannya dan diposisikan ulang saat jendela Game mengubah ukuran dan bentuk. Selain itu, jangkar memiliki dua "mode": bersama dan terpisah. Ketika jangkar bersama-sama sebagai satu titik, objek tahu di mana itu dengan menentukan jarak (dalam piksel) dari porosnya dari jangkar. Namun, saat jangkar dipisah, elemen UI mendasarkan kotak pembatasnya pada seberapa jauh (sekali lagi dalam piksel) setiap sudutnya dari setiap sudut jangkar yang dipisah. Membingungkan? Mari kita coba!

Menggunakan Transformasi Persegi

Transformasi rect dan anchor dapat membingungkan, jadi ikuti langkah-langkah berikut untuk lebih memahaminya:

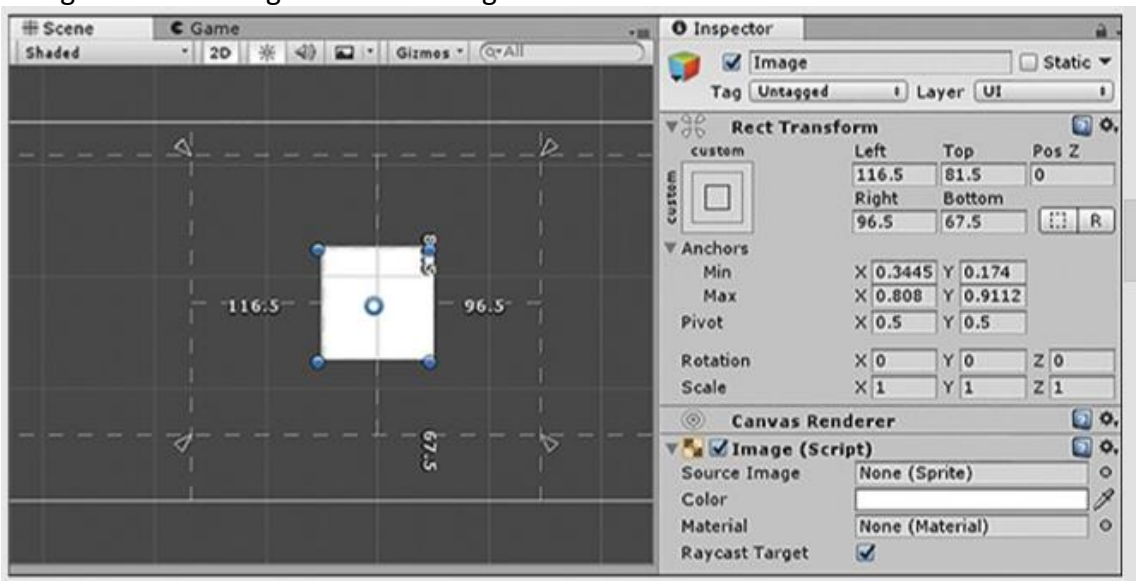
1. Buat scene atau new project.

2. Tambahkan gambar UI (dengan memilih GameObject > UI > Image). Perhatikan bahwa jika Anda menambahkan gambar ke scene tanpa kanvas, Unity secara otomatis menempatkan kanvas di scene Anda dan kemudian meletakkan gambar di atasnya.
3. Perkecil sehingga Anda dapat melihat seluruh gambar dan kanvas. Perhatikan bahwa bekerja dengan UI jauh lebih mudah saat scene Anda dalam mode 2D (yang Anda masukkan dengan mengklik tombol 2D di bagian atas tampilan Scene) dan Anda menggunakan alat Rect (tombol pintas: T).
4. Coba drag gambar di sekitar kanvas. Coba juga drag jangkar di sekitar kanvas. Perhatikan bahwa garis menunjukkan seberapa jauh poros gambar dari jangkar. Perhatikan juga properti dari transformasi Rect di Inspector dan bagaimana mereka berubah



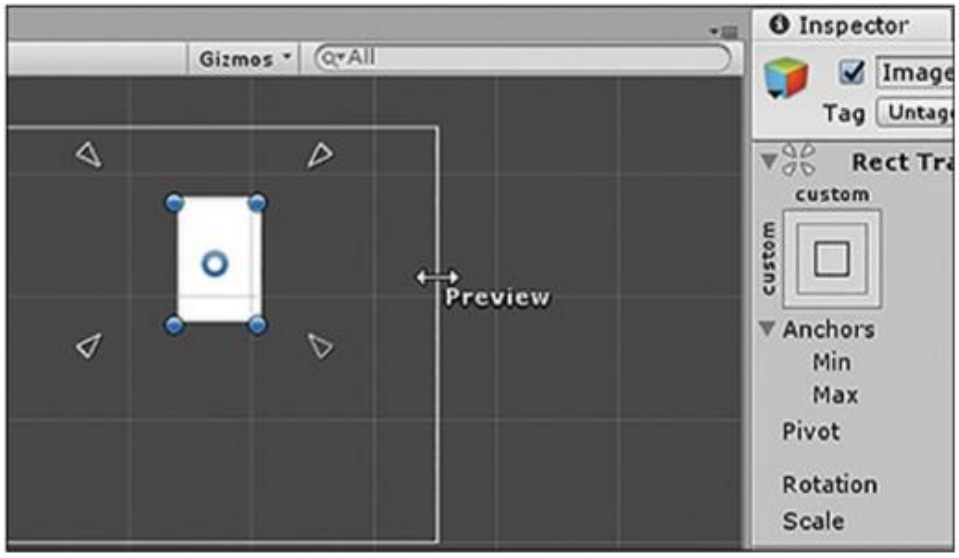
Gambar 6.3 Jangkar pada satu titik.

5. Sekarang coba pisahkan jangkar Anda. Anda dapat melakukan ini dengan menyeret salah satu sudut jangkar menjauh dari yang lain. Dengan pemisahan jangkar, gerakkan gambar Anda lagi. Perhatikan bagaimana sifat-sifat transformasi Rect berubah



Gambar 6.4 Jangkar setelah dibelah.

Jadi, apa sebenarnya yang dilakukan dengan memisahkan jangkar (atau menyatukannya)? Secara sederhana, jangkar yang merupakan satu titik memperbaiki elemen UI Anda di tempat relatif terhadap tempat itu. Jadi jika kanvas berubah ukuran, elemennya tidak. Memisahkan jangkar menyebabkan elemen memperbaiki sudutnya relatif terhadap sudut jangkar. Jika kanvas berubah ukuran, begitu juga elemennya. Anda dapat dengan mudah melihat preview perilaku ini di editor Unity. Menggunakan contoh sebelumnya, jika Anda memilih gambar lalu klik dan drag batas kanvas (atau induk lainnya, jika Anda memiliki lebih banyak elemen), kata Preview akan muncul, dan Anda dapat melihat apa yang akan terjadi saat menggunakan resolusi yang berbeda. Cobalah dengan jangkar tunggal dan jangkar terpisah dan perhatikan betapa berbedanya perilaku mereka.



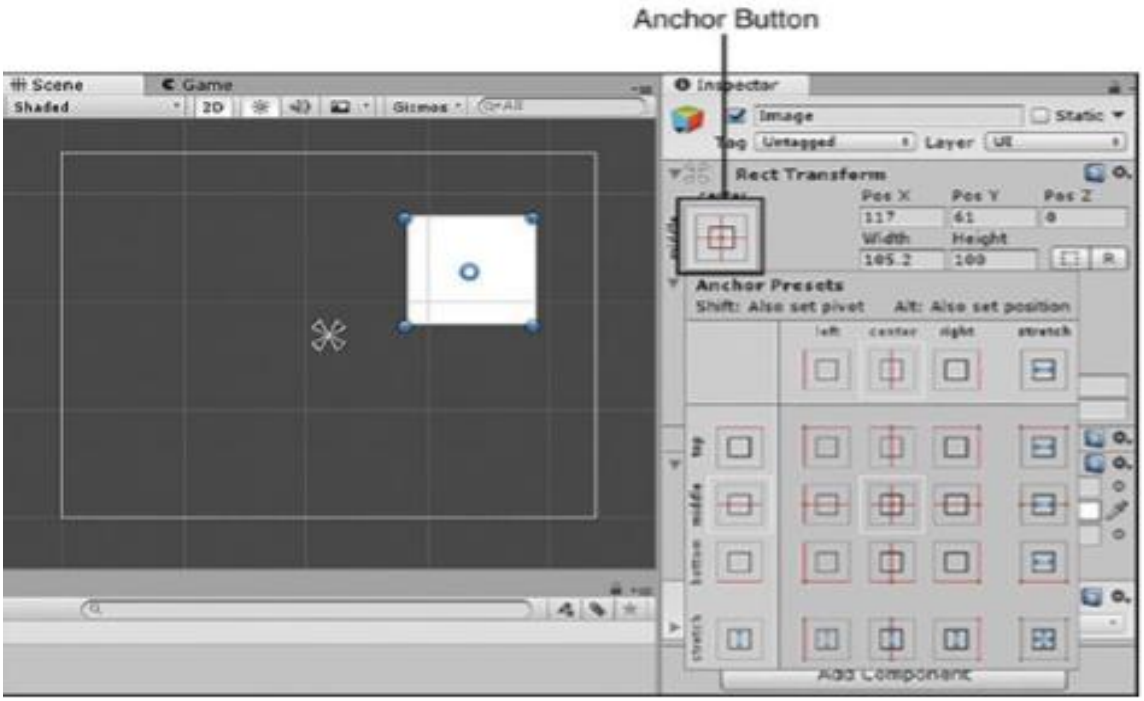
Gambar 6.5 Preview perubahan kanvas.

Mendapatkan Jangkar dengan Benar

Jangkar mungkin tampak sedikit aneh pada awalnya, tetapi memahaminya adalah kunci untuk memahami UI. Dapatkan jangkar, dan segala sesuatu yang lain hanya jatuh ke tempatnya. Saat bekerja dengan elemen UI, sangat bagus untuk membiasakan diri selalu menempatkan jangkar dan kemudian menempatkan objek karena objek menempel pada jangkarnya tetapi tidak sebaliknya. Ketika Anda terbiasa dengan kebiasaan itu (meletakkan jangkar, menempatkan objek, menempatkan jangkar, menempatkan objek, dan sebagainya), semuanya menjadi jauh lebih mudah. Investasikan waktu untuk bermain dengan jangkar sampai Anda memahaminya.

Tombol Jangkar

Anda tidak selalu harus menyeret jangkar secara manual di sekitar scene Anda untuk menempatkannya. Anda juga dapat mengetikkan nilainya ke dalam properti Anchors di Inspector (nilai 1 adalah 100%, .5 adalah 50%, dan seterusnya). Bahkan jika itu terlalu banyak bekerja untuk Anda, Anda dapat menggunakan tombol jangkar yang nyaman yang memungkinkan Anda untuk menempatkan jangkar (dan juga poros dan posisi) di salah satu dari 24 lokasi yang telah ditetapkan



Gambar 6.6 Tombol jangkar.

Komponen Kanvas Tambahan

Sejauh ini, kami telah berbicara sedikit tentang kanvas tetapi masih belum menyebutkan komponen Kanvas yang sebenarnya. Sejujurnya, tidak banyak komponen itu sendiri yang perlu Anda perhatikan. Anda perlu tahu tentang mode render, dan kita akan melihatnya secara mendetail nanti dalam jam ini. Bergantung pada versi Unity yang Anda gunakan, Anda mungkin memiliki beberapa komponen tambahan. Sekali lagi, ini sangat mudah digunakan, jadi tidak dibahas secara mendetail di sini (terlalu banyak hal bagus lainnya untuk dicapai). Komponen Canvas Scaler memungkinkan Anda untuk menentukan bagaimana, jika sama sekali, Anda ingin elemen UI Anda diubah ukurannya saat layar perangkat target Anda berubah (misalnya, melihat UI yang sama di halaman web versus perangkat iPad Retina DPI tinggi) . Komponen Graphical Raycaster, yang bekerja dengan objek EventSystem, memungkinkan UI Anda menerima klik tombol dan sentuhan layar. Ini memungkinkan Anda untuk menggunakan raycasting tanpa perlu menyeret seluruh mesin fisika untuk melakukannya.

6.4 ELEMEN UI

Pada titik ini, Anda mungkin cukup bosan dengan kanvas, jadi mari bekerja dengan beberapa elemen UI (juga disebut kontrol UI). Unity memiliki beberapa kontrol bawaan yang tersedia untuk Anda mulai. Namun, jangan khawatir jika Anda tidak melihat kontrol yang Anda inginkan. Pustaka UI Unity adalah sumber terbuka, dan banyak kontrol khusus dibuat oleh anggota komunitas setiap saat. Faktanya, jika Anda siap menghadapi tantangan, Anda bahkan dapat membuat kontrol Anda sendiri dan membaginya dengan orang lain. Unity memiliki banyak kontrol yang dapat Anda tambahkan ke scene. Kebanyakan dari mereka adalah kombinasi sederhana dan variasi dari dua elemen dasar: gambar dan teks. Ini masuk akal jika Anda memikirkannya: Panel hanyalah gambar berukuran penuh, tombol hanyalah gambar dengan beberapa teks, dan penggeser sebenarnya adalah tiga gambar yang ditumpuk menjadi satu. Faktanya, seluruh UI dibangun untuk menjadi sistem blok bangunan dasar yang dapat ditumpuk untuk mendapatkan fungsionalitas yang Anda inginkan.

Gambar

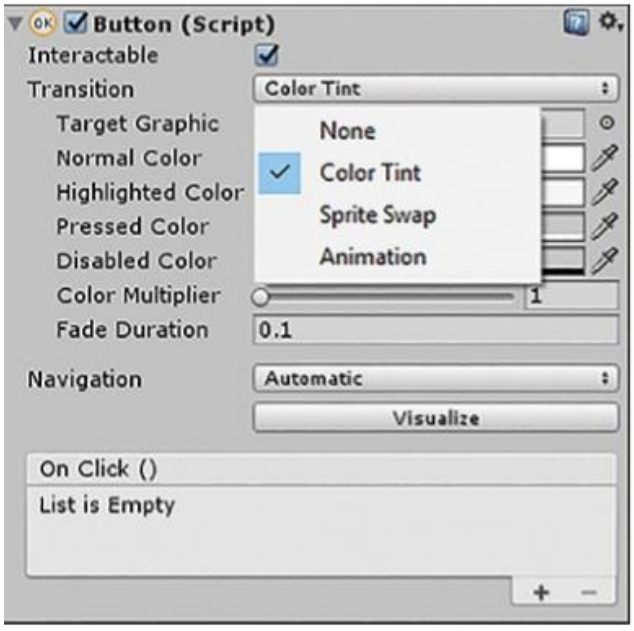
Gambar adalah bagian dasar dari sebuah UI. Mereka dapat berkisar dari gambar latar belakang, hingga tombol, hingga logo, hingga bilah kesehatan, hingga semua yang ada di antaranya. Jika Anda menyelesaikan latihan Coba Sendiri di awal jam ini, maka Anda sudah terbiasa dengan gambar, tetapi sekarang Anda akan melihat lebih dekat. Seperti yang Anda lihat sebelumnya, Anda dapat menambahkan gambar ke kanvas dengan memilih GameObject > UI > Image. Tabel 14.1 mencantumkan properti gambar, yang hanya merupakan objek game dengan komponen Gambar.

Material UI

Seperti yang Anda lihat pada Gambar 14.7, ada properti Material untuk komponen gambar di "Using an Image" Coba Sendiri. Perlu dicatat bahwa properti Material sepenuhnya opsional dan tidak diperlukan untuk UI ini. Selanjutnya, dalam mode render kanvas saat ini (dijelaskan lebih detail nanti di jam ini), properti Material tidak melakukan banyak hal. Namun, dalam mode lain, properti Material memungkinkan Anda menerapkan efek lampu dan shader ke elemen UI. Objek teks (yang sebenarnya hanyalah komponen teks) adalah elemen yang Anda gunakan untuk menampilkan teks kepada pengguna. Jika Anda pernah menggunakan kontrol pemformatan teks sebelumnya (pikirkan software blog, pengolah kata seperti Word atau WordPad, atau di mana pun Anda akan menggunakan dan menata teks), komponen Teks akan sangat familiar. Anda dapat menambahkan komponen Teks ke kanvas dengan memilih GameObject > UI > Text.

Buttons

Buttons adalah elemen yang memungkinkan input klik dari pengguna. Mereka mungkin tampak rumit pada pandangan pertama, tetapi ingat bahwa, seperti yang disebutkan sebelumnya, tombol sebenarnya hanyalah sebuah gambar dengan anak objek teks dan sedikit lebih banyak fungsi. Anda dapat menambahkan tombol ke scene Anda dengan memilih GameObject > UI > Button. Di mana tombol berbeda dari salah satu kontrol lain yang telah Anda lihat sejauh ini adalah tombol itu dapat berinteraksi. Karena itu, ia memiliki beberapa properti dan fitur yang menarik. Misalnya, tombol dapat memiliki transisi, dapat dinavigasi, dan dapat memiliki event handler OnClick.



Gambar 6.7 Pemilih jenis transisi.

On Klik ()

Ketika pengguna selesai mengagumi berbagai efek transisi tombol Anda, mereka akhirnya dapat mengklik beberapa tombol. Anda dapat menggunakan properti On Click () di bagian bawah Inspector untuk memanggil fungsi dari skrip dan untuk mengakses banyak komponen lainnya. Anda dapat meneruskan parameter ke metode apa pun yang Anda panggil, yang berarti perancang dapat mengontrol perilaku tanpa memasukkan kode. Penggunaan lanjutan dari fungsi ini mungkin untuk memanggil metode pada objek game atau membuat kamera melihat target secara langsung.

Menyortir Elemen

Sekarang Anda sudah familiar dengan berbagai elemen, ini saat yang tepat untuk menyebutkan bagaimana mereka digambar. Anda mungkin telah memperhatikan bahwa komponen Canvas yang Anda lihat sebelumnya pada jam ini memiliki properti Sorting Layer (seperti yang Anda lihat dengan gambar 2D di jam lain). Properti ini hanya digunakan untuk mengurutkan di antara beberapa kanvas dalam scene yang sama. Untuk mengurutkan elemen UI pada kanvas yang sama, Anda menggunakan urutan objek dalam tampilan Hierarki. Oleh karena itu, jika Anda ingin objek digambar di atas objek lain, Anda memindahkannya lebih rendah dalam tampilan Hierarki sehingga digambar nanti.

Presets

Unity 2018.1 menambahkan konsep preset komponen. Preset adalah properti komponen yang disimpan (seperti komponen Teks UI) yang dapat diterapkan untuk menyiapkan komponen baru dengan cepat. Menu preset berada di sudut kanan atas komponen, di sebelah roda pengaturan dalam tampilan Inspector. Sementara preset dapat bekerja untuk semua jenis komponen, mereka secara khusus disebutkan di sini, bukan sebelumnya, karena seberapa baik mereka bekerja dengan UI. Kasus penggunaan yang sangat umum adalah menginginkan semua teks dalam game Anda cocok. Anda tidak selalu ingin membuat semua teks Anda menjadi Prefab, tetapi Anda dapat dengan cepat menerapkan prasetel teks.

Canvas Render Modes

Unity menawarkan tiga opsi canggih untuk cara UI Anda dirender ke layar. Anda dapat memilih mode di Inspector dengan memilih Canvas dan memilih Render Mode. Anda kemudian melihat mode yang ditunjukkan pada Gambar 14.11. Penggunaan setiap mode kanvas sangat kompleks, jadi Anda tidak boleh mencoba menguasainya sekarang. Alih-alih, tujuannya di sini adalah untuk menjelaskan tiga mode (Ruang Layar–Overlay, Ruang Layar–Kamera, dan Ruang Dunia) sehingga Anda dapat memilih yang terbaik untuk gim Anda.

Gambar 6.8 Tiga mode render kanvas yang berbeda.

Ruang Layar–Hamparan

Screen Space–Overlay, yang merupakan mode default, adalah versi mode kanvas yang paling mudah digunakan dan juga paling tidak bertenaga. UI dalam mode Screen Space–Overlay menarik di atas semua yang ada di layar, terlepas dari pengaturan kamera atau posisi kamera di dunia. Faktanya, di mana UI ini muncul di tampilan Scene tidak memiliki hubungan dengan objek di dunia karena tidak benar-benar dirender oleh kamera. UI muncul di tampilan Scene pada posisi tetap, dengan kiri bawah di (0, 0, 0) di dunia. Scale UI berbeda dari scale

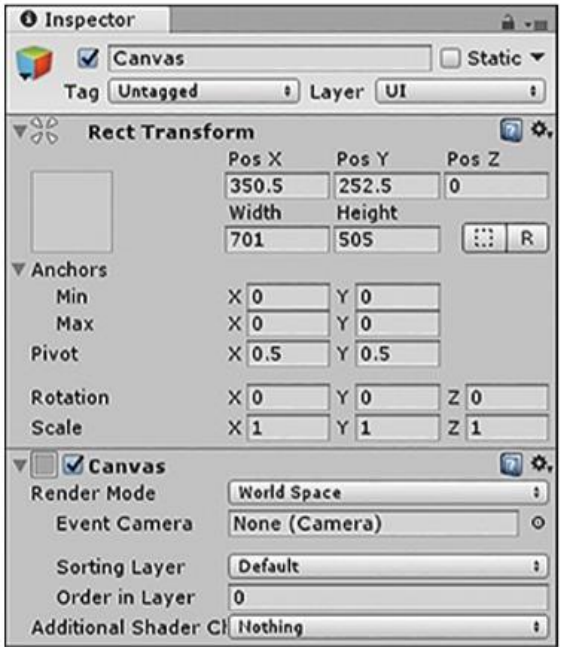
dunia, dan apa yang Anda lihat di kanvas berada pada scale 1 unit dunia untuk setiap piksel dalam tampilan Game Anda. Jika Anda menggunakan jenis UI ini di game Anda dan merasa tempatnya di dunia tidak nyaman saat Anda bekerja, Anda dapat menyembunyikannya untuk menghindarinya. Untuk melakukannya, Anda dapat mengklik drop-down Layers di editor dan menyembunyikan ikon mata di sebelah layer UI (lihat Gambar 6.9). UI kemudian disembunyikan di tampilan Scene saja (melalui itu akan tetap ada saat Anda menjalankan game Anda). Pastikan untuk tidak lupa mengaktifkannya kembali, atau Anda mungkin bingung mengapa UI Anda tidak muncul!

Gambar 6.9 Menyembunyikan UI

Screen Space–Camera

Screen Space–Mode Kamera mirip dengan Screen Space–Overlay, tetapi UI dirender oleh kamera pilihan Anda. Anda dapat memutar dan scaling elemen UI untuk membuat antarmuka 3D yang jauh lebih dinamis. Tidak seperti mode Screen Space–Overlay, mode ini menggunakan kamera untuk merender UI. Ini berarti efek seperti pencahayaan memengaruhi UI, dan objek bahkan dapat lewat di antara kamera dan UI. Ini dapat membutuhkan beberapa pekerjaan ekstra, tetapi hasilnya adalah antarmuka Anda dapat terasa lebih seperti bagian dari dunia. Perhatikan bahwa dalam mode ini, UI tetap dalam posisi tetap relatif terhadap kamera yang Anda pilih untuk merendernya. Menggerakkan kamera juga menggerakkan kanvas. Ini bisa menjadi ide yang baik untuk menggunakan kamera kedua hanya untuk merender kanvas Anda (jadi tidak menghalangi scene Anda yang lain).

Ruang Dunia Mode UI terakhir yang perlu dipertimbangkan adalah mode Ruang Dunia. Bayangkan sebuah museum virtual, di mana setiap objek yang Anda lihat memiliki informasi rinci tentang objek tepat di sebelahnya. Selanjutnya, informasi pop-up ini dapat menyertakan tombol untuk memungkinkan Anda membaca lebih lanjut atau pergi ke bagian lain museum. Jika Anda dapat membayangkannya, maka Anda hanya menggores permukaan dari apa yang dapat Anda lakukan dengan kanvas mode Ruang Dunia. Perhatikan bahwa transformasi Rect dari kanvas dalam mode World Space tidak lagi berwarna abu-abu, dan komponen Canvas itu sendiri dapat diedit dan diubah ukurannya. Karena dalam mode ini kanvas sebenarnya adalah objek game di dunia, kanvas tidak lagi digambar di sisa game Anda, seperti HUD. Sebaliknya, itu diperbaiki di dunia dan dapat menjadi bagian dari atau dicampur dengan objek scene Anda lainnya.



Gambar 6.10 Transformasi Rect, tersedia dengan mode World Space.

6.5 MENAMBAHKAN KONTROL SENTUH

Sebelum kita mulai membuat APK, sebaiknya tambahkan kontrol sentuh. Saat ini, Anda dapat menggunakan aplikasi Anda di perangkat Android dengan keyboard Bluetooth, tetapi itu bukan cara yang sangat nyaman bagi kebanyakan orang untuk bermain. Kebanyakan orang bahkan tidak memiliki keyboard Bluetooth. Anda ingin mereka dapat memainkannya hanya dengan ponsel mereka. Untungnya, menambahkan kontrol sentuh bukanlah proses yang terlalu rumit. Jika kami membuat game endless runner, menambahkan kontrol sentuh akan sangat sederhana. Dalam hal ini, yang perlu kita lakukan hanyalah menggunakan baris Input.GetMouseButtonDown(0) alih-alih Input.GetKey(KeyCode.Space) atau apa pun yang kita gunakan untuk melompat. Di Unity, klik mouse dan layar sentuh terdaftar sebagai hal yang sama persis, dan karena kita tidak perlu tahu di mana pada layar yang diketuk pengguna, ini akan lebih dari cukup untuk mengendalikan permainan kita. Kita akan melihat cara membuat endless runner di bab mendatang. Jika itu yang ingin Anda lakukan, Anda dapat melompat ke bagian selanjutnya tentang membangun APK. Jika tidak, tetap bersama saya dan kita akan melihat cara menerapkan kontrol sentuh yang tepat.

Mendesain Kontrol

Hal pertama yang ingin Anda lakukan adalah mendesain beberapa kontrol sentuh untuk melihat bagian saat ditempatkan di atas game Anda. Mereka harus jelas dan mudah ditemukan, tetapi juga penting bahwa mereka tidak mengalihkan perhatian player atau menutupi elemen penting dari permainan. Oleh karena itu, memilih sesuatu yang akan terlihat agak tembus pandang adalah pilihan yang baik. Penting juga bahwa tombolnya cocok dengan estetika dunia game Anda. Warna yang Anda pilih harus menonjol di antara levelnya tetapi tanpa berbenturan dengan gaya yang mencolok. Saat player Anda maju melalui gim Anda, adalah normal jika palet warna dunia gim Anda berubah: mungkin satu level diatur di bawah air dengan banyak warna biru dan hijau, dan level lain diatur dalam ruang dengan banyak hitam dan putih. Jika Anda membuat tombol Anda merah atau hijau, Anda akan menemukan bahwa tombol itu terkadang terlihat jelek di dunia game.

Untuk alasan ini, saya membuat tombol saya abu-abu terang dengan garis abu-abu tipis yang sedikit lebih gelap. Saya juga menerapkan filter pixelized dengan editor gambar

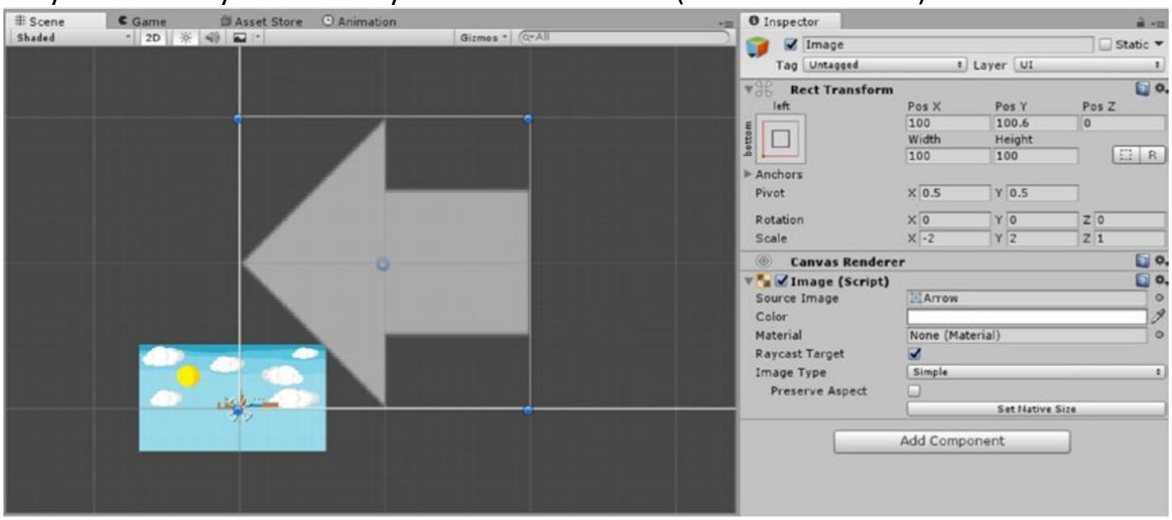
GIMP dan mengatur opacity menjadi 80 persen. Hasilnya harus sesuatu yang tidak akan terlihat terlalu mengganggu dan tidak akan terasa tidak pada tempatnya.

Gambar 6.11 Sebuah panah

Gambar 6.12 Sebuah tombol

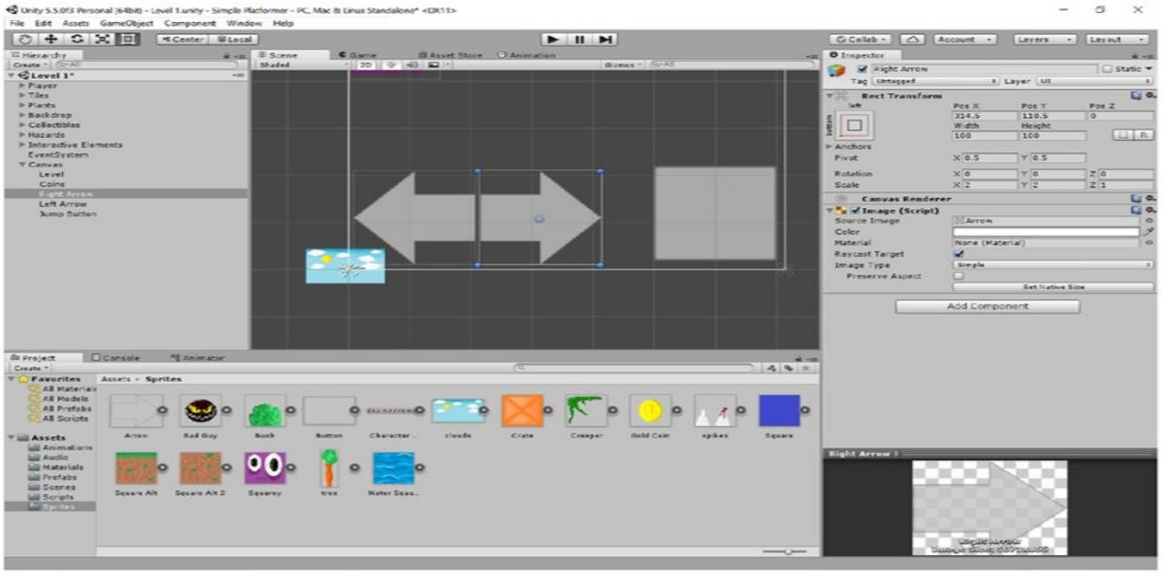
Perhatikan bahwa saya hanya perlu membuat satu panah arah. Itu karena saya cukup membalikkan gambar untuk membuat panah yang berlawanan—tidak perlu menghabiskan waktu menggambar dua.

Menambahkan Kontrol Kita Sekarang kita perlu menambahkan ini ke game kita dan membuatnya melakukan sesuatu. Pertama, tambahkan gambar ke folder Sprite proyek Anda seperti yang Anda lakukan lainnya. Sekarang klik kanan di Hierarchy Anda di suatu tempat di bawah kanvas Anda—Anda ingin elemen baru ini menjadi anak dari Canvas GameObject—dan pilih UI Gambar. Sebuah gambar akan muncul di game Anda yang mungkin akan terlihat seperti kotak putih besar. Pilih elemen ini dan di mana dikatakan Sumber Gambar, drag dan jatuhkan sprite panah yang Anda buat dari folder Sprite Anda. Di mana dikatakan Jangkar, pilih Kiri Bawah. Drag dan posisikan panah sehingga berada di sudut kiri kanvas (yang mungkin akan tampak masif pada titik ini) lalu ubah scale horizontal menjadi angka negatif sehingga panah mengarah ke kiri, bukan ke kanan. Dengan kata lain, ubah Lebar dari 1 menjadi -1 sehingga akan terlipat kembali dengan sendirinya. Bergantung pada seberapa besar Anda menggambar panah, Anda perlu bermain-main untuk memastikan bahwa gambar-gambar ini berukuran tepat. Anda dapat mengubah ini nanti setelah APK berjalan di ponsel Anda, tetapi untuk saat ini saya telah menyetel milik saya ke X = –2 dan Y = 2 (lihat Gambar 6.13).



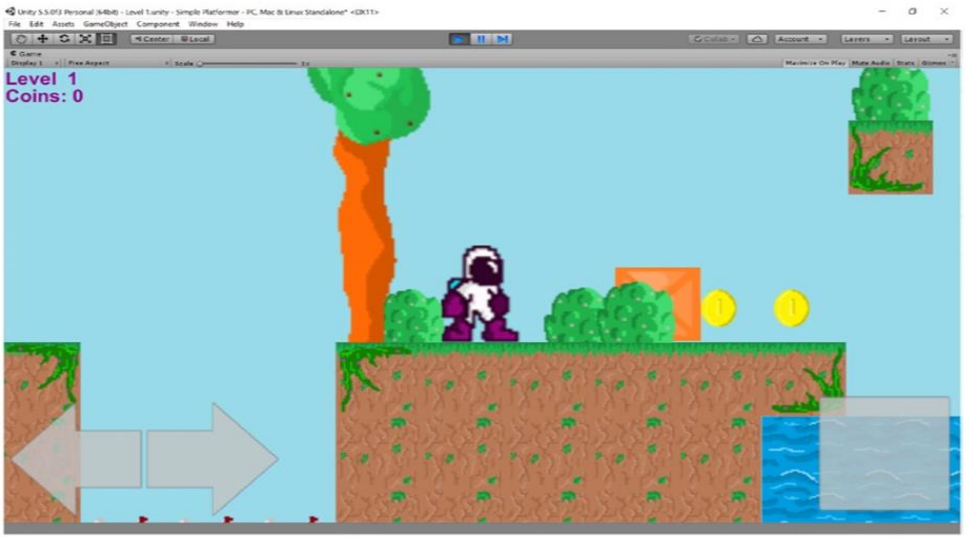
Gambar 6.13 Memposisikan kontrol pertama

Sekarang lakukan hal yang sama untuk panah kedua. Kali ini posisinya akan sedikit ke kanan, dan jangkar akan tetap berada di kiri bawah. Tentu saja, Scale kali ini akan berada dalam angka positif. Setelah itu, Anda dapat menambahkan tombol lompat, yang akan menjadi gambar "tombol" generik kami. Yang ini akan ditambatkan ke kanan bawah layar. Lihat Gambar 6.13. Ganti nama tombol yang sesuai. Apa yang akan Anda temukan adalah bahwa tombol lompat dan panah kanan mungkin tumpang tindih pada titik ini atau hanya terlihat sangat berdekatan (seperti pada Gambar 6.14), tetapi Anda tidak perlu khawatir tentang itu. Dengan mengatur gambar untuk berlabuh ke sudut bawah layar, Anda menyatakan bahwa semua informasi posisi akan relatif terhadap sudut itu. Unity tidak tahu berapa ukuran layar ponsel atau perangkat apa pun yang akan Anda mainkan, dan dengan demikian kanvas cenderung berbentuk agak aneh. Tetapi selama tombol lompat diatur pada jarak tertentu dari sudut kanan dan hal yang sama berlaku untuk panah dan sudut kiri, mereka harus berada di posisi yang benar setelah Anda menekan tombol play.'



Gambar 6.14 Tombolnya belum terlihat benar, tetapi percayalah

Tentu saja, untuk preview, Anda dapat menekan tombol rotate dan melihat tampilannya di layar komputer Anda (Gambar 6.15). Saat memposisikan panah Anda, ada baiknya meninggalkan sedikit ruang di sekitar tepinya untuk memastikan mereka tidak terlalu sempit.



Gambar 6.15 Lihat? Panah kami terlihat bagus!!

Mengkodekan Kontrol Sekarang setelah Anda memiliki tombol, saatnya untuk membuatnya benar-benar melakukan sesuatu. Dengan mengingat hal itu, kita perlu membuat GameObject kosong yang akan bertindak seperti wadah untuk elemen-elemen ini. Klik kanan kanvas Anda, pilih Buat Kosong, lalu jangkar objek baru ini ke bagian bawah layar. Klik Stretch sehingga menjadi selebar layar dan kemudian drag elemen ke sini di Hierarchy. Panggil TouchController wadah baru Anda. Masuk ke skrip Player Anda (yang, seperti yang kita pelajari di Bab 6, benar-benar sebuah kelas) dan kita akan menambahkan dua Boolean publik. Ingat, bool adalah variabel yang bisa benar atau salah—1 atau 0—dan karena bersifat publik, variabel tersebut dapat diakses oleh kelas (skrip) lain dalam game kita. Variabel baru ini akan disebut moveRight dan moveLeft, dan Anda akan menggunakannya untuk melakukan ini (jangan tempelkan kode ini dulu):

```
if (moveright)
{
rb.velocity = new Vector2(movespeed, rb.velocity.y);
}
if (moveleft) { rb.velocity = new Vector2(-movespeed, rb.velocity.y);
}
```

Perhatikan bahwa ini melakukan sesuatu yang sangat mirip dengan menekan panah kanan dan kiri secara manual. Cara kerja elemen gambar yang dapat disentuh ini adalah mereka hanya mengizinkan kita untuk mendaftar saat mereka disadap dan saat dirilis. Itu berarti kami tidak dapat menanyakan Unity apakah tombolnya "sedang ditekan". Sebagai gantinya, kita perlu mengatur Boolean kita menjadi benar atau salah berdasarkan kapan tombol diketuk dan dilepaskan. Alasan saya memberi tahu Anda untuk tidak menempelkan kode itu dulu adalah karena ada cara yang lebih mudah untuk melakukan ini. Kami sudah memiliki banyak kode untuk menangani player yang berjalan ke kiri dan ke kanan, dan saat ini mencakup hal-hal seperti animasi—jadi kami tidak ingin mengulanginya. Sebagai gantinya kita akan menggunakan perintah yang disebut OR. Ini pada dasarnya memungkinkan kita untuk bertanya apakah salah satu dari dua hal sedang terjadi. Dalam hal ini, kami menanyakan apakah player menekan tombol panah atau salah satu Boolean kami benar. Dalam C#, kita menulis OR menggunakan simbol ||. Jadi, kode kita akan terlihat seperti ini sekarang:

```
if (moveLeft || Input.GetKey(KeyCode.LeftArrow))
        {
        rb.velocity = new Vector2(-movespeed, rb.velocity.y);
        anim.SetBool("Walking", true);
        if (facing == 1)
        {
        transform.localScale = new Vector3(-1f, 1f, 1f);
        facing = 0;
        }
         } else if (moveRight || Input.GetKey(KeyCode.RightArrow))
        {
        rb.velocity = new Vector2(movespeed, rb.velocity.y);
        anim.SetBool("Walking", true);
        if (facing == 0)
        {
        transform.localScale = new Vector3(1f, 1f, 1f);
        facing = 1;
        }
        } else
```

```
{
anim.SetBool("Walking", false);
}
```

Sekarang, ketika Anda menekan kanan dan kiri, karakter Anda harus tetap bergerak. Tetapi jika Anda menyetel salah satu Boolean ke true (ingat, semua variabel sama dengan 0 secara default saat pertama kali dibuat—yaitu, false), maka player akan bergerak secara otomatis. Demikian juga, saya ingin Anda memindahkan kode yang menangani aksi lompatan karakter player Anda ke metode publik baru. Metode publik adalah metode—potongan kode instruksional—yang dapat dieksekusi dari kelas lain (skrip). Itu berarti kita sekarang dapat memaksa player untuk melompat dengan mengaktifkannya dari skrip eksternal. Kami masih ingin mendaftarkan penekanan tombol dalam metode Pembaruan kami, tetapi alih-alih memasukkan kode lompatan, kami merujuk metode publik baru yang berisi kode tersebut.
Jadi, Anda akan membuat metode publik seperti ini:

```
public void jump() {
   if (onGround) {
   rb.velocity = new Vector2(rb.velocity.x, jumppower);
   }
   }
```

Dan kemudian dari dalam metode Perbarui, Anda cukup mengatakan ini:

```
if (Input.GetKey(KeyCode.Space)) {
jump(); }
```

The whole thing should look like this:

```
void Update() {
if (moveLeft || Input.GetKey(KeyCode.LeftArrow))
{
rb.velocity = new Vector2(-movespeed, rb.velocity.y);
anim.SetBool("Walking", true);
if (facing == 1)
{
transform.localScale = new Vector3(-1f, 1f, 1f);
facing = 0;
}
} else if (moveRight || Input.GetKey(KeyCode.RightArrow))
{
rb.velocity = new Vector2(movespeed, rb.velocity.y);
anim.SetBool("Walking", true);
if (facing == 0)
{
transform.localScale = new Vector3(1f, 1f, 1f);
facing = 1;
}
} else
{
anim.SetBool("Walking", false);
}
if (Input.GetKey(KeyCode.Space))
{
```

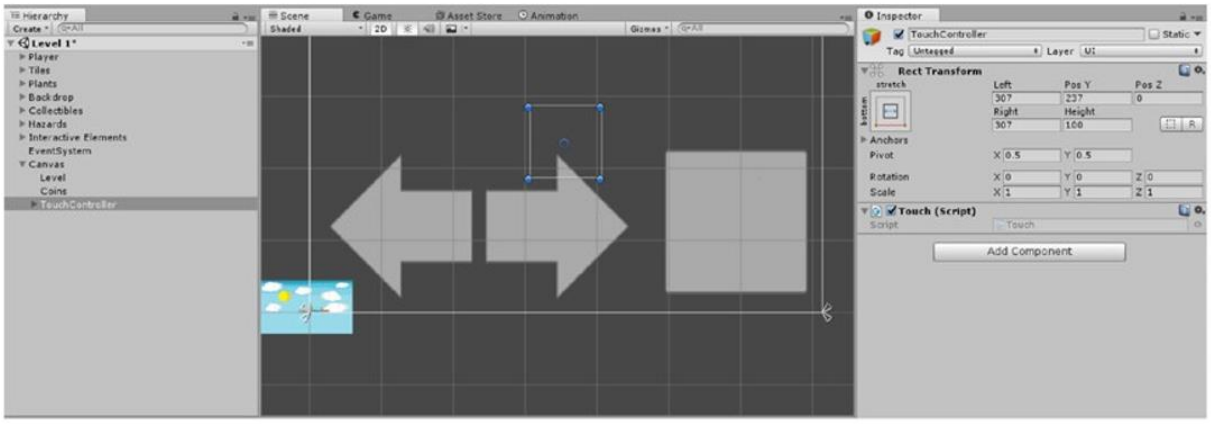
```
jump();
}
}
public void jump() {
if (onGround) { rb.velocity = new Vector2(rb.velocity.x, jumppower);
}
}
```

Ini penting karena kami pada dasarnya telah memberi diri kami titik akses yang dapat kami gunakan untuk mengontrol pemutar dari luar skrip. Dan kami akan menggunakannya dari dalam skrip yang akan mengontrol tombol kami. Jika Anda kesulitan memahami apa yang terjadi di sini, pertimbangkan untuk membaca kembali bagian pemrograman berorientasi objek (OOP) di Bab 6. Anda mengerti? Itu selalu baik untuk mempelajari hal-hal teori saat Anda pergi. Oke, jadi sekarang kita sudah melakukannya. Saatnya membuat tombol responsif. Mulailah dengan membuat skrip baru lainnya, kali ini bernama Touch. Touch akan berisi kode berikut:

```
public class Touch : MonoBehaviour {
private Player player;
void Start()
{
        player = FindObjectOfType<Player>();
        }
        public void PressLeftArrow()
        {
        player.moveRight = false;
        player.moveLeft = true;
        }
        public void PressRightArrow()
        {
        player.moveRight = true;
        player.moveLeft = false;
        }
        public void ReleaseLeftArrow()
        {
        player.moveLeft = false;
        }
        public void ReleaseRightArrow()
        {
        player.moveRight = false;
        }
        public void Jump()
        {
        player.Jump();
        }
        }
```

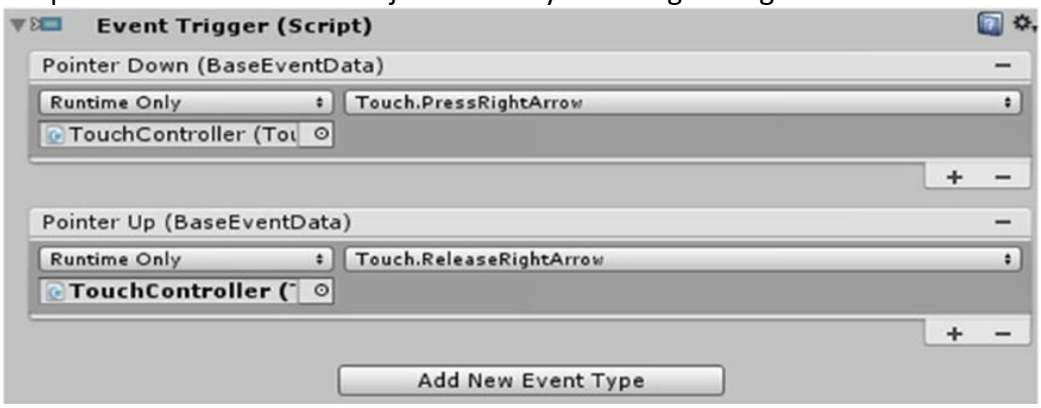
Ini pada dasarnya adalah kumpulan metode publik, yang masing-masing akan berinteraksi dengan skrip Player (kelas) dalam satu atau lain cara. Seperti yang mungkin sudah Anda duga,

sekarang kita akan membuatnya sehingga setiap tombol di layar memicu salah satu metode ini. Sekarang kembali ke Unity dan tambahkan skrip Touch baru ini sebagai komponen dari GameObject kosong TouchController yang kita buat sebelumnya.



Gambar 6.16 Tambahkan skrip Touch ke GameObject yang kosong

Sekarang kita akan menambahkan komponen ke panah kiri—kali ini jenis baru yang disebut Pemicu Peristiwa. Pergi ke Add Component➤ Event➤ Event Trigger. Sekarang klik Add New Event Type➤ Pointer Down. Klik simbol plus (+) kecil yang muncul di sebelah kanan dan kemudian ambil GameObject TouchController dan drag ke dalam kotak None (Object). Kemudian klik menu tarik-turun di sebelah kanan dan pilih Touch ➤ Tekan LeftArrow(). Pada dasarnya, Anda memberi tahu Unity bahwa Anda ingin event Pointer Down (tindakan menekan tombol) untuk memicu metode publik Press Left Arrow yang ada di dalam skrip Touch. Klik Add New Event Type lalu pilih Pointer Up. Ini mencatat tindakan jari yang diangkat dari panah. Sekarang pilih Touch➤ ReleaseLeftArrow() untuk masuk ke sini. Seharusnya terlihat seperti Gambar dibawah ini jika semuanya berfungsi dengan baik.

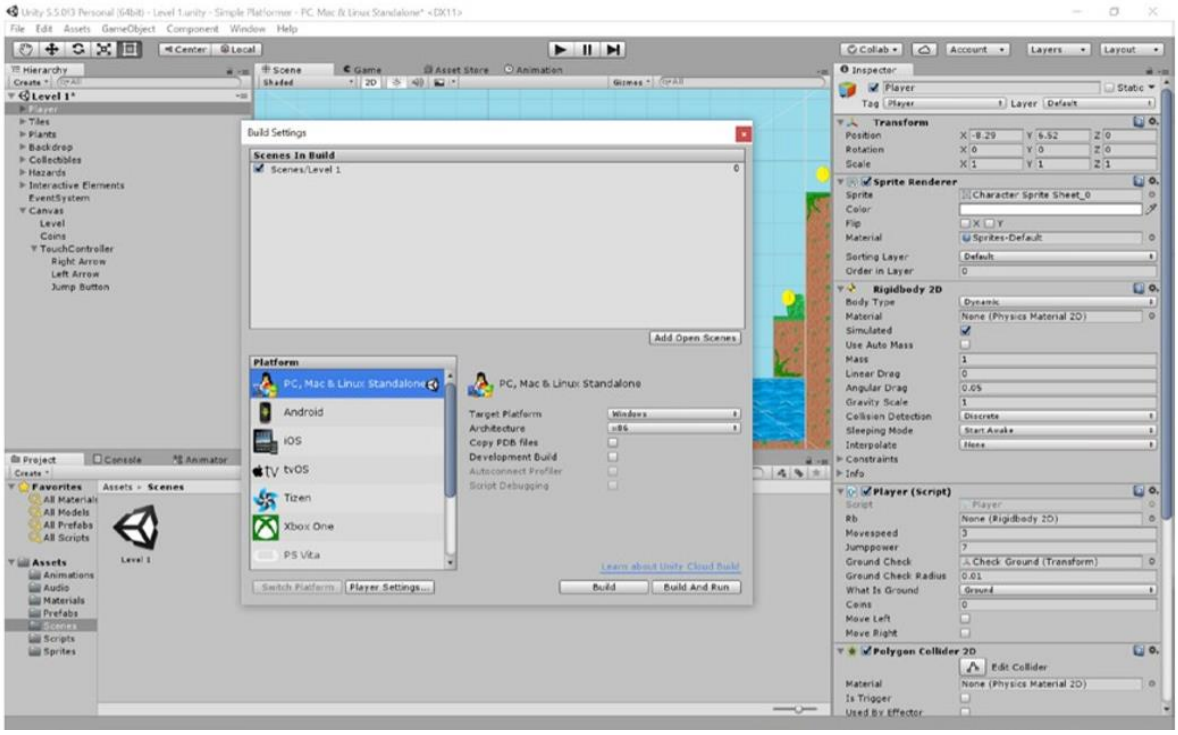


Gambar 6.17 Pemicu event ditambahkan

Seperti yang mungkin sudah Anda duga, Anda perlu melakukan hal yang sama untuk panah kanan tetapi menggunakan metode panah kanan masing-masing. Untuk tombol lompat, Anda akan melakukan sesuatu yang sedikit berbeda dengan mengabaikan jenis event penunjuk ke atas dan memilih metode Jump() untuk penunjuk ke bawah. Klik mainkan dan Anda harus dapat menguji ini. Jika Anda tidak memiliki laptop layar sentuh untuk mencobanya, maka cukup klik tombol dengan mouse Anda akan melakukan trik yang sama. Jangan khawatir jika saat ini tidak terasa sangat responsif—ini akan menjadi cerita yang berbeda setelah dijalankan di perangkat Android. Ngomong-ngomong soal....

6.6 MEMBUAT APK PERTAMA ANDA

Sekarang setelah Anda memiliki bentuk input yang sesuai, Anda akhirnya dapat benar-benar menguji semua kerja keras Anda di perangkat Android. Pertama, pastikan Anda telah menyimpan scene Anda lagi dengan menekan Ctrl+S. Selanjutnya, buka File➤Build Settings. Anda akan melihat kotak di bagian atas jendela yang bertuliskan Scenes In Build—ini pada dasarnya menunjukkan scene mana yang Anda buat yang ingin Anda sertakan dalam produk akhir. Untuk menambahkan Level 1 Anda, cukup drag dari folder Scenes Anda di jendela Project dan letakkan di area Scenes in Build. Seharusnya terlihat seperti Gambar 6.18.

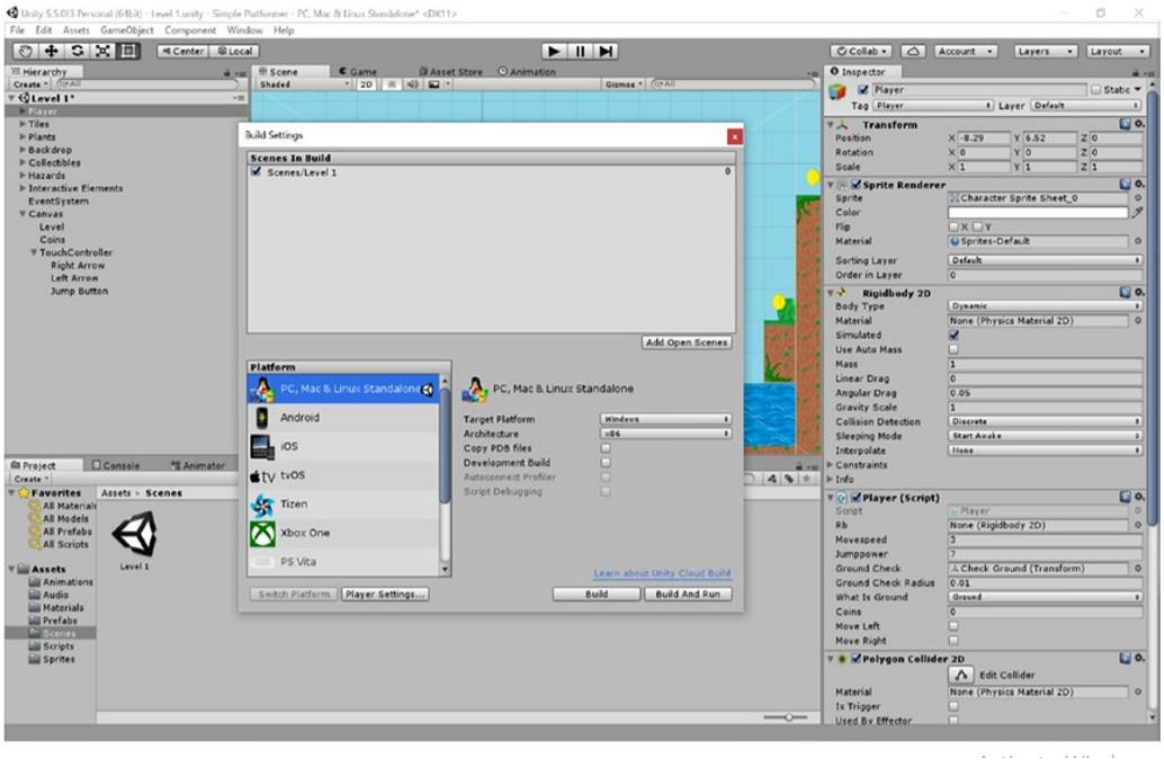


Gambar 6.18 Level 1 saat ini adalah satu-satunya scene di build kami

Saat Anda memiliki lebih banyak scene (dan Anda akan melakukannya), Anda harus memastikan bahwa scene di atas adalah scene yang ingin Anda jalankan terlebih dahulu. Itu biasanya berarti layar splash atau menu (tapi ingat, jika Anda memiliki lisensi Unity gratis, layar splash Anda akan didahului oleh Unity). Untuk saat ini, jangan khawatir tentang kompresi tekstur. Itu berguna untuk membuat game 3D dan akan membantu Anda mengoptimalkan aplikasi Anda. Untuk tujuan kami saat ini, itu tidak perlu (aplikasi kami cukup kecil dan tidak terlalu intensif sumber daya), dan tidak semua jenis kompresi didukung oleh semua platform Android. Saya membahas kompresi tekstur lebih lanjut di buku ini. Anda akan melihat bahwa jendela ini juga memiliki opsi untuk memilih platform, dan saat ini sepertinya tertulis PC, Mac & Linux Standalone. Anda perlu mengubahnya dengan mengklik opsi Android dan kemudian mengklik Switch Platform.

Pengaturan Player

Selanjutnya, klik tombol Pengaturan Player di bawah kotak gulir Platform dan Anda akan menemukan bahwa beberapa opsi baru terbuka untuk Anda mainkan di Inspector. Di sinilah Anda dapat menentukan banyak properti APK yang akan Anda buat: hal-hal seperti ikon, nama paket, dan orientasi



Gambar 6.19 Pengaturan Player adalah tempat Anda mengatur properti untuk APK baru Anda

Sebelum kita masuk ke pengaturan, isi opsi di atas. Di sini, Anda dapat memasukkan nama untuk perusahaan Anda dan nama aplikasi Anda. Jika Anda membiarkan ini apa adanya, maka perusahaan akan menjadi DefaultCompany, dan aplikasi akan dipanggil apa pun yang Anda sebut Proyek Anda. Ada juga opsi untuk menambahkan ikon di sini. Kami tidak akan mengkhawatirkan hal itu sekarang—kami akan membahasnya lagi saat membahas pengunggahan dan pemasaran aplikasi Anda. Untuk saat ini, kami akan tetap menggunakan ikon Unity default. Sekarang, apa yang dilakukan semua opsi ini?

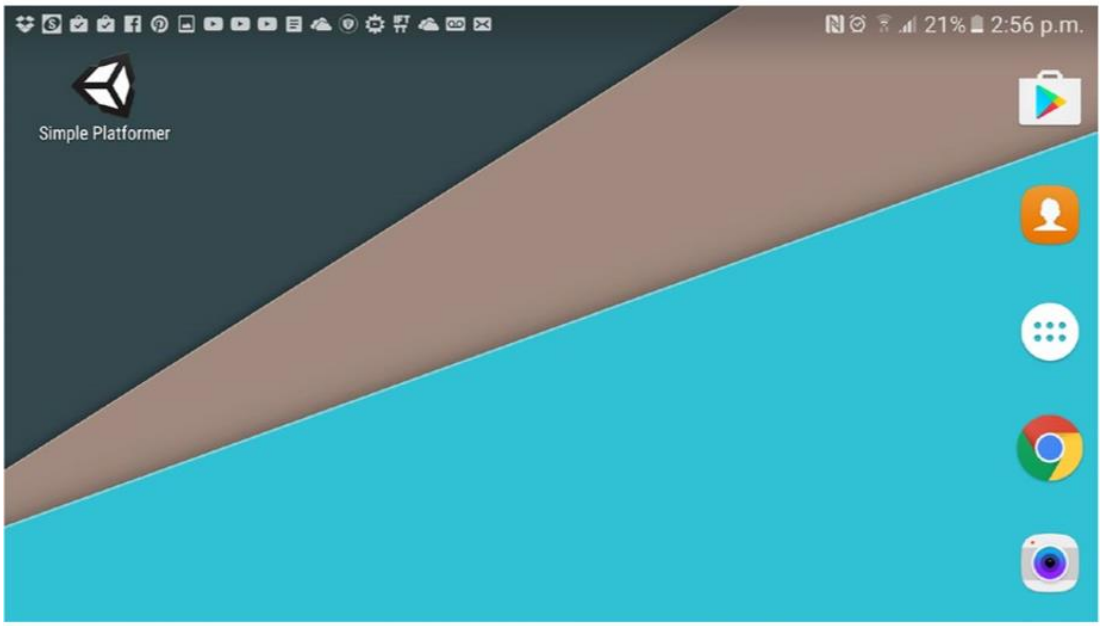
Resolusi dan Presentasi

Hal pertama yang akan kita lihat adalah Resolusi dan Presentasi. Saat ini, Orientasi Default mungkin diatur ke Rotasi Otomatis, dan di bawahnya ada kotak centang yang menunjukkan orientasi mana yang diizinkan—saat ini, jawabannya mungkin semuanya. Jika Anda ingin membuat game puzzle (dibahas dalam bab mendatang), ada kemungkinan Anda ingin mendukung orientasi potret. Bahkan ada sejumlah kecil game aksi potret seperti Fotonica dan Sonic Jump. Tetapi sebagian besar, lebih masuk akal untuk tetap menggunakan lanskap, yang akan mencegah player Anda merasa terlalu sempit dan yang akan menampilkan layar paling banyak. Controller yang menahan ponsel saat Anda bermain juga cenderung hanya mendukung lanskap. Jadi, pilih orientasi di kotak Orientasi Default atau hapus centang pada dua opsi potret di bawah

Ikon

Bagian selanjutnya adalah bagian Ikon. Seperti yang saya katakan sebelumnya, ikon adalah sesuatu yang akan kita bahas nanti, tetapi seperti yang Anda lihat, ada ruang di sini untuk menambahkan ikon dengan berbagai resolusi berbeda. Jika Anda ingin meletakkan sesuatu di sini, tidak apa-apa menggunakan satu gambar saja, dalam hal ini lebih baik menggunakan yang beresolusi lebih tinggi daripada yang beresolusi lebih rendah. Upscaling menghasilkan kualitas gambar yang lebih baik daripada downscaling. Saya akan membahas ini

lebih detail nanti—tidak apa-apa membiarkan ini kosong untuk saat ini. Gambar 7-10 menunjukkan tampilan ikon default setelah dipasang.



Gambar 6.20 Segera ini akan menjadi aplikasi Anda

Gambar percikan

Selanjutnya adalah Splash Image, yang akan kami biarkan kosong lagi—terutama karena Anda seharusnya membiarkan gambar default di sini dengan lisensi gratis.

Pengaturan Lain

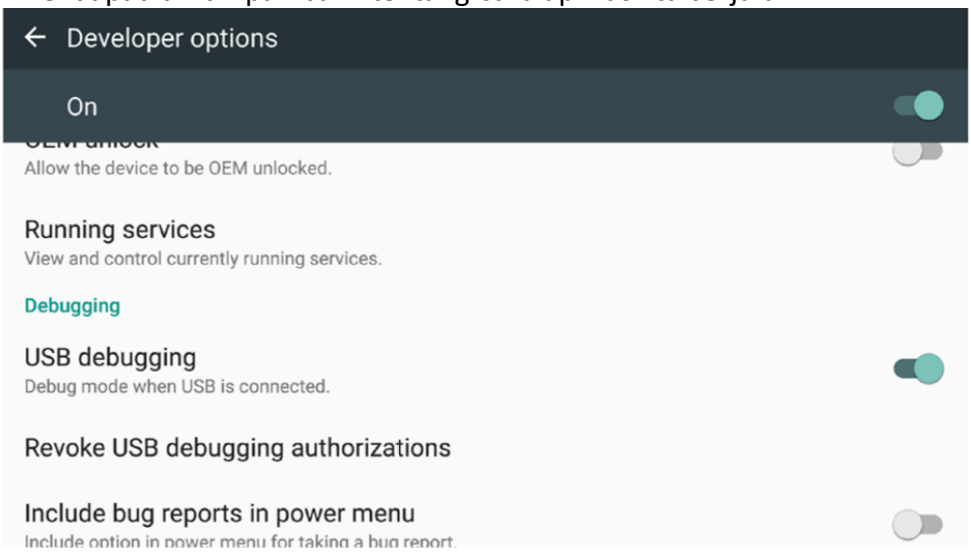
Pengaturan lain memberi kita banyak hal untuk dimainkan. Anda dapat mengubah pengaturan yang berkaitan dengan rendering, serta level API minimum, izin menulis, lokasi pemasangan, dan lainnya. Sebagian besar dari ini cukup jelas, dan sisanya akan kita bahas nanti di bab selanjutnya.

Anda baik-baik saja untuk melewati bagian ini dan membiarkan semuanya sebagai default lagi, tetapi satu atau dua hal mungkin layak untuk dilihat di sini. Bundle Identifier, misalnya, adalah tempat Anda memasukkan nama paket Anda. Nomenklatur yang benar untuk ini adalah sebagai berikut: com.YourCompanyHere.YourAppNameHere. Anda harus menyetel ini sebelum aplikasi Anda dibuat, jadi lanjutkan dan masukkan nama paket menggunakan detail Anda sendiri. Tidak masalah apa yang Anda pilih untuk saat ini, tetapi pikirkan baik-baik sebelum Anda mempublikasikannya. Terlepas dari hal lain, setelah aplikasi diunggah ke Play Store, Anda tidak akan dapat mengubahnya lagi.

Versi dan kode versi masing-masing untuk kepentingan kami dan Android. Versi adalah versi seperti yang kami lihat dan seperti yang dilihat pengguna kami. Namun, kode versi perlu diubah setiap kali Anda memperbarui aplikasi di Play Store. Bahkan jika Anda membuat perubahan terkecil dan kemudian mengunggah APK baru, Anda harus memastikan bahwa kode versi baru lebih tinggi dari yang terakhir. Sementara itu, API level minimum menentukan versi Android terendah yang ingin Anda dukung. Secara default, ini mungkin disetel ke Android 2.3.1 (Gingerbread). Pada saat penulisan, Google baru saja merilis preview developer untuk Android O, dan versi terbaru yang tersedia untuk pengguna adalah 7.1 (Nougat). Semakin rendah Anda membuat level API, semakin banyak orang yang dapat mengunduh aplikasi Anda. Tetapi jika Anda membuatnya terlalu rendah, Anda tidak akan dapat mengakses beberapa fitur Android yang lebih baru. Sekali lagi, saya membahas semua ini secara lebih rinci nanti dalam buku ini.

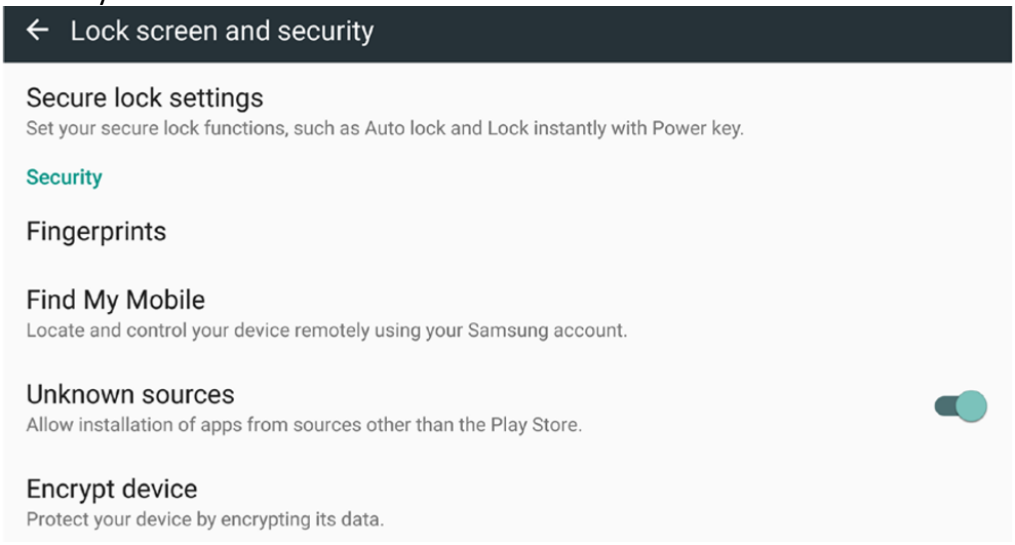
6.7 MEMPERSIAPKAN PONSEL ANDA

Satu hal lagi yang harus dilakukan sebelum Anda dapat mencoba menjalankan game di ponsel adalah menyiapkan ponsel yang dimaksud. Pertama, itu berarti Anda harus mengizinkan USB debugging. Sayangnya, saya tidak dapat memberikan petunjuk langkah demi langkah, karena setiap ponsel Android berbeda (itulah keajaiban dan frustrasi bekerja dengan Android). Debug USB memungkinkan Anda menginstal aplikasi melalui koneksi USB dan kemudian mendapatkan umpan balik tentang cara aplikasi itu berjalan



Gambar 6.21 Mengizinkan debugging USB

Biasanya, opsi ini dapat ditemukan di bagian dalam pengaturan ponsel Anda yang disebut Opsi Developer. Di beberapa ponsel ini disembunyikan, jadi lakukan pencarian Google untuk mengetahui cara mengaktifkan USB debugging pada hardware khusus Anda. Gambar 7-11 menunjukkan opsi ini di perangkat Samsung Galaxy. Pengaturan lain yang perlu Anda ubah adalah yang mengatakan "Izinkan pemasangan aplikasi dari sumber selain Play Store." Ini biasanya memiliki judul Sumber Tidak Dikenal dan dapat ditemukan di bagian Aplikasi di pengaturan Anda, atau di bagian Layar Kunci dan Keamanan. Sekali lagi, pencarian Google cepat akan membantu Anda. Seperti yang Anda harapkan, pengaturan ini memastikan bahwa ponsel Anda akan menerima APK dari sumber lain—seperti PC Anda—jadi Anda harus mengaktifkannya.



Gambar 6.22 Anda perlu mencentang opsi Sumber Tidak Dikenal

Terakhir, pastikan Anda telah mengatur driver di komputer untuk ponsel Anda. Ini kemungkinan akan terjadi saat pertama kali Anda menghubungkannya untuk mentransfer foto, tetapi untuk berjaga-jaga, Anda mungkin perlu melakukan pencarian lain dan mengambil file driver tersebut untuk ponsel Anda. Tetapi jika Anda tidak dapat mengetahuinya, ada cara lain untuk mengaktifkan dan menjalankan aplikasi di ponsel Anda.

Tarik Pemicu

Sekarang yang tersisa untuk dilakukan adalah membangun aplikasi Anda dan menjalankannya. Silakan dan colokkan telepon ke PC Anda melalui port USB lalu tekan Build and Run. Jika semuanya berjalan sesuai rencana, Unity akan membangun APK dan kemudian menginstalnya di ponsel Anda. Setelah menampilkan layar splash, itu akan langsung muncul di depan Anda. Kesuksesan!

Jika semuanya berjalan sesuai rencana, Anda sekarang harus memiliki versi aplikasi yang berfungsi di ponsel Anda dengan kontrol sentuh. Anda mungkin menemukan bahwa elemen UI agak kecil, jadi pindahkan ini dan ubah ukurannya sesuai keinginan Anda.

Gambar 6.23 UI itu perlu sedikit lebih besar

Gunakan momen ini untuk menikmati pencapaian Anda. Anda baru saja membuat aplikasi Android pertama yang berfungsi. Silakan dan tunjukkan pada Ibu.
Tapi jangan terlalu senang dengan diri sendiri—perjalanan masih panjang.

Gambar 6.24 Kita berhasil!

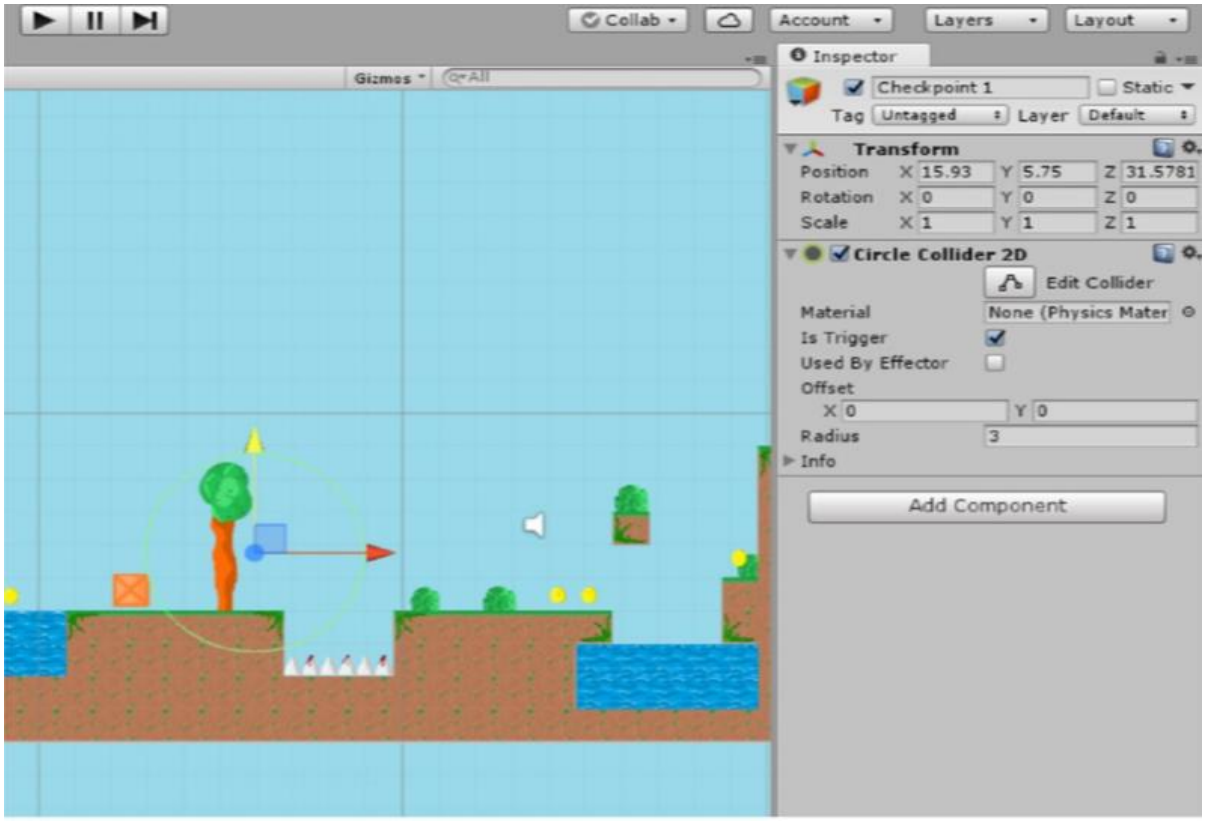
6.8 MEMPERLUAS DUNIA GAME DENGAN CHECKPOINT, LEVEL, DAN SIMPAN FILE

Banyak dari buku ini adalah opsional. Sungguh, Anda sudah berada di titik di mana Anda bisa membuat game yang hampir selesai. Sekarang berjalan di Android, memiliki animasi dan suara, dan dengan mengekstrapolasi dari apa yang telah Anda pelajari, Anda mungkin dapat membuat banyak elemen baru dan membungkusnya menjadi game "lengkap". Tentu saja, saya harap Anda akan tetap bersama saya sampai akhir, karena menurut saya ini akan menghasilkan produk jadi yang lebih baik dan lebih banyak pengetahuan coding untuk Anda. (Selain itu, Anda akan belajar cara membuat aplikasi realitas virtual.) Dengan demikian, setidaknya masih ada beberapa elemen yang belum kami bahas yang cukup wajib jika Anda ingin game Anda terasa lengkap. Itulah yang tercakup dalam bab ini. Pertama, jika level Anda akan lebih dari beberapa platform, Anda perlu memperkenalkan checkpoint sehingga player Anda tidak frustrasi karena terus-menerus dikirim kembali ke awal. Kedua, Anda mungkin juga ingin membuat lebih dari satu level dan menemukan cara untuk bertransisi di antara mereka. Dan jika Anda memiliki lebih dari satu level, Anda akan memerlukan semacam sistem pemilihan level (menu) dan cara untuk menyimpan kemajuan player. Ini benar-benar satu-satunya hal penting yang tersisa untuk Anda pelajari sebelum Anda dapat menyelesaikan permainan yang menantang dan menyenangkan. Jadi mari kita lakukan.

Menambahkan Checkpoint

Sebelum Anda mulai menambahkan checkpoint, masuk akal untuk membuat level Anda sedikit lebih lama. Ini adalah bagian yang menyenangkan, jadi salin dan tempel beberapa sprite tanah lagi, tambahkan lebih banyak paku dan kolam air, dan biarkan imajinasi Anda menjadi liar. Dalam bab 10, kita akan membahas apa yang membuat desain level bagus—jadi jangan terlalu banyak menghabiskan waktu dan usaha untuk ini dulu. Pertimbangkan desain level ini sebagai pengganti yang hanya akan memberi Anda sesuatu untuk dijalankan saat Anda bermain.

Level 1 saya sangat datar dan horizontal, menjaga hal-hal sederhana untuk player baru. Sekarang kita akan membuat checkpoint pertama kita. Ini hanya akan menjadi GameObject kosong yang akan kita masukkan ke dalam game di berbagai lokasi. Dan coba tebak apa yang akan kita sebut yang pertama? Checkpoint 1. Cara paling jelas untuk menangani penempatan checkpoint adalah dengan menempatkannya tepat sebelum player menghadapi tantangan baru yang signifikan atau tepat setelahnya. Kemudian dalam permainan, kami dapat mencoba menggabungkan beberapa bahaya untuk membuat urutan tantangan dan dengan demikian meningkatkan sedikit kesulitan. Tetapi untuk saat ini, mari kita mulai dengan satu checkpoint baru untuk memulai dan menempatkannya tepat sebelum lubang paku pertama. Anda ingin menjadikan objek kosong ini sebagai pemicu, yang, mungkin Anda ingat, berarti kami dapat mendeteksi ketika seseorang melewati sangkar collision tetapi tanpa menabrak sesuatu. Pertama, beri dia penumbuk lingkaran lalu centang kotak IsTrigger di Inspector. Ini berarti kami dapat mengetahui kapan pengguna berjalan melewati checkpoint, tetapi mereka tidak akan lebih bijaksana. Buat objek ini cukup besar, karena penting agar player tidak melewati checkpoint secara tidak sengaja. Anda dapat melakukannya menggunakan alat pengubahan ukuran atau dengan memasukkan Radius di Inspector. Milik saya adalah 3, yang lebih dari cukup besar untuk mencegahnya dielakkan.



Gambar 6.25 Lihatlah checkpoint 1

Seperti yang mungkin sudah Anda duga, saatnya untuk sedikit lebih banyak skrip (kelas). Jadi buat skrip dan beri nama Checkpoint. Kami juga akan mengedit skrip Player kami, jadi buka juga di Visual Studio.

Scripting a More Fitting Death

Sebenarnya, kita akan mengedit script Player 1 terlebih dahulu. Jika Anda membaca bagian tentang pemrograman berorientasi objek di Bab 6, maka Anda tahu bahwa karakter Player kita sebenarnya adalah konstruksi yang disebut objek. Objek ini memiliki properti (variabel) dan metode (perilaku), dan melalui ini objek kita yang lain dapat berinteraksi dengannya. Jika kita ingin memindahkan pemutar kita, masuk akal untuk melakukannya dengan memanipulasi variabel dalam skrip Player. Kita mulai dengan membuat dua float publik lagi: startx dan starty. Ini akan menjadi titik awal player kami saat respawn. Awal permainan secara efektif adalah checkpoint pertama kami, jadi hal pertama yang harus kami lakukan untuk menelurkan player kami di awal permainan adalah mencari tahu di mana mereka berada di dunia sehingga kami dapat mengirim mereka kembali ke titik yang tepat ini masing-masing. waktu mereka mati. Saat ini, kami mengirim player kembali ke set koordinat tertentu yang kami edit, dan jika kami memindahkan player dalam tampilan Scene, kami harus memperbarui angka-angka ini setiap saat. Ini akan menjadi masalah yang lebih besar ketika kita mulai membuat beberapa level dan menggunakan skrip yang sama.

Apa yang akan kita lakukan adalah memeriksa posisi player saat objek pertama kali dibuat dan membuatnya menjadi posisi respawn. Untuk melakukan ini, Anda hanya perlu menambahkan sedikit kode berikut ke metode Mulai:

```
startx = transform.position.x;
starty = transform.position.y;
```

Kode sebelumnya mendapatkan posisi GameObject saat pertama kali dibuat dan menyimpan koordinat X dan Y secara terpisah. Sekarang temukan metode Kematian Anda. Ini akan ada di skrip Bahaya Anda atau skrip Player Anda (jika Anda salah satu siswa terbaik saya dan Anda memindahkannya, itu). Either way, Anda sekarang akan menukar angka untuk variabel baru Anda. Jika metode Kematian ada dalam skrip Player, Anda dapat menulis ini secara sederhana sebagai berikut:
transform.position = new Vector2(startx, starty);
Jika tidak, jika masih dalam skrip Hazards, akan terlihat seperti ini:
player.transform.position = new Vector2(player.startx, player.starty);

Either way, player kita sekarang respawn ke posisi yang kita baca di awal. Saya sarankan Anda memindahkan metode Kematian Anda sekarang sehingga ada di skrip Player. Jika Anda belum menemukan cara untuk melakukan ini dan membutuhkan sedikit bantuan, cukup perbarui skrip Anda sebagai berikut. Skrip player:

```
public class Player : MonoBehaviour {
        public Rigidbody2D rb;
        public int movespeed;
        public int jumppower;
        public Transform groundCheck;
        public float groundCheckRadius;
        public LayerMask whatIsGround;
        private bool onGround;
        public int coins;
        private Animator anim;
        private int facing;
        public bool moveLeft;
        public bool moveRight;
        public float startx;
        public float starty;
        public GameObject Blood;
        void Start () {
                rb = GetComponent<Rigidbody2D>();
                anim = GetComponent<Animator>();
                facing = 1;
                startx = transform.position.x;
                starty = transform.position.y;
        }
void FixedUpdate()
{
        onGround = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius,
        whatIsGround);    }
        void Update() {
        if (moveLeft || Input.GetKey(KeyCode.LeftArrow))
        {
        rb.velocity = new Vector2(-movespeed, rb.velocity.y);
        anim.SetBool("Walking", true);
        if (facing == 1)
        {
```

```
        transform.localScale = new Vector3(-1f, 1f, 1f);            facing = 0;
        }
        } else if (moveRight || Input.GetKey(KeyCode.RightArrow))       ]
        {
        rb.velocity = new Vector2(movespeed, rb.velocity.y);
        anim.SetBool("Walking", true);
        if (facing == 0)
        {
        transform.localScale = new Vector3(1f, 1f, 1f);             facing = 1;
        }
        } else
        {
        anim.SetBool("Walking", false);
        }
        if (Input.GetKey(KeyCode.Space))
        {
        Jump();
        }
        }
        public void Jump() {
        if (onGround)
        {
        rb.velocity = new Vector2(rb.velocity.x, jumppower);
        }
        }
        public void Death()
        {
        StartCoroutine("respawndelay");
}
public IEnumerator respawndelay()
        {
                Instantiate(Blood, transform.position, transform.rotation);
                enabled = false;
                GetComponent<Rigidbody2D>().velocity = Vector3.zero;
                GetComponent<Renderer>().enabled = false;
                yield return new WaitForSeconds(1);
                transform.position = new Vector2(startx, starty);
                GetComponent<Renderer>().enabled = true;
                enabled = true;
                }
                }
                Hazards script:
                public class Hazards : MonoBehaviour {
                private Player player;
                // Use this for initialization    void Start()
                {
                player = FindObjectOfType<Player>();
                }
```

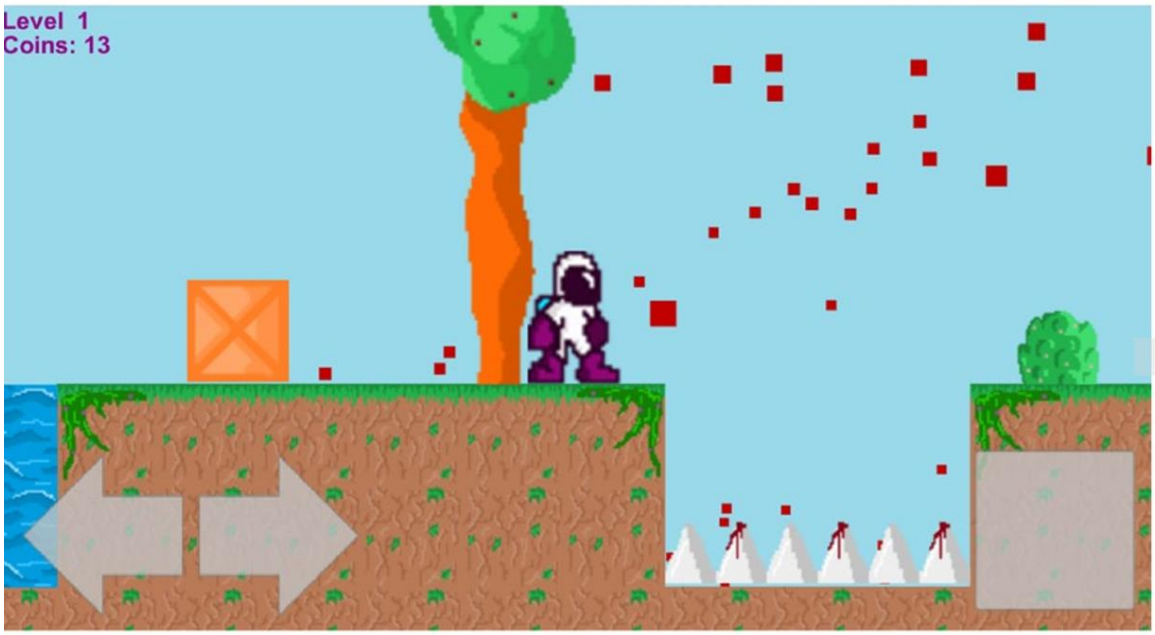
```
        // Update is called once per frame
        void Update()
        {
        }
        void OnTriggerEnter2D(Collider2D other)
        {
        if (other.tag == "Player")
        {
        player.Death();
        }
    }
}
```

Scripting Checkpoint

Anda mungkin sudah mengetahui apa yang akan terjadi selanjutnya. Yang perlu kita lakukan dengan checkpoint kita adalah mengubah nilai startx dan starty ketika mereka memasuki collider. Skrip Checkpoint baru kami sesederhana ini:

```
public class Checkpoint : MonoBehaviour {
private Player player;
void Start()
{
        player = FindObjectOfType<Player>();
        }
        void Update()
        {
        }
        void OnTriggerEnter2D(Collider2D other)
        {
        if (other.tag == "Player")
        {
player.startx = transform.position.x;
player.starty = transform.position.y;
        }
    }
}
```

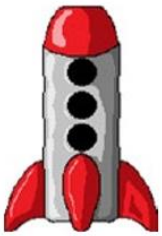
Dan jangan lupa, Anda juga perlu melampirkan skrip ini ke checkpoint yang dimaksud. Kemudian buatlah menjadi Prefab sehingga Anda dapat menambahkan lebih banyak checkpoint di sekitar level dengan mudah di masa mendatang. Cobalah ini dan Anda akan menemukan bahwa Anda sekarang respawn di checkpoint daripada di awal permainan. Bahkan, Anda seharusnya muncul cukup cepat untuk dihujani darah Anda sendiri.



Gambar 6.26 Menambah checkpoint

Sekarang buat beberapa checkpoint lagi di tempat-tempat cerdas dan mainkan sedikit untuk melihat mana yang paling berhasil. Atur mereka dalam Hierarchy Anda dengan cara yang logis juga, mungkin dengan memberi mereka objek induk yang disebut Checkpoints. Tentu saja, Anda dapat memberikan semacam indikator yang terlihat untuk menunjukkan checkpoint Anda—seperti pos yang ditemukan di game Sonic the Hedgehog—tetapi akhir-akhir ini cukup umum bagi player untuk muncul kembali di titik yang berbeda dalam permainan. Kami menerima ini sebagai bagian dari penangguhan ketidakpercayaan saat kami memuat, dan itu telah menjadi bagian dari bahasa video game.

Kami telah menempuh perjalanan panjang, tetapi permainan kami masih hanya memiliki satu level. Saatnya untuk memperkenalkan beberapa bentuk kemajuan nyata. Untuk melakukan itu, kami hanya akan membuat GameObject lain yang akan mewakili akhir level dan menjadikannya pemicu. Melihat Kevin adalah seorang angkasawan, masuk akal jika akhir levelnya adalah semacam roket luar angkasa. Nanti, kita akan mengani masikan ini; untuk saat ini, hanya mencapai roket ruang angkasa akan mengakhiri level.



Gambar 6.27 Kapal roket ini menandakan akhir dari setiap level

Membuat Level Baru

Sebelum kita dapat menambahkan skrip, pertama-tama kita perlu membuat level lain. Dan sebelum Anda melakukannya, pastikan untuk membuat Player GameObject Anda menjadi prefab (drag Player ke folder Prefabs dan pastikan untuk membawa Main Camera dan Check Ground bersama Anda). Dengan begitu, setiap perubahan yang Anda buat akan tercermin secara global di semua level yang Anda buat. Lakukan hal yang sama untuk Canvas Anda dan semua anaknya. Setelah Anda selesai melakukannya, coba gunakan trik kecil yang

mudah ini untuk membuat level baru: cukup klik File - Save Scene As dan beri nama Level 2. Pastikan itu masuk ke folder Scenes yang sama dengan Level 1

Cara cepat untuk membuat level baru

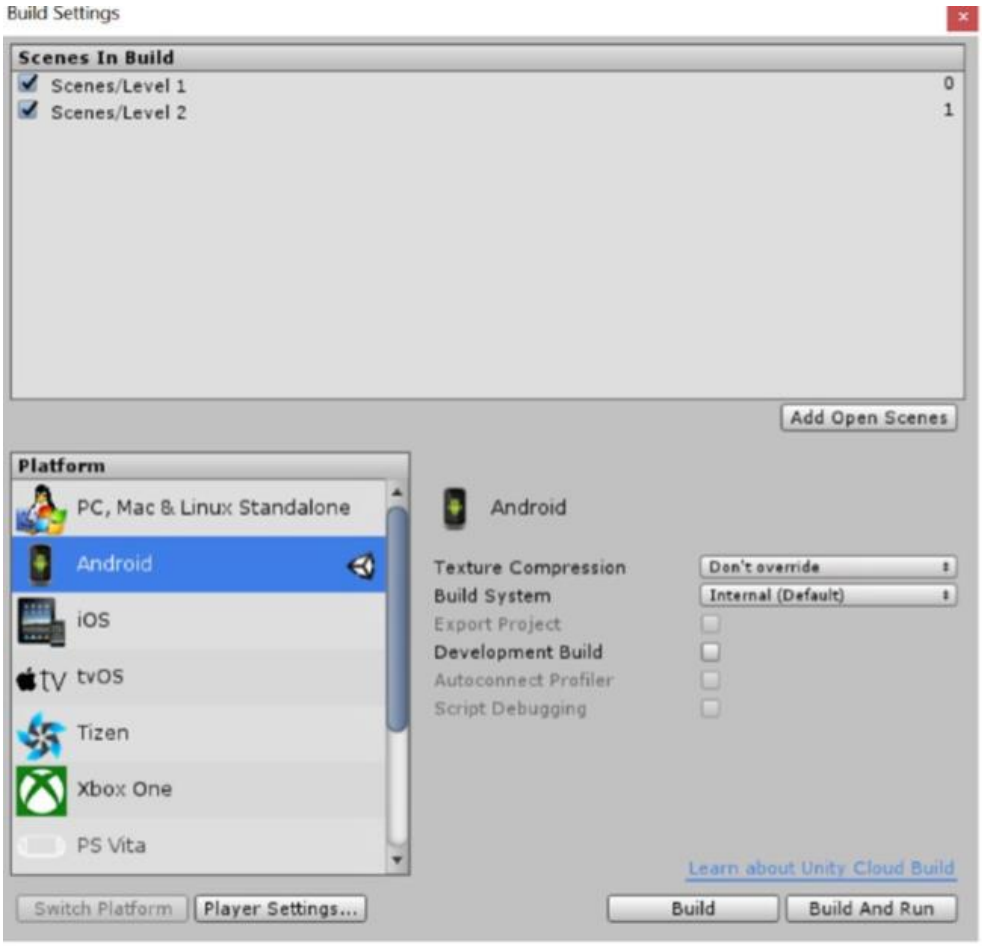
Ini adalah jalan pintas yang bagus karena itu berarti Anda sudah memiliki semua prefab Anda dan Anda dapat menyiapkan dan menjalankan semuanya dengan lebih cepat. Hapus saja beberapa elemen, pindahkan beberapa hal, lalu tekan Ctrl+S untuk menyimpan tata letak baru. (Atau, Anda dapat membuat scene yang sama sekali baru hanya dengan mengklik kanan folder Scenes Anda dan kemudian memilih Create➤Scene). Sekarang jika Anda menavigasi ke folder Scenes di panel Project Anda dan klik dua kali level satu, itu harus melompat di antara dua tata letak. Perhatikan bahwa semuanya baru dalam scene baru—bahkan pemutar dan kamera. Namun demikian, semuanya masih merupakan instance dari objek yang sama yang hidup di folder Prefabs Anda, jadi mengedit skrip atau properti akan memengaruhi semuanya di semua level.

Menaikkan Level

Sekarang permainan kami memiliki lebih dari satu level, kami siap untuk memungkinkan transisi di antara mereka. Kembali ke Level 1 dan tambahkan sprite ke level pertama Anda seperti biasa. Jadikan itu GameObject dengan penumbuk poligon, centang Is Trigger, buat skrip baru bernama EndLevel, dan tambahkan sebagai komponen ke kapal roket. EndLevel akan terlihat seperti ini:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class EndLevel : MonoBehaviour {
                public string nextLevel;
                void Start()
                {
                }
                void Update()
                {
                }
                void OnTriggerEnter2D(Collider2D other)
                {
                if (other.tag == "Player")
                {
                SceneManager.LoadScene(nextLevel);
                }
        }
}
```

itu karena saya menggunakan kelas tambahan dari Unity yang disebut SceneManagement. Kelas ini memungkinkan kita menggunakan perintah yang memuat scene berikutnya. Scene yang dimaksud sementara itu adalah string publik, yang akan kita beri nama di Inspector sebagai Level 2. Itu akan memudahkan kita untuk memperbarui tujuan level kita untuk setiap scene sambil tetap menggunakan skrip dan objek yang sama. Sebelum Anda mengklik play, ada satu hal lagi yang perlu Anda lakukan: kembali ke Build Settings dan tambahkan Scene 2 ke game Anda (cukup drag dari folder Scenes dan jatuhkan ke jendela.

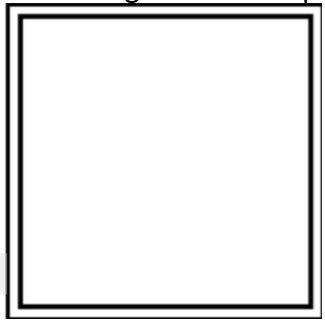


Gambar 6.28 Drag Level 2 ke pengaturan build sehingga Anda dapat memuatnya

Sekarang coba selesaikan level dengan mencapai roket. Anda akan menemukan bahwa level berikutnya segera dimuat dan Anda siap untuk menghadapi tantangan berikutnya: membuat pemilihan level.

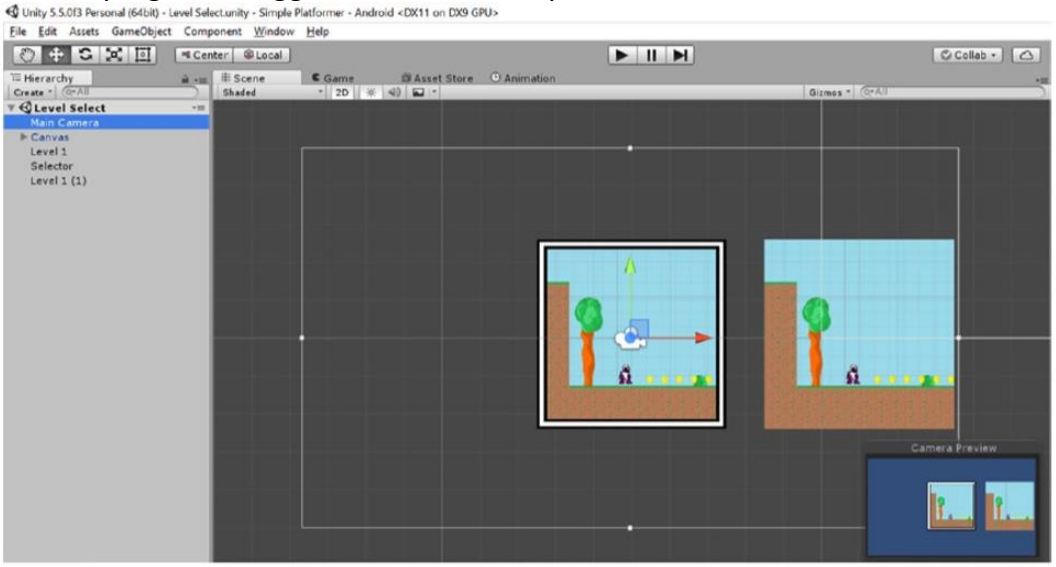
Membangun Layar Pemilihan Level

Di sebagian besar game seluler—dan game PC dan konsol dalam hal ini—player dapat langsung melompat ke level tertentu selama mereka sebelumnya telah menyelesaikan level sebelumnya. Ini memungkinkan mereka memutar ulang momen favorit mereka, kembali untuk menemukan rahasia tersembunyi, dan mengalahkan skor tertinggi mereka. Untuk memungkinkan ini, kami perlu menyediakan beberapa cara bagi player kami untuk melihat level dan memilihnya. Dengan kata lain, kita perlu membangun pemilihan level. Ini berarti Anda perlu membuat scene lain, tetapi yang ini akan benar-benar kosong. Kami akan menyebutnya Level Select. Setelah itu siap, saatnya untuk berkenalan kembali dengan Squarey, hanya saja kali ini dia telah kehilangan sedikit kepersonalannya.



Gambar 6.29 Ini akan menjadi pemilih kami

Sebenarnya ini bukan Squarey sama sekali, melainkan indikator atau pemilih, artinya kita perlu memiliki transparansi di tengahnya. Inilah yang akan menunjukkan kepada kita level mana yang ingin kita pilih, jadi kita juga akan membutuhkan dua tampilan, satu untuk setiap level, yang ukurannya akan sama. Saya telah membuat ini 500 x 500. Anda dapat melakukan ini dengan mengambil screenshoot dari dua level Anda (mengabaikan fakta bahwa mereka pada dasarnya sama pada saat ini) dan menyimpannya sebagai sprite. Sekarang, atur dua gambar level ke dalam scene sehingga terlihat kamera dan disejajarkan dengan baik. Kemudian tempatkan pemilih Anda di atas (dengan koordinat yang sama persis) dan pastikan memiliki nilai yang lebih tinggi untuk Order in Layer.



Gambar 6.30 Awal dari scene Level Select kami

Menulis Script Kontrol

Sekarang kita akan membuat skrip kontrol baru yang akan berfungsi seperti skrip Player. Untuk sebagian besar, kami akan mengontrol pemilih seperti kami akan mengontrol player. Ini adalah karakter player kami untuk saat ini. Buat skrip dan beri nama Selector. Kemudian gunakan kode berikut:

```
public class Selector : MonoBehaviour {
  public bool moveLeft;   public bool moveRight;
      void Start()
      {
  }
      void Update()
      {
            if (transform.position.x > -5 && (moveLeft || Input.GetKeyDown(KeyCode.LeftArrow)))
      {
      transform.position = new Vector2(transform.position.x - 6, transform. position.y);
      moveLeft = false;
}
else if (transform.position.x < 1 && (moveRight || Input.GetKeyDown(KeyCode.RightArrow)))
{
transform.position = new Vector2(transform.position.x + 6, transform. position.y);
moveRight = false;
}
```

```
        if (Input.GetKey(KeyCode.Space))
        {
        Select();
        }
        }
        public void Select()
        {
        }
}
```

Gambar level saya berjarak 6 unit terpisah, jadi itulah seberapa banyak pemilih akan bergerak dengan setiap langkah.

Seperti yang Anda lihat, ini sangat mirip dengan skrip Player kami yang biasa, meskipun jelas tanpa metode Death atau animasi. Ada juga satu atau dua perbedaan lain yang juga perlu Anda ketahui. Kami telah membuat metode Select, tetapi untuk saat ini kosong. Alih-alih, yang terjadi hanyalah pemilih bergerak 6 unit ke kiri atau kanan saat pengguna mengetuk tombol panah. Perhatikan bahwa kami menggunakan GetKeyDown sekarang, jadi pengguna harus mengetuk daripada menahan panah. Saya juga mengatur moveLeft dan moveRight ke false segera setelah kotak bergerak satu langkah untuk alasan yang sama. Terakhir, saya telah menambahkan sedikit kode untuk memeriksa apakah pemilih tidak akan berpindah dari tepi kiri atau kanan layar. Anda harus memperbarui ini setiap kali Anda menambahkan level baru atau berpotensi menggunakan sesuatu seperti numberOfLevels * 6 untuk menghitung seberapa jauh pemilih dapat bergerak ke kanan.

Jika Anda menyeret objek Kamera Utama untuk menjadikannya anak dari pemilih di Hierarki, maka layar akan "menggulir" saat pemilih kami bergerak. Saat ini, jika Anda menguji ini, itu akan berfungsi selama Anda menggunakan tombol kursor di komputer Anda. Sekarang Anda perlu membuat skrip Touch baru khusus untuk Selector. Kita dapat menambahkan kode ini ke skrip Touch yang sama yang telah kita buat dan memeriksa objek yang dilampirkannya, tetapi membuat sesuatu yang baru mungkin lebih sederhana. Jadi buat skrip baru lainnya, kali ini bernama LevelSelectTouch. Yang ini secara efektif merupakan pengulangan dari Sentuhan sebelumnya, membuat segalanya menyenangkan dan mudah bagi kami:

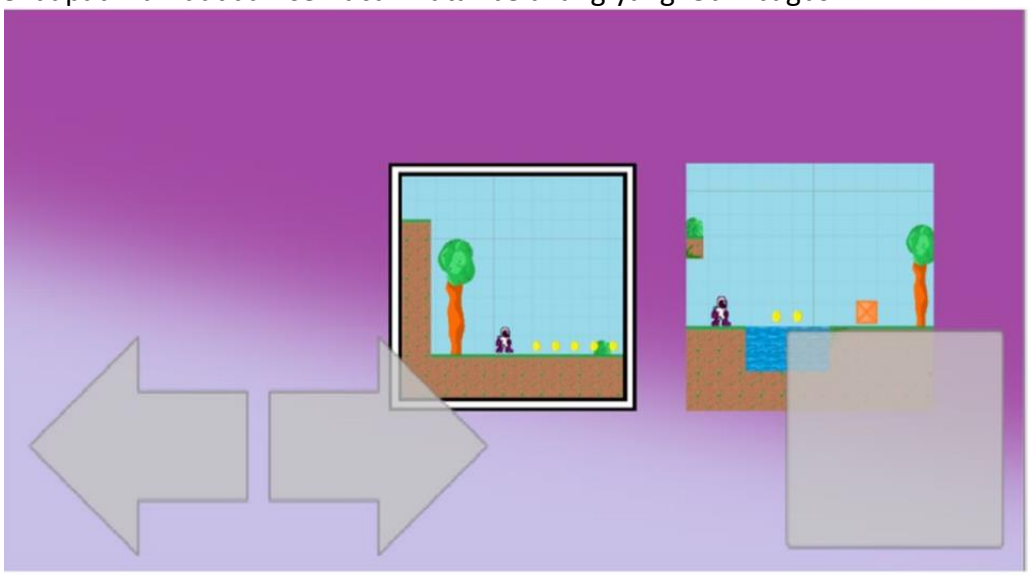
```
public class LevelSelectTouch : MonoBehaviour {
        private Selector selector;
        void Start()
        {
        selector = FindObjectOfType<Selector>();
        }
        public void PressLeftArrow()
        {
        selector.moveRight = false;
        selector.moveLeft = true;
        }
        public void PressRightArrow()
        {
        selector.moveRight = true;
        selector.moveLeft = false;
        }
        public void ReleaseLeftArrow()
```

```
        {
        selector.moveLeft = false;
        }
        public void ReleaseRightArrow()
        {
        selector.moveRight = false;
        }
        public void Select()
        {
        selector.Select();
        }
}
```

Tambahkan skrip ini ke TouchController GameObject di Hierarchy Anda—bukan ke prefab. Ingat, kami hanya ingin perubahan ini memengaruhi contoh kontrol sentuh ini dan bukan yang digunakan di Level 1 dan 2. Sekarang Anda hanya perlu menyiapkan kontrol untuk bekerja dengan skrip ini. Buka Panah Kanan, Panah Kiri, lalu Lompat di Inspector dan konfigurasikan ulang pemicu peristiwa untuk masing-masing sehingga sesuai dengan metode yang benar dalam skrip baru. Jika Anda buntu, periksa kembali dan lihat bagaimana kami melakukannya terakhir kali—prosesnya sama persis (ada di Bab 7). Tentu saja, tombol lompat akan dikaitkan dengan metode Select dalam kasus ini. Dan sementara kami merusak Prefab kami, hapus Level dan Koin GameObjects dari kanvas karena mereka tidak masuk akal dalam konteks ini.

Siap Diluncurkan

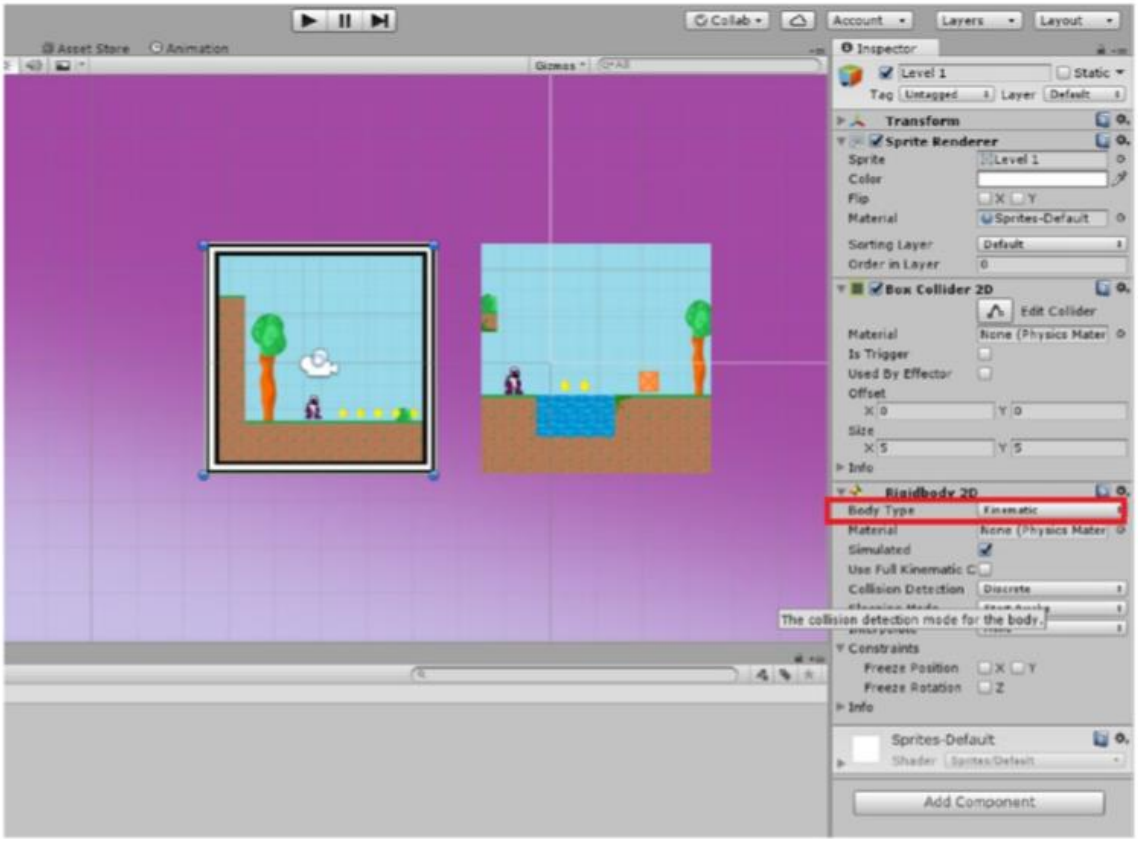
Cobalah ini dan Anda akan menemukan bahwa Anda sekarang dapat memindahkan pemilih dengan kursor atau kontrol di layar. Kelihatannya cukup bagus, meskipun mungkin bisa mendapat manfaat dari semacam latar belakang yang lebih bagus.



Gambar 6.31 Di sana kita pergi, jauh lebih baik.

Yang benar-benar dibutuhkan adalah kemampuan untuk benar-benar memilih level. Cara yang baik untuk melakukan ini adalah dengan memberikan nama yang tepat untuk gambar level kita, yang akan sama dengan nama scene kita (jadi, Level 1 dan Level 2) dan membuat pemilih itu sendiri sebagai pemicu dengan penumbuk kotak. Kami juga akan menambahkan Rigidbodies, yang digunakan untuk deteksi collision kami. Kami jelas tidak ingin level kami

turun dari bagian bawah layar, jadi klik menu dropdown kecil di sebelah Body Type untuk memilih Kinematic.



Gambar 6.32 Atur Jenis Tubuh ke Kinematis.

Sekarang perbarui skrip Selector seperti itu, ingat untuk menambahkan yang baru menggunakan line up top juga:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Selector : MonoBehaviour {
        public bool moveLeft;
        public bool moveRight;
        private string levelChoice;
        void Start()
        {
}
void Update()   {
        if (transform.position.x > -5 && (moveLeft || Input.GetKeyDown(KeyCode.LeftArrow)))
        {
        transform.position = new Vector2(transform.position.x - 6, transform. position.y);
        moveLeft = false;
        }                e lse if (transform.position.x < 7 && (moveRight ||
        Input.GetKeyDown(KeyCode.RightArrow)))
        {
        transform.position = new Vector2(transform.position.x + 6, transform.position.y);
```

```
        moveRight = false;
        }
        if (Input.GetKey(KeyCode.Space))
        {
        Select();
        }
        }
        public void Select()
        {
        SceneManager.LoadScene(levelChoice);
        }
        void OnTriggerEnter2D(Collider2D other)   {
        levelChoice = other.name;
        }
}
```

Kode itu hanya mencari collision dan kemudian mendapatkan nama objek yang melanggar untuk disimpan sebagai string yang disebut levelChoice. Saat Anda menekan tombol lompat, levelChoice kemudian dimuat dengan cara yang sama seperti kita memuatnya sebelumnya. Cobalah ini dan Anda akan menemukan bahwa Anda sekarang dapat melompat ke level mana pun yang Anda pilih. Jangan lupa untuk menambahkan scene Level Select ke Build Settings. Mari luangkan waktu sejenak untuk merenungkan apa yang telah Anda capai di sini: Anda menggunakan semua trik yang sama yang telah Anda gunakan tetapi kali ini Anda membuat menu daripada level dalam game. Ini adalah indikasi awal betapa serbagunanya alat yang diberikan Unity kepada kita. Tidak terlalu sulit untuk membayangkan membuat game puzzle atau bahkan semacam alat produktivitas.

Selamatkan Progress

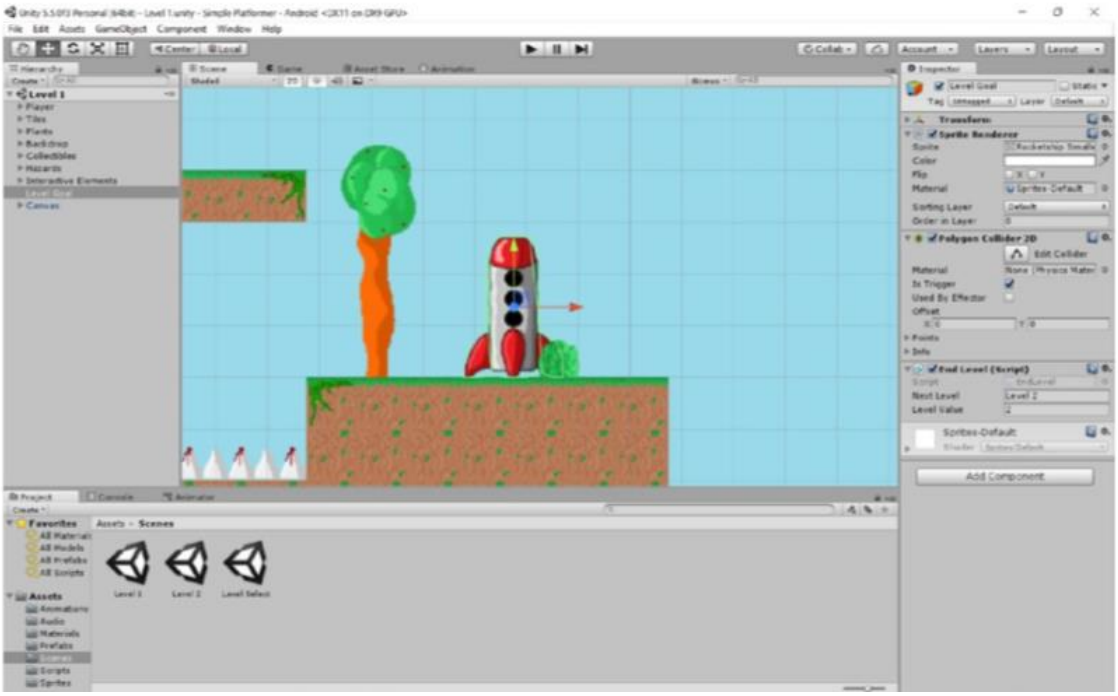
Pemilihan level tidak banyak berguna sampai kami dapat menyimpan kemajuan player kami. Kami ingin ada rasa pencapaian dan kemajuan saat mereka bermain melalui level, yang berarti bahwa opsi untuk bermain lebih jauh akan terbuka saat setiap level selesai. Kemajuan ini harus bertahan dari sesi bermain ke sesi bermain juga, karena tidak menyenangkan harus memulai lagi dari awal setiap saat.

Jadi kami membutuhkan cara untuk menyimpan kemajuan kami, dan Unity sebenarnya memberi kami sejumlah opsi, mulai dari menggunakan preferensi pemutar hingga serialisasi atau membuat file teks. Secara teknis, yang seharusnya kita gunakan adalah serialisasi—ini memungkinkan Anda menyimpan lebih banyak informasi dengan lebih cepat. Tanpa terlalu banyak detail di sini, itu berarti mengubah objek menjadi byte. Ini sedikit rumit, jadi untuk saat ini kita akan menggunakan PlayerPrefs karena cepat dan kotor dan lebih mudah untuk memahaminya. PlayerPrefs seharusnya digunakan untuk menyimpan preferensi seperti kualitas gambar, atau apakah Anda ingin suara aktif—pengaturan, dengan kata lain. Tapi sejujurnya, banyak developer indie menggunakan metode ini secara eksklusif, dan jika yang perlu Anda lakukan hanyalah menyimpan beberapa skor teratas dan nama level, itu akan baik-baik saja. Sangat mudah untuk menyimpan level segera setelah kami memuatnya. Cukup perbarui skrip EndLevel yang terpasang pada roket di Level 1 agar terlihat seperti ini:

```
public class EndLevel : MonoBehaviour {
        public string nextLevel;
        public int levelValue;
        void Start()
```

```
        {
        }
        void Update()
        {
        }
        void OnTriggerEnter2D(Collider2D other)
        {
        if (other.tag == "Player")
        {
        SaveLevel(levelValue);
        SceneManager.LoadScene(nextLevel);
        }
        }
        public void SaveLevel(int level)
        {
        PlayerPrefs.SetInt("farthestLevel", level);
        }
}
```

Satu baris itu: PlayerPrefs.SetInt("farthestLevel", level); adalah semua yang diperlukan. Ini membuat integer baru dengan kunci terjauhLevel dan meletakkannya di PlayerPrefs. Kita hanya perlu menambahkan variabel publik levelValue di Inspector (Gambar 8-11), dan sekarang menyentuh roket akan memuat scene berikutnya dan memperbarui variabel yang disimpan.

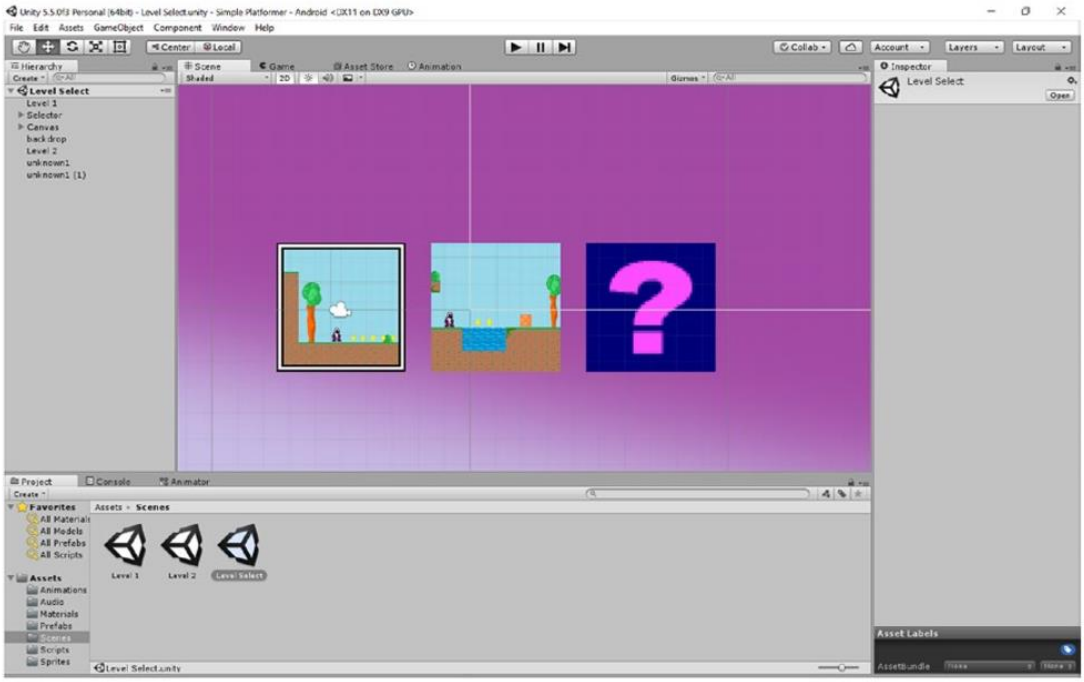


Gambar 6.33 Roket memiliki nilai Level 2

Untuk memanfaatkan ini, kita perlu membuat layar pemilihan level kita sedikit lebih pintar sehingga dapat menunjukkan kepada kita saat level belum siap untuk dimuat. Buat sesuatu yang akan ditampilkan sebagai pengganti level yang akan datang. Saya menggunakan tanda tanya yang harus cocok dengan latar belakang saya

Gambar 6.34 Ada apa di balik pintu nomor dua?

Buat dua dari ini dan tempatkan yang pertama sehingga berada di belakang Level 2 lebih jauh ke belakang dalam urutan (jika Anda melakukannya dengan benar, itu tidak akan terlihat). Yang kedua berjalan ke kanan di mana Level 3 nanti. Ini seharusnya tidak memiliki clasher; itu hanya gambar yang akan ditampilkan ketika gambar di depannya hilang.



Gambar 6.35 Sebenarnya ada dua tanda tanya di sini — yang pertama ada di belakang Level 2

Kami sekarang akan membuat skrip lain yang disebut LevelLoader dan melampirkannya ke gambar Level 2:

```
public class LevelLoader : MonoBehaviour {
        public int thisLevel;
        private Selector selector;
        void Start () {
        selector = FindObjectOfType<Selector>();
        }
        void Update ()
        {
        if (selector.farthestLevel < thisLevel) {
        this.tag = "off";
        GetComponent<Renderer>().enabled = false;
        } else
        {
```

```
        this.tag = "on";
        GetComponent<Renderer>().enabled = true;
        }
        }
}
```

Jadi, gambar-gambar ini sekarang menjadi pemuat level kami. Ketika scene Level Select dibuat, semuanya akan muncul sebagaimana mestinya, dan kemudian mereka akan memeriksa untuk melihat apakah player sudah cukup jauh. Jika player tidak melakukannya, mereka akan menghilang (GetComponent<Rendered>().enabled = false) dan menonaktifkan tag mereka. Lampirkan ini ke GameObject dan kemudian masukkan bilangan bulat publik thisLevel di Inspector, yang tentu saja harus 2. Seperti yang Anda lihat, kode yang perlu kita periksa levelnya akan berada di skrip Selector (karena kita hanya ingin lakukan ini sekali), yang memiliki properti publik loadLevel. Untuk mendapatkan nilai ini, yang harus kita lakukan hanyalah menambahkan satu baris kode ke metode Start di skrip itu:
PlayerPrefs.GetInt("farthestLevel");

Tentu saja, kita juga perlu mendefinisikan bilangan bulat publik di bagian atas. Sekarang, ketika pemilih dibuat, ia akan memeriksa level terjauh yang telah dicapai player dengan melihat PlayerPrefs dan menyimpan nilai ini sebagai bilangan bulat publik. Gambar LevelLoader kemudian akan hilang jika player belum cukup jauh untuk memainkannya. Ketika player memilih ?, mereka masih benar-benar memilih "Tingkat 2 yang tidak terlihat", tetapi karena tag dimatikan, itu tidak akan dimuat.

6.9 MENAMBAHKAN LEBIH BANYAK ELEMEN GAME

Anda telah menghabiskan beberapa bab terakhir bekerja keras untuk menciptakan dunia game yang berfungsi. Anda telah mengembangkan mesin, membuat file simpanan, dan membuat karakter Anda bergerak dan berinteraksi dengan dunia sebagaimana mestinya. Mudah-mudahan, Anda bersenang-senang melakukannya, tetapi mungkin terasa sedikit menantang dan cukup teknis pada titik-titik di sepanjang jalan. Nah, sekarang saatnya menikmati pencapaian Anda sebentar. Anda telah menciptakan dunia ini. Mari bersenang-senang di dalamnya. Lagi pula, permainan tipikal akan melibatkan banyak rintangan, bahaya, dan peningkatan kekuatan yang berbeda, yang masing-masing biasanya akan menciptakan tantangan gameplay yang unik dan pertemuan yang menyenangkan. Sonic memiliki pegas, cincin, tiang, Badnik, lubang paku, zamrud chaos, dan loop-de-loop. Mario memiliki jamur, Bullet Bills, hantu, kotak tanda tanya, dan Yoshi(s). Super Meat Boy memiliki portal, gergaji raksasa, misil, dan tumpukan jarum bekas. Sudah waktunya bagi Anda untuk menjadi kreatif dan mulai memperkenalkan lebih banyak elemen ke dalam game Anda sendiri. Dan bagian terbaiknya? Menciptakan tantangan ini hampir sama menyenangkannya dengan memainkannya nanti. Dalam bab ini, Anda akan belajar cara membuat semua jenis bahaya lingkungan dan musuh dan Anda akan dapat mundur kapan pun Anda ingin menambahkan elemen lain ke dunia game Anda sendiri. Saya berharap ini juga akan menjadi sumber inspirasi untuk membantu Anda menghadapi rintangan dan tantangan Anda sendiri. Tentu saja, kita juga akan mempelajari beberapa konsep baru. Pada akhirnya, Anda juga akan belajar cara menjarah Asset Store sehingga Anda dapat mengakses efek partikel, skrip, dan sprite yang telah dibuat dengan penuh kasih oleh orang lain dan menggunakannya dalam permainan Anda sendiri. Siap? Mari bersenang - senang!

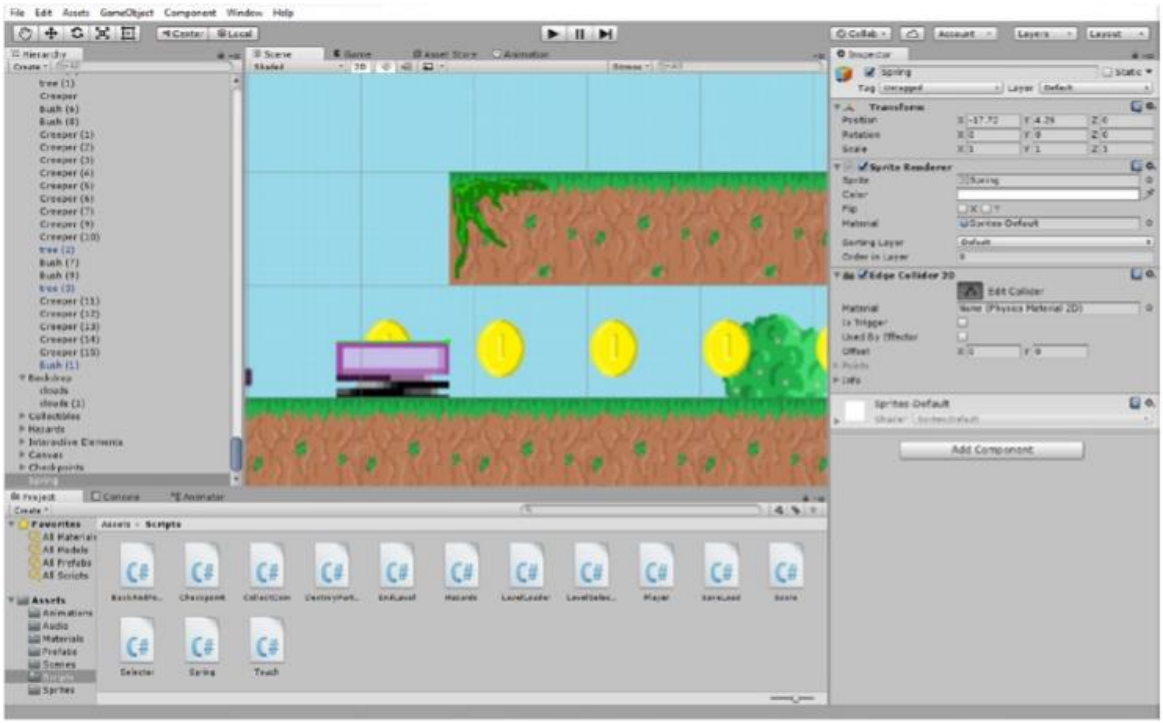
Beberapa Objek Game Umum dan Perilakunya

Meskipun setiap platformer berbeda dan Anda harus melakukan apa yang Anda bisa untuk membedakan diri Anda dan menonjol dari yang lain, memang benar bahwa kiasan tertentu muncul berulang kali. Ini normal dalam genre apa pun dalam bentuk media apa pun, jadi jangan khawatir jika Anda kembali ke "favorit lama". Jadi, dengan asumsi Anda akan menggunakan beberapa aset dan objek yang lebih umum dalam desain game Anda, bagian ini akan menunjukkan cara membangun elemen dasar seperti pegas dan balok bergerak.

Springs

Objek game pertama yang akan kita buat adalah pegas, atau "bounce pad." Diperdebatkan dipopulerkan oleh Sonic the Hedgehog, pegas sekarang menjadi kiasan umum di platformer yang digunakan sebagai sarana untuk mendorong player naik level. Lihatlah di sekitar Unity IDE dan Anda mungkin menemukan sesuatu yang sepertinya akan berhasil: Anda dapat menambahkan properti "goyang" ke Bahan Fisika 2D. Sayangnya, bukan itu yang kami cari karena ini akan membuat tanah bertindak lebih seperti permukaan goyang yang sebenarnya. Artinya, itu akan mendorong karakter lebih tinggi semakin jauh mereka jatuh dan akhirnya mengembalikan lebih sedikit energi. Anda dapat melakukan beberapa hal menyenangkan dengan itu, tetapi itu tidak akan bertindak seperti yang kita inginkan. Sebagai gantinya, kita akan membuat sprite pegas dan menambahkannya ke level Anda, seperti yang biasa Anda lakukan sekarang. Perhatikan bahwa kita menggunakan penumbuk tepi di sepanjang tepi atas (bukan penumbuk kotak biasa).

Gambar 6.36 Spring



Gambar 6.37 Sebuah spring di tingkat

Selanjutnya, Anda akan membuat skrip Spring dan menambahkan kode berikut:

```
public class Spring : MonoBehaviour
        {
```

```
        private Player player;   // Use this for initialization
        void Start()
        {
        player = FindObjectOfType<Player>();
        }
        // Update is called once per frame
        void Update()
        {
        }
        void OnCollisionEnter2D(Collision2D other)
        {
        if (other.gameObject.tag == "Player")
        {
        player.SuperJump();
        }
    }
}
```

Seperti yang mungkin sudah Anda duga, Anda juga akan menambahkan metode SuperJump ke skrip Player Anda:

```
public void SuperJump() {
rb.velocity = new Vector2(rb.velocity.x, jumpPower * 2);
}
```

Tentu saja, ingatlah untuk menambahkan skrip Spring baru ke GameObject Anda dan menjadikannya prefab—hal yang biasa. Sekarang ketika Anda menyentuh pegas, Kevin akan diluncurkan ke udara dengan ketinggian dua kali lipat dari biasanya dia bisa melompat. Saya telah menjaga ketinggian proporsional dengan tinggi lompatannya untuk berjaga-jaga jika kami memutuskan untuk mengubah scale level. Jika mau, Anda juga dapat menambahkan animasi dan suara ke pegas.

Platform Bergerak

Sebuah kiasan umum dalam permainan platform apapun adalah platform bergerak. Anda memiliki platform yang bergerak ke kiri dan ke kanan dan membawa Anda melewati jurang dan Anda memiliki platform yang bergerak ke atas dan ke bawah dan bertindak seperti elevator. Kita sudah bisa membuat segalanya bergerak ke kiri dan ke kanan—kita sudah melakukannya dengan musuh kita. Masalahnya adalah jika Anda menempelkan skrip gerakan ini ke sebidang tanah, Kevin tidak akan bergerak dengannya. Sebaliknya, tanah akan bergerak keluar dari bawahnya, dan dia akan jatuh darinya. Tidak baik.

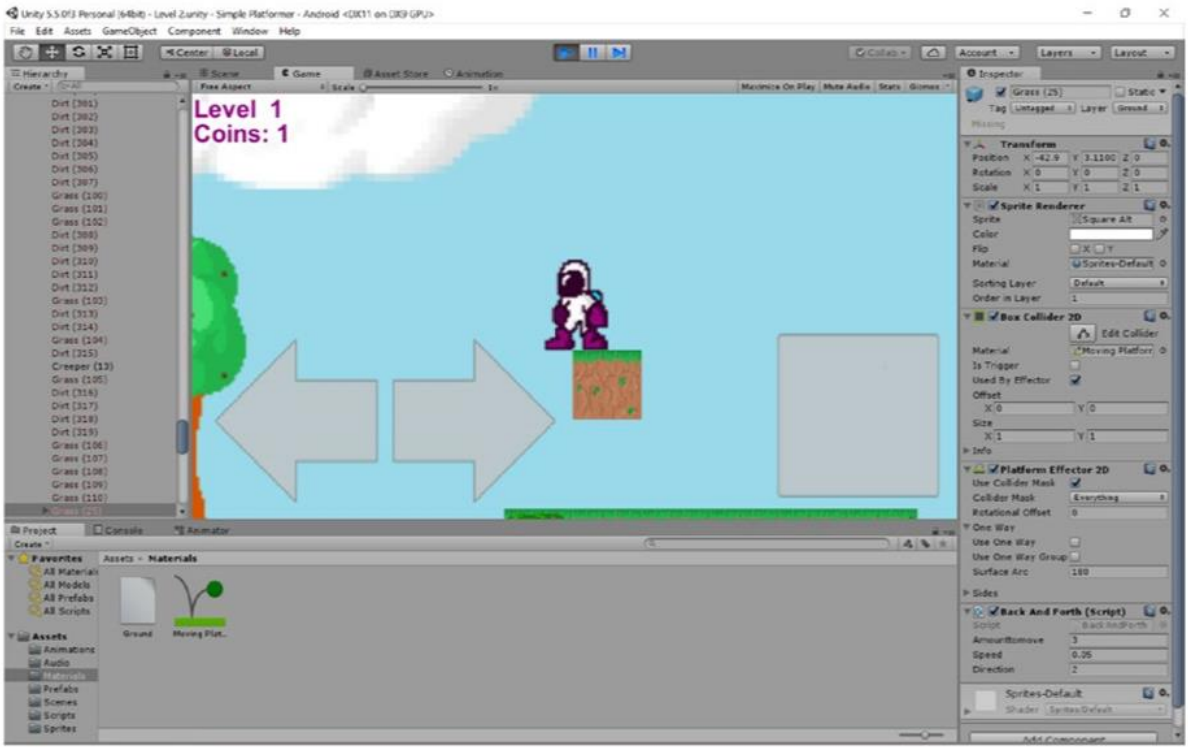
Sementara itu, jika platform bergerak ke atas dan ke bawah dan player Anda berada di atasnya, dia akan terguncang dan ketakutan dan mungkin jatuh ke lantai. Kami pada dasarnya perlu memodifikasi skrip ini sehingga kami dapat menempel di permukaan atas dan bepergian dengannya. Bagaimana kita bisa melakukan ini? Saya akan memberi Anda waktu untuk berpikir ... apa yang telah kami gunakan sebelumnya dalam buku ini yang memungkinkan GameObject untuk bergerak dalam kaitannya dengan GameObject lain? Mengerti? Jawabannya adalah kita perlu menjadikan Kevin sebagai anak dari GameObject tempat dia berdiri. Untuk melakukan itu, buka skrip gerakan Anda (kami menyebutnya BackAndForth) dan bersiaplah untuk membuat beberapa perubahan. Kami tidak hanya mengubah skrip sehingga karakter kami menjadi anak platform ketika mereka bersentuhan dengannya, kami

juga menambahkan dimensi gerakan lain sehingga juga dapat bergerak ke atas dan ke bawah. Variabel integer arah sekarang menjadi publik, artinya kita dapat mengeditnya dari Inspector. Ini juga tidak lagi disetel ke 0 dalam metode onStart(), tetapi ingat bahwa bilangan bulat selalu mulai hidup sebagai 0 jika tidak disetel. Itu berarti bahwa perilaku musuh kita tidak akan berubah—mereka akan terus bergerak ke kiri dan ke kanan karena variabel arah akan default ke 0. Namun, untuk platform, kita memiliki opsi untuk menyetelnya ke 2 atau 3, yang akan membuat itu bergerak ke atas lalu ke bawah, atau ke bawah lalu ke atas. Setelah semua itu, BackAndForth sekarang akan terlihat seperti ini:

```
public class BackAndForth : MonoBehaviour {
        public double amounttomove;
        public float speed;
        private float startx;
        private float starty;
        public int direction;
        private Player player;
        // Use this for initialization
        void Start()
        {
        startx = gameObject.transform.position.x;
        starty = gameObject.transform.position.y;
        player = FindObjectOfType<Player>();
        }
        // Update is called once per frame
        void Update()
        {
        if (gameObject.transform.position.x < startx + amounttomove && direction == 0)
        {
        gameObject.transform.position = new Vector2(gameObject.transform.position.x +
        speed, gameObject.transform.position.y);
        }
        else if (gameObject.transform.position.x >= startx + amounttomove && direction == 0)
        {
        direction = 1;
        }
        else if (gameObject.transform.position.x > startx && direction == 1)
        {
        gameObject.transform.position = new Vector2(gameObject.transform.position. x -
        speed, gameObject.transform.position.y);
        }       else if (gameObject.transform.position.x <= startx && direction == 1)
        {
        direction = 0;
        }
        if (gameObject.transform.position.y < starty + amounttomove && direction == 3)
        {                                       gameObject.transform.position = new
        Vector2(gameObject.transform.position.x, gameObject.transform.position.y + speed);
        }
        else if (gameObject.transform.position.y >= starty + amounttomove && direction == 3)
        {
```

```
        direction = 2;
        }
        else if (gameObject.transform.position.y > starty && direction == 2)
        { gameObject.transform.position = new
        Vector2(gameObject.transform.position.x, gameObject.transform.position.y - speed);
        }
        else if (gameObject.transform.position.y <= starty && direction == 2)
        {
        direction = 3;
        }
        }
        void OnCollisionEnter2D(Collision2D other)
        {
        if (other.gameObject.tag == "Player") { player.transform.parent =
        gameObject.transform;
        }
        }
        private void OnCollisionExit2D(Collision2D other)
        {
        if (other.gameObject.tag == "Player")
        {
        player.transform.parent = null;
        }
        }
}
```

Saya juga menyarankan agar Anda membuat Bahan Fisika 2D baru untuk platform dan mengatur gesekan ke sesuatu yang tinggi. Hal ini untuk mencegah Kevin meluncur terlalu banyak di platform, yang terlihat sedikit aneh dalam hubungannya dengan gerakan. Ini juga merupakan ide yang baik untuk menggunakan penumbuk tepi dan menambahkan efektor platform. Centang Digunakan oleh Efektor dan Gunakan Satu Arah, dan ini akan mencegah player kita terlindas platform atau menempel ke samping dan bergerak sebagai anak platform. Jika ini mencegah player untuk melompat dari platform, Anda mungkin ingin sedikit meningkatkan radius Ground Check Anda. Ada cara lain untuk mencapai hal yang sama yang mungkin sedikit lebih elegan, tetapi ini adalah "perbaikan" mudah yang akan membuat platform bergerak Anda aktif dan berjalan. Jika semuanya berhasil, Kevin sekarang harus bergerak dengan platform apakah itu ke kiri dan ke kanan atau ke atas dan ke bawah (Gambar 6.38). Ini menciptakan banyak peluang tantangan platform, jadi bersenang-senanglah dengannya.



Gambar 6.38 Bergerak dengan platform

Runtuhnya Platform

Anda tahu apa lagi yang hebat? Runtuhnya platform. Ini adalah platform yang runtuh di bawah kaki ketika Anda mendarat di atasnya dan dengan demikian mendorong Anda untuk berlari dan melompat dengan cepat untuk menghindari jatuh ke malapetaka Anda. Dengan hal semacam ini, penting bagi Anda untuk mengomunikasikan kepada player tentang sifat tantangan yang akan mereka hadapi. Tidak adil untuk meninju player Anda dengan menjatuhkan platform dari bawahnya tanpa peringatan, dan untuk alasan itu biasanya disarankan untuk memiliki indikator visual bahwa tanahnya tidak cukup stabil. Untuk alasan itu, saya membuat desain untuk platform yang runtuh.

Gambar 6.39 Ubin platform yang runtuh

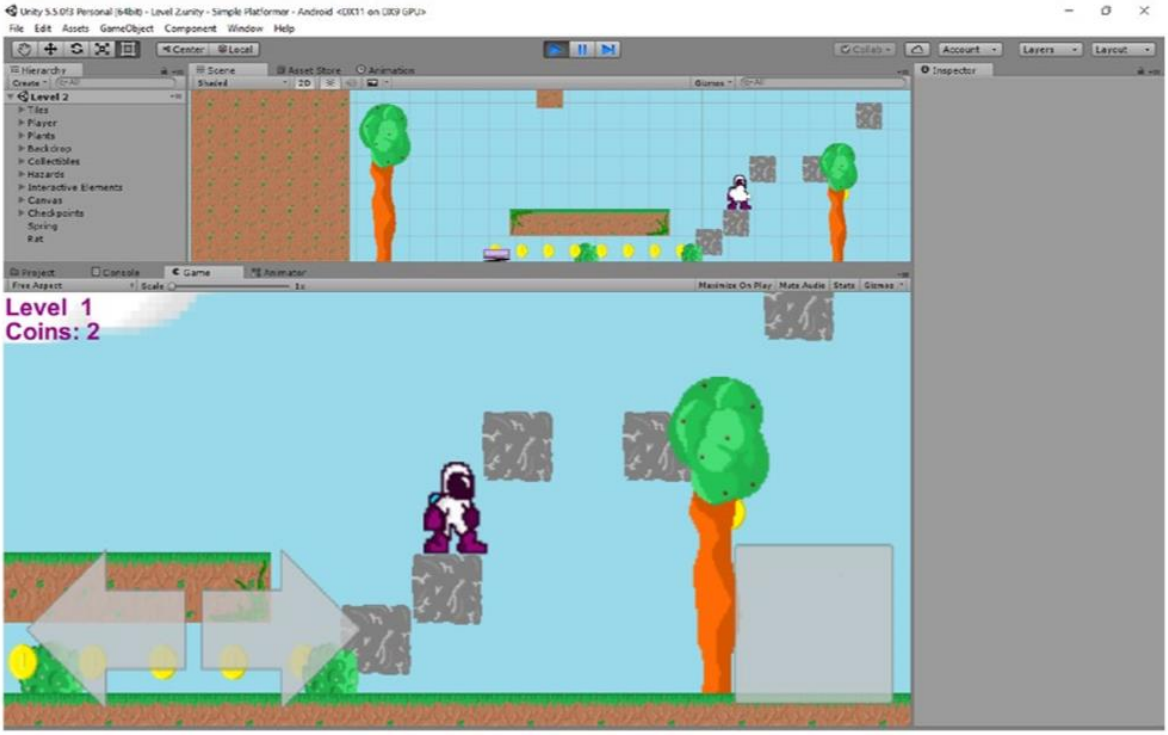
Kami ingin ubin platform ini mulai runtuh begitu kami mendarat di atasnya, jadi kami akan membuat skrip baru bernama Crumble. Skrip ini hanya akan memulai penghitung waktu segera setelah player menyentuh objek dan kemudian menyebabkan objek jatuh dan menghilang setelah penghitung waktu selesai. Kodenya terlihat seperti ini:

```
public class Crumble : MonoBehaviour {
    private Player player;
    private Rigidbody2D rb;
    public int timeToCollapse;
    private int timeLeft;
    public int timeToRestore;
    private int restoreTime;
```

```
        private float startY;
        private float startX;
        // Use this for initialization
        void Start () {
        rb = GetComponent<Rigidbody2D>();
        player = FindObjectOfType<Player>();
        startX = transform.position.x;
        startY = transform.position.y;
        timeLeft = -70;
}
// Update is called once per frame    void Update () {
        if (timeLeft > -70)      {
        timeLeft = timeLeft - 1;
        }
        if (timeLeft == 0)
        {
        rb.constraints = RigidbodyConstraints2D.None;
        }
        if (timeLeft == -62)
        {
        GetComponent<Renderer>().enabled = false;
        restoreTime = timeToRestore;
        }
        if (restoreTime > 0)
        {
        restoreTime = restoreTime - 1;
        }
        if (restoreTime == 2)
        {
          transform.position = new Vector3(startX, startY);
          transform.rotation = Quaternion.identity;
          GetComponent<Rigidbody2D>().velocity = Vector3.zero;
          rb.constraints = RigidbodyConstraints2D.FreezeAll;
          GetComponent<Renderer>().enabled = true;
          }
          }
          void OnCollisionEnter2D(Collision2D other)
          {
          if (other.gameObject.tag == "Player")
          {
          timeLeft = timeToCollapse;
          }
        }
}
```

Saat player menyentuh bercollision, itu memulai hitungan mundur, yang akan menjadi nilai yang dipilih di Inspector. Ketika hitungan mundur melewati nol, batasan dihilangkan dari Rigidbody, memungkinkannya jatuh dan berputar di udara. Penghitung waktu terus

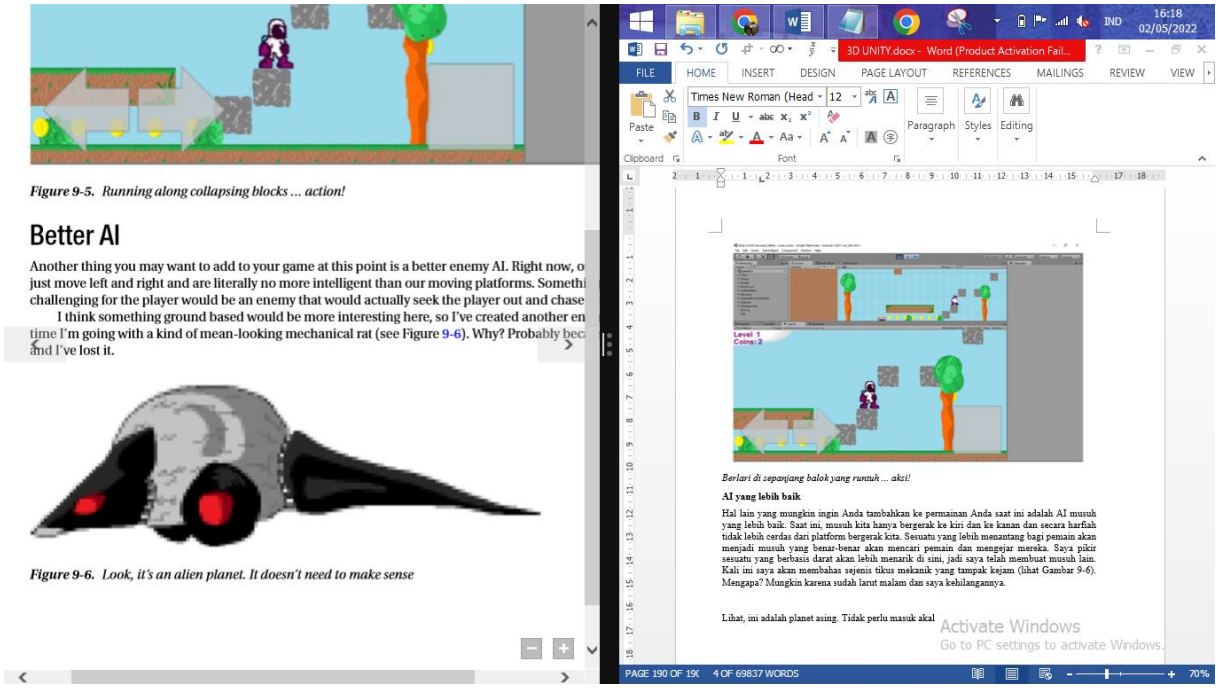
menghitung mundur melewati nol hingga –70 sehingga kita punya waktu untuk melihat platform jatuh dan menghilang. Tepat sebelum waktu mencapai titik itu, objek akan berhenti dirender dan menjadi tidak terlihat. Ini kemudian akan memulai hitungan mundur baru: penghitung waktu pemulihan. Ini juga diatur di Inspector, dan ketika ini menghitung mundur ke nol, objek dikembalikan ke posisi semula, dengan batasan yang baru dibekukan, rotasi disetel ke nol, dan rendering kembali. Dari sudut pandang player, ini kemudian membuat ubin yang dapat berdiri untuk jangka waktu terbatas sebelum jatuh dari layar dan akhirnya menghilang. Anda mungkin ingin menambahkan animasi "gemuruh" dan sound effect untuk drama tambahan, dan dari sana Anda dapat memperkenalkan beberapa platform refleksif yang keren. Itu semua harus terlihat sesuatu



Gambar 6.40 Berlari di sepanjang balok yang runtuh ... aksi!

AI yang lebih baik

Hal lain yang mungkin ingin Anda tambahkan ke permainan Anda saat ini adalah AI musuh yang lebih baik. Saat ini, musuh kita hanya bergerak ke kiri dan ke kanan dan secara harfiah tidak lebih cerdas dari platform bergerak kita. Sesuatu yang lebih menantang bagi player akan menjadi musuh yang benar-benar akan mencari player dan mengejar mereka. Saya pikir sesuatu yang berbasis darat akan lebih menarik di sini, jadi saya telah membuat musuh lain. Kali ini saya akan membahas sejenis tikus mekanik yang tampak kejam. Mengapa? Mungkin karena sudah larut malam dan saya kehilangannya.



Gambar 6.41 Lihat, ini adalah planet asing. Tidak perlu masuk akal

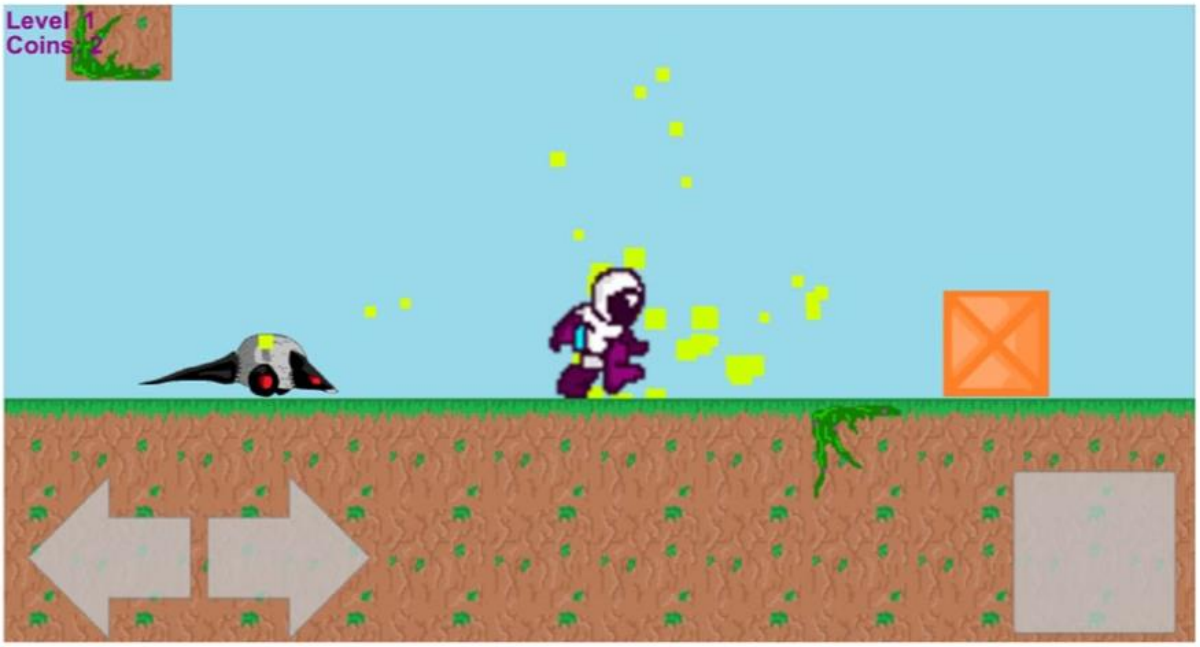
Si kecil ini akan menggunakan skrip baru, yang akan disebut GroundEnemy. Perilaku dasar karakter ini adalah mengikuti player di tanah. Jadi, jika posisi horizontal kita lebih besar dari posisi player, kita kurangi nilai X. Jika lebih kecil, kita naikkan nilainya. Kami juga membutuhkan tikus untuk membalik ketika dia mengubah arah, seperti player. Script sederhana ini terlihat seperti ini:

```
public class GoundEnemy : MonoBehaviour {
private Player player;
private int facing;
public float enemySpeed;
void Start ()
{
player = FindObjectOfType<Player>();
}
void Update ()
{
        if (gameObject.transform.position.x > player.transform.position.x)
        {
        gameObject.transform.position = new
        Vector2(gameObject.transform.position.x - enemySpeed,
        gameObject.transform.position.y);
        if (facing == 0)
        {
        facing = 1;
        transform.localScale = new Vector3(.2f, .2f, 1f);
        }
        }
                if (gameObject.transform.position.x < player.transform.position.x)
        {
```

```
                gameObject.transform.position = new
                Vector2(gameObject.transform.position.x + enemySpeed,
                gameObject.transform.position.y);
                if (facing == 1)
                {
                facing = 0;
                transform.localScale = new Vector3(-.2f, .2f, 1f);
                        }
                }
        }
}
```

Fakta menyenangkan: game pertama yang pernah saya buat didasarkan pada skrip ini (kecuali dalam BASIC untuk ZX Spectrum). Ini adalah titik di mana saya "mendapat" pemrograman. Saya membuat titik yang bisa bergerak di sekitar layar dan akan dikejar oleh titik kedua. Tujuan player adalah untuk mengelabui musuh agar mendarat di ranjau kecil (titik merah). Mungkin itu bukan fakta yang menyenangkan … terkadang saya bingung.

Tentu saja, Anda juga harus menambahkan skrip Hazards jika Anda ingin orang jahat itu benar-benar mematikan (dan Anda harus menambahkan metode onCollissionEnter2D sehingga tumbal dan pemicu akan membunuh player).



Gambar 6.42 Lari Kevin, ada semacam robot tikus

Bagaimanapun, skrip ini agak terlalu sederhana saat ini. Seperti itu, musuh akan mulai mengejar player segera setelah permainan dimulai dan mungkin berakhir terdampar di lubang di suatu tempat. Tidak hanya itu tetapi dia sangat mudah dibodohi dan akan terhalang oleh hampir semua rintangan. Untuk mengatasi masalah ini, pertama-tama kita akan membuatnya beraksi setelah player mencapai jarak tertentu dan kemudian berhenti mengikuti setelah player pergi. Kedekatan akan menjadi bilangan bulat publik yang dapat kita atur di Inspector. Tip yang berguna adalah memastikan bahwa Anda bermain-main dengan rentang yang berbeda dan kecepatan yang berbeda. Idealnya, Anda tidak ingin musuh mulai bergerak sampai player dapat melihatnya di layar. Demikian juga, kecepatan ideal adalah kecepatan

yang memungkinkan player untuk melarikan diri tetapi hanya setelah pengejaran yang menegangkan.

Perhatikan bagaimana mengutak-atik angka-angka ini bahkan hanya sedikit mengubah kecepatan dan ketegangan permainan Anda secara signifikan. Ini mirip dengan menjadi sutradara film, dan kita akan membahas aspek-aspek semacam ini lebih banyak di bab berikutnya. Sementara kita melakukannya, mengapa kita tidak menjadi sedikit lebih kreatif dan memberi musuh kita kemampuan untuk melintasi lingkungan sedikit lebih banyak? Misalnya, akan sangat bagus jika Roborat (ya, saya akan memanggilnya Roborat) akan melompati balok untuk mencoba dan menjangkau player dan jika dia bisa keluar dari lubang. Untuk melakukan ini, kami akan menggunakan fitur baru: raycasts.

Menggunakan Raycasts

Raycast sedikit mirip dengan sensor terbalik di mobil Anda; mereka mengirimkan sinar yang memeriksa collision dan kemudian mengembalikan "benar" jika ada pukulan. Apa yang ingin kita lakukan adalah memberi musuh kita kemampuan untuk melihat apakah ada sesuatu yang menghalangi jalannya dan kemudian melompatinya. Itu berarti ia juga membutuhkan groundCheck sendiri (agar tidak terus melompat dan terbang di udara). Tangani ini dengan cara yang sama persis seperti yang Anda lakukan untuk player: buat GameObject kosong dengan radius kecil dan kemudian minta untuk memeriksa ground untuk mengatur Boolean. Anda dapat menyalin dan menempelkan kode secara langsung, dan kode tersebut akan terlihat seperti ini setelah ditempatkan:

```
public class GoundEnemy : MonoBehaviour {
        private Player player;
        private int facing;
        public float enemySpeed;
        private bool chaseIsOn;
        public int attackRange;
        public Transform groundCheck;
        public float groundCheckRadius;
        public LayerMask whatIsGround;
        private bool onGround;
        void Start () {
        player = FindObjectOfType<Player>();
        }
        void FixedUpdate()
        {
        onGround = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius,
        whatIsGround);
}
```

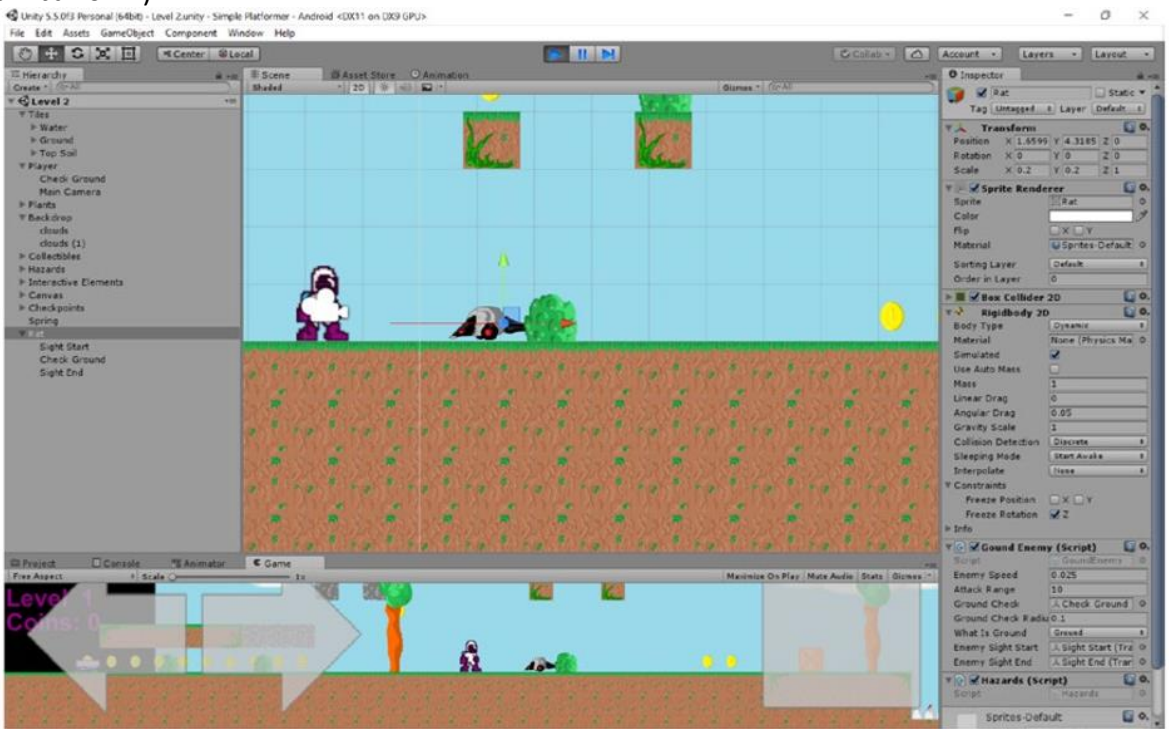
Anda juga harus memiliki sesuatu yang terlihat seperti Gambar 6.43 di Inspector

Gambar 6.43 Ingat, Anda membuat GameObject kosong tepat di bawah karakter untuk digunakan sebagai transformasi dan kemudian menambahkannya ke Inspector

Raycast adalah garis yang tidak terlihat, dan kami akan menggunakan lebih banyak transformasi dan GameObjects kosong untuk menentukan posisinya. Untuk memeriksa apakah ini berfungsi, kita akan menggunakan fungsi debug untuk menggambar garis lurus di antara dua titik. Ini adalah fitur Unity yang berguna yang memungkinkan Anda menggambar langsung ke layar dengan cara yang hanya terlihat di tampilan Scene. Para player tidak akan melihatnya, tetapi kami dapat menggunakannya untuk menguji permainan kami.

Jadi, buat dua GameObject kosong baru yang merupakan anak-anak tikus (yang terdengar seperti judul buku yang aneh: Children of the Rat). Yang pertama harus menjadi pusat mati, yang akan kita sebut Sight Start. Yang kedua akan menjadi dua unit di depan tikus dan akan disebut Sight End. Sekarang kita akan membuat dua transformasi publik baru, yang akan menjadi musuhSightStart dan musuhSightEnd. Sekali lagi, kita akan menggunakan Inspector untuk meletakkan dua objek kosong yang baru saja kita buat ke sana. Jika Anda melakukan ini dengan benar, Anda seharusnya dapat menambahkan baris ini:
Debug.DrawLine(enemySightStart.position, enemySightEnd.position, Color.red);

Dan kemudian lihat garis merah muncul di tampilan Scene antara dua titik (lihat Gambar 6.44).



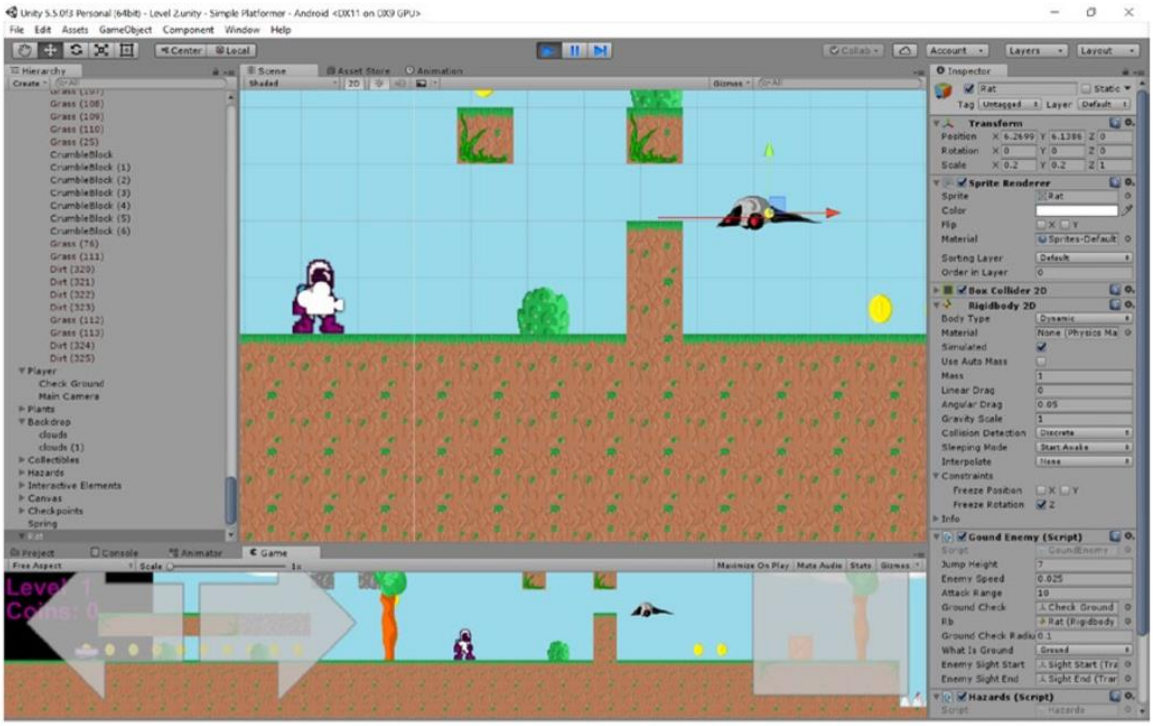
Gambar 6.44 Sekarang kita hanya akan menukar baris ini dengan raycast.

Setelah transformasi diatur, tikus Anda akan terlihat seperti sedang berkelahi raycast kami akan pergi tepat di mana garis saat ini, tetapi dapat berguna untuk menjaga garis tetap, mengingat raycasts benar-benar tidak terlihat dan sebaliknya bisa sulit untuk memvisualisasikan. Untungnya, menggunakan raycast kami cukup sederhana—terutama karena Anda terbiasa menggunakan lingkaran tumpang tindih. Kami akan menggunakan Physics2D.Linecast untuk pekerjaan ini. Ada jenis raycast lain, seperti Circlecast, tetapi untuk permainan 2D dengan aturan sederhana, garis adalah pilihan yang paling efisien. Kita perlu memberikan fungsi ini titik awal dan titik akhir (seperti yang kita lakukan dengan garis kita) dan kemudian kita juga akan memberikan layer mask. Kami tidak ingin musuh melompati player, jadi layer yang dicari adalah Ground. Ini akan masuk ke dalam pembaruan dan hanya akan beroperasi jika chaseIsOn Boolean benar (yaitu, jika player telah terlihat):

```
if (Physics2D.Linecast(enemySightStart.position, enemySightEnd.position, whatIsGround)) {
Jump();
}
```

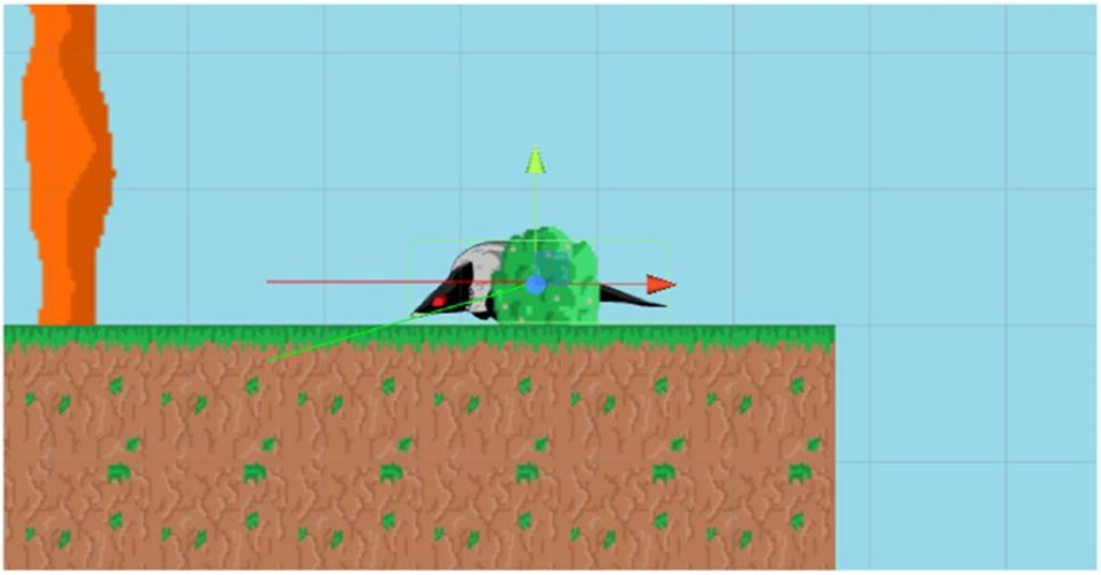
Seperti yang Anda lihat, saya juga telah membuat metode Jump seperti yang kami lakukan untuk player. Ini harus akrab:

```
private void Jump()
        {
        if (onGround)
        {
        rb.velocity = new Vector2(rb.velocity.x, jumpHeight);
        }
}
```



Gambar 6.45 Tikus-tikus yang melompat—scene umum bagi seorang mantan warga London seperti saya

Kabar baiknya adalah ketika tikus berbalik, objek Sight End juga akan terbalik karena itu adalah anak tikus. Tidak perlu lebih banyak kode untuk membuat tikus mencoba melompati jurang juga; kita hanya perlu raycast kedua yang melihat ke tanah tepat di bawah yang pertama. Tambahkan itu dan pastikan musuh melompat ketika titik itu tidak tumpang tindih dengan tanah.



Gambar 6.46 Teman tikus mencari lantai

Coding Perilaku Musuh

Saya juga menambahkan sesuatu yang lain, yang pada dasarnya adalah kode yang sama dari skrip BackAndForth. Saya ingin tikus bergerak ke kiri dan ke kanan sehingga "berpatroli" sampai mulai mengejar player. Tidaklah terlihat alami untuk memiliki musuh yang tetap sempurna sampai player terlihat ... meskipun itu akan menyeramkan, saya setuju. Kami akan membuat kode ini sedikit lebih pintar, dengan membuat musuh mengubah arah saat berpatroli jika mendekati tepi sehingga tidak berpatroli dari platform atau ke dinding. Saya juga akan memindahkan kodenya sedikit sehingga tidak semuanya berada di fungsi Update—yang terlihat sedikit jelek. Jika Anda ingin mengambil rute yang mudah, Anda cukup menyalin dan menempelkan kode ini untuk membuat musuh darat Anda sendiri:

```
public class GoundEnemy : MonoBehaviour {
    private Player player;
    private int facing;
    public int jumpHeight;
    public float enemySpeed;
    private bool chaseIsOn;
    public int attackRange;
    public Transform groundCheck;
    public Rigidbody2D rb;
    public float groundCheckRadius;
    public LayerMask whatIsGround;
    private bool onGround;
    public Transform enemySightStart;
    public Transform enemySightEnd;
    public Transform enemySightEnd2;
    private float startX;
    public double amountToMove;
    void Start () {
    player = FindObjectOfType<Player>();
    rb = GetComponent<Rigidbody2D>();
    startX = gameObject.transform.position.x;
```

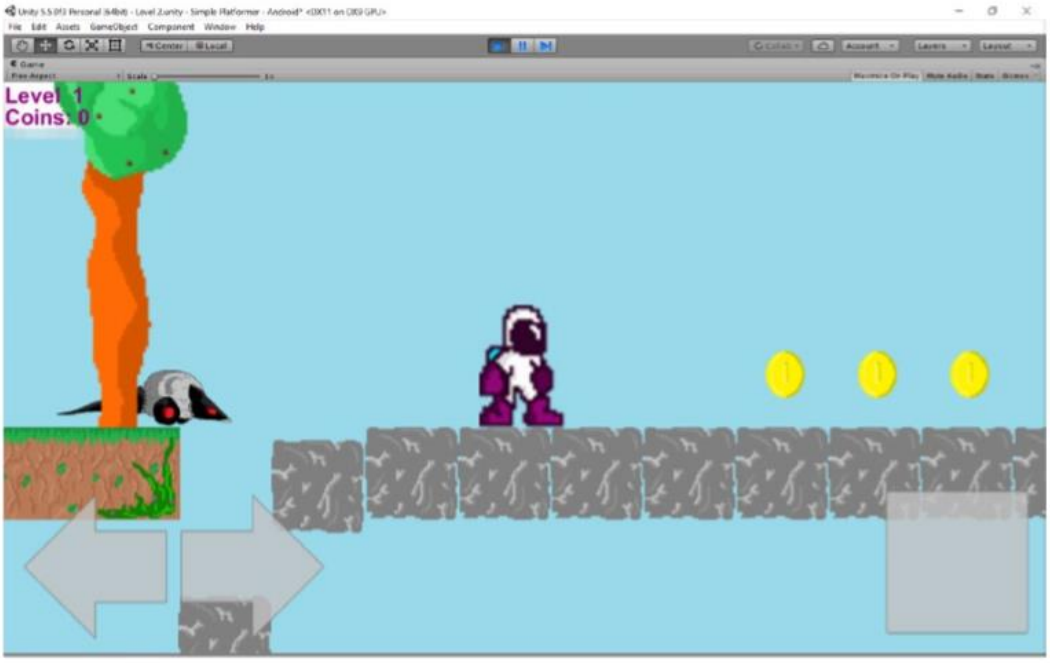
```
        facing = 3;
        }
void FixedUpdate()
{
                onGround = Physics2D.OverlapCircle(groundCheck.position,
                groundCheckRadius, whatIsGround);
                Debug.DrawLine(enemySightStart.position, enemySightEnd.position,
                Color.red);
                Debug.DrawLine(enemySightStart.position, enemySightEnd2.position,
                Color.green);
                }
                void Update()
                {
                if (gameObject.transform.position.x - player.transform.position.x <
                attackRange && gameObject.transform.position.x -
                player.transform.position.x > -attackRange && chaseIsOn == false)
                {
                chaseIsOn = true;
                }
                if (gameObject.transform.position.x - player.transform.position.x >
                attackRange || gameObject.transform.position.x - player.transform.position.x
                < -attackRange && chaseIsOn == true)
                {
                if (chaseIsOn)
                {
                startX = gameObject.transform.position.x;
                }
                chaseIsOn = false;
                }
                if (chaseIsOn)
                {
                Pursuit();
                } else
                {
                Patrol();
                }
                }
                private void Patrol()
                {
                if (facing == 3)
                {
                facing = 0;
                transform.lo
                calScale = new Vector3(-.2f, .2f, 1f);
                }
                if (gameObject.transform.position.x < startX + amountToMove && facing == 0)
                {
```

```
gameObject.transform.position = new Vector2(gameObject.transform.position.x + enemySpeed / 2, gameObject.transform.position.y);
}
else if (gameObject.transform.position.x >= startX + amountToMove && facing == 0)
{
facing = 1;
transform.localScale = new Vector3(.2f, .2f, 1f);
}
else if (gameObject.transform.position.x > startX && facing == 1)
{
gameObject.transform.position = new Vector2(gameObject.transform.position.x - enemySpeed / 2, gameObject.transform.position.y);
}
else if (gameObject.transform.position.x <= startX && facing == 1)
{
facing = 0;
transform.localScale = new Vector3(-.2f, .2f, 1f);
}
if (Physics2D.Linecast(enemySightStart.position, enemySightEnd2.position, whatIsGround) == false || Physics2D.Linecast(enemySightStart.position, enemySightEnd.position, whatIsGround))
{
if (facing == 1)
{
facing = 0;
transform.localScale = new Vector3(-.2f, .2f, 1f);
}
else
{
facing = 1;
transform.localScale = new Vector3(.2f, .2f, 1f);
}
}
}
private void Pursuit()  {
if (Physics2D.Linecast(enemySightStart.position, enemySightEnd.position, whatIsGround) || Physics2D.Linecast(enemySightStart.position, enemySightEnd2. position, whatIsGround) == false)
{
Jump();
}
if (gameObject.transform.position.x > player.transform.position.x)
{
```

```
                gameObject.transform.position = new
                Vector2(gameObject.transform.position.x - enemySpeed,
                gameObject.transform.position.y);
                if (facing == 0 || facing == 3)
                {
                facing = 1;
                transform.localScale = new Vector3(.2f, .2f, 1f);
                }
                }
                if (gameObject.transform.position.x < player.transform.position.x)
                { gameObject.transform.position = new
                Vector2(gameObject.transform.position.x + enemySpeed,
                gameObject.transform.position.y);
                if (facing == 1 || facing == 3)
                {
                facing = 0;
                transform.localScale = new Vector3(-.2f, .2f, 1f);
                }
        }
}
private void Jump()  {
                if (onGround)
                {
                rb.velocity = new Vector2(rb.velocity.x, jumpHeight);
                }
                }
                void OnCollisionEnter2D(Collision2D collision)
                {
                if (collision.gameObject.tag == "Enemy")
                {
                Physics2D.IgnoreCollision(collision.collider, GetComponent<Collider2D>());
                }
        }
}
```

Ini masih cukup sederhana saat AI musuh berjalan, tetapi menghasilkan beberapa perilaku yang cukup menyenangkan. Orang jahat kita sekarang akan perlahan-lahan berpatroli (saya setel ini ke kecepatan setengah) sampai player mendekat. Menjadi tikus, dia bisa mencium bau Kevin, dan begitu Kevin terlalu dekat, tikus itu mengejar dan melompati rintangan dan lubang untuk mengejar. Jika dia menyentuh Kevin, kita mati. Jika Kevin berhasil kabur tepat waktu, tikus itu kehilangan minat dan berpatroli di mana pun ia berada. Metode terakhir—metode onCollission2D—dilakukan untuk mencegah tikus saling bercollision. Saya memasukkan ini sehingga Anda bisa membuat "lubang" tikus untuk faktor kotor. Anda harus menandai tikus sebagai Musuh agar bisa berfungsi.

Jika Roborat menemukan dirinya terdampar di sebuah platform (lihat Gambar 6.47), dia akan sering membeku. Jadi, dia tidak sempurna. Tapi dia masih cukup menarik dan tentu saja cukup dinamis untuk menciptakan banyak peluang gameplay.



Gambar 6.47 Begitu lama, pengisap!

Dan merasa bangga: Anda baru saja menciptakan kecerdasan buatan pertama Anda. Semua dalam satu hari kerja.

Mempersenjatai Player

Tikus kami ternyata menjadi ancaman yang cukup kejam dan tentu saja cukup menantang untuk membuat player kami kesulitan. Sudah waktunya kami memberi player kami cara untuk melawan. Membuat peluru yang dapat ditembakkan player relatif mudah, meskipun kita perlu melakukan sedikit juggling untuk memastikan bahwa kita mereferensikan instance yang tepat dari objek Bullet kita. Izinkan saya untuk menjelaskan. Pertama, kita perlu membuat GameObject baru, yang disebut Bullet. Ini adalah peluru kami, yang akan memiliki variabel publik untuk arah dan kecepatannya. Itu juga akan memiliki Collider. Scriptnya terlihat seperti ini:

```
public class Bullet : MonoBehaviour {
        public float speed;
        public int direction;
        private int timeLeft;
        public GameObject Blood;
        void Start () {
        timeLeft = 100;
        }
        void Update () {
        timeLeft = timeLeft - 1;
        if (timeLeft < 1)
        {
        Destroy(gameObject);
        }
        if (direction == 0)
        {
```

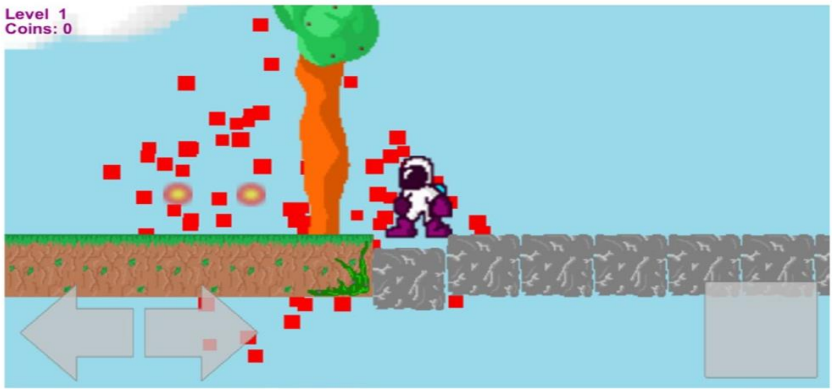
```
            gameObject.transform.position = new
            Vector2(gameObject.transform.position.x - speed,
            gameObject.transform.position.y);
            }
            else if (direction == 1)
            {
            gameObject.transform.position = new
            Vector2(gameObject.transform.position.x + speed,
            gameObject.transform.position.y);
            }
            }
            void OnCollisionEnter2D(Collision2D other)
            {
            if (other.gameObject.tag == "Enemy") { Destroy(other.gameObject);
            Instantiate(Blood, transform.position, transform.rotation);
            }
            else if (other.gameObject.tag == "Player")
            { Physics2D.IgnoreCollision(other.collider, GetComponent<Collider2D>());
            }
            else
            {
            Destroy(gameObject);
            }
        }
}
```

Perhatikan bahwa membunuh seekor tikus akan menyebabkan efek partikel berdarah yang sama yang terjadi ketika player mati. Itu berarti Anda perlu menambahkan GameObject publik di Inspector sama seperti sebelumnya. Perhatikan juga bahwa metode onCollision kami memeriksa tag objek, sehingga tikus dihancurkan dengan darah, player diabaikan, dan apa pun menghancurkan peluru. Peluru juga habis waktu setelah durasi yang ditetapkan dan penghancuran diri untuk menghapus dirinya sendiri dari memori, seperti yang dilakukan efek partikel sebelumnya. Demikian juga, kami juga ingin membuat objek publik baru bernama Bullet di skrip Player. Anda perlu menempatkan Prefab peluru di sana melalui Inspector. Anda kemudian akan menambahkan sedikit kode berikut:

```
if (Input.GetKeyDown(KeyCode.LeftControl)) {
var newBullet = Instantiate(bullet, transform.position, transform.rotation);
var bulletScript = newBullet.GetComponent<Bullet>(); bulletScript.direction = facing;
}
```

Ini adalah bagian baru. Di sini, kami ingin tidak hanya membuat instance objek baru tetapi juga mengatur beberapa properti untuk objek tersebut saat objek tersebut dibuat. Untuk melakukan itu, kita perlu menggunakan GetComponent untuk mendapatkan referensi ke skrip dari instance ini. Dari sana, kita kemudian dapat mengakses variabel publik dan mengubahnya. Pada akhirnya, Anda harus mendapatkan peluru yang dapat menembus musuh (Tentu saja, nanti Anda akan ingin masuk ke kanvas Anda dan menambahkan tombol "api". Saya meninggalkan ini karena saya pikir peluru membuat player kami sedikit dikuasai. Bagian ini di sini hanya untuk Anda sendiri referensi dan agar Anda dapat menambahkan peluru dan

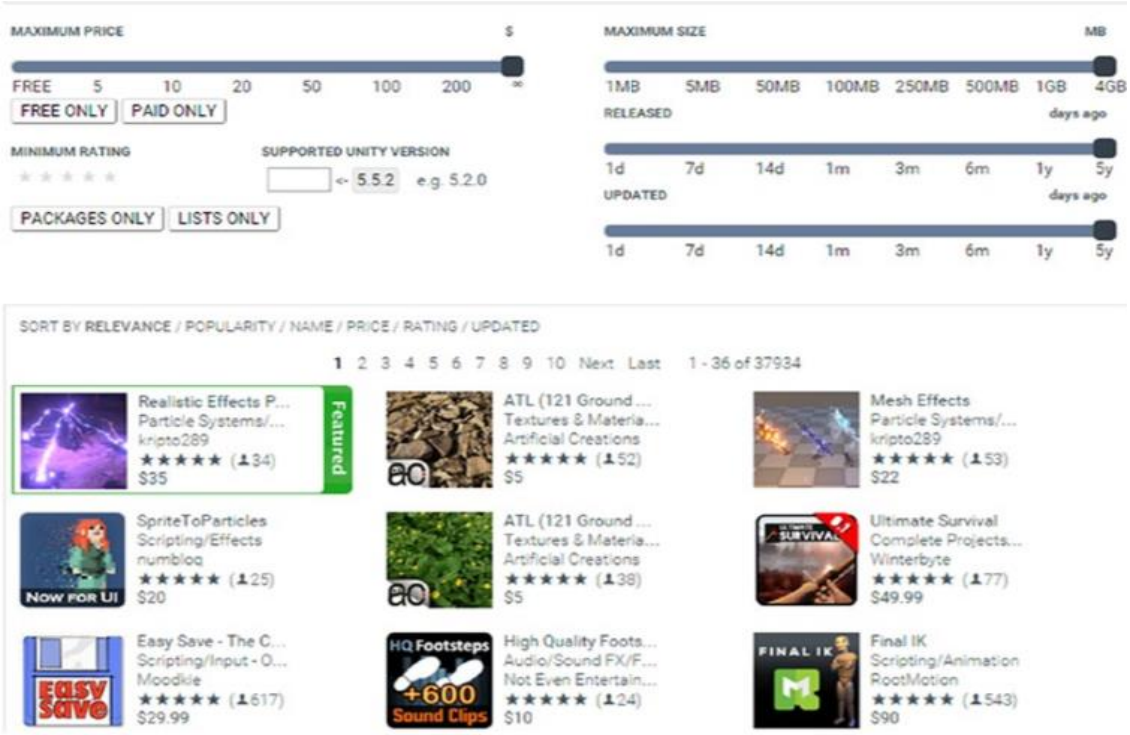
senjata ke permainan Anda jika Anda mau. Demikian juga, Anda juga perlu menambahkan semacam senjata ke sprite player Anda dan mungkin animasi baru.)



Gambar 6.48 Sayangnya, Roborat yang malang

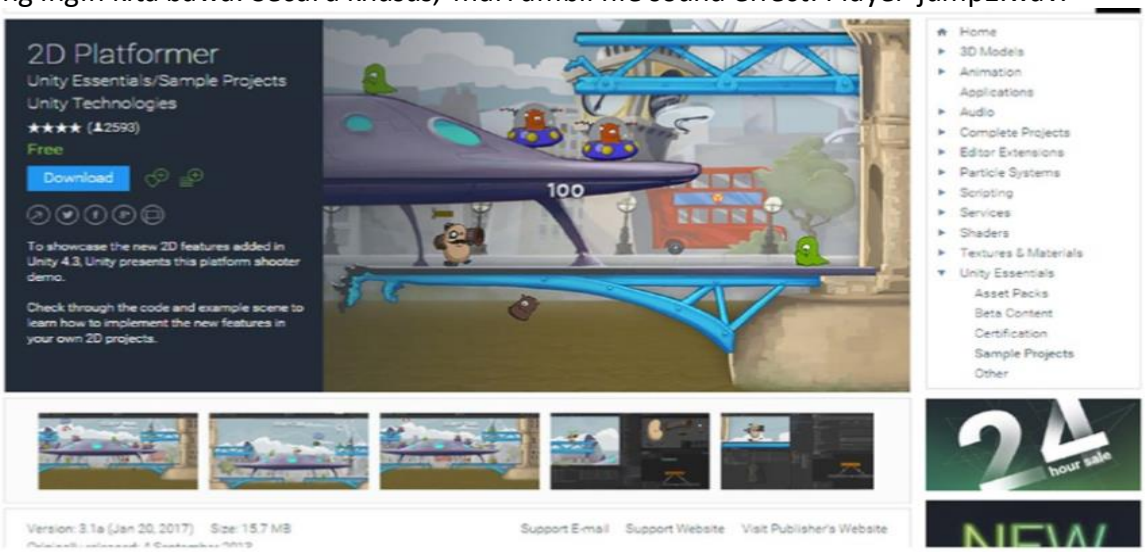
Menggunakan Asset dari Asset Store

Saya dapat terus berbicara dengan Anda tentang cara membuat objek dan perilaku yang berbeda sampai wajah saya pucat, tetapi saya tidak akan pernah menunjukkan kepada Anda bagaimana membuat semua yang mungkin Anda butuhkan. Bagaimana dengan portal untuk menteleportasi player? Bagaimana dengan sakelar yang membuka pintu? Bagaimana dengan musuh terbang? Atau lompat ganda? Atau powerup? Mudah-mudahan, Anda akan dapat menemukan beberapa hal ini sendiri sekarang. Bab ini telah memperkenalkan raycast, membahas lebih detail tentang instantiating, dan secara umum menambah pengetahuan Anda. Dengan menggunakan informasi baru itu dan membangun apa yang sudah Anda ketahui, Anda seharusnya dapat menemukan solusi kreatif untuk hampir semua masalah yang Anda impikan. Ingat, tidak ada yang namanya kekurangan sumber daya, hanya kekurangan akal. Tetapi jika Anda tidak dapat mengetahuinya sendiri, atau Anda tidak punya waktu atau keinginan, Anda dapat menemukan pabrikan yang telah dibuat oleh orang lain (termasuk Unity Technologies) dan menambahkannya ke proyek Anda sendiri.



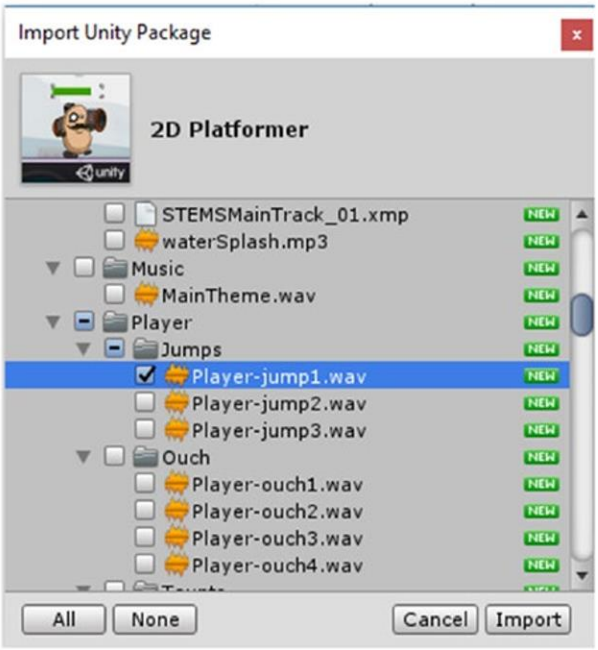
Gambar 6.49 Asset Store dengan segala kemegahannya

Untuk mulai menjelajah di sini, cukup pilih tab Asset Store dan lihat-lihat. Anda akan melihat banyak hal—mulai dari efek partikel, skrip, paket sprite, hingga demo keseluruhan game. Anda akan melihat bahwa ada penggeser untuk memungkinkan Anda mengatur harga (banyak aset gratis) dan ukuran file dan Anda dapat memilih dari kategori di sisi kanan. Jika Anda mengklik nama penerbit—misalnya, Unity Technologies—Anda dapat melihat semua aset dan paket mereka. Saya ingin Anda menemukan pilihan aset yang disebut Platformer 2D dari Unity Technologies (Gambar 6.50). Ini pada dasarnya adalah game 2D lengkap, tetapi daripada menggunakan semuanya, untuk saat ini mari kita coba memilih hanya satu elemen yang ingin kita bawa. Secara khusus, mari ambil file sound effect: Player-jump1.wav.



Gambar 6.50 Ini akan berhasil dengan baik

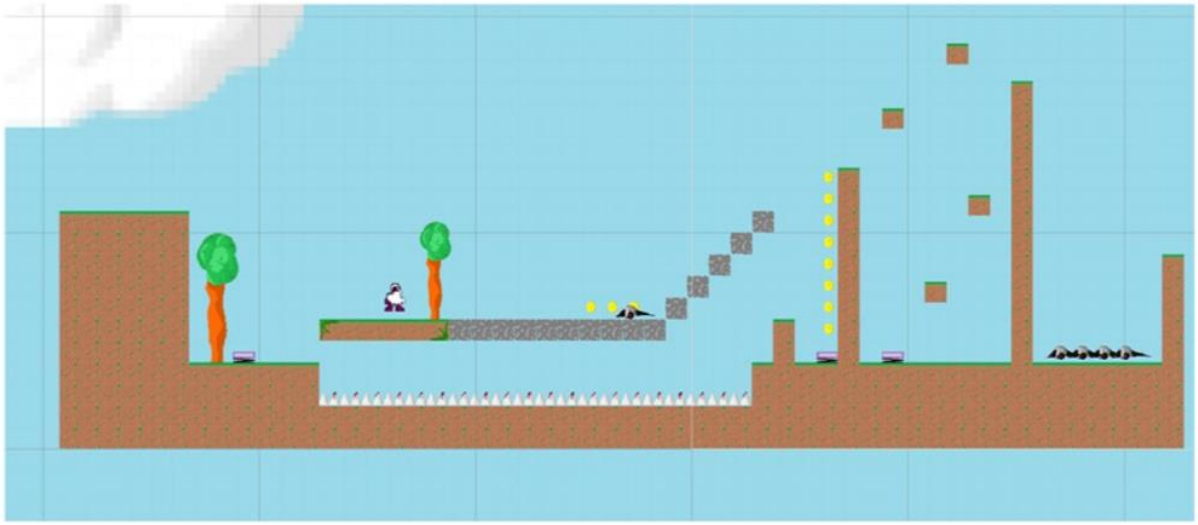
Klik Unduh di toko dan kemudian klik Impor. Unity akan memperingatkan Anda bahwa Anda berisiko menimpa kerja keras Anda, tetapi jangan khawatir tentang itu—Anda akan dapat memilih dengan tepat apa yang ingin Anda tambahkan di layar berikutnya. Jadi klik OK, batalkan pilihan semuanya dari daftar di jendela yang terbuka (klik Tidak Ada), lalu pilih ulang secara manual hanya sound effect.



Gambar 6.51 Hanya pilih yang Anda inginkan

Sekarang klik Impor dan setelah sedetik Anda akan menemukan subfolder baru di folder Audio Anda. Klik melalui ini dan Anda akhirnya akan mendapatkan sound effect yang Anda inginkan. Sekarang Anda dapat membuat sumber audio seperti sebelumnya dan memutarnya saat Kevin melompat. Suara ini jelas salah baginya, tetapi mudah-mudahan Anda melihat kemungkinan di sini—Anda dapat menemukan hampir semua hal yang Anda inginkan di Asset Store, dan meskipun Anda mungkin perlu membayar untuk beberapa, Anda akan mendapati bahwa biasanya tidak. terlalu mahal. Seringkali kualitas dan profesionalisme sumber daya di sini akan melampaui apa yang dapat Anda lakukan sendiri, dan ini akan mempercepat developeran sekaligus menghasilkan produk akhir dengan nilai produksi yang lebih tinggi dan lebih "kilau". Saya tidak ingin merekomendasikan sesuatu yang spesifik, mengingat isi toko berubah sepanjang waktu dan itu mungkin memberi tanggal pada buku ini. Saat ini, bagaimanapun, ada pilihan 2D Essentials-curated yang mencakup beberapa hal keren seperti efek partikel cuaca, pencahayaan dinamis, air 2D reflektif, dan "Kamera Pro 2D."

Mudah-mudahan, pikiran Anda sekarang terguncang pada kemungkinan, tetapi yang paling penting adalah Anda tidak terbawa suasana. Desain game yang bagus tidak hanya berarti melemparkan semua hal keren yang Anda bisa ke arah player—ini juga tentang pengendalian diri seperti hal lainnya. Dengan kekuatan besar datang tanggung jawab besar.



Gambar 6.52 Level 2 saya terlihat seperti ini sekarang, yang semuanya salah. Cari tahu alasannya di bab berikutnya.

6.10 MEMBUAT DAN MENGOPTIMALKAN GAME MENYENANGKAN

Selamat, Anda sekarang dapat membuat game dengan Unity! Tidak, serius, jika Anda berhenti membaca sekarang, saya cukup yakin Anda bisa membangun game lengkap dengan level dan segalanya. Dan Anda mungkin bahkan dapat menemukan cara untuk merilisnya di Play Store setelah sedikit membaca. Ya, Anda bisa membuat game. Tapi bisakah Anda membuat game yang bagus? Karena itu adalah dua hal yang sangat berbeda. Ingat: dengan kekuatan besar datang tanggung jawab besar. Saya merasa saya akan merugikan dunia jika saya mengajari Anda cara membuat game dan kemudian membebaskan Anda tanpa panduan tentang apa yang membuat game menyenangkan. Itulah yang akan kita lihat dalam bab ini. Kami juga akan membahas sedikit tentang pengoptimalan (agar game Anda berjalan lebih lancar dan menggunakan lebih sedikit ruang) dan bahkan cara membuat level Anda sedikit lebih cantik. Ini adalah ceri untuk menjadi yang teratas dari cookies pemrograman Anda. Ayo pergi!

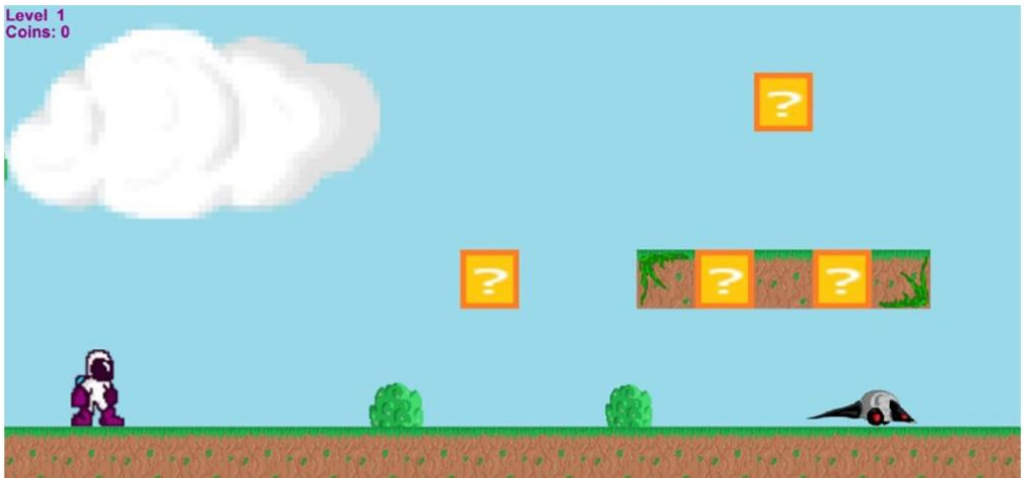
Orientasi dan Tutorial

Ingat hari-hari ketika Anda membeli game komputer dari toko dan kemudian menunggu di toko pakaian sementara ibumu selesai berbelanja? Anda mungkin senang duduk di sana karena Anda memiliki manual untuk dibaca, yang dipenuhi dengan latar belakang, tips, dan penjelasan tentang cara kerja semua yang ada di dalam game. Itu memenuhi Anda dengan antisipasi untuk permainan dan memastikan Anda memiliki ide bagus tentang bagaimana memulai setelah kartrid / disk dimasukkan.

Saat ini, game jarang datang dengan manual, dan itu terutama berlaku untuk game seluler. Tetapi itu tidak berarti Anda dapat berasumsi bahwa player Anda langsung tahu cara bermain. Faktanya, Anda bahkan tidak boleh berasumsi bahwa mereka pernah memainkan video game sebelumnya. Karena beberapa player Anda mungkin tidak akan memilikinya, dan mereka masih merupakan pelanggan yang ingin Anda pertahankan. Setiap permainan adalah yang pertama bagi seseorang. Jadi, menjadi tugas Anda untuk mengajari player "on the job", dan itu berarti Anda memerlukan level tutorial. Sebenarnya, tingkat tutorial hampir sama dengan anakronisme seperti instruksi manual itu sendiri. Angkat tangan Anda jika Anda dapat mengingat level tutorial terakhir yang benar-benar Anda nikmati. Tidak, tidak menyangka! Level pertama yang baik kemudian harus menginstruksikan player tentang bagaimana semuanya beroperasi tanpa secara eksplisit memberi tahu mereka satu hal pun. Ini berarti Anda harus menggunakan isyarat visual, serta kiasan gameplay yang menciptakan dasar pengetahuan dan kemudian membangunnya.

Membedah Level Pembukaan yang Sempurna

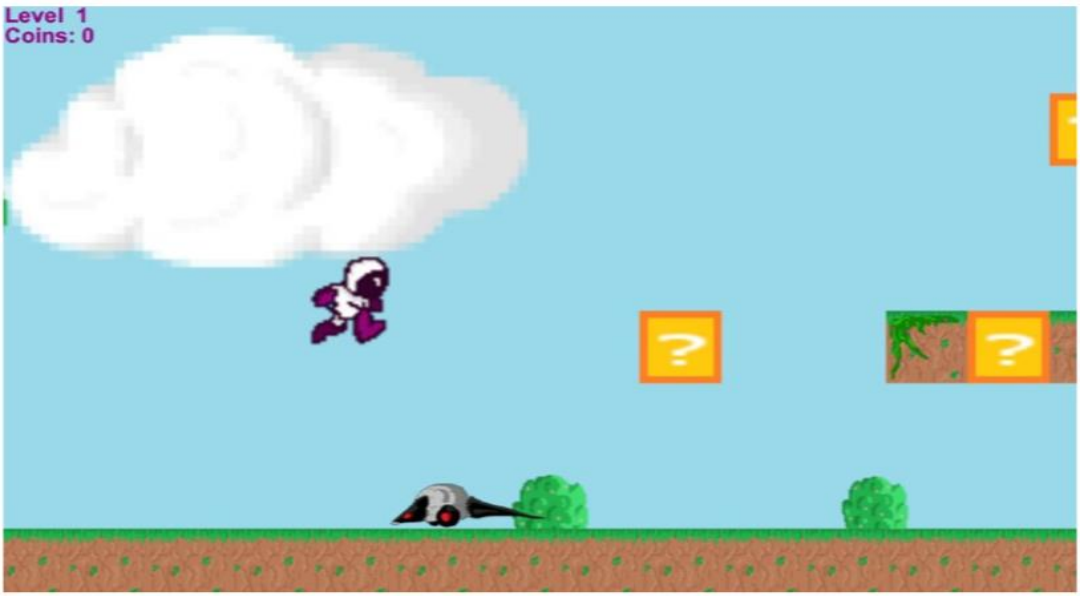
Meskipun kurangnya manual mungkin merupakan masalah modern, contoh sempurna dari level pembukaan yang secara implisit mengajarkan cara bermain kepada player dapat ditemukan di salah satu game klasik sepanjang masa: Super Mario Bros. Pembukaan game ini level, yang disebut Dunia 1-1, adalah salah satu level yang paling dianalisis dan sangat dipuji dalam semua sejarah game—dan untuk alasan yang bagus. Mari kita lihat cara kerjanya. Demi kenyamanan Anda, saya telah membuat ulang tata letak tingkat pertama menggunakan aset dari platformer sederhana kami (Gambar 10-1). Ini mungkin atau mungkin bukan penistaan.



Gambar 6.53 Hmm, ini anehnya akrab …

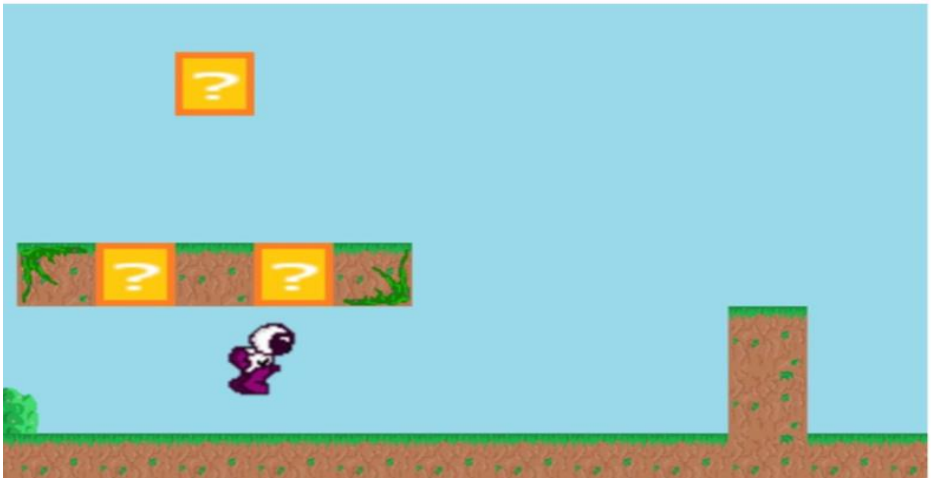
Sejak layar pertama, player mulai belajar cara bermain. Di sini, mereka disambut oleh protagonis mereka, Mario (atau dalam kasus kami, Kevin). Mario telah diposisikan di paling kiri layar dengan kamera mendorong ke depan ke kanan. Ini segera dan tanpa kata memberitahu player: ke kanan. Saat Mario berjalan ke kanan, dia akan melihat Goomba yang marah menuju ke arahnya (yang telah kami ganti dengan RoboRat kami). Tahukah Anda bahwa kita secara evolusioner diprogram untuk merasakan stres setiap kali sesuatu bergerak langsung ke arah kita? Itu sebabnya perjalanan adalah mimpi buruk. Pola gerakan ini

dikombinasikan dengan alis marah Goomba seharusnya cukup untuk memberi tahu kita bahwa kita perlu menghindari musuh. Satu-satunya cara yang bisa kita lakukan adalah melompat—jika tidak, kita mati.



Gambar 6.54 Lompat atau mati

Melompat adalah mekanik utama di Mario, dan pembukaan ini memastikan bahwa player memahami cara kerjanya sebelum mereka melangkah lebih jauh. Jika mereka kehilangan seluruh hidup mereka dan kembali ke awal, maka mereka tidak kehilangan apa-apa karena mereka belum membuat kemajuan apa pun. Jadi ini adalah tempat yang bagus untuk bereksperimen. Hal berikutnya yang akan ditemui Mario adalah kotak tanda tanya. Kotak ini memohon untuk disentuh oleh isyarat visual yang kuat yaitu tanda tanya. Simbol universal ini mengatakan, "Ooh, apa yang ada di sini?" ("Ooh" adalah opsional). Ketika Mario menyentuh tanda tanya dengan memantulkannya, dia akan menemukan bahwa itu menghasilkan jamur. Desain level berikutnya sedemikian rupa sehingga Mario hampir dipaksa untuk mengumpulkan jamur tersebut. Ini akan muncul dari kotak, perjalanan ke kanan di atas Mario, dan kemudian memantul kembali dari pipa ke kiri. Mario kemungkinan besar berada di bawah platform berikutnya pada saat ini dan bahkan jika dia mencoba melompat dan menghindari jamur, jamur itu mungkin masih akan mengenainya. Gambar 6.55 memperkirakan hal ini dalam game kami.



Gambar 6.55 Apakah Mario ini, jamur sekarang akan muncul

Dengan demikian, player belajar bagaimana mereka bisa menjadi "Super Mario." Levelnya berlanjut dalam nada ini, mengajari player setiap keterampilan yang mereka butuhkan dengan cara yang sederhana dan tanpa kata-kata. Beberapa level berikutnya berkaitan dengan memberi player banyak waktu untuk melatih keterampilan itu sebelum merangkai beberapa rintangan bersama untuk membuat tantangan nyata. Seiring berjalannya permainan, jumlah rintangan yang harus dielakkan secara berurutan akan meningkat, dan pada akhirnya kebiasaan dan tikungan baru akan mulai diperkenalkan.

Memastikan Player Anda Memahami Game Anda

Ini semua mungkin terdengar seperti akal sehat bagi Anda (bukankah Anda pintar), tetapi sangat mudah untuk melupakan poin-poin ini ketika Anda sedang dalam pergolakan desain level. Lihatlah apa yang sudah Anda buat. Saya tahu Anda hanya bermain-main pada saat ini, tetapi saya berani bertaruh bahwa Anda telah memasang beberapa jebakan yang sangat jahat. Sangat mudah untuk terbawa dengan ini dan secara keliru berpikir bahwa sulit = menyenangkan. Filosofi ini akan membuat banyak orang menyerah pada permainan Anda sebelum mereka memberikannya kesempatan yang tepat. Ini terutama benar jika player tidak memiliki banyak pengalaman sebelumnya dengan game.

Maka, sesuatu yang harus selalu Anda lakukan adalah memberikan permainan Anda kepada orang-orang untuk dicoba. Keuntungan mengembangkan untuk Android adalah Anda dapat membawa ponsel Anda ke pub dan menyebarkannya untuk melihat bagaimana teman Anda mengelolanya. Apa yang mungkin Anda temukan adalah bahwa hal-hal yang tampak jelas bagi Anda adalah tumpul atau tidak adil bagi player pemula. Anda akan melihat di mana player Anda terjebak dan pada titik mana mereka mempertimbangkan untuk menyerah. Jika Anda melakukan ini dengan benar, mereka seharusnya bisa melewati setidaknya beberapa level pertama tanpa menjadi terlalu frustrasi. Setidaknya butuh waktu lama bagi mereka untuk ketagihan. Tip dari Shigeru Miyamotu, pemikir jenius di balik Mario dan World 1-1, adalah mendesain level pertama Anda terakhir. Ini membuatnya lebih mudah untuk mengambil langkah mundur dan menghindari godaan untuk menjadi sadis dengan desain level Anda.

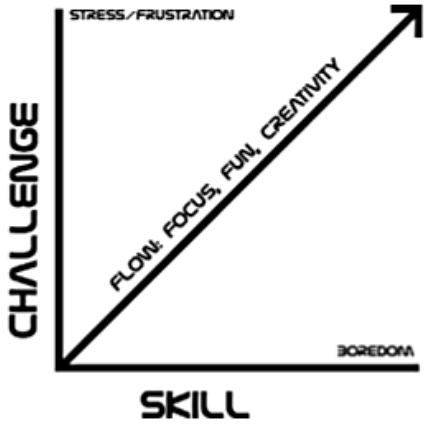
Kurva Kesulitan

Anda harus mulai memperkenalkan tantangan yang signifikan di beberapa titik dalam gim Anda, karena jika tidak, itu akan menjadi membosankan. Kesenangan terletak di sweet spot antara tidak mungkin dan terlalu mudah. Mengapa? Karena dari sisi neurologis, game memang menyenangkan selama kita belajar. Ya, saya akan masuk lebih dalam. Otak Anda berevolusi untuk membantu Anda bertahan hidup. Dan apa yang membuat manusia begitu mahir bertahan hidup? Kemampuan kita untuk beradaptasi dan belajar. Kami berkembang karena kami belajar bagaimana memanfaatkan lingkungan di sekitar kami dan mengatasi perubahan iklim dan keadaan. Reaksi kita menjadi lebih baik dengan latihan, dan pola gerakan yang ditetapkan menjadi tertanam dalam melalui pengulangan.

Otak ingin terus belajar, jadi otak menghargai pembelajaran itu dengan melepaskan neurotransmiter dan hormon tertentu. Saat Anda bekerja menuju suatu tujuan, otak melepaskan dopamin untuk membuat Anda tetap fokus. Ketika Anda mencapai tujuan itu, ia melepaskan endorfin—yang terasa luar biasa. Ini mendorong otak untuk memperbaiki dirinya sendiri, sehingga Anda memiliki peluang lebih baik untuk mencapai hal yang sama lagi. Game menggunakan sound effect untuk memberi sinyal hadiah, yang memperkuat respons itu. Jika Anda menyajikan otak dengan terlalu sedikit stimulasi atau tantangan, itu menjadi bosan. Kebosanan itu buruk bagi kita karena secara harfiah dapat menyebabkan otak berhenti berkembang. Saat bosan, kita akan segera mencari hal lain untuk dilakukan. Namun demikian,

ketika Anda memberi otak tantangan yang mustahil, otak dengan cepat menjadi kempis dan menyerah.

Tetapi jika Anda diberi tantangan yang cukup sulit untuk membutuhkan banyak pekerjaan tetapi tidak terlalu sulit untuk tidak mungkin, maka ini dapat merangsang dan melibatkan Anda saat otak belajar, beradaptasi, dan tumbuh. Jika permainan mendapatkan ini dengan benar, maka player akan memasuki apa yang disebut dalam ilmu saraf sebagai keadaan aliran—keadaan pikiran di mana kita sepenuhnya fokus pada tugas yang ada sampai pada titik di mana waktu di sekitar kita hampir tampak melebar dan pelan - pelan. Studi pencitraan otak menunjukkan beberapa perubahan menarik dalam cara otak bekerja pada saat ini; ia memasuki keadaan yang disebut hipofrontalitas di mana daerah frontal otak menjadi tertekan, dan kita mulai bertindak murni berdasarkan naluri.



Gambar 6.56 Agar player Anda terlibat dan bersenang-senang, kurva kesulitan Anda harus benar-benar sesuai dengan tingkat kemampuan mereka

Anda mungkin pernah mengalami ini di beberapa titik selama penembak neraka di mana Anda telah menari di sekitar ratusan peluru di layar dalam keadaan trance, atau selama pertempuran bos yang intens di mana Anda hanya memiliki satu kehidupan yang tersisa. Otak terlibat karena sedang belajar dan tumbuh, dan Anda dapat merasakan hasil kerja ini ketika Anda kembali ke tingkat sebelumnya dan menemukan bahwa memori otot baru Anda sekarang membuat tantangan yang sebelumnya tampak mustahil menjadi mudah. Sebagai seorang desainer game, tugas Anda adalah memastikan bahwa intensitas tantangan meningkat dengan sempurna seiring dengan peningkatan keterampilan dan pengalaman player, dengan tujuan akhir untuk mempertahankan mereka di tempat yang tepat. Lebih baik lagi, Anda harus memberikan kedalaman permainan Anda sehingga mereka dapat kembali ke level sebelumnya dan menggunakan keterampilan baru mereka untuk mendapatkan waktu yang lebih baik atau menemukan koleksi tersembunyi. (Konon, mondar-mandir juga penting, dan Anda perlu memberi player ruang sesekali untuk bernapas sehingga mereka dapat pulih.)

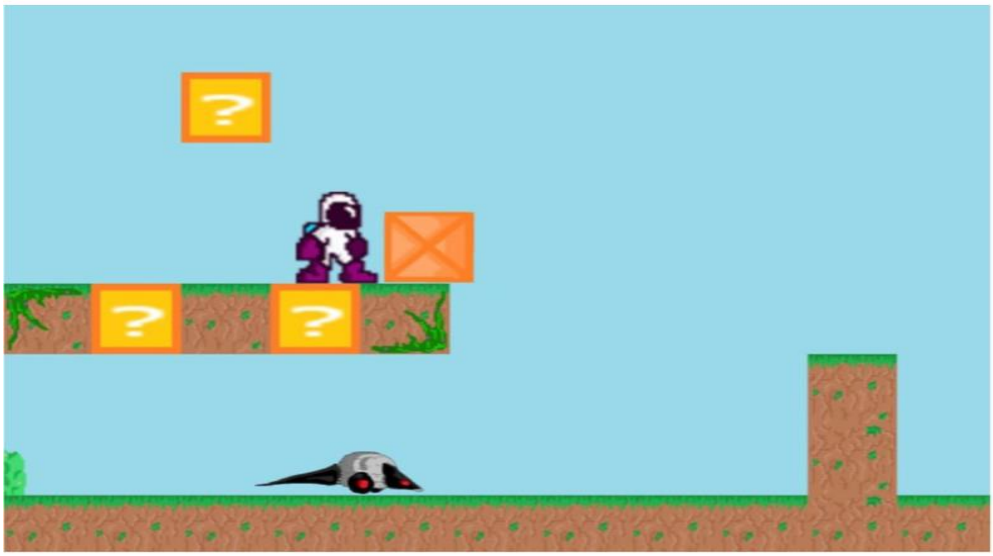
Cara Lain untuk Membuat Game Anda Menyenangkan

Jadi, sangat penting bagi Anda untuk terus mengajari player saat mereka maju melalui permainan dan menjaga agar tantangan tetap adil tetapi bermanfaat. Tapi itu bukan satu-satunya cara untuk membuat game Anda menyenangkan. Alat lain yang berguna adalah variasi. Salah satu cara untuk merangsang keadaan aliran di dunia nyata adalah dengan menempatkan seseorang di lingkungan baru, dan ini adalah sesuatu yang dapat kita gunakan untuk keuntungan kita. Otak bangun dan memperhatikan ketika lingkungan tidak dikenal karena ini sekali lagi merupakan kesempatan belajar. Itulah mengapa Anda harus terus memperkenalkan mekanik baru dan mengubah lingkungan Anda. Inilah sebabnya mengapa sangat umum untuk melihat "level salju" dan "level gunung berapi" dalam game. Anda bisa

lebih inventif dari itu, tentu saja, tetapi yang paling penting adalah Anda terus mengubah palet dan nadanya. Itu menciptakan rasa penemuan dan mendorong player Anda untuk ingin terus maju.

Pemecahan teka-teki adalah kiasan lain yang dinikmati player dalam permainan. Sekali lagi, ada penghargaan neurologis yang datang dari momen "eureka" itu dan dari membiarkan semuanya berjalan pada tempatnya. Jadi bagaimana Anda merancang teka-teki yang baik? Jawabannya adalah dengan memperkenalkan elemen ke permainan Anda dan kemudian meminta player untuk mencari cara baru untuk menggabungkan dan menggunakan elemen tersebut. Jadi, kotak yang Anda gunakan untuk memanjat tebing menjadi senjata yang bisa Anda jatuhkan pada musuh. Ini membutuhkan pemikiran lateral dan menantang otak untuk mengatasi kemantapan fungsional—godaan untuk melihat objek dan elemen hanya dalam konteks yang awalnya diperkenalkan.

Cara terbaik untuk meningkatkan tantangan dengan teka-teki Anda adalah secara bertahap meningkatkan jumlah langkah yang perlu diambil player untuk menyelesaikannya. Terakhir, cara yang bagus untuk memberi penghargaan kepada player Anda dan melibatkan mereka lebih jauh adalah dengan memberdayakan mereka untuk memberi dampak pada dunia di sekitar mereka. Hal ini sering kali berkaitan dengan kaitan utama gameplay Anda—mekanik yang membedakan game Anda dan memungkinkan karakter Anda untuk menavigasi dunia dengan cara yang unik. Jika mekanik ini juga membiarkan player melihat efek yang mereka alami di sekitar mereka, maka itu akan membantu membuat mereka merasa lebih kuat, yang dapat menghasilkan banyak kesenangan. Itu sebabnya game seperti Angry Birds atau bahkan Just Cause pada dasarnya berputar di sekitar menyebabkan kehancuran dalam jumlah besar. Itu membuat player merasa kuat. Game lain seperti Godus mengambil satu langkah lebih jauh dengan membiarkan player bermain sebagai dewa. Konon, melemahkan player untuk menciptakan rasa tegang, terisolasi, dan bahaya bisa menjadi cara yang baik untuk meningkatkan fokus dan perhatian mereka dan membuat kemenangan mereka lebih berharga. Ini diilustrasikan dengan sempurna dalam game seperti Limbo.



Gambar 6.57 Elemen kotak

Gameplay yang Muncul

Hal lain yang perlu dipertimbangkan adalah aspek permainan Anda yang tidak dapat Anda desain. Dunia Anda akan menjadi rangkaian permutasi yang selalu berubah, yang akan didasarkan pada peristiwa acak dan tindakan player Anda. Anda tidak dapat mengantisipasi setiap skenario, artinya beberapa kemungkinan gameplay akan keluar dari tangan Anda. Tapi

ini bukan hal yang buruk. Ini sebenarnya hal yang hebat. Inilah bagaimana gameplay yang muncul: ketika elemen yang Anda buat berinteraksi dengan cara yang tidak terduga, menciptakan tantangan baru dan situasi unik bagi para player. Misalnya, jika Roborat dapat memicu balok yang jatuh, dalam situasi yang tepat hal itu dapat mengakibatkan gambar player dan tikus yang melompat melintasi puing-puing yang jatuh. Gameplay yang muncul luar biasa karena memberi setiap player cerita unik mereka sendiri untuk diceritakan dan memastikan bahwa setiap sesi permainan berbeda. Anda tinggal membuat elemen, mengaduknya bersama-sama dalam panci besar, dan kemudian menunggu keajaiban terjadi.

Interaksi Antara Hardware, Game engine, Format, dan Gameplay

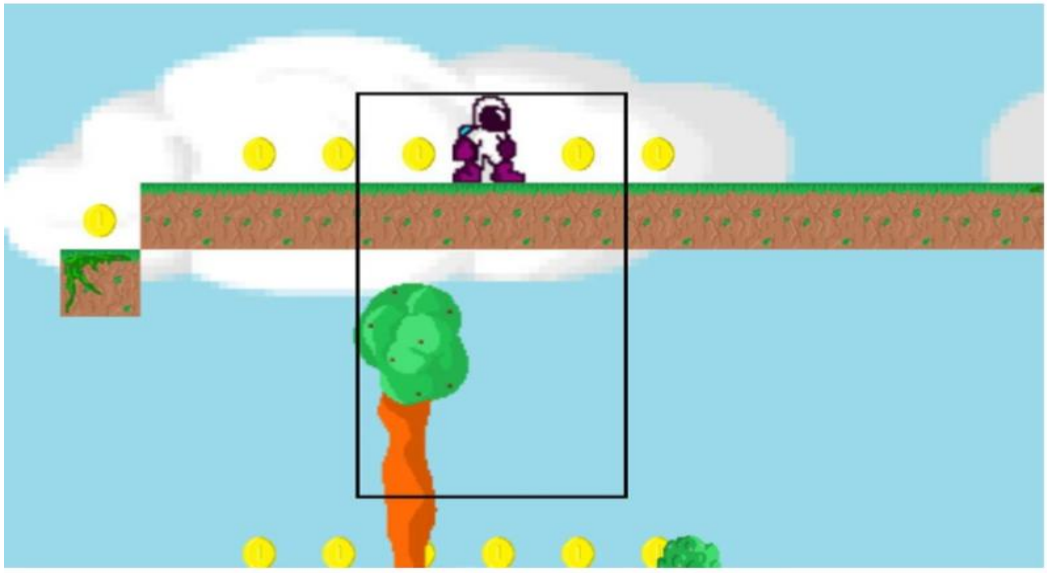
Dalam bab 5, saya menyebutkan bahwa fisika game dan elemen yang Anda buat tidak akan terpisahkan dari gameplay dan tantangannya. Yang saya maksud adalah bahwa ada interaksi dua arah di sini yang harus dipertimbangkan selama desain Anda. Keputusan yang Anda buat mengenai berfungsinya dunia game akan memiliki konsecookiesnsi langsung terkait cara permainan Anda dimainkan dan tantangan yang mungkin terjadi. Misalnya, jumlah gesekan yang Anda tambahkan ke permukaan akan mengubah kesulitan yang disajikan oleh serangkaian platform bergerak, seperti yang kita lihat sebelumnya. Begitu juga dengan ukuran tombol arah di layar dan jari-jari player itu sendiri. Semua ini perlu diperhitungkan saat membuat urutan yang menantang dan saat mendesain fisika game di tempat pertama. Seperti yang akan kita lihat, hubungan dua arah ini jauh lebih dalam....

Membuat Kamera Hebat

Salah satu contoh terbaik tentang bagaimana mekanisme permainan dan pemrograman game engine Anda berpotongan dapat dilihat dengan kamera. Saat ini, Kamera Anda sangat sederhana: ini adalah anak dari Player dan bergerak dengan kecepatan yang sama persis dengan yang dilakukan player. Anda mungkin tidak terlalu memikirkan hal ini, tetapi jika Anda sekarang kembali ke game platform favorit Anda, Anda mungkin memperhatikan bahwa ini bukanlah cara sebagian besar game berperilaku. Misalnya, kita melihat dalam contoh awal Mario bahwa kamera mulai jauh di sebelah kanan player, menunjukkan arah yang harus mereka gerakkan. Ini juga merupakan posisi di mana Anda akan melihat kamera dalam "permainan pelari tak berujung", dan dalam hal ini kamera mengambil posisi itu untuk memastikan player memiliki banyak kesempatan untuk melihat rintangan yang akan datang dan oleh karena itu lebih banyak waktu untuk bereaksi . Player tidak dapat berlari mundur dalam jenis permainan ini, jadi apa gunanya memiliki banyak ruang tidak berguna di sebelah kirinya? Semakin cepat permainannya, semakin menarik kamera ke belakang dengan FOV (bidang pandang) yang lebih luas untuk menunjukkan lebih banyak apa yang akan terjadi.

Dalam game dengan banyak platforming, penting untuk mencegah mual. Dalam hal ini, kamera terkadang memiliki zona netral di tengah tempat player dapat bergerak dan kemudian menggulir hanya setelah mereka keluar dari pusat ini. Platformer lain memecahkan masalah ini dengan membuat kamera "jepret" ke platform mana pun yang disentuh karakter pada waktu tertentu. Pada Gambar 10-6, kotak hitam menunjukkan zona netral kita. Ada banyak ruang atas dan bawah, tetapi tidak begitu banyak ruang kiri dan kanan. Jadi jika player bergerak ke kiri atau ke kanan, kamera akan segera melacak dengan hanya sedikit penundaan (artinya mereka akan merasa lebih cepat saat menghindari rintangan). Namun, akan ada lebih banyak ruang bagi player untuk melompat-lompat tanpa kamera terombang-ambing. Ini akan cocok untuk desain tingkat yang kurang vertikal dengan banyak lompatan melintasi celah. Lihat bagaimana perilaku kamera mencerminkan tata letak level dan sebaliknya? Ini adalah jenis perilaku kamera yang sama yang terlihat di game 2D Sonic the Hedgehog, yang sebenarnya sangat penting karena banyaknya bukit dan gradien dalam game tersebut. Jika

kamera hanya mengikuti Sonic, itu akan terus bergerak naik dan turun ke tingkat yang memuakkan — terutama pada kecepatan itu.



Gambar 6.58 Pendekatan berbeda untuk kamera kami

Dalam kasus lain, kamera dapat digunakan untuk efek dramatis—mengisyaratkan bahaya yang ada di depan atau melambat saat player mendekati tantangan besar. Jika kamera berhenti bergerak maju, player akan langsung bertanya-tanya apakah mereka harus melanjutkan dan mulai bertanya-tanya apa yang ada di luar bidang penglihatan mereka (FOV). Jadi, jika desain game Anda tidak berfungsi sebagaimana mestinya, pertimbangkan apakah ada jumlah sinergi yang tepat antara dunia yang Anda impikan dan pergerakan kamera. Bisakah game dibuat lebih menyenangkan dengan mengkodekan beberapa perilaku yang lebih maju ke dalam kamera Anda atau bahkan hanya dengan memindahkannya sedikit ke belakang?

6.11 HARDWARE DAN MODEL BISNIS

Bukan hanya fisika dan kode yang akan menentukan apa yang mungkin dan apa yang menyenangkan dalam desain game Anda. Ini juga hardware yang Anda targetkan dan model bisnis yang ingin Anda gunakan. Untuk contoh bagaimana hardware dan monetisasi dapat berdampak langsung pada cara permainan dimainkan, lihat saja mesin arkade video lokal Anda. Permainan arcade secara konvensional sangat sulit dan memiliki sistem kehidupan karena mereka ingin player memasukkan lebih banyak koin. Demikian juga, mereka harus mudah dipelajari dan menantang untuk dikuasai sehingga orang-orang akan terus kembali ke puncak tangga lagu skor tinggi. Ketika game bermigrasi ke PC, mereka mulai menjadi jauh lebih kompleks dan rumit. Mereka menjadi lebih rumit dengan diperkenalkannya file simpan dan hardware yang lebih kuat.

Menariknya, game seluler telah mengambil kembali sedikit. Game seluler di layar kecil cocok untuk alur game yang lebih "seukuran gigitan" (lihat Gambar 6.59), sedangkan pengenalan opsi monetisasi alternatif seperti "gratis untuk dimainkan" berarti bahwa game sekali lagi perlu memberi kita insentif untuk tetap pengeluaran.

Gambar 6.59 Breath of the Wild berfungsi dengan baik sebagai game portabel karena kemampuannya untuk masuk dan keluar dengan mudah

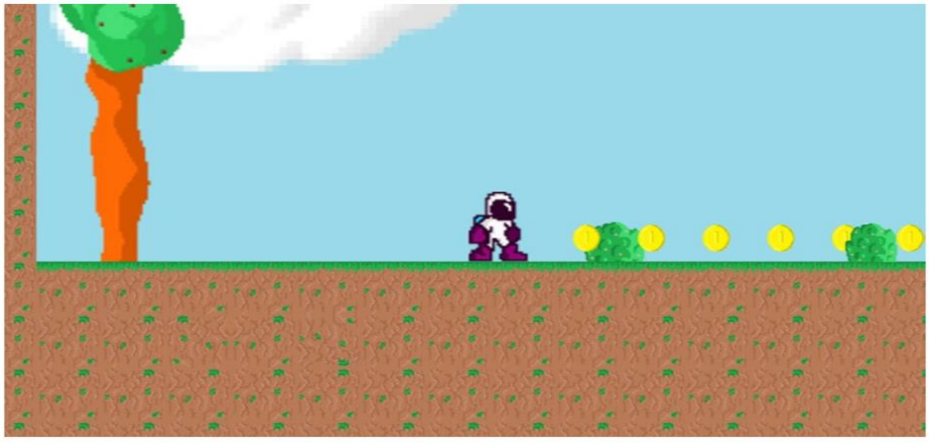
Fungsionalitas online sementara itu berarti bahwa daftar "skor tinggi" sekali lagi menjadi lebih penting. Intinya di sini adalah tidak ada yang harus disertakan dalam game Anda karena "itulah yang dilakukan game." Segala sesuatu harus memiliki tujuan, dan tujuan itu akan ditentukan oleh berbagai faktor yang berbeda. Apakah Anda ingin membuat game "duduk" dengan pembayaran satu kali yang ditujukan untuk perangkat tablet atau endless runner gratis untuk dimainkan untuk gamer kasual akan sepenuhnya mengubah cara Anda membuat desain level. Ini berarti Anda memerlukan beberapa konsep dari seluruh permainan akhir Anda sebelum Anda mulai mendesain level pertama itu. Dan untuk berpikir Anda baru saja akan mulai menjatuhkan barang-barang di beberapa tempat.

6.12 MEMBUAT GAME ANDA TERLIHAT LUAR BIASA

Meskipun gameplay bisa dibilang lebih penting daripada penampilan, tetap sangat penting bahwa game Anda memiliki keduanya. Kami telah melihat bahwa grafik dalam game Anda akan berdampak pada cara kerja game Anda; grafis dapat menyampaikan rasa tempat dan dapat memberikan isyarat untuk interaksi. Namun, pada saat yang sama, itu akan menjadi screenshoot dan cuplikan gameplay lebih dari apa pun yang membantu Anda menjual game Anda. Dengan kata lain, inilah saatnya kita melihat untuk menambahkan sedikit cat ke game yang telah kita buat. Apa saja cara kami dapat meningkatkan tampilannya di foto?

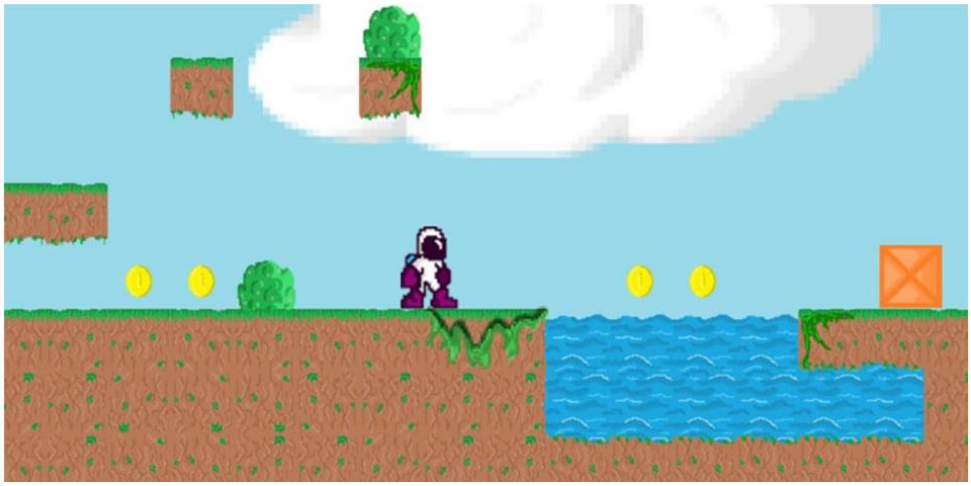
Cara Mudah Membuat Game Anda Lebih Menarik

Jika Anda melihat apa yang telah kami buat, akan adil untuk mengatakan bahwa saat ini tidak begitu menarik. Ini belum terlihat seperti game profesional, dan itulah yang ingin kami perbaiki di sini.



Gambar 6.60 Tampilan saat ini

Tapi apa sebenarnya yang salah? Apa yang hilang? Masalah pertama adalah bahwa semuanya sangat anorganik. Platform terbuat dari garis lurus dan semuanya seragam. Cara cepat untuk mengubahnya adalah dengan memutar beberapa di antaranya 90 atau 180 derajat. Itu cara yang efisien untuk menggunakan kembali aset yang sama dan menjaga ukuran file tetap rendah, tetapi masih menambahkan beberapa variasi pada tampilannya. Demikian juga, kita harus mempertimbangkan untuk menggunakan beberapa sprite yang lebih detail untuk tepi platform kita. Itu akan memberikan efek pembusukan alami dan dengan cepat membuat segalanya terlihat lebih nyata. Kita dapat menambahkan lebih banyak detail, seperti tanaman merambat yang kita gunakan sebelumnya, untuk membuat setiap bagian tanah terlihat sedikit berbeda. Pada dasarnya, kami ingin semuanya terlihat acak mungkin, dan kami dapat melakukannya dengan sedikit kode.



Gambar 6.61 Tidak banyak yang berubah, tetapi terlihat sedikit lebih organik

Karena itu, masalah lain dengan dunia game kami adalah statis. Lihatlah ke luar jendela Anda, dan Anda akan melihat bahwa ada sesuatu yang selalu bergerak, apakah itu cabang pohon yang tertiup angin atau air hujan yang menetes dari pipa. Hal yang sama berlaku untuk gim terbaik, dan itulah sebabnya hampir semuanya dianimasikan, mulai dari bunga hingga bintang di latar belakang. Itu tidak hanya membuat dunia Anda terasa hidup, tetapi juga membawa lebih banyak karakter dan kepersonalan ke dalam gim Anda dan membuatnya lebih menarik untuk dilihat. Ada titik potong, meskipun. Kami tidak ingin mengalihkan perhatian para player kami dari elemen-elemen penting. Tentu saja, kami kehilangan animasi di sepanjang game saat ini. Anda akan ingin memberikan animasi orang jahat dan animasi player Anda untuk melakukan hal-hal seperti melompat atau menembak. Ini adalah cara lain untuk membantu player merasa bahwa mereka sedang berinteraksi dengan dunia. Pegas harus bergoyang saat terpental. Masalah terakhir adalah bahwa permainan tidak memiliki kedalaman. Latar belakang yang kami rancang cukup datar dan kurang menarik hingga kami mencapai awan atau matahari. Ini memperburuk perasaan bahwa semuanya telah dipotong dari potongan karton, jadi Anda harus memperbaikinya dengan menambahkan beberapa layer lagi.

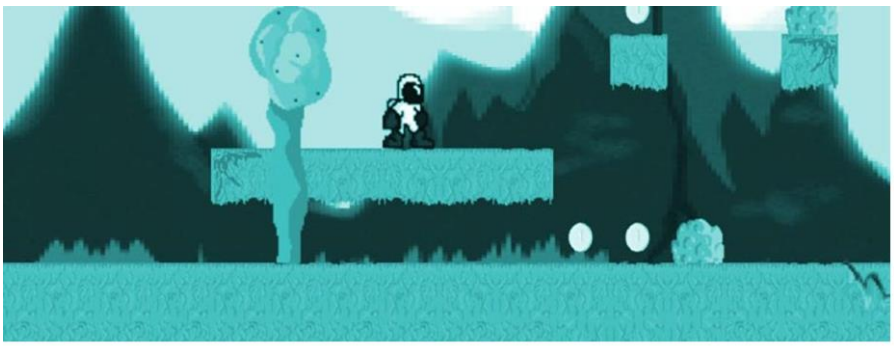
Saya juga menambahkan beberapa lagi di latar belakang dan memperkenalkan layer pegunungan. Gunung-gunung itu bergerak dengan kecepatan lain dan membantu memastikan bahwa dunia game kita tidak pernah terlihat benar-benar kosong di latar belakang.

Gambar 6.62 Dengan beberapa perubahan ini, permainan kami mulai terlihat lebih menarik

Cara Membuat Sprite Berpenampilan Hebat dan Memilih Bahasa Desain untuk Game Anda

Meskipun menambahkan elemen-elemen ini dapat melakukan banyak hal untuk meningkatkan tampilan dan nuansa permainan Anda, mereka semua mengharuskan Anda untuk memiliki beberapa keterampilan dasar dalam hal membuat sprite Anda sendiri. Apa yang Anda lakukan jika Anda tidak memiliki tulang artistik di tubuh Anda? Salah satu opsi adalah mengalihdayakan karya seni Anda. Situs seperti Fiverr, Freelancer, dan UpWork memungkinkan Anda terhubung dengan pekerja lepas yang menawarkan berbagai layanan termasuk seni dan desain. Ini juga merupakan tempat yang bagus untuk mendapatkan musik latar dan sound effect.

Opsi kedua adalah membuat game bergaya yang menggunakan gaya seni unik yang sangat mengurangi jumlah pekerjaan yang perlu Anda lakukan. Banyak permainan hari ini menggunakan gaya seni hitam dan putih, siluet (seperti Limbo yang disebutkan di atas), atau berbagai tampilan retro (seperti yang terlihat di VVVVVV, yang sepertinya dirancang pada ZX Spectrum). Gambar 10-11 menunjukkan seperti apa tampilan game kami jika didesain untuk Game Boy.

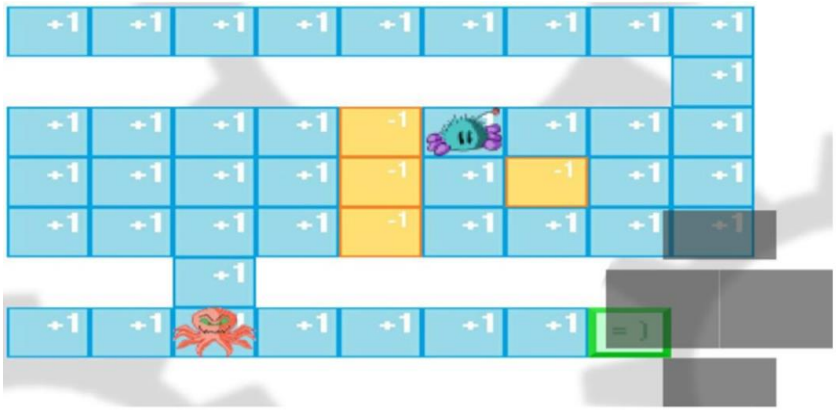


Gambar 6.63 Gaya Retro

Menggunakan gaya seni tertentu seperti ini dapat membuat game Anda menonjol dan menarik perhatian di Play Store, sekaligus memberikan identitas yang kuat. Jika Anda memilih sesuatu yang agak minimal, Anda juga akan menghemat banyak waktu dan menghilangkan kebutuhan untuk menjadi hebat dalam desain. Untuk game kami, kami menggunakan gaya seni piksel. Ini adalah tampilan lain yang terinspirasi retro yang memberi game kami rasa nostalgia dan membebaskan kami dari kebutuhan untuk membuat sprite fotorealistik. Jadi bagaimana Anda mencapai gaya ini? Jawabannya sederhana: cukup gunakan software pengedit gambar seperti GIMP atau bahkan MSPaint, lalu perbesar sedekat mungkin. Jika memungkinkan, pilih Show Grid Lines di pengaturan. Sekarang, dengan menggunakan alat

pensil yang memiliki opacity 100%, Anda dapat menggambar garis luar untuk sprite Anda. Anda harus dapat melihat piksel individu saat Anda menggambar.

Luangkan waktu dan hati-hati saat Anda menggambar sprite Anda dan pastikan untuk mengawasi setiap pola yang terbentuk. Misalnya, jika Anda menggambar gradien, Anda mungkin memperhatikan bahwa piksel bergerak satu ke atas, tiga kali melintasi. Ini akan menghasilkan sesuatu yang terlihat lebih konsisten dan terkontrol. Untungnya, Anda bisa menekan Ctrl+Z jika ada kesalahan. Tip lainnya adalah mempertimbangkan untuk menggunakan layer jika tersedia (GIMP dan Photoshop menawarkan fitur ini, tetapi MSPaint tidak) dan dengan cara itu menelusuri gambar yang ingin Anda ubah menjadi seni piksel.



Gambar 6.64 Aplikasi awal yang saya buat menggunakan palet warna pastel dan tampilan yang terinspirasi dari Sudoku: Debugger: Brain Untraining

Anda dapat menguraikan sprite Anda atau Anda dapat menggunakan warna blok. Saya juga merekomendasikan menambahkan bayangan. Itu biasanya berarti Anda akan menggunakan tiga warna: satu untuk isian utama, satu untuk bayangan, dan satu untuk sorotan. Pastikan bayangan secara konsisten berada di sisi yang sama di semua sprite Anda yang akan berada di scene yang sama—jika tidak, akan terlihat membingungkan karena tidak akan jelas di mana sumber cahayanya. Terakhir, ekspor gambar Anda. Sekarang, Anda mungkin menemukan bahwa itu terlihat kecil ketika Anda melakukan ini, tetapi itu dapat diperbaiki ketika Anda mengimpornya ke Unity dan mengatur Pixels Per Unit dan Scale. Buat gambar kecil lebih besar dan piksel akan benar-benar muncul.

Optimasi

Bab ini adalah tentang mengambil game fungsional Anda dan mengubahnya menjadi game yang luar biasa. Untuk tujuan ini, kami memiliki satu item tersisa di agenda kami: pengoptimalan. Kami telah melihat detail permukaannya—sekarang kami perlu melihat lagi apa yang ada di dalamnya. Pertama, apa sih yang saya maksud dengan optimasi? Pada dasarnya, saya sedang berbicara tentang membuat game Anda berjalan dengan lancar dan mudah untuk diedit, ditingkatkan, dan diperbarui di masa mendatang. Kode yang baik harus menggunakan baris sesedikit mungkin, dengan semuanya diatur dengan rapi sehingga mudah bagi Anda untuk menemukan elemen mana pun yang Anda butuhkan.

Tips Untuk Kode Lebih Baik

Setiap kali Anda menulis kode, Anda harus memperhatikan masa depan. Suatu hari, Anda akan ingin memperbarui game Anda untuk memperbaiki bug atau menambahkan fitur baru (sekali lagi, ini sangat umum di seluler) dan akan kembali lagi setelah pergi untuk sementara waktu. Di dunia yang ideal, ini seharusnya menjadi pengalaman yang tidak menyakitkan. Semuanya mudah dipahami dan Anda tidak perlu menghabiskan waktu lama untuk menyipitkan mata di layar. Anda harus tahu di mana semuanya dan apa yang perlu Anda ubah

untuk mencapai hasil yang Anda inginkan. Ini menjadi lebih penting jika Anda bekerja dalam tim. Seperti disebutkan, kode yang lebih baik juga berarti lebih sedikit kode. Semakin banyak kode yang ada di halaman, semakin sulit untuk menemukan apa pun yang Anda cari, dan semakin banyak langkah yang mungkin diambil setiap proses. Lebih banyak langkah = eksekusi lebih lambat. Jadi bagaimana Anda bisa mulai membuat program yang lebih elegan? Berikut adalah beberapa tip untuk Anda mulai:

1. Tempatkan beberapa variabel pada satu baris:

   ```
   public float startX;
   float startY;
   becomes
   public float startX, startY;
   ```

2. Selalu pastikan untuk menggunakan variabel dengan nama yang masuk akal yang menggambarkan fungsinya. Ini mungkin terdengar jelas, tetapi Anda akan terkejut betapa seringnya programmer menggunakan label yang sepenuhnya acak. Jika variabel Anda memberi tahu karakter seberapa tinggi mereka harus melompat, itu harus disebut sesuatu seperti jumpHeight. Ini juga berarti menghindari singkatan (jh), yang akan cepat menjadi tumpul dan membingungkan. Faktanya, situasi yang ideal adalah variabel Anda memungkinkan kode Anda dibaca seperti bahasa Inggris. Saat menggunakan Boolean khususnya, ini bisa benar atau salah, yang berarti Anda dapat membuat baris seperti ini:

   ```
   if (playerIsGrounded) {
   ```

   Ini memberi tahu kita semua yang perlu kita ketahui, bahkan jika kita tidak tahu satu baris pemrograman.

3. Gunakan kotak unta. Ini berarti bahwa setiap kata baru dalam variabel dimulai dengan huruf kapital untuk membantu pembaca memecahnya (terkadang ini tidak termasuk kata pertama). Misalnya: jumpheight harus ditulis sebagai jumpHeight atau JumpHeight. Ini tidak hanya membantu keterbacaan lebih lanjut, tetapi Anda juga akan melihat bahwa Unity memecah variabel-variabel ini menjadi kata-kata individual saat Anda melihat variabel di Inspector.

4. Hindari penggunaan “angka ajaib”. Dengan kata lain, jangan menetapkan signifikansi acak ke angka sebagai cara untuk mengatasi tantangan pengkodean. Saya melakukan ini di Bab 9 ketika saya menggunakan pengatur waktu yang melampaui nol untuk balok-balok yang jatuh. Timer berhenti ketika mencapai –70. Mengapa dikurangi tujuh puluh? Salah satu cara untuk menghindari ini adalah dengan menggunakan konstanta. Konstanta adalah jenis variabel yang memiliki nilai tetap dan tidak dapat diubah setelah didefinisikan. Ini tidak memiliki overhead memori, dan tujuan utamanya biasanya untuk keterbacaan. Kita bisa membuat bilangan bulat konstan misalnya dengan nilai –70 dan menyebutnya endOfFallAnimation. Sekarang blok jatuh kita akan berhenti di endOfFallAnimation daripada di –70. Jauh lebih masuk akal! Ingat skrip Player kami dan penggunaan 1 untuk mewakili "kanan" dan 0 untuk mewakili "kiri"? Jika Anda mengambil jeda dari kode Anda dan kemudian kembali ke sana, ini juga bisa sangat membingungkan. Jadi alih-alih mengapa tidak menggunakan ini:

   ```
   const int left = 0, right = 1;
   ```

   Sekarang kita bisa mengatakan

   ```
   if (facing == right)
   ```

   yang jauh lebih mudah bagi kita untuk membaca kembali. (Namun, ini masih ditampilkan sebagai 0 dan 1 di Inspector.)

Keuntungan lain menggunakan konstanta adalah jauh lebih mudah untuk mencari dan mengganti nilai nanti jika Anda ingin membuat perubahan.

5. Jelaskan mengapa tidak apa. Saat menulis komentar, menjelaskan tujuan suatu metode jauh lebih penting daripada menjelaskan apa yang dilakukannya. Apa relevansi dari fungsi ini? Bagaimana hubungannya dengan sisa skrip?
6. Hindari menulis kode yang sama dua kali jika memungkinkan. Semakin banyak kode yang dapat Anda tempatkan dalam metode yang berbeda, semakin mudah untuk menemukan apa yang Anda cari dengan cepat dan semakin sedikit Anda harus mengetik secara total. Menggunakan metode juga memungkinkan Anda untuk menyalin dan menempelkan seluruh potongan kode dari satu skrip ke skrip berikutnya.
7. Gunakan loop! Loop adalah bagian dari kode yang berulang sampai kondisi tertentu terpenuhi atau rusak. Misalnya, loop sementara terlihat seperti ini:

```
int count = 1;
while (count <= 4)
{
count = count + 1;
}
```

Ini hanya menghitung sampai empat dan kemudian berhenti, tetapi kita dapat menggunakan struktur ini untuk menjalankan perintah yang sama sebanyak empat kali. Sebenarnya, untuk loop yang menggunakan variabel incrementing, sering kali masuk akal untuk menggunakan "for". Ini adalah contoh menyelesaikan hal yang sama dengan lebih sedikit baris kode. Untuk loop terlihat seperti ini:

```
for ( init;
condition;
increment )
{
statement(s);
}
```

Apa pun jenis loop yang Anda gunakan, mereka memiliki tujuan yang mirip dengan metode yang membantu Anda mengelompokkan kode dan mencegah Anda menulis banyak fungsi berulang kali.

8. Gunakan tag dan layer pintar. Sama seperti Anda harus peka dengan konvensi penamaan variabel, hal yang sama berlaku untuk nama yang Anda tetapkan saat berada di dalam Unity IDE. Sekarang, Anda tahu untuk menggunakan pola asuh yang benar dan membuat Prefab daripada berurusan dengan contoh juga.

Performa Dan Kompatibilitas

Kiat sebelumnya akan membantu membuat kode Anda logis dan mudah dibaca dan dalam beberapa kasus juga sedikit lebih cepat. Namun, sungguh, hambatan utama dalam hal kecepatan akan berada di luar skrip Anda.

Gambar yang lebih kecil

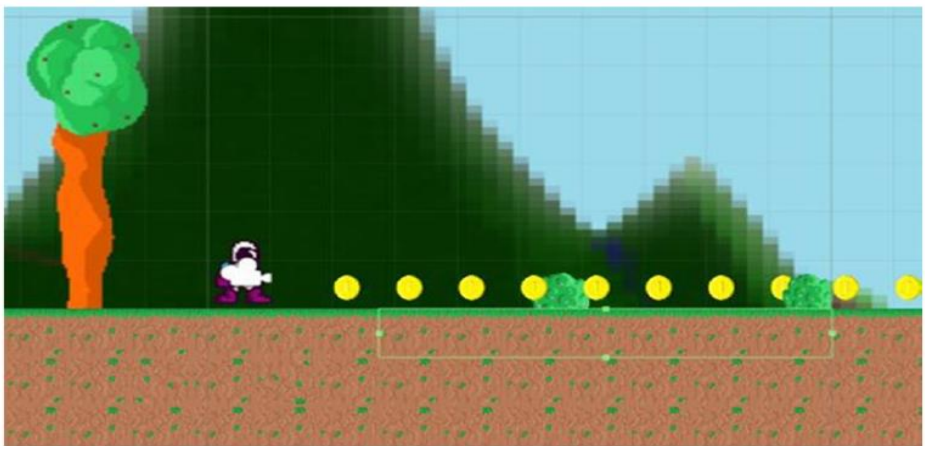
Misalnya, Anda perlu memastikan bahwa Anda menggunakan gambar yang tidak terlalu besar. Semakin besar gambar, semakin besar ukuran file aplikasi Anda dan semakin lama waktu yang dibutuhkan untuk memuat. Saya ingin memberi tahu Anda bahwa ukuran aplikasi Anda tidak terlalu penting, tetapi itu bohong: Saya personal memiliki beberapa komentar negatif dari pengguna saya sendiri ketika ukuran APK menjadi terlalu besar, jadi ini adalah sesuatu orang benar-benar peduli. Perhatikan bahwa Anda dapat menampilkan layar pemuatan jika perlu dengan memuat scene dari rutinitas bersama (seperti yang kami gunakan untuk metode Player.Death kami) dan kemudian menampilkan UI pemuatan di atas scene

lama. Namun, kami tetap ingin waktu pemuatan sesingkat mungkin, jadi Anda harus menghindari menempelkan gambar besar dalam scene jika tidak perlu. Ini adalah alasan lain mengapa memilih gaya seni piksel sangat masuk akal: ini memungkinkan Anda menyimpan ukuran file yang lebih kecil dan kemudian meningkatkannya tanpa harus khawatir tentang pikselasi. Memilih jenis kompresi gambar yang tepat (format gambar JPG daripada PNG jika kualitas sedikit berkurang) juga akan membantu dalam hal ini. Begitu juga dapat menggunakan kembali aset, itulah sebabnya memutar ubin adalah langkah yang baik sebelumnya. Unity akan menambahkan kompresi tambahan untuk gambar Anda saat Anda membangun APK, dan Anda dapat mengatur jenis kompresi tekstur yang ingin Anda gunakan dalam pengaturan build. Kompresi tambahan ini akan memengaruhi kecepatan dan ukuran aplikasi Anda, tetapi juga kompatibilitasnya dan apakah itu mendukung alfa (transparansi) atau tidak. Dari dokumentasi Unity sendiri:

Collision

Performa seharusnya tidak terlalu menjadi perhatian jika Anda membuat game 2D di Unity. Kecuali Anda memiliki elemen yang tak terhitung jumlahnya di layar, semua menjalankan animasi dan skrip yang rumit, sebagian besar ponsel Android akan dapat menangani sebagian besar hal yang Anda lakukan. Tapi itu tidak berarti tidak ada manfaat untuk menjaga tuntutan aplikasi Anda serendah mungkin (pertimbangkan menguras baterai dan menyimpan aplikasi lain di memori, misalnya), dan Anda tentu ingin menghindari kemungkinan aplikasi Anda menjadi tidak responsif .

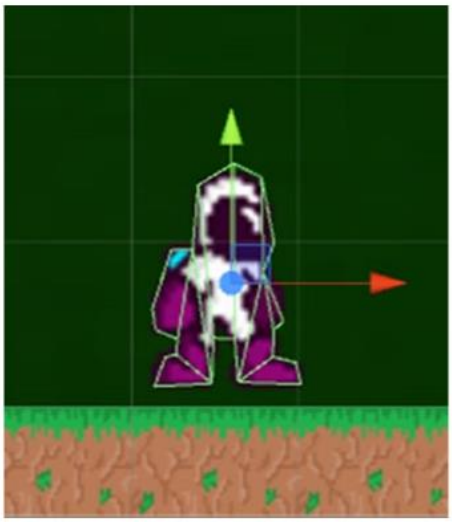
Saat runtime, salah satu hal terbesar yang perlu dipertimbangkan adalah berapa banyak clasher yang Anda miliki. Ukuran alat penumbuk tidak menjadi masalah, tetapi yang menjadi masalah adalah jumlah alat penumbuk dan kerumitan alat penumbuk tersebut. Ubin kami menggunakan penumbuk individu, misalnya, yang membuat developeran lebih mudah dan memungkinkan kami menggunakan Prefab. Ini adalah praktik terbaik untuk tujuan kami sebagian besar karena biaya dalam kinerja jauh lebih besar daripada fleksibilitas dan kemudahan yang dapat kami tambahkan pembaruan di masa mendatang



Gambar 6.65 Saya telah menggambar satu kotak penumbuk di sekitar sekelompok ubin di sini

Anda dapat membuat balok menjadi anak-anak dari yang memiliki clasher dan menyimpannya sebagai Prefab untuk menerapkannya dengan cepat ke dalam permainan Anda. Atau, Anda cukup menggambar kotak platform yang lebih besar dengan Collider yang lebih besar. Perlu diingat juga bahwa ubin di bawah permukaan sebenarnya tidak membutuhkan penumbuk. Menghapus Collider dari sini mungkin adalah salah satu cara tercepat dan termudah untuk membuat aplikasi kita berperforma lebih baik. Bahkan lebih buruk daripada memiliki banyak penumbuk kecil adalah menggunakan penumbuk poligon

kompleks dengan banyak titik dan sudut yang berbeda (lihat Gambar 6.66). Ini menciptakan lebih banyak matematika untuk Unity, karena perlu mengetahui bagaimana setiap titik akan berinteraksi dengan permukaan yang ditabraknya. Inilah mengapa masuk akal untuk menggunakan penumbuk kotak untuk karakter Anda (atau penumbuk poligon yang merupakan kotak yang sedikit berubah bentuk) daripada menggunakan penumbuk poligon yang sempurna mengikuti kontur karakter.



Gambar 6.66 Collision yang terlalu rumit

Bahkan penumbuk yang terlalu rumit pada Gambar 10-14 kemungkinan tidak akan menyebabkan perlambatan yang nyata, tetapi jika Anda memiliki banyak objek dengan penumbuk seperti ini, hal-hal dapat mulai menjadi sedikit berombak. Pada akhirnya, ini sia-sia karena tidak akan berdampak berarti pada cara player benar-benar memainkan permainan.

6.13 MEMBUAT JENIS GAME LAINNYA

Selama bab ini, kita telah membahas banyak hal tentang interaksi antara mekanika gameplay, desain, dan hardware. Namun sejauh ini, kami belum benar-benar mempertimbangkan sifat platform yang kami kembangkan. Bagaimanapun, platformer berasal dari NES dan komputer awal lainnya dan tidak secara alami cocok untuk input layar sentuh perangkat seluler. Pastinya masih ada pasar untuk game platform di Android, dan ini adalah pilihan yang sangat baik untuk tutorial karena memungkinkan kami bereksperimen dengan banyak konsep berbeda. Tetapi jika Anda ingin mencoba genre yang lebih ideal untuk seluler, Anda dapat memilih untuk mengembangkan pelari tak terbatas. Ini terlihat dan bertindak seperti platformer kecuali satu perbedaan utama: player terus berlari ke depan. Contoh yang bagus termasuk Canabalt, Sonic Runners, Super Mario Run, Temple Run, dan Jetpack Joyride. Di sini, player hanya membutuhkan satu input—lompatan—yang membuat layar tidak berantakan (tidak ada lagi tombol panah yang menutupi ruang bermain) dan menyediakan gameplay yang sempurna untuk masuk dan keluar dengan cepat.

Untuk membuat apa yang telah Anda buat menjadi pelari tak terbatas, Anda cukup memodifikasi skrip Player untuk berjalan maju secara otomatis. Anda kemudian dapat merancang level dengan pemikiran ini atau jika Anda ingin benar-benar "tak terbatas", minta level Anda menghasilkan sendiri dengan cepat (disebut generasi prosedural). Ini berarti Anda harus memperkenalkan algoritme yang membuat platform baru secara acak (dan mungkin menghancurkan yang lama) sambil memastikan selalu ada rute untuk player. Menggunakan ubin platform yang lebih besar sering kali merupakan ide bagus di sini, dan tentu saja Anda

membutuhkan kesulitan dan kecepatan untuk meningkatkan secara bertahap. Anda juga dapat menghilangkan gravitasi dari mesin fisika untuk mengubahnya menjadi semacam penembak luar angkasa, atau bahkan permainan top-down.

Game Puzzle dan Lainnya

Masalah potensial dengan platformer, first-person shooters, dan game balap di Android adalah bahwa mereka pada dasarnya melibatkan perkuatan genre game lama ke hardware baru. Sebaliknya, bisa dibilang game Android yang paling inventif dan menarik adalah game yang menemukan cara baru untuk memanfaatkan hardware. Angry Birds adalah contoh yang baik karena menggunakan layar sentuh dengan cara yang sangat alami untuk membuka kemungkinan gameplay baru. Room and Monument Valley melangkah lebih jauh dengan membiarkan player berinteraksi langsung dengan dunia game dengan menjangkau dan menyentuh, memutar, dan menyeret berbagai elemen dan bahkan menyertakan kontrol kemiringan pada waktu tertentu. Ingat bagaimana kami mengatakan bahwa player suka merasa bahwa mereka memengaruhi dunia game? Anda bisa membuat game Anda menggunakan akselerometer ponsel sesederhana ini:

```
rb.velocity = new Vector2(Input.acceleration.x, rb.velocity.y);
```

Bagaimana jika membalik ponsel dapat menyebabkan musuh dan barang koleksi meluncur di layar? Demikian juga, Anda juga dapat menggunakan multitouch dengan sangat mudah, membuka berbagai kemungkinan lain:

```
void Update ()
{
Touch myTouch = Input.GetTouch(0);
Touch[] myTouches = Input.touches;
for(int i = 0; i < Input.touchCount;  i++)
{
//Do something with the touches
}
```

Jangan dibatasi oleh gagasan lama tentang apa itu "permainan". Anda dapat mengatur kondisi apa pun untuk mengakhiri level, apakah itu membuat bola bergulir ke target yang bertindak seperti pemicu atau menghitung ketika player telah mengumpulkan setiap koin di layar. Bahkan tidak perlu ada objek "player" sama sekali — lihat saja Tetris, hit seluler asli. Oh, dan berbicara tentang berbagai jenis game seluler, saya punya sesuatu yang menarik untuk Anda di bab berikutnya. Pertama, kita akan membahas bagaimana Anda akan membuat game 3D untuk Android dengan grafik realistis (ya, Anda bisa melakukannya). Kemudian kita akan membahas bagaimana Anda benar-benar dapat memasuki dunia itu menggunakan Samsung Galaxy Gear atau headset Daydream View Google.

Ini adalah batas baru yang menarik bagi developer seluler, dan kami akan memastikan Anda berada di puncak gelombang itu.

Gambar 6.67 Sekarang itu mulai terlihat seperti permainan yang ingin saya mainkan!

BAB 7
MENGHUBUNGKAN GAME ANDA KE INTERNET

Dalam bab ini Anda akan mempelajari cara mengirim dan menerima data melalui jaringan. Proyek-proyek yang dibangun di bab-bab sebelumnya mewakili berbagai genre permainan, tetapi semuanya telah diisolasi ke mesin player. Seperti yang Anda ketahui, terhubung ke internet dan bertukar data semakin penting untuk game di semua genre. Banyak permainan ada hampir seluruhnya melalui internet, dengan koneksi konstan ke komunitas player lain; game semacam ini disebut sebagai MMO (massively multiplayer online) dan paling dikenal melalui MMORPG (MMO role-playing game). Bahkan ketika game tidak memerlukan konektivitas konstan seperti itu, video game modern biasanya menggabungkan fitur seperti melaporkan skor ke daftar global skor tinggi. Unity menyediakan dukungan untuk jaringan semacam itu, jadi kami akan membahas fitur-fitur itu.

Unity mendukung berbagai pendekatan untuk komunikasi jaringan, karena pendekatan yang berbeda lebih cocok untuk kebutuhan yang berbeda. Namun, bab ini sebagian besar akan membahas jenis komunikasi internet yang paling umum: mengeluarkan permintaan HTTP.

7.1 APA ITU PERMINTAAN HTTP?

Saya berasumsi sebagian besar pembaca tahu apa itu permintaan HTTP, tetapi ini adalah petunjuk singkat untuk berjaga-jaga: HTTP adalah protokol komunikasi untuk mengirim permintaan ke dan menerima tanggapan dari server web. Misalnya, ketika Anda mengklik tautan di halaman web, browser Anda (klien) mengirimkan permintaan ke alamat tertentu, dan kemudian server itu merespons dengan halaman baru. Permintaan HTTP dapat diatur ke berbagai metode, khususnya GET atau POST untuk mengambil atau mengirim data. Permintaan HTTP dapat diandalkan, dan itulah sebabnya sebagian besar internet dibangun di sekitarnya. Permintaan itu sendiri, serta infrastruktur untuk menangani permintaan tersebut, dirancang untuk menjadi kuat dan menangani berbagai kegagalan dalam jaringan.

Dalam game online yang dibangun berdasarkan permintaan HTTP, game yang dikembangkan di Unity pada dasarnya adalah klien tebal yang berkomunikasi dengan server dalam gaya Ajax. Sebagai perbandingan yang baik, bayangkan bagaimana aplikasi web satu halaman modern bekerja (sebagai lawan dari developeran web jadul berdasarkan halaman web yang dihasilkan sisi server). Keakraban pendekatan ini dapat menyesatkan bagi developer web berpengalaman. Video game sering kali memiliki persyaratan kinerja yang jauh lebih ketat daripada aplikasi web, dan perbedaan ini dapat memengaruhi keputusan desain.

Scale waktu bisa sangat berbeda antara aplikasi web dan videogame. Setengah detik bisa tampak seperti menunggu sebentar untuk memperbarui situs web, tetapi menjeda bahkan hanya sebagian kecil dari waktu itu dapat menyiksa di tengah-tengah permainan aksi intensitas tinggi. Konsep "cepat" pasti relatif terhadap situasi.

Game online biasanya terhubung ke server yang khusus ditujukan untuk game tersebut; untuk tujuan pembelajaran, namun, kami akan terhubung ke beberapa sumber data internet yang tersedia secara gratis, termasuk data cuaca dan gambar yang dapat kami unduh. Bagian terakhir dari bab ini memang mengharuskan Anda untuk mengatur server web kustom; bagian itu opsional karena persyaratan itu, meskipun saya akan menjelaskan cara mudah untuk melakukannya dengan software sumber terbuka.

Rencana untuk bab ini adalah membahas beberapa penggunaan permintaan HTTP sehingga Anda dapat mempelajari cara kerjanya dalam Unity:

1. Siapkan scene luar ruangan (khususnya, bangun langit yang dapat bereaksi terhadap data cuaca).
2. Tulis kode untuk meminta data cuaca dari internet.
3. Parsing respon dan kemudian ubah scene berdasarkan data.
4. Download dan tampilkan gambar dari internet.
5. Posting data ke server Anda sendiri (dalam hal ini, log tentang cuaca saat itu).

Gim sebenarnya yang akan Anda gunakan untuk proyek bab ini tidak terlalu penting. Semua yang ada di bab ini akan menambahkan skrip baru ke proyek yang sudah ada dan tidak akan mengubah kode apa pun yang ada. Untuk kode sampel, saya menggunakan demo gerakan dari bab 2, sebagian besar agar kita dapat melihat langit dalam tampilan orang pertama saat dimodifikasi. Proyek untuk bab ini tidak secara langsung terkait dengan gameplay, tetapi jelas untuk sebagian besar game yang Anda buat, Anda ingin jaringan terkait dengan gameplay (misalnya, memunculkan musuh berdasarkan respons dari server).

7.2 MEMBUAT SCENE LUAR RUANGAN

Karena kita akan mengunduh data cuaca, pertama-tama kita akan menyiapkan area luar ruangan di mana cuaca akan terlihat. Bagian tersulit dari itu adalah langit, tetapi pertama-tama mari luangkan waktu sejenak untuk menerapkan tekstur yang tampak di luar ruangan pada geometri level.

Ingatlah untuk mengubah ukuran gambar yang diunduh ke pangkat 2, seperti 256x256. Kemudian impor gambar ke dalam proyek Unity, buat bahan, dan tetapkan gambar ke bahan (yaitu, drag gambar ke dalam slot tekstur bahan). Drag bahan ke dinding atau lantai di scene, dan tingkatkan ubin di bahan (coba angka seperti 8 atau 9 dalam satu atau kedua arah) sehingga gambar tidak akan diregangkan dengan cara yang jelek. Setelah tanah dan dinding dirawat, saatnya untuk mengatasi langit.

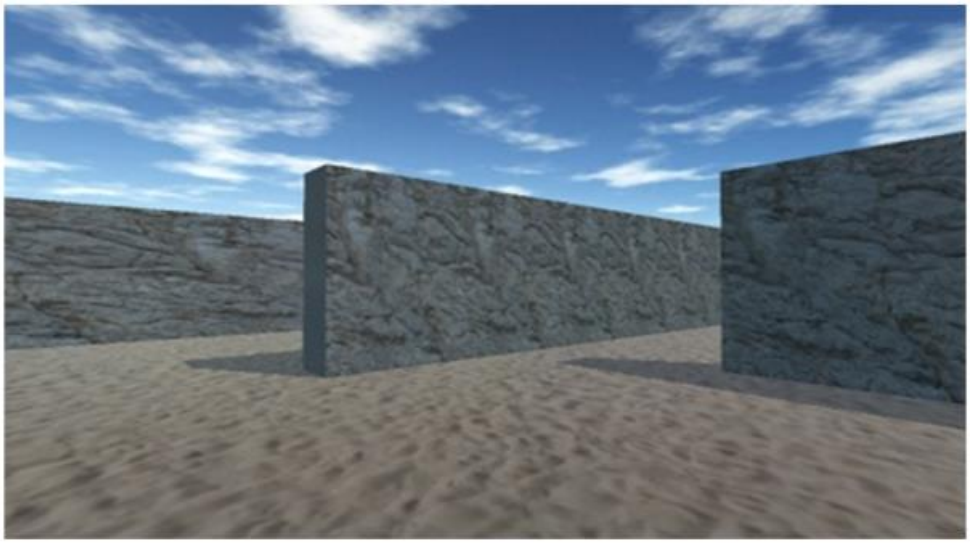
Menghasilkan visual langit menggunakan skybox

Mulailah dengan mengimpor gambar skybox seperti yang Anda lakukan di bab 4: buka www.93i.de untuk mengunduh gambar skybox. Kali ini dapatkan gambar untuk set DarkStormy selain TropicalSunnyDay (langit akan lebih kompleks dalam proyek ini). Impor tekstur ini ke tampilan Project, dan (seperti yang dijelaskan di bab 4) atur Wrap Mode ke Clamp.

Sekarang buat material baru yang akan digunakan untuk skybox ini. Di bagian atas pengaturan untuk bahan ini, klik menu Shader untuk melihat daftar drop-down dengan semua shader yang tersedia. Pindah ke bagian Skybox dan pilih 6-Sided di submenu itu. Dengan aktifnya shader ini, material sekarang memiliki enam slot tekstur (bukan hanya slot tekstur Albedo kecil yang dimiliki shader standar).

Drag gambar skybox SunnyDay ke slot tekstur materi baru. Nama gambar sesuai dengan slot tekstur untuk menetapkannya (atas, depan, dan seterusnya) Setelah keenam tekstur terhubung, Anda dapat menggunakan materi baru ini sebagai skybox untuk scene.

Tetapkan material skybox ini di jendela Lighting (Window > Lighting). Tetapkan material untuk skybox Anda ke slot Skybox di bagian atas jendela (drag material ke atas atau klik tombol lingkaran kecil di sebelah slot). Tekan Play dan Anda akan melihat sesuatu seperti gambar dibawah ini.



Gambar 7.1 Scene dengan gambar latar langit

Hebat, sekarang Anda memiliki scene luar ruangan! Skybox adalah cara elegan untuk menciptakan ilusi atmosfer luas yang mengelilingi player. Namun skybox shader yang dibangun ke dalam Unity memang memiliki satu batasan signifikan: gambar tidak akan pernah bisa berubah sehingga menghasilkan langit yang tampak benar-benar statis. Kami akan mengatasi batasan itu dengan membuat shader khusus baru.

7.3 MENYIAPKAN SUASANA YANG DIKENDALIKAN OLEH KODE

Gambar dalam rangkaian TropicalSunnyDay terlihat bagus untuk hari yang cerah, tetapi bagaimana jika kita ingin beralih antara cuaca cerah dan mendung? Ini akan membutuhkan kumpulan gambar langit kedua (beberapa gambar langit berawan) jadi kita membutuhkan shader baru untuk skybox.

Seperti yang dijelaskan di bab sebelumnya, shader adalah program singkat dengan instruksi tentang cara merender gambar. Itu menyiratkan bahwa Anda dapat memprogram shader baru, dan itulah kenyataannya. Kami akan membuat shader baru yang mengambil dua set gambar skybox dan transisi di antara keduanya. Untungnya shader untuk tujuan ini sudah ada di kumpulan skrip wiki Komunitas Unify: http://wiki.unity3d.com/index.php?title=SkyboxBlended .

Di Unity buat skrip shader baru: buka menu Buat seperti saat Anda membuat skrip C# baru, tetapi pilih Shader sebagai gantinya. Beri nama aset SkyboxBlended lalu klik dua kali shader untuk membuka skrip. Salin kode dari halaman wiki itu dan tempel ke skrip shader. Baris atas mengatakan Shader "Skybox/Blended", yang memberitahu Unity untuk menambahkan shader baru ke dalam daftar shader di bawah kategori Skybox (kategori yang sama dengan skybox biasa).

Kami tidak akan membahas semua detail program shader sekarang. Pemrograman shader adalah topik grafik komputer yang cukup canggih dan dengan demikian berada di luar cakupan buku ini. Anda mungkin ingin mencarinya setelah Anda menyelesaikan buku ini; jika demikian, mulai di sini: http://docs.unity3d.com/Manual/ShadersOverview.html .

Sekarang Anda dapat mengatur materi Anda ke shader Skybox Blended. Ada 12 slot tekstur, dalam dua set enam gambar. Tetapkan gambar TropicalSunnyDay ke enam tekstur pertama seperti sebelumnya; untuk tekstur yang tersisa, gunakan kumpulan gambar skybox DarkStormy.

Shader baru ini juga menambahkan slider Blend di dekat bagian atas pengaturan. Nilai Blend mengontrol seberapa banyak dari setiap kumpulan gambar skybox yang ingin Anda tampilkan; saat Anda menyesuaikan penggeser dari satu sisi ke sisi lain, skybox bertransisi dari cerah ke mendung. Anda dapat menguji dengan menyesuaikan penggeser dan memainkan game, tetapi menyesuaikan langit secara manual tidak terlalu membantu saat game sedang berjalan, jadi mari kita tulis beberapa kode untuk mentransisikan langit.

Buat objek kosong di scene dan beri nama Controller. Buat skrip baru dan beri nama WeatherController. Drag skrip itu ke objek kosong, lalu tulis daftar berikut di skrip itu.

Skrip WeatherController yang bertransisi dari cerah ke mendung

```
using UnityEngine;
using System.Collections;

public class WeatherController : MonoBehaviour {
   [SerializeField] private Material sky;
   [SerializeField] private Light sun;

   private float _fullIntensity;

   private float _cloudValue = 0f;

   void Start() {
      _fullIntensity = sun.intensity;
   }

   void Update() {
      SetOvercast(_cloudValue);
      _cloudValue += .005f;
   }

   private void SetOvercast(float value) {
      sky.SetFloat("_Blend", value);
      sun.intensity = _fullIntensity - (_fullIntensity * value);
   }
}
```

Saya akan menunjukkan beberapa hal dalam kode ini, tetapi kunci metode baru adalah SetFloat(), yang muncul hampir di bagian bawah. Semuanya sampai saat itu harus cukup akrab, tetapi yang satu itu baru. Metode ini menetapkan nilai angka pada materi. Nilai mana yang secara khusus merupakan parameter pertama untuk metode itu. Dalam hal ini, material memiliki properti yang disebut Blend (perhatikan bahwa properti material dalam kode dimulai dengan garis bawah).

Adapun sisa kode, beberapa variabel didefinisikan, termasuk material dan cahaya. Untuk bahan yang Anda ingin referensikan bahan skybox campuran yang baru saja kita buat, tapi ada apa dengan cahayanya? Itu agar scene juga akan menjadi gelap saat transisi dari cerah ke mendung; saat nilai Blend meningkat, kita akan mematikan lampu. Cahaya terarah dalam scene bertindak sebagai cahaya utama dan memberikan penerangan di mana-mana; drag cahaya itu ke dalam Inspector.

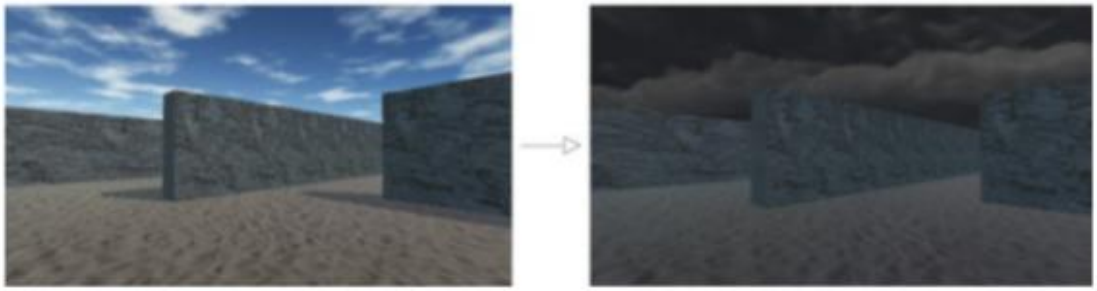
Sistem pencahayaan canggih (disebut Enlighten) di Unity memperhitungkan skybox untuk mencapai hasil yang realistis. Namun, pendekatan pencahayaan ini tidak akan berfungsi dengan baik dengan skybox yang berubah, jadi Anda mungkin ingin mematikannya. Di jendela Lighting Anda dapat mematikan Continuous Baking (istilah ini didefinisikan dalam bab 7) di bagian bawah; sekarang hanya akan diperbarui ketika Anda mengklik tombol. Atur Blend of the skybox ke tengah untuk tampilan rata-rata, lalu klik Build di bagian bawah jendela Lighting

untuk membuat lightmaps (folder bernama Scene ditambahkan ke proyek Anda; jangan sentuh folder itu).

Ketika skrip dimulai, itu menginisialisasi intensitas cahaya. Skrip akan menyimpan nilai awal dan menganggapnya sebagai intensitas "penuh". Intensitas penuh ini akan digunakan nanti dalam skrip saat meredupkan cahaya.

Kemudian kode menambah nilai setiap frame dan menggunakan nilai itu untuk menyesuaikan langit. Secara khusus, ia memanggil SetOvercast() setiap frame, dan fungsi itu merangkum beberapa penyesuaian yang dilakukan pada scene. Saya sudah menjelaskan apa yang dilakukan SetFloat() sehingga kita tidak akan membahasnya lagi, dan baris terakhir menyesuaikan intensitas cahaya.

Sekarang mainkan scene untuk melihat kode berjalan. Anda akan melihat apa yang digambarkan oleh Gambar 7.2: selama beberapa detik Anda akan melihat transisi scene dari hari yang cerah ke hari yang gelap dan mendung.



Gambar 7.2 Sebelum dan sesudah: transisi scene dari cerah ke mendung

Satu kekhasan yang tidak terduga tentang Unity adalah bahwa perubahan "Blend" pada materi bersifat permanen. Unity mengatur ulang objek dalam scene saat game berhenti berjalan, tetapi aset yang ditautkan langsung dari tampilan Project (seperti materi skybox) diubah secara permanen. Ini hanya terjadi di dalam editor Unity (perubahan tidak terbawa di antara permainan setelah game di-deploy di luar editor) dan dengan demikian dapat mengakibatkan bug yang membuat frustrasi jika Anda melupakannya. Sangat keren menyaksikan transisi scene dari cerah ke mendung. Tapi ini semua hanyalah pengaturan untuk tujuan sebenarnya: membuat cuaca di dalam game sinkron dengan kondisi cuaca dunia nyata. Untuk itu, kita perlu mulai mengunduh data cuaca dari internet.

Mengunduh data cuaca dari layanan internet

Sekarang setelah kita mengatur scene luar ruangan, kita dapat menulis kode yang akan mengunduh data cuaca dan memodifikasi scene berdasarkan data tersebut. Tugas ini akan memberikan contoh yang baik untuk mengambil data menggunakan permintaan HTTP. Layanan web untuk data cuaca gratis adalah OpenWeatherMap; Anda akan menggunakan API mereka (antarmuka pemrograman aplikasi, cara untuk mengakses layanan mereka menggunakan perintah kode alih-alih antarmuka grafis) yang terletak di http://openweathermap.org/api .Layanan web atau API web adalah server yang terhubung ke internet yang mengembalikan data berdasarkan permintaan. Tidak ada perbedaan teknis antara API web dan situs web; situs web adalah layanan web yang mengembalikan data untuk halaman web, dan browser menafsirkan data HTML sebagai dokumen yang terlihat.

Kode yang akan Anda tulis akan terstruktur di sekitar arsitektur Manajer yang sama dari bab 8. Kali ini Anda akan memiliki kelas WeatherManager yang diinisialisasi dari manajer pusat manajer. WeatherManager akan bertanggung jawab untuk mengambil dan menyimpan data cuaca, tetapi untuk melakukannya diperlukan kemampuan untuk berkomunikasi dengan internet.

Untuk mencapai itu, Anda akan membuat kelas utilitas bernama NetworkService. NetworkService akan menangani detail koneksi ke internet dan membuat permintaan HTTP. WeatherManager kemudian dapat memberi tahu NetworkService untuk membuat permintaan tersebut dan mengembalikan responsnya. Gambar dibawah menunjukkan bagaimana struktur kode ini akan beroperasi.

Agar ini berfungsi, jelas WeatherManager harus memiliki akses ke objek NetworkService. Anda akan mengatasinya dengan membuat objek di Manajer dan kemudian menyuntikkan objek NetworkService ke berbagai manajer saat mereka diinisialisasi. Dengan cara ini, WeatherManager tidak hanya akan memiliki referensi ke NetworkService, tetapi juga manajer lain yang Anda buat nanti.

Untuk mulai membawa arsitektur kode Manajer dari bab 8, pertama-tama salin ManagerStatus dan IGameManager (ingat bahwa IGameManager adalah antarmuka yang harus diterapkan oleh semua manajer, sedangkan ManagerStatus adalah enum yang digunakan IGameManager). Anda perlu memodifikasi IGameManager sedikit untuk mengakomodasi kelas NetworkService baru, jadi buat skrip baru bernama NetworkService (biarkan kosong untuk saat ini; Anda akan mengisinya nanti) dan kemudian sesuaikan IGameManager seperti yang ditunjukkan pada daftar 9.2.

Menyesuaikan IGameManager untuk menyertakan NetworkService

```
public interface IGameManager {
   ManagerStatus status {get;}

   void Startup(NetworkService service);
}
```

Selanjutnya mari kita buat WeatherManager untuk mengimplementasikan antarmuka yang sedikit disesuaikan ini. Buat skrip C# baru (lihat daftar berikut).

Skrip awal untuk WeatherManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class WeatherManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   // Add cloud value here (listing 9.8)
   private NetworkService _network;

   public void Startup(NetworkService service) {
      Debug.Log("Weather manager starting...");

      _network = service;

      status = ManagerStatus.Started;
   }
}
```

Pass awal di WeatherManager ini tidak benar-benar melakukan apa-apa. Untuk saat ini hanya jumlah minimum yang diperlukan IGameManager yang diimplementasikan oleh kelas: mendeklarasikan properti status dari antarmuka, serta mengimplementasikan fungsi Startup(). Anda akan mengisi kerangka kerja kosong ini selama beberapa bagian berikutnya.

Terakhir, salin Manajer dari bab 8 dan sesuaikan untuk memulai WeatherManager (lihat daftar berikutnya).

Managers.cs disesuaikan untuk menginisialisasi WeatherManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

[RequireComponent(typeof(WeatherManager))]

public class Managers : MonoBehaviour {
   public static WeatherManager Weather {get; private set;}

   private List<IGameManager> _startSequence;

   void Awake() {
      Weather = GetComponent<WeatherManager>();

      _startSequence = new List<IGameManager>();
      _startSequence.Add(Weather);
      StartCoroutine(StartupManagers());
   }

   private IEnumerator StartupManagers() {
      NetworkService network = new NetworkService();

      foreach (IGameManager manager in _startSequence) {
         manager.Startup(network);
      }


      yield return null;

      int numModules = _startSequence.Count;
      int numReady = 0;

      while (numReady < numModules) {
         int lastReady = numReady;
         numReady = 0;

         foreach (IGameManager manager in _startSequence) {
            if (manager.status == ManagerStatus.Started) {
               numReady++;
            }
         }

         if (numReady > lastReady)
            Debug.Log("Progress: " + numReady + "/" + numModules);

         yield return null;
      }

      Debug.Log("All managers started up");
   }
}
```

Dan itu semua yang diperlukan secara kode untuk arsitektur kode Manajer. Seperti yang Anda lakukan di bab sebelumnya, buat objek manajer game di scene, lalu lampirkan Manajer dan WeatherManager ke objek kosong. Meskipun pengelola belum melakukan apa pun, Anda dapat melihat pesan pengaktifan di konsol jika sudah diatur dengan benar. Wah, ada beberapa hal "boilerplate" yang harus dihindari! Sekarang kita bisa melanjutkan menulis kode jaringan.

7.4 MEMINTA DATA WWW MENGGUNAKAN COROUTINE

NetworkService saat ini merupakan skrip kosong, sehingga Anda dapat menulis kode di dalamnya untuk membuat permintaan HTTP. Kelas utama yang perlu Anda ketahui adalah WWW. Unity menyediakan kelas WWW untuk berkomunikasi dengan internet. Membuat instance objek WWW menggunakan URL akan mengirim permintaan ke URL tersebut.

Coroutine dapat bekerja dengan kelas WWW untuk menunggu permintaan selesai. Coroutine pertama kali diperkenalkan kembali di bab 3, di mana kami menggunakannya untuk menjeda beberapa kode untuk jangka waktu tertentu. Ingat penjelasan yang diberikan di sana: coroutine adalah fungsi khusus yang tampaknya berjalan di latar belakang suatu program, dalam siklus berulang yang berjalan di tengah jalan dan kemudian kembali ke program lainnya. Saat digunakan bersama dengan metode StartCoroutine(), kata kunci yield menyebabkan coroutine berhenti sementara, mengembalikan aliran program dan mengambil lagi dari titik itu pada frame berikutnya.

Dalam babsebelumnya coroutine dihasilkan di WaitForSeconds(), sebuah objek yang menyebabkan fungsi berhenti selama beberapa detik tertentu. Menghasilkan coroutine dengan WWW akan menjeda fungsi sampai permintaan jaringan itu selesai. Alur program di sini mirip dengan membuat panggilan Ajax asinkron dalam aplikasi web: pertama Anda mengirim permintaan, kemudian Anda melanjutkan dengan sisa program, dan setelah beberapa waktu Anda menerima tanggapan. Baiklah, mari kita terapkan hal ini dalam kode kita. Pertama buka skrip NetworkService dan ganti templat default dengan konten daftar berikut.

Membuat permintaan HTTP di NetworkServices

```
using UnityEngine;
using System.Collections;
using System;

public class NetworkService {
   private const string xmlApi =
"http://api.openweathermap.org/data/2.5/weather?q=Chicago,us&mode=xml";

   private bool IsResponseValid(WWW www) {
      if (www.error != null) {
         Debug.Log("bad connection");
         return false;
      }
      else if (string.IsNullOrEmpty(www.text)) {
         Debug.Log("bad data");
         return false;
      }
      else {   // all good
         return true;
      }
   }


   private IEnumerator CallAPI(string url, Action<string> callback) {
      WWW www = new WWW(url);
      yield return www;

      if (!IsResponseValid(www))
         yield break;

      callback(www.text);
   }

   public IEnumerator GetWeatherXML(Action<string> callback) {
      return CallAPI(xmlApi, callback);
   }
}
```

Ingat desain kode yang dijelaskan sebelumnya: WeatherManager akan memberi tahu NetworkService untuk mengambil data. Jadi semua kode ini belum benar-benar berjalan; Anda sedang menyiapkan kode yang akan dipanggil oleh WeatherManager nanti. Untuk menjelajahi daftar kode ini, mari kita mulai dari bawah dan naik ke atas.

Menulis metode coroutine yang mengalir satu sama lain

GetWeatherXML() adalah metode coroutine yang dapat digunakan kode luar untuk memberi tahu NetworkService agar membuat permintaan HTTP. Perhatikan bahwa fungsi ini memiliki IEnumerator untuk tipe pengembaliannya; metode yang digunakan dalam coroutine harus memiliki IEnumerator yang dideklarasikan sebagai tipe kembalian. Mungkin terlihat aneh pada awalnya bahwa GetWeatherXML() tidak memiliki pernyataan hasil. Coroutine dijeda oleh pernyataan hasil, yang menyiratkan bahwa setiap coroutine harus menghasilkan suatu tempat. Ternyata hasil dapat mengalir melalui beberapa metode. Jika metode coroutine awal itu sendiri memanggil metode lain, dan metode lain itu menghasilkan sebagian dari jalan, maka coroutine akan berhenti di dalam metode kedua itu dan melanjutkan di sana. Jadi, pernyataan hasil di CallAPI() menjeda coroutine yang dimulai di GetWeatherXML(); Gambar 9.4 menunjukkan aliran kode ini.

7.5 MEMAHAMI CARA KERJA CALLBACK

Ketika coroutine dimulai, metode dipanggil dengan parameter yang disebut callback, dan callback memiliki type Action. Tapi apa itu Action? Type Actions adalah delegasi (C# memiliki beberapa pendekatan untuk delegasi, tetapi yang ini adalah yang paling sederhana). Delegasi adalah referensi ke beberapa metode/fungsi lain. Mereka memungkinkan Anda untuk menyimpan fungsi (atau lebih tepatnya pointer ke fungsi) dalam variabel dan meneruskan fungsi itu sebagai parameter ke fungsi lain.

Jika Anda tidak terbiasa dengan konsep delegasi, sadari bahwa mereka memungkinkan Anda untuk menyebarkan fungsi seperti yang Anda lakukan pada angka dan string. Tanpa delegasi, Anda tidak dapat menyebarkan fungsi untuk dipanggil nanti—Anda hanya dapat memanggil fungsi tersebut secara langsung. Dengan delegasi, Anda dapat memberi tahu kode tentang metode lain untuk dipanggil nanti. Ini berguna untuk banyak tujuan, terutama untuk mengimplementasikan fungsi callback.

Callback adalah fungsi yang digunakan untuk berkomunikasi kembali ke objek pemanggil. Objek A dapat memberi tahu Objek B tentang salah satu metode di A. B nantinya dapat memanggil metode A untuk berkomunikasi kembali ke A. Misalnya, dalam hal ini callback digunakan untuk mengomunikasikan kembali data respons setelah menunggu permintaan HTTP selesai. Di CallAPI() kode pertama-tama membuat permintaan HTTP, lalu menghasilkan hingga permintaan itu selesai, dan akhirnya menggunakan callback() untuk mengirim kembali respons.

Perhatikan sintaks <> yang digunakan dengan kata kunci Action; jenis yang ditulis dalam kurung sudut menyatakan parameter yang diperlukan agar sesuai dengan Tindakan ini. Dengan kata lain, fungsi yang ditunjuk oleh Tindakan ini harus mengambil parameter yang cocok dengan tipe yang dideklarasikan. Dalam hal ini parameternya adalah string tunggal, jadi metode callback harus memiliki tanda tangan seperti ini:

MethodName(string value)

Konsep callback mungkin lebih masuk akal setelah Anda melihatnya beraksi, yang akan Anda lakukan dalam daftar 9.6; penjelasan awal ini adalah agar Anda mengenali apa yang terjadi

ketika Anda melihat kode tambahan itu. IsResponseValid() memeriksa kesalahan dalam respons HTTP. Ada dua jenis kesalahan: permintaan bisa saja gagal karena koneksi internet yang buruk, atau data yang dikembalikan mungkin salah format. Nilai const dideklarasikan dengan URL untuk membuat permintaan. (Kebetulan, Anda dapat mengubah URL ini untuk mendapatkan cuaca di lokasi yang berbeda.)

Memanfaatkan kode Jaringan

Itu membungkus kode di NetworkService. Sekarang mari kita gunakan NetworkService di WeatherManager; daftar berikutnya menunjukkan tambahan pada skrip itu.

Menyesuaikan WeatherManager untuk menggunakan NetworkService

```
...
public void Startup(NetworkService service) {
   Debug.Log("Weather manager starting...");

   _network = service;
   StartCoroutine(_network.GetWeatherXML(OnXMLDataLoaded));

   status = ManagerStatus.Initializing;
}

public void OnXMLDataLoaded(string data) {
   Debug.Log(data);

   status = ManagerStatus.Started;
}
...
```

Tiga perubahan utama dibuat pada kode di pengelola ini: memulai coroutine untuk mengunduh data dari internet, menyetel status pengaktifan yang berbeda, dan menentukan metode callback untuk menerima respons. Memulai coroutine itu sederhana. Sebagian besar kerumitan di balik coroutine sudah ditangani di NetworkService, jadi Anda hanya perlu memanggil StartCoroutine() di sini. Kemudian Anda menetapkan status startup yang berbeda, karena pengelola sebenarnya belum selesai melakukan inisialisasi; perlu menerima data dari internet sebelum startup selesai.

Selalu mulai metode jaringan menggunakan StartCoroutine(); jangan hanya memanggil fungsi secara normal. Ini mudah dilupakan karena membuat objek WWW di luar coroutine tidak menghasilkan kesalahan kompiler apa pun. Saat Anda memanggil metode StartCoroutine(), Anda perlu memanggil metode tersebut. Artinya, ketikkan tanda kurung—()—dan jangan hanya memberikan nama fungsinya. Dalam hal ini, metode coroutine membutuhkan fungsi callback sebagai satu parameternya, jadi mari kita definisikan fungsi itu. Kami akan menggunakan OnXMLDataLoaded() untuk callback; perhatikan bahwa metode ini memiliki parameter string, yang sesuai dengan deklarasi Action<string> dari NetworkService. Fungsi callback tidak melakukan banyak hal sekarang; baris debug hanya mencetak data yang diterima ke konsol untuk memverifikasi bahwa data telah diterima dengan benar. Kemudian baris terakhir dari fungsi mengubah status startup manajer untuk mengatakan bahwa itu benar-benar dimulai. Tekan Mainkan untuk menjalankan kode. Dengan asumsi Anda memiliki koneksi internet yang solid, Anda akan melihat banyak data muncul di konsol. Data ini hanyalah string panjang, tetapi string diformat dengan cara tertentu yang dapat kita manfaatkan.

Mengurai XML

Data yang ada sebagai string panjang biasanya memiliki bit informasi individual yang tertanam di dalam string. Anda mengekstrak bit informasi tersebut dengan mengurai string. Parsing berarti menganalisis sepotong data dan membaginya menjadi potongan-potongan informasi yang terpisah. Untuk mengurai string, itu perlu diformat dengan cara yang memungkinkan Anda (atau lebih tepatnya, kode parser) untuk mengidentifikasi bagian yang terpisah. Ada beberapa format standar yang biasa digunakan untuk mentransfer data melalui internet; format standar yang paling umum adalah XML.

XML adalah singkatan dari Extensible Markup Language. Ini adalah seperangkat aturan untuk menyandikan dokumen dengan cara yang terstruktur, mirip dengan halaman web HTML. Untungnya, Unity (atau lebih tepatnya Mono, kerangka kode yang dibangun ke dalam Unity) menyediakan fungsionalitas untuk mem-parsing XML. Data cuaca yang kami minta diformat dalam XML, jadi kami akan menambahkan kode ke WeatherManager untuk mengurai respons dan mengekstrak kekeruhan. Masukkan URL ke browser web untuk melihat kodenya; ada banyak di sana, tetapi kami hanya tertarik pada simpul yang berisi sesuatu seperti <clouds value="40" name="scattered cloud"/>.

Selain menambahkan kode untuk mengurai XML, kita akan menggunakan sistem messenger yang sama seperti yang kita lakukan di bab 6. Itu karena setelah data cuaca diunduh dan diuraikan, kita masih perlu menginformasikan scene tentang itu. Buat skrip bernama Messenger dan rekatkan kode dari halaman ini di wiki Unify: http://wiki.unity3d.com/index.php/CSharpMessenger_Extended.

Maka Anda perlu membuat skrip yang disebut GameEvent (lihat daftar berikutnya). Seperti yang dijelaskan di bab 6, sistem messenger ini sangat bagus untuk menyediakan cara terpisah untuk mengkomunikasikan event ke bagian lain dari program.

Kode Event Game:

```
public static class GameEvent {
public const string WEATHER_UPDATED = "WEATHER_UPDATED";
}
```

Setelah sistem messenger terpasang, sesuaikan WeatherManager seperti yang ditunjukkan pada daftar berikut.

Mengurai XML di WeatherManager

```
...
using System;
using System.Xml;
...
public float cloudValue {get; private set;}
...
public void OnXMLDataLoaded(string data) {
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(data);
    XmlNode root = doc.DocumentElement;

    XmlNode node = root.SelectSingleNode("clouds");
    string value = node.Attributes["value"].Value;
    cloudValue = Convert.ToInt32(value) / 100f;
    Debug.Log("Value: " + cloudValue);

    Messenger.Broadcast(GameEvent.WEATHER_UPDATED);

    status = ManagerStatus.Started;
}
...
```

Anda dapat melihat bahwa perubahan paling penting dibuat di dalam OnXMLDataLoaded(). Sebelumnya metode ini hanya mencatat data ke konsol untuk memverifikasi bahwa data masuk dengan benar. Daftar ini menambahkan banyak kode untuk mengurai XML. Pertama buat dokumen XML kosong baru; ini adalah wadah kosong yang dapat Anda isi dengan struktur XML yang diurai. Baris berikutnya mem-parsing string data ke dalam struktur yang dikandung oleh dokumen XML. Kemudian kita mulai dari akar pohon XML sehingga semuanya dapat mencari pohon dalam kode berikutnya.

Pada titik ini Anda dapat mencari node dalam struktur XML untuk menarik keluar bit informasi individu. Dalam hal ini, <clouds> adalah satu-satunya node yang kami minati. Pertama, temukan node tersebut dalam dokumen XML, lalu ekstrak atribut value dari node tersebut. Data ini mendefinisikan nilai cloud sebagai bilangan bulat 0-100, tetapi kita akan membutuhkannya sebagai float 0-1 untuk menyesuaikan scene nanti. Mengonversi itu adalah sedikit matematika sederhana yang ditambahkan ke kode.

Terakhir, setelah mengekstrak nilai kekeruhan dari data lengkap, siarkan pesan bahwa data cuaca telah diperbarui. Saat ini tidak ada yang mendengarkan pesan itu, tetapi penyiar tidak perlu tahu apa-apa tentang pendengar (memang, itulah inti dari sistem utusan yang dipisahkan). Nanti kita akan menambahkan pendengar ke scene. Bagus—kami telah menulis kode untuk mengurai data XML! Tetapi sebelum kita melanjutkan untuk menerapkan nilai ini ke scene yang terlihat, saya ingin membahas opsi lain untuk transfer data.

Mengurai JSON

Sebelum melanjutkan ke langkah berikutnya dalam proyek, mari kita jelajahi format alternatif untuk mentransfer data. XML adalah salah satu format umum untuk data yang ditransfer melalui internet, tetapi format umum lainnya disebut JSON. JSON adalah singkatan dari JavaScript Object Notation. Mirip dengan tujuan XML, JSON dirancang untuk menjadi alternatif yang ringan. Meskipun sintaks untuk JSON awalnya berasal dari JavaScript, formatnya tidak spesifik bahasa dan siap digunakan dengan berbagai bahasa pemrograman.
Tidak seperti XML, Mono tidak dilengkapi dengan parser untuk format ini. Ada sejumlah parser JSON bagus yang tersedia yang dapat Anda unduh, seperti MiniJSON (https://Gist.github.com/darktable/1411710). Buat skrip bernama MiniJSON dan rekatkan kode itu. Sekarang Anda dapat menggunakan perpustakaan ini untuk mengurai data JSON. Kami telah mendapatkan XML dari OpenWeatherMap API, tetapi kebetulan mereka juga

dapat mengirim data yang sama dengan format JSON. Untuk melakukannya, ubah NetworkService sesuai dengan daftar berikutnya.

Membuat permintaan NetworkService JSON alih-alih XML

```
...
private const string jsonApi =
"http://api.openweathermap.org/data/2.5/weather?q=Chicago,us";
...
public IEnumerator GetWeatherJSON(Action<string> callback) {
   return CallAPI(jsonApi, callback);
}
...
```

Ini hampir sama dengan kode untuk mengunduh data XML, kecuali bahwa URL-nya sedikit berbeda. Data yang dikembalikan dari permintaan ini memiliki nilai yang sama, tetapi diformat secara berbeda. Kali ini kami mencari potongan seperti "awan":{"all":40}. Tidak ada banyak kode tambahan yang diperlukan kali ini. Itu karena kami menyiapkan kode untuk permintaan ke dalam fungsi terpisah yang dibagi dengan baik, sehingga setiap permintaan HTTP berikutnya akan mudah ditambahkan. Bagus! Sekarang mari kita ubah WeatherManager untuk meminta data JSON alih-alih XML (lihat daftar berikut).

Memodifikasi WeatherManager untuk meminta JSON sebagai gantinya

```
...
using MiniJSON;
...
public void Startup(NetworkService service) {
   Debug.Log("Weather manager starting...");

   _network = service;
   StartCoroutine(_network.GetWeatherJSON(OnJSONDataLoaded));

   status = ManagerStatus.Initializing;
}
...
public void OnJSONDataLoaded(string data) {
   Dictionary<string, object> dict;
   dict = Json.Deserialize(data) as Dictionary<string,object>;

   Dictionary<string, object> clouds =
     (Dictionary<string,object>)dict["clouds"];
   cloudValue = (long)clouds["all"] / 100f;
   Debug.Log("Value: " + cloudValue);

   Messenger.Broadcast(GameEvent.WEATHER_UPDATED);

   status = ManagerStatus.Started;
}
...
```

Seperti yang Anda lihat, kode untuk bekerja dengan JSON terlihat mirip dengan kode untuk XML. Satu-satunya perbedaan nyata adalah bahwa parser JSON ini bekerja dengan Kamus standar alih-alih wadah dokumen khusus seperti yang dilakukan XML. Ada perintah untuk deserialize, dan itu mungkin kata yang asing. Deserialize berarti hampir sama dengan parse. Ini adalah kebalikan dari serialisasi, yang berarti mengkodekan kumpulan data ke dalam bentuk yang dapat ditransfer dan disimpan, seperti string JSON. Selain sintaks yang berbeda, semua langkahnya persis sama. Ekstrak nilai dari potongan data (untuk beberapa alasan nilai

disebut selama ini, tapi itu hanya kekhasan API) dan lakukan beberapa matematika sederhana untuk mengonversi nilai menjadi float 0-1. Setelah itu selesai, saatnya untuk menerapkan nilai ke scene yang terlihat.

Mempengaruhi scene berdasarkan Data Cuaca

Terlepas dari bagaimana tepatnya data diformat, setelah nilai kekeruhan diekstraksi dari data respons, kita dapat menggunakan nilai tersebut dalam metode SetOvercast() dari WeatherController. Baik XML atau JSON, string data akhirnya diuraikan menjadi serangkaian kata dan angka. Metode SetOvercast() mengambil angka sebagai parameter. Bagian selanjutnya kita akan menggunakan nomor yang bertambah setiap frame, tetapi kami dapat dengan mudah menggunakan nomor yang dikembalikan oleh API cuaca. Daftar berikutnya menampilkan skrip WeatherController lengkap lagi setelah modifikasi.

WeatherController yang bereaksi terhadap data cuaca yang diunduh

```
using UnityEngine;
using System.Collections;

public class WeatherController : MonoBehaviour {
    [SerializeField] private Material sky;
    [SerializeField] private Light sun;

    private float _fullIntensity;

    void Awake() {
        Messenger.AddListener(GameEvent.WEATHER_UPDATED, OnWeatherUpdated);
    }
    void OnDestroy() {
        Messenger.RemoveListener(GameEvent.WEATHER_UPDATED, OnWeatherUpdated);
    }


    void Start() {
        _fullIntensity = sun.intensity;
    }

    private void OnWeatherUpdated() {
        SetOvercast(Managers.Weather.cloudValue);
    }

    private void SetOvercast(float value) {
        sky.SetFloat("_Blend", value);
        sun.intensity = _fullIntensity - (_fullIntensity * value);
    }
}
```

Perhatikan bahwa perubahan tidak hanya penambahan; beberapa bit kode uji telah dihapus. Secara khusus, kami menghapus nilai kekeruhan lokal yang bertambah setiap frame; kami tidak membutuhkannya lagi, karena kami akan menggunakan nilai dari WeatherManager. Listener ditambahkan dan dihapus di Awake()/OnDestroy() (ini adalah fungsi MonoBehaviour yang dipanggil saat objek bangun atau dihapus). Listener ini adalah bagian dari sistem pesan siaran, dan akan memanggil OnWeatherUpdated() saat pesan itu diterima. OnWeatherUpdated() mengambil nilai kekeruhan dari WeatherManager dan memanggil SetOvercast() menggunakan nilai itu. Dengan cara ini, tampilan scene dikendalikan oleh data cuaca yang diunduh.

7.6 JARINGAN GAME DI LUAR HTTP

Permintaan HTTP kuat dan andal, tetapi latensi antara membuat permintaan dan menerima respons bisa sedikit lambat untuk banyak game. Oleh karena itu, permintaan HTTP adalah cara yang baik untuk melakukan pesan yang relatif lambat ke server (seperti bergerak dalam game berbasis giliran, atau mengirimkan skor tinggi untuk game apa pun), tetapi sesuatu seperti FPS multiplayer akan memerlukan pendekatan berbeda untuk jaringan.

Pendekatan yang berbeda ini melibatkan berbagai teknologi komunikasi, serta teknik untuk mengimbangi kelambatan. Misalnya, Unity menggunakan perpustakaan jaringan RakNet melalui sistem yang disebut panggilan prosedur jarak jauh (RPC). Ujung tombak untuk game aksi berjejaring adalah topik kompleks yang melampaui cakupan buku ini. Anda dapat mencari informasi lebih lanjut sendiri, mulai dari sini: http://docs.unity3d.com/Manual/NetworkReferenceGuide.html.

Menambahkan billboard berjaringan

Meskipun respons dari API web hampir selalu berupa string teks yang diformat dalam XML atau JSON, banyak jenis data lain yang ditransfer melalui internet. Selain data teks, jenis data yang paling umum diminta adalah gambar. Objek WWW Unity juga dapat digunakan untuk mengunduh gambar. Anda akan mempelajari tugas ini dengan membuat papan iklan yang menampilkan gambar yang diunduh dari internet. Anda perlu mengkode dua langkah: mengunduh gambar untuk ditampilkan, dan menerapkan gambar itu ke objek billboard. Kemudian, sebagai langkah ketiga, Anda akan meningkatkan kode sehingga gambar akan disimpan untuk digunakan di beberapa billboard.

Memuat gambar dari internet

Pertama mari kita tulis kode untuk mengunduh gambar. Anda akan mengunduh beberapa fotografi lanskap domain publik (lihat gambar 7.3) untuk diuji. Gambar yang diunduh belum akan terlihat di papan iklan; Saya akan menunjukkan kepada Anda skrip untuk menampilkan gambar di bagian berikutnya, tetapi sebelum itu, mari siapkan kode yang akan mengambil gambar.

Gambar 7.3 Fotografi lanskap

Arsitektur kode untuk mengunduh gambar terlihat hampir sama dengan arsitektur untuk mengunduh data. Modul pengelola baru (disebut ImagesManager) akan bertanggung jawab atas gambar yang diunduh untuk ditampilkan. Sekali lagi, detail koneksi ke internet dan pengiriman permintaan HTTP akan ditangani di NetworkService, dan ImagesManager akan memanggil NetworkService untuk mengunduh gambar untuk itu. Penambahan kode pertama ada di NetworkService. Daftar berikut menambahkan unduhan gambar ke skrip itu.

Mengunduh gambar di NetworkService

```
...
private const string webImage =
"http://upload.wikimedia.org/wikipedia/commons/c/c5/
     Moraine_Lake_17092005.jpg";
...
public IEnumerator DownloadImage(Action<Texture2D> callback) {
   WWW www = new WWW(webImage);
   yield return www;
   callback(www.texture);
}
...
```

Kode yang mengunduh gambar terlihat hampir identik dengan kode untuk mengunduh data. Perbedaan utama adalah jenis metode callback; perhatikan bahwa callback kali ini menggunakan Texture2D alih-alih string. Itu karena Anda mengirim kembali respons yang relevan: Anda mengunduh serangkaian data sebelumnya—sekarang Anda mengunduh gambar. Daftar berikutnya berisi kode untuk ImagesManager baru. Buat skrip baru dan masukkan kode itu.

Membuat ImagesManager untuk mengambil dan menyimpan gambar

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;

public class ImagesManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   private NetworkService _network;

   private Texture2D _webImage;

   public void Startup(NetworkService service) {
      Debug.Log("Images manager starting...");

      _network = service;


      status = ManagerStatus.Started;
   }
   public void GetWebImage(Action<Texture2D> callback) {
      if (_webImage == null) {
         StartCoroutine(_network.DownloadImage(callback));
      }
      else {
         callback(_webImage);
      }
   }
}
```

Bagian yang paling menarik dari kode ini adalah GetWebImage(); segala sesuatu yang lain dalam skrip ini terdiri dari properti standar dan metode yang mengimplementasikan antarmuka manajer. Saat GetWebImage() dipanggil, itu akan mengembalikan (melalui fungsi callback) gambar web. Pertama akan memeriksa apakah _webImage sudah memiliki gambar yang disimpan: jika tidak, itu akan memanggil panggilan jaringan untuk mengunduh gambar.

Jika _webImage sudah memiliki gambar yang disimpan, GetWebImage() akan mengirim kembali gambar yang disimpan (daripada mengunduh gambar lagi). Saat ini gambar yang diunduh tidak pernah disimpan, yang berarti _webImage akan selalu kosong. Kode yang menentukan apa yang harus dilakukan ketika _webImage tidak kosong sudah ada, jadi Anda akan menyesuaikan kode untuk menyimpan gambar itu di bagian berikut. Penyesuaian ini ada di bagian terpisah karena melibatkan beberapa sihir kode yang rumit. Tentu saja, seperti semua modul manajer, ImagesManager perlu ditambahkan ke Manajer; daftar berikut merinci penambahan ke Managers.cs.

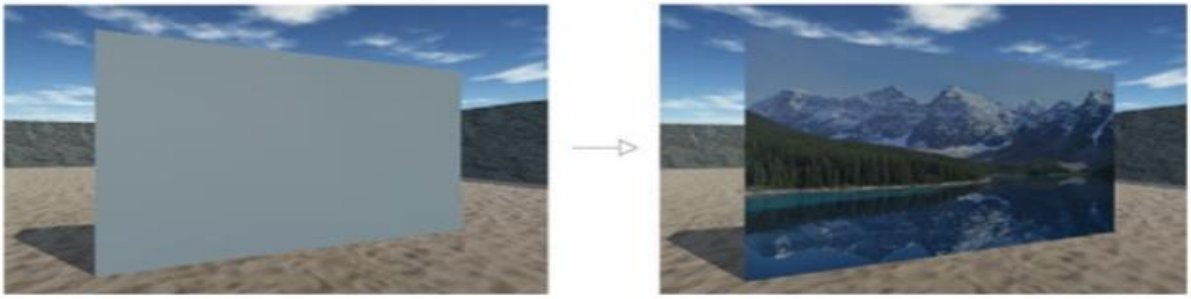
Menambahkan manajer baru ke Managers.cs

```
...
[RequireComponent(typeof(ImagesManager))]
...
public static ImagesManager Images{get; private set;}
... void Awake() {
Weather = GetComponent<WeatherManager>(); Images=
    GetComponent<ImagesManager>();
startSequence = new List<IGameManager>();
startSequence.Add(Weather);
_startSequence.Add(Images);
StartCoroutine(StartupManagers()); }
...
```

Tidak seperti cara kami menyiapkan WeatherManager, GetWebImage() di ImagesManager tidak dipanggil secara otomatis saat startup. Sebaliknya, kode menunggu sampai dipanggil; yang akan terjadi di bagian selanjutnya.

Menampilkan gambar di billboard

ImagesManager yang baru saja Anda tulis tidak melakukan apa pun sampai dipanggil, jadi sekarang kita akan membuat objek billboard yang akan memanggil metode di ImagesManager. Pertama buat kubus baru dan kemudian letakkan di tengah scene, pada sesuatu seperti Posisi 0 1.5 -5 dan Scale 5 3 .5 (lihat gambar 7.4).



Gambar 7.4 Objek billboard, sebelum dan sesudah menampilkan gambar yang diunduh

Anda akan membuat perangkat yang beroperasi seperti monitor yang berubah warna di bab 8. Salin skrip DeviceOperator dan letakkan di pemutar. Seperti yang Anda ingat, skrip itu akan mengoperasikan perangkat terdekat saat tombol Fire3 ditekan (yang didefinisikan dalam pengaturan input proyek sebagai tombol Command kiri). Buat juga skrip untuk

perangkat billboard yang disebut WebLoadingBillboard, letakkan skrip itu di objek billboard, dan masukkan kode dari daftar berikutnya.

Skrip perangkat WebLoadingBillboard

```
using UnityEngine;
using System.Collections;

public class WebLoadingBillboard : MonoBehaviour {
   public void Operate() {
      Managers.Images.GetWebImage(OnWebImage);
   }

   private void OnWebImage(Texture2D image) {
      GetComponent<Renderer>().material.mainTexture = image;
   }
}
```

Kode ini melakukan dua hal utama: memanggil ImagesManager.GetWebImage() saat perangkat dioperasikan, dan menerapkan gambar dari fungsi callback. Tekstur diterapkan pada bahan sehingga Anda dapat mengubah tekstur pada bahan yang ada di billboard. Gambar diatas menunjukkan seperti apa tampilan billboard setelah Anda memainkan game tersebut.

AssetBundles: Cara mengunduh jenis aset lainnya

Mengunduh gambar cukup mudah menggunakan objek WWW, tetapi bagaimana dengan jenis aset lainnya, seperti objek mesh dan prefab? WWW memiliki properti untuk teks dan gambar, tetapi aset lainnya sedikit lebih rumit. Unity dapat mengunduh semua jenis aset melalui mekanisme yang disebut AssetBundles. Singkat cerita, pertama-tama Anda mengemas beberapa aset ke dalam bundel, dan kemudian Unity dapat mengekstrak aset setelah mengunduh bundel. Detail pembuatan dan pengunduhan AssetBundles berada di luar cakupan buku ini; jika Anda ingin mempelajari lebih lanjut, mulailah dengan membaca bagian manual Unity ini: http://docs.unity3d.com/Manual/AssetBundlesIntro.html

Bagus, gambar yang diunduh ditampilkan di billboard! Tetapi kode ini dapat dioptimalkan lebih lanjut untuk bekerja dengan banyak billboard. Mari kita atasi pengoptimalan itu di bagian selanjutnya.

7.7 CACHING GAMBAR YANG DIUNDUH UNTUK DIGUNAKAN KEMBALI

Seperti disebutkan di bagian 9.3.1, ImagesManager belum menyimpan gambar yang diunduh. Itu berarti gambar akan diunduh berulang kali untuk beberapa billboard. Ini tidak efisien, karena itu akan menjadi gambar yang sama setiap saat. Untuk mengatasi ini, kami akan menyesuaikan ImagesManager untuk menyimpan gambar yang telah diunduh. Cache berarti menyimpan secara lokal. Konteks yang paling umum (tetapi tidak hanya!) melibatkan gambar yang diunduh dari internet.

Kuncinya adalah menyediakan fungsi callback di ImagesManager yang pertama-tama menyimpan gambar, lalu memanggil callback dari WebLoadingBillboard. Ini sulit dilakukan (berlawanan dengan kode saat ini yang menggunakan callback dari WebLoadingBillboard) karena kode tersebut tidak mengetahui sebelumnya seperti apa callback dari WebLoadingBillboard. Dengan kata lain, tidak ada cara untuk menulis metode di ImagesManager yang memanggil metode tertentu di WebLoadingBillboard karena kodenya tidak tahu apa metode spesifik itu. Cara mengatasi teka-teki ini adalah dengan menggunakan fungsi lambda. Fungsi lambda (juga disebut fungsi anonim) adalah fungsi yang tidak memiliki

nama. Fungsi seperti itu biasanya dibuat dengan cepat di dalam fungsi lain. Fungsi Lambda adalah fitur kode rumit yang didukung dalam sejumlah bahasa pemrograman, termasuk C#. Dengan menggunakan fungsi lambda untuk callback di ImagesManager, kode dapat membuat fungsi callback dengan cepat menggunakan metode yang diteruskan dari WebLoadingBillboard. Dengan demikian Anda tidak perlu mengetahui metode untuk memanggil sebelumnya, karena fungsi lambda ini tidak ada sebelumnya! Daftar berikut menunjukkan bagaimana melakukan voodoo ini di ImagesManager.

Fungsi Lambda untuk callback di ImagesManager

```
...
using System;
...
public void GetWebImage(Action<Texture2D> callback) {
   if (_webImage == null) {
      StartCoroutine(_network.DownloadImage((Texture2D image) => {
         _webImage = image;
         callback(_webImage);
      }));
   }
   else {
      callback(_webImage);
   }
}
...
```

Perubahan utama adalah pada fungsi yang diteruskan ke NetworkService.DownloadImage(). Sebelumnya kode melewati metode callback yang sama dari WebLoadingBanner. Namun, setelah perubahan, callback yang dikirim ke NetworkService adalah fungsi lambda terpisah yang dideklarasikan di tempat yang memanggil metode dari WebLoadingBanner. Perhatikan sintaks untuk mendeklarasikan metode lambda: () => {}. Menjadikan callback sebagai fungsi terpisah memungkinkan melakukan lebih dari sekadar memanggil metode di WebLoadingBanner; secara khusus, fungsi lambda juga menyimpan salinan lokal dari gambar yang diunduh. Jadi GetWebImage() hanya perlu mengunduh gambar pertama kali; semua panggilan berikutnya akan menggunakan gambar yang disimpan secara lokal.

Karena pengoptimalan ini berlaku untuk panggilan berikutnya, efeknya hanya akan terlihat di beberapa papan iklan. Mari kita duplikat objek billboard sehingga akan ada billboard kedua di scene. Pilih objek billboard, tekan Duplicate (lihat di bawah menu Edit atau klik kanan), dan pindahkan duplikat ke atas (misalnya, ubah posisi X menjadi 18).

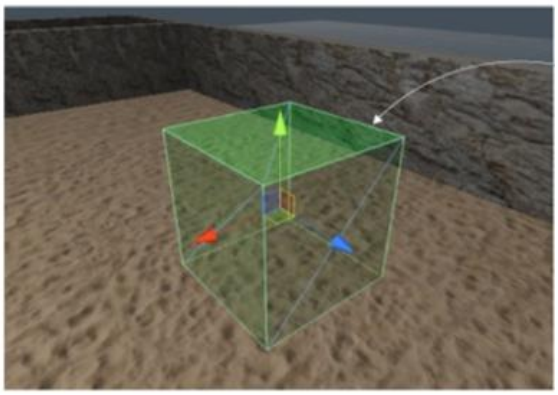
Sekarang mainkan permainannya dan lihat apa yang terjadi. Saat Anda mengoperasikan billboard pertama, akan ada jeda yang terlihat saat gambar diunduh dari internet. Namun ketika Anda berjalan ke billboard kedua, gambar akan langsung muncul karena sudah diunduh. Ini adalah pengoptimalan penting untuk mengunduh gambar (ada alasan browser web menyimpan gambar secara default). Ada satu lagi tugas jaringan utama yang harus diselesaikan: mengirim data kembali ke server.

7.8 MEMPOSTING DATA KE SERVER WEB

Kami telah membahas beberapa contoh pengunduhan data, tetapi kami masih perlu melihat contoh pengiriman data. Bagian terakhir ini memang mengharuskan Anda memiliki server untuk mengirim permintaan, jadi bagian ini opsional. Tetapi mudah untuk mengunduh

software sumber terbuka untuk menyiapkan server untuk diuji. Saya merekomendasikan XAMPP untuk server uji. Buka www.apachefriends.org untuk mengunduh XAMPP. Setelah terinstal dan server berjalan, Anda dapat mengakses folder htdocs XAMPP dengan alamat http://localhost/seperti halnya server di internet. Setelah Anda menjalankan dan menjalankan XAMPP, buat folder bernama ch9 di htdocs; di situlah Anda akan meletakkan skrip sisi server.

Apakah Anda menggunakan XAMPP atau server web Anda sendiri yang ada, tugas sebenarnya adalah mengirim data cuaca ke server saat player mencapai checkpoint di scene. Checkpoint ini akan menjadi volume pemicu, sama seperti pemicu pintu di bab 8. Anda perlu membuat objek kubus baru, memposisikan objek ke satu sisi scene, menyetel penumbuk ke Pemicu, dan menerapkan bahan semitransparan seperti Anda lakukan di bab sebelumnya (ingat, atur Mode Rendering materi). Gambar 9.7 menunjukkan objek checkpoint dengan bahan semitransparan hijau yang diterapkan.



Gambar 7.5 Objek checkpoint yang memicu pengiriman data

Sekarang objek pemicu ada di dalam scene, mari kita tulis kode yang dipanggilnya.

Melacak cuaca saat ini: mengirim permintaan pos

Kode yang dipanggil oleh objek checkpoint akan mengalir melalui beberapa skrip. Seperti kode untuk mengunduh data, kode untuk mengirim data akan melibatkan WeatherManager yang memberi tahu NetworkService untuk membuat permintaan, sementara NetworkService menangani detail komunikasi HTTP. Daftar berikutnya menunjukkan penyesuaian yang perlu Anda lakukan pada NetworkService.

Pertama, perhatikan bahwa CallAPI() memiliki parameter baru. Ini adalah tabel argumen untuk dikirim bersama dengan permintaan HTTP. Di dalam CallAPI() objek WWWForm dapat dibuat sesuai dengan tabel argumen tersebut. Biasanya WWW mengirimkan permintaan GET, tetapi WWWForm akan mengubahnya menjadi permintaan POST untuk mengirim data. Semua perubahan lain dalam kode bereaksi terhadap perubahan sentral itu (misalnya, memodifikasi kode GetWhatever() karena parameter CallAPI()).

Daftar berikutnya menunjukkan apa yang perlu Anda tambahkan di WeatherManager.

Menambahkan kode ke WeatherManager yang mengirim data

```
...
public void LogWeather(string name) {
StartCoroutine(_network.LogWeather(name, cloudValue, OnLogged));
} private void OnLogged(string response) {
Debug.Log(response); }
```

...

Terakhir, manfaatkan kode itu dengan menambahkan skrip checkpoint ke volume pemicu di scene. Buat skrip bernama CheckpointTrigger, letakkan skrip itu di volume pemicu, dan masukkan konten daftar berikutnya.

Skrip CheckpointTrigger untuk volume pemicu

```
using UnityEngine;
using System.Collections;

public class CheckpointTrigger : MonoBehaviour {
   public string identifier;

   private bool _triggered;

   void OnTriggerEnter(Collider other) {
      if (_triggered) {return;}

      Managers.Weather.LogWeather(identifier);
      _triggered = true;
   }
}
```

Slot Identifier akan muncul di Inspector; beri nama seperti checkpoint1. Jalankan kode dan data akan dikirim saat Anda memasuki checkpoint. Responsnya akan menunjukkan kesalahan, karena tidak ada skrip di server untuk menerima permintaan. Itu langkah terakhir di bagian ini.

7.9 KODE SISI SERVER DI PHP

Server harus memiliki skrip untuk menerima data yang dikirim dari game. Skrip server pengkodean berada di luar cakupan buku ini, jadi kami tidak akan membahasnya secara mendetail di sini. Kami hanya akan menyiapkan skrip PHP karena itu adalah pendekatan termudah. Buat file teks di htdocs (atau di mana pun server web Anda berada) dan beri nama file api.php (lihat daftar 9.20).

Skrip server yang ditulis dalam PHP yang menerima data kami

```
<?php

$message = $_POST['message'];
$cloudiness = $_POST['cloud_value'];
$timestamp = $_POST['timestamp'];
$combined = $message." cloudiness=".$cloudiness." time=".$timestamp."\n";

$filename = "data.txt";
file_put_contents($filename, $combined, FILE_APPEND | LOCK_EX);

echo "Logged";

?>
```

Perhatikan bahwa skrip ini menulis data yang diterima ke dalam data.txt, jadi Anda juga perlu meletakkan file teks dengan nama itu di server. Setelah api.php terpasang, Anda akan melihat log cuaca muncul di data.txt saat memicu checkpoint dalam game. Besar!

BAB 8
ANIMASI

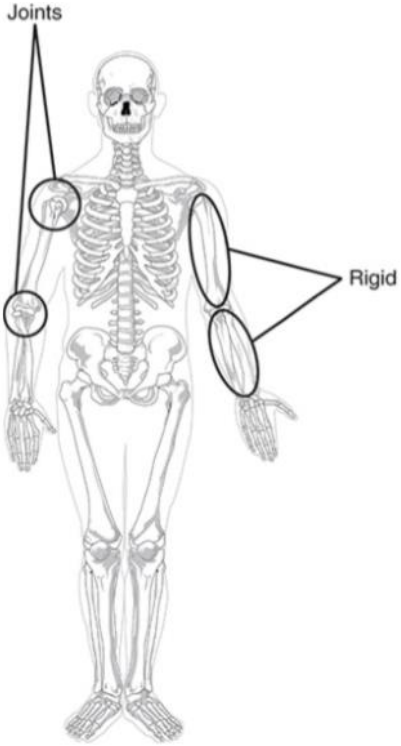
Pada jam ini, Anda akan belajar tentang animasi di Unity. Anda akan mulai dengan mempelajari dengan tepat apa itu animasi dan apa yang diperlukan agar dapat berfungsi. Setelah itu, Anda akan melihat berbagai jenis animasi. Dari sana, Anda akan belajar cara membuat animasi kustom Anda sendiri dengan alat animasi Unity.

8.1 DASAR-DASAR ANIMASI

Animasi adalah kumpulan gerakan visual yang dibuat sebelumnya. Dalam permainan 2D, animasi melibatkan beberapa gambar berurutan yang dapat dibalik dengan sangat cepat untuk memberikan tampilan gerakan (seperti buku flip kuno). Animasi di dunia 3D jauh berbeda. Dalam game 3D, Anda menggunakan model untuk mewakili entitas game. Anda tidak bisa begitu saja beralih di antara model untuk memberikan ilusi gerak. Sebaliknya, Anda harus benar-benar memindahkan bagian-bagian model. Melakukannya membutuhkan rig dan animasi. Selanjutnya, animasi juga dapat dianggap sebagai "otomatisasi"; yaitu, Anda dapat menggunakan animasi untuk mengotomatiskan perubahan objek seperti ukuran Collider, nilai variabel skrip, atau bahkan warna material.

Rig

Mencapai tindakan animasi yang kompleks, seperti berjalan, tanpa rig tidak mungkin (atau sangat sulit). Tanpa rig, komputer tidak memiliki cara untuk mengetahui bagian mana dari model yang seharusnya bergerak dan bagaimana mereka seharusnya bergerak. Jadi, apa sebenarnya rig itu? Sama seperti kerangka manusia (lihat Gambar 8.1), rig menentukan bagian-bagian dari model yang kaku, yang sering disebut tulang. Ini juga menentukan bagian mana yang bisa ditekuk; bagian yang dapat ditekuk ini disebut sendi



Gambar 8.1 Kerangka sebagai rig.

Tulang dan sendi bekerja sama untuk menentukan struktur fisik model. Struktur inilah yang digunakan untuk benar-benar menghidupkan model. Perlu dicatat bahwa animasi 2D, animasi sederhana, dan animasi pada objek sederhana tidak memerlukan rig tertentu atau kompleks.

Animasi

Setelah model memiliki rig (atau tidak, dalam kasus animasi sederhana), dapat diberikan animasi. Pada tingkat teknis, animasi hanyalah serangkaian instruksi untuk properti atau rig. Instruksi ini dapat dimainkan seperti film. Mereka bahkan dapat dijeda, dilewati, atau dibalik. Selanjutnya, dengan rig yang tepat, mengubah aksi model semudah mengubah animasi. Bagian terbaik dari semuanya adalah jika Anda memiliki dua model yang sama sekali berbeda yang memiliki rigging yang sama (atau rigging yang berbeda tetapi serupa, seperti yang akan Anda temukan di Hour 18, "Animator"), Anda dapat menerapkan animasi yang sama ke masing-masing model secara identik. . Jadi, orc, manusia, raksasa, dan manusia serigala semua bisa melakukan tarian yang sama persis.

3d Artists Wanted

Kebenaran tentang animasi adalah bahwa sebagian besar pekerjaan dilakukan di luar program seperti Unity. Secara umum, pemodelan, tekstur, rigging, dan animasi semuanya dibuat oleh para profesional, yang dikenal sebagai seniman 3D, dalam program seperti Blender, Maya, dan 3ds Max. Menciptakan aset-aset ini membutuhkan keterampilan dan latihan yang signifikan. Oleh karena itu, ciptaan mereka tidak tercakup dalam teks ini. Sebagai gantinya, buku ini menunjukkan kepada Anda bagaimana membangun pengalaman interaktif di Unity dengan menyatukan aset yang sudah dibuat. Ingatlah bahwa membuat game lebih dari sekadar menyusun potongan-potongan. Anda mungkin membuat game, tetapi seniman membuat game terlihat bagus

8.2 JENIS ANIMASI

Sejauh ini Anda telah membaca tentang hal-hal seperti animasi, rig, dan otomatisasi. Istilah-istilah ini mungkin tampak agak tidak masuk akal pada saat ini, dan Anda mungkin bertanya-tanya bagaimana kaitannya dan apa sebenarnya yang Anda butuhkan untuk menggunakan animasi dalam permainan. Bagian ini membahas berbagai jenis animasi dan membantu Anda memahami cara kerjanya sehingga Anda dapat mulai membuatnya.

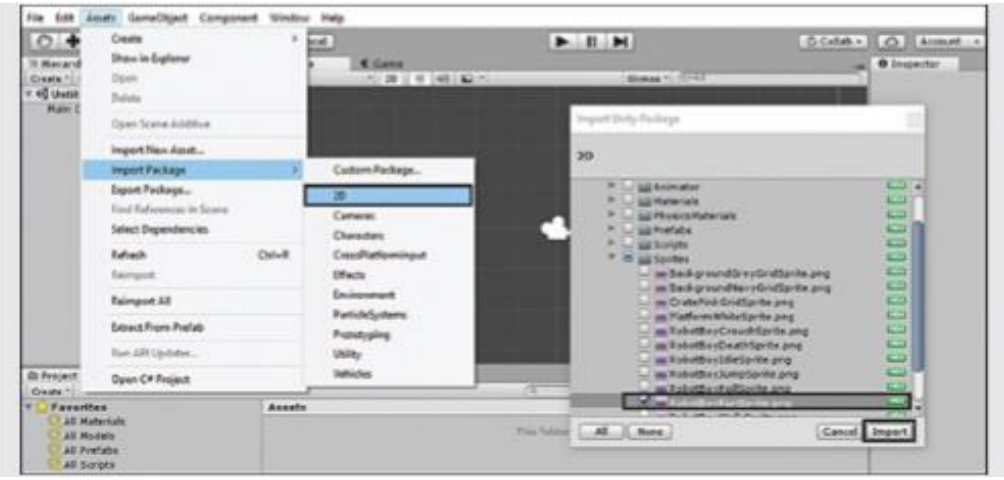
Animasi 2D

Dalam arti tertentu, animasi 2D adalah jenis animasi yang paling sederhana. Seperti yang dijelaskan sebelumnya di jam ini, fungsi animasi 2D sangat mirip dengan buku flip (atau kartun animasi atau bahkan film berbasis film). Ide di balik animasi 2D adalah bahwa gambar disajikan secara berurutan dengan kecepatan yang sangat cepat, menipu mata untuk melihat gerakan. Menyiapkan animasi 2D dalam Unity sangat mudah, tetapi memodifikasinya lebih sulit. Alasannya adalah bahwa animasi 2D membutuhkan aset seni (gambar yang ditampilkan) agar dapat berfungsi. Setiap perubahan pada animasi mengharuskan Anda (atau seniman) untuk membuat perubahan pada gambar itu sendiri di software lain, seperti Photoshop atau Gimp. Tidak mungkin membuat perubahan pada gambar itu sendiri di dalam Unity.

Mengiris Sprite

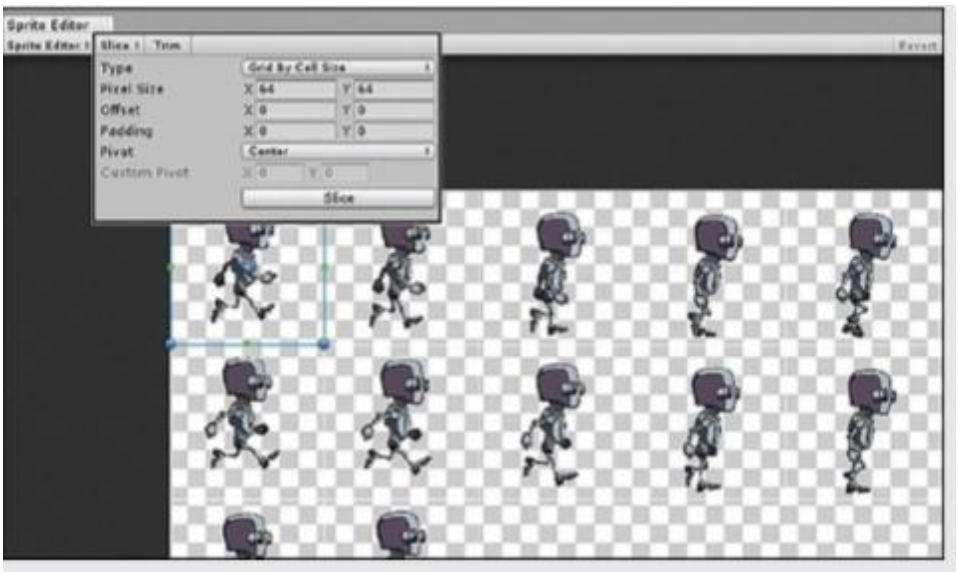
Sheet untuk Animasi Dalam latihan ini, Anda akan menyiapkan sprite sheet untuk animasi. Proyek yang dibuat dalam latihan ini akan digunakan nanti, jadi pastikan untuk menyimpannya. Ikuti langkah-langkah berikut:

1. Buat proyek 2D baru.
2. Impor gambar RobotBoyRunSprite.png dari paket aset 2D. (Meskipun gambar ini telah disiapkan untuk animasi—ini adalah karakter animasi dalam paket aset 2D—Anda memeriksanya dalam latihan ini sebagai tinjauan.) Anda dapat melakukannya dengan memilih Aset > Impor Paket > 2D dan hanya mengimpor RobotBoyRunSprite .png aset Sebagai alternatif, Anda dapat menemukan aset RobotBoyRunSprite.png dalam file buku untuk Jam 17.



Gambar 8.2 Mengimpor lembar sprite.

3. Pilih file RobotBoyRunSprite.png yang baru diimpor dalam tampilan Proyek.
4. Pada tampilan Inspector, pastikan Sprite Mode diatur ke Multiple lalu klik Sprite Editor. Di sudut kiri atas, klik Irisan lalu pilih Kotak menurut Ukuran Sel sebagai tipenya. Perhatikan ukuran grid dan irisan sprite yang dihasilkan



Gambar 8.3 Mengiris lembaran sprite.

5. Tutup jendela Editor Sprite.

Menemukan Kembali Roda

Jam ini Anda menggunakan aset dari paket aset 2D Unity. Anda mungkin telah memperhatikan bahwa aset ini sudah dianimasikan, yang berarti Anda sedang melakukan

pekerjaan yang telah selesai. Dengan cara ini, Anda dapat melihat pekerjaan yang awalnya digunakan untuk menghidupkan karakter-karakter ini. Lebih baik lagi, Anda dapat menjelajahi aset yang telah selesai dalam paket aset 2D untuk mengetahui bagaimana aset tersebut disatukan dan untuk melihat bagaimana contoh yang lebih kompleks dapat dicapai.

8.3 MEMBUAT ANIMASI

Anda telah menyiapkan aset Anda, jadi Anda sekarang siap untuk mengubahnya menjadi animasi. Ada cara sederhana untuk melakukan ini, dan ada juga cara rumit untuk melakukannya. Cara yang rumit melibatkan pembuatan aset animasi, menentukan properti penyaji sprite, menambahkan bingkai kunci, dan memberikan nilai. Karena Anda belum belajar bagaimana melakukan semua itu (walaupun Anda akan melakukannya di bagian selanjutnya), mari kita lanjutkan dengan rute sederhana. Unity memiliki alur kerja otomatis yang kuat, dan Anda akan memanfaatkannya untuk membuat animasi Anda.

Gunakan proyek yang Anda buat "Mengiris Sprite Sheet untuk Animasi" dan ikuti langkah-langkah berikut untuk membuat animasi:

1. Buka proyek yang Anda buat di "Mengiris Sprite Sheet untuk Animasi."
2. Temukan aset RobotBoyRunSprite dalam tampilan Proyek. Luaskan laci sprite (dengan mengklik panah kecil di sisi kanan sprite) untuk melihat semua subsprite.
3. Pilih semua sprite dari sprite sheet itu dengan memilih sprite pertama dan kemudian pilih sprite terakhir sambil menahan tombol Shift. Kemudian drag semua bingkai ke tampilan Scene (atau tampilan Hierarki; salah satu berfungsi) dan lepaskan

Gambar 8.4 Membuat animasi

4. Jika Anda diminta dengan dialog Simpan Sebagai, pilih nama dan lokasi untuk animasi baru Anda. Jika Anda tidak diminta, dua aset baru akan dibuat di folder yang sama dengan lembar sprite. Either way, Unity sekarang telah mengotomatiskan proses pembuatan karakter sprite animasi di scene Anda. Dua aset yang dibuat (aset animasi dan aset lain yang disebut controller animator) dibahas secara lebih rinci di Jam 18.

5. Jalankan scene Anda, dan Anda akan melihat anak robot animasi memainkan urutan larinya. Dalam tampilan Inspector, Anda dapat melihat properti Sprite dari komponen Sprite Renderer yang bersepeda melalui berbagai bingkai animasi.

Lebih Banyak Animasi

Sekarang Anda tahu cara membuat satu animasi 2D, tetapi bagaimana jika Anda menginginkan serangkaian animasi (seperti berjalan, berlari, diam, melompat, dan sebagainya) yang semuanya bekerja bersama? Untungnya, konsep yang baru saja Anda pelajari terus bekerja dalam skenario yang lebih kompleks. Namun, mendapatkan animasi untuk bekerja bersama membutuhkan pemahaman yang lebih besar tentang sistem animasi Unity. Anda akan mempelajari sistem itu dengan sangat rinci di Jam 18, di mana Anda akan menggunakan animasi 3D yang diimpor. Ingatlah bahwa di mana pun Anda dapat menggunakan animasi 3D, Anda juga dapat menggunakan jenis animasi lainnya. Dengan demikian, konsep yang akan anda pelajari di Hour 18 sama-sama berlaku untuk animasi 2D dan kustom.

Alat Animasi

Unity memiliki seperangkat alat animasi bawaan yang dapat Anda gunakan untuk membuat dan memodifikasi animasi tanpa meninggalkan editor. Anda telah menggunakannya secara tidak sadar ketika Anda membuat animasi 2D, dan sekarang saatnya untuk menggali dan melihat apa yang dapat Anda lakukan.

Jendela Animasi

Untuk mulai menggunakan alat animasi Unity, Anda perlu membuka jendela Animasi. Anda dapat melakukannya dengan mengklik Window > Animation (bukan Animator). Ini membuka tampilan baru yang dapat Anda ubah ukurannya dan berlabuh seperti jendela Unity lainnya. Secara umum, merupakan ide yang baik untuk memasang jendela ini sehingga Anda dapat menggunakannya dan bagian editor lainnya tanpa tumpang tindih. Gambar 17.5 menunjukkan jendela Animasi dan berbagai elemennya. Perhatikan bahwa untuk tujuan gambar ini, sprite animasi 2D dari latihan sebelumnya dipilih. Lihat apakah Anda dapat mengidentifikasi bagaimana Unity menggunakan sprite yang Anda drag ke dalam scene untuk membuat animasi yang Anda lihat saat menjalankan proyek Anda. Tabel 17.1 berjalan melalui beberapa bagian terpenting dari jendela Animasi.

Membuat Animasi Baru

Setelah Anda terbiasa dengan alat animasi, Anda siap untuk menggunakannya. Membuat animasi melibatkan penempatan bingkai kunci dan kemudian memilih nilai untuknya. Nilai bingkai di antara semua bingkai utama dihitung agar Anda dapat bertransisi dengan mulus di antara mereka—misalnya, untuk membuat objek bergerak ke atas dan ke bawah. Anda dapat dengan mudah mencapai ini dengan memilih untuk mengubah posisi transformasi objek dan menambahkan tiga bingkai utama. Yang pertama akan memiliki nilai sumbu y yang "rendah", yang kedua akan lebih tinggi, dan yang ketiga akan sama dengan yang pertama. Akibatnya, benda tersebut akan terpental ke atas dan ke bawah. Ini semua cukup konseptual, jadi kerjakan contoh berikut untuk merasakan prosesnya dengan lebih baik.

Waktu Animasi

Nilai pada timeline mungkin tampak agak aneh pada pandangan pertama. Faktanya, membaca timeline membantu Anda memahami laju sampel klip animasi yang diberikan. Berdasarkan kecepatan sampel default 60 frame per detik, timeline akan menghitung frame 0 hingga 59 dan kemudian, alih-alih frame 60, ia memiliki 1:00 (untuk 1 detik). Oleh karena itu, waktu 1:30 berarti 1 detik dan 30 frame. Ini mudah ketika Anda memiliki animasi 60 bingkai

per detik, tetapi apa yang terjadi dengan animasi 12 bingkai per detik? Maka penghitungannya adalah "1, 2, 3,…11, 12, satu detik." Dengan kata lain, Anda bisa melihat "0:10, 0:11, 1:00, 1:01, 1:02,…1:10, 1:11, 2:00,…" Hal terpenting untuk ambil dari ini adalah bahwa angka sebelum titik dua adalah detik, dan angka yang mengikutinya adalah bingkai.

Memindahkan Timeline

Jika Anda ingin memperkecil atau menggeser timeline untuk melihat lebih banyak bingkai atau melihat bingkai yang berbeda, Anda dapat melakukannya dengan mudah. Jendela menggunakan gaya navigasi yang sama dengan tampilan Scene dalam mode 2D. Artinya, menggulir roda mouse memperbesar dan memperkecil timeline, sambil menahan Alt (Option di Mac) dan menyeret panci di sekitar.

Mode Rekam

Meskipun alat yang Anda gunakan sejauh ini sangat mudah digunakan, ada cara yang lebih mudah untuk bekerja dengan animasi. Elemen penting dalam alat animasi adalah mode Rekam. Dalam mode Rekam, setiap perubahan yang Anda buat pada objek direkam ke dalam animasi. Ini bisa menjadi cara yang sangat ampuh untuk membuat penambahan animasi yang cepat dan akurat. Ini juga bisa sangat berbahaya. Pertimbangkan apa yang akan terjadi jika Anda lupa bahwa Anda berada dalam mode Rekam dan membuat banyak perubahan pada suatu objek. Perubahan-perubahan itu akan direkam ke dalam animasi dan diulang-ulang setiap kali dimainkan. Oleh karena itu, umumnya disarankan untuk selalu memastikan bahwa Anda tidak dalam mode Rekam saat Anda tidak ingin menggunakannya. Ini adalah peringatan yang cukup menakutkan untuk alat yang bahkan belum Anda gunakan, tetapi jangan khawatir. Sebenarnya tidak terlalu buruk (dan sangat kuat untuk boot). Dengan sedikit disiplin, ini adalah alat yang fantastis. Dalam skenario terburuk, jika Unity merekam sesuatu untuk Anda yang tidak Anda inginkan, Anda dapat menghapusnya sendiri secara manual dari animasi.

Editor Curves

Alat terakhir yang akan Anda lihat pada jam ini adalah Editor Curves. Sejauh ini, Anda telah berada dalam tampilan Dopesheet, yang merupakan tampilan dengan bingkai utama yang terdaftar dan diperinci secara datar. Anda mungkin telah memperhatikan bahwa saat Anda mengarahkan nilai bingkai kunci, Anda tidak mengontrol nilai di antaranya. Jika Anda bertanya-tanya bagaimana nilai ditentukan, jangan bertanya-tanya lagi. Unity memadukan nilai (dalam proses yang disebut interpolasi) di antara bingkai kunci untuk menciptakan transisi yang mulus. Di Editor Curves, Anda dapat melihat seperti apa tampilannya. Untuk masuk ke Editor Curves, cukup klik tombol berlabel Curves di bagian bawah tampilan Animasi Dalam mode ini, Anda dapat beralih nilai mana yang ingin Anda lihat dengan mengklik propertinya di sebelah kiri. Di Editor Curves, Anda dapat melihat dengan tepat bagaimana nilai sedang ditransisikan di antara bingkai utama. Anda dapat menyeret bingkai kunci untuk mengubah nilainya atau bahkan mengklik dua kali kurva untuk membuat bingkai kunci baru pada titik di mana Anda mengklik. Jika Anda tidak menyukai bagaimana Unity menghasilkan nilai di antara bingkai utama, Anda dapat mengklik kanan bingkai kunci dan memilih Free Smooth. Unity kemudian memberi Anda dua pegangan yang dapat Anda pindahkan untuk mengubah bagaimana nilai dihaluskan. Jangan ragu untuk bermain-main dan lihat kegilaan macam apa yang bisa Anda buat dengan kurva.

Menggunakan Editor Curves

Anda mungkin telah memperhatikan bahwa kubus dari "Using Mode Rekam" Coba Sendiri tidak berputar dengan mulus dan malah memiliki gerakan start-and-stop yang lambat.

Dalam latihan ini Anda akan memodifikasi animasi untuk memberikan gerakan berputar yang mulus pada kubus. Ikuti langkah-langkah ini:

1. Buka scene dengan kubus pemintalan Anda dari "Using Mode Rekam" Coba Sendiri. Dalam tampilan Animasi, klik tombol Curves untuk beralih ke Editor Curves
2. Klik properti Rotation.y untuk menunjukkan kurvanya di timeline. Jika kurvanya kecil atau tidak pas di jendela, cukup gerakkan kursor mouse Anda ke timeline dan tekan tombol F.
3. Meluruskan kurva rotasi akan memberikan kubus Anda animasi halus yang bagus, jadi klik kanan bingkai utama pertama (pada waktu 0:00) dan pilih Otomatis. Lakukan hal yang sama untuk keyframe terakhir.

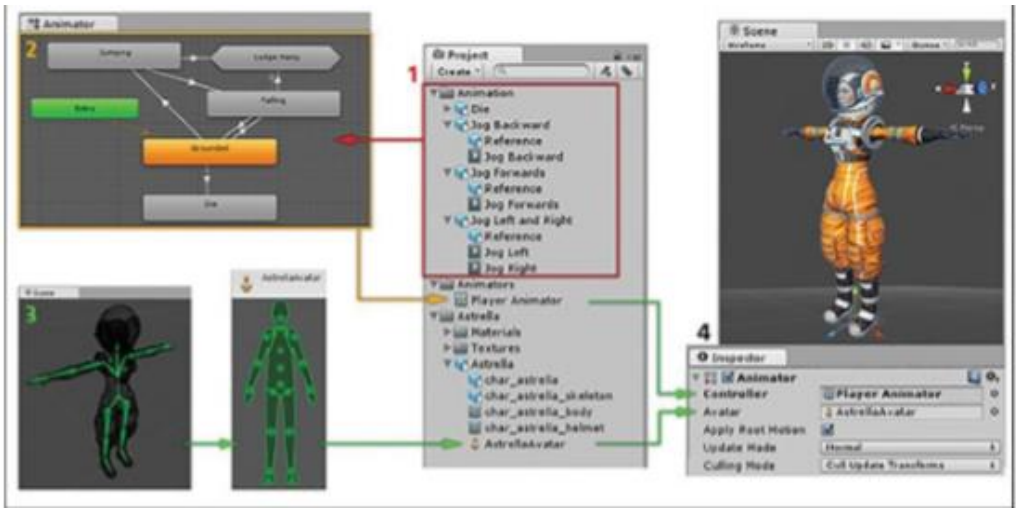
Gambar 8.5 Memodifikasi kurva rotasi.

5. Sekarang setelah kurva diluruskan, masuk ke mode rotate untuk melihat animasi kubus yang dimodifikasi.

Animator

Dasar-dasar Animator

Semua animasi di Unity dimulai dengan komponen Animator. Di Jam 17, saat Anda membuat dan mempelajari animasi, Anda menggunakan animator tanpa benar-benar menyadarinya. Pada intinya, sistem animasi Unity (Mecanim) terdiri dari tiga bagian: klip animasi, controller Animator, dan komponen Animator. Ketiga bagian ini semuanya ada untuk membuat karakter Anda menjadi hidup.



Gambar 8.6 Bagaimana bagian-bagian dari animasi humanoid berhubungan satu sama lain.

Klip animasi (lihat #1 pada Gambar 18.1) adalah berbagai gerakan yang Anda impor atau buat di Unity. Controller Animator (#2) berisi animasi Anda dan menentukan klip mana yang harus diputar pada saat tertentu. Model memiliki sesuatu yang disebut avatar (#3) yang bertindak sebagai "penerjemah" antara controller Animator dan pengaturan model. Anda biasanya dapat mengabaikan avatar karena sudah diatur dan digunakan secara otomatis. Terakhir, controller Animator (yang bisa Anda sebut sebagai "controller") dan avatar disatukan pada model menggunakan komponen Animator (#4). Sepertinya banyak yang harus diingat? Jangan khawatir. Sebagian besar hal ini bersifat intuitif atau otomatis. Salah satu hal terbaik tentang sistem animasi Unity adalah Anda dapat menggunakannya untuk "menargetkan ulang" animasi ke objek game lain. Jika Anda menganimasikan kubus, Anda juga dapat menerapkan animasi itu ke bola. Jika Anda menganimasikan karakter, Anda dapat menerapkan animasi ke karakter lain dengan pengaturan yang sama (atau pengaturan yang berbeda, seperti yang akan segera Anda lihat). Ini berarti Anda dapat, misalnya, memiliki orc dan seorang pria yang melakukan tarian bahagia yang sama bersama-sama.

8.4 MENGANALISIS KASUS PENGGUNAAN TERTENTU

Untuk mendapatkan hasil maksimal dari jam ini, Anda akan bekerja dengan kasus penggunaan yang sangat spesifik: animasi 3D pada model humanoid (kasus penggunaan yang sangat umum tentunya). Ini akan memungkinkan Anda untuk belajar tentang animasi 3D, mengimpor model dan animasi, bekerja dengan rig, dan menggunakan sistem penargetan ulang humanoid Unity yang mengagumkan. Ingatlah bahwa, dengan pengecualian penargetan ulang humanoid, semua yang tercakup dalam jam ini berlaku sepenuhnya untuk semua jenis animasi lainnya. Jadi, jika Anda sedang membangun sistem animasi 2D multi-bagian, semua pengetahuan yang dipelajari di sini tetap penting.

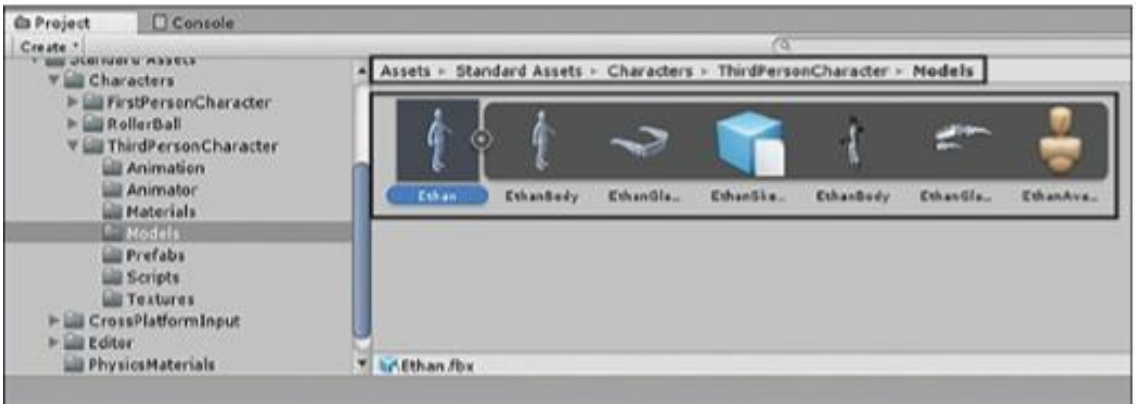
Tinjau Ulang Rigging

Untuk mulai membangun sistem animasi yang kompleks, pertama-tama Anda harus memastikan bahwa rigging model Anda sudah siap. Ingat dari Jam 17, “Animasi,” bahwa model dan animasi harus dicurangi dengan cara yang persis sama agar dapat berfungsi. Ini berarti akan sangat sulit untuk membuat animasi yang dibuat untuk satu model untuk bekerja pada model yang berbeda. Oleh karena itu, animasi dan model umumnya dibuat khusus untuk bekerja sama. Jika Anda menggunakan model humanoid (dua lengan, dua kaki, kepala, dan dada), Anda memiliki kemampuan untuk memanfaatkan alat penargetan ulang animasi Mecanim. Dengan sistem Mecanim, humanoid dapat dipetakan ulang riggingnya di editor tanpa menggunakan alat pemodelan 3D apa pun. Hasilnya adalah setiap animasi yang dibuat untuk model humanoid dapat bekerja dengan humanoid lain yang Anda miliki. Ini berarti animator (orang-orang, bukan aset Unity) dapat menghasilkan perpustakaan animasi yang besar yang dapat diterapkan ke berbagai macam model menggunakan banyak rig yang berbeda.

Mengimpor Model

Untuk jam ini, Anda akan menggunakan Ethan, model dari paket aset standar Karakter. Model ini hadir dengan banyak item yang berbeda, dan Anda akan memeriksa setiap bagian untuk memastikan bahwa itu dikonfigurasi dengan benar. Untuk mengimpor model, pilih Aset > Impor Paket > Karakter. Biarkan semuanya diperiksa dan klik Impor. Sekarang lanjutkan dan temukan Ethan di tab Project Anda di bawah Assets\Standard Assets\Characters \ThirdPersonCharacter\Models (lihat Gambar 8.7). Jika Anda mengklik panah kecil di sebelah

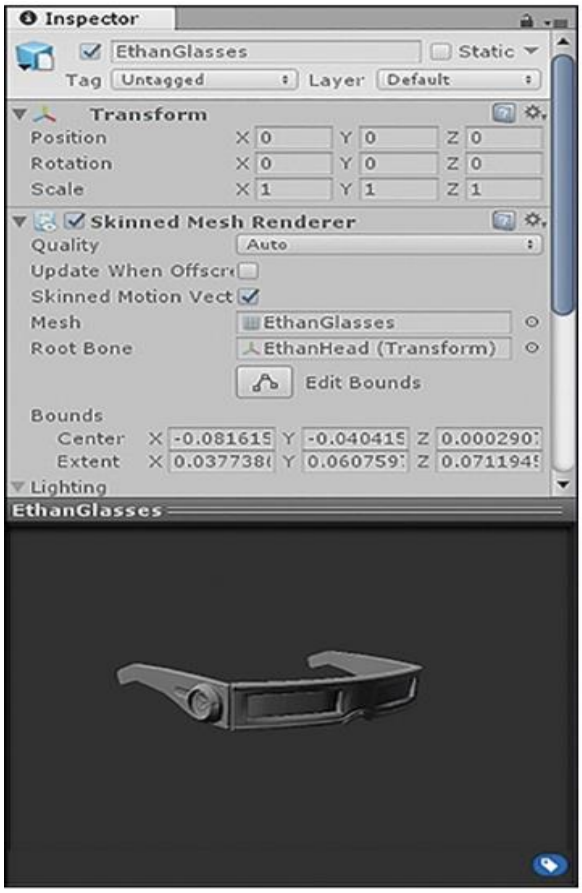
kanan file Ethan, Anda dapat memperluas model untuk melihat semua bagian penyusunnya (lihat Gambar 8.7). Bagaimana bagian-bagian ini disusun tergantung pada bagaimana model diekspor dari aplikasi 3D yang digunakan untuk membuatnya. Komponen-komponen ini, dari kiri ke kanan, tubuh Ethan dengan tekstur, kacamata bertekstur, definisi kerangka, mesh EthanBody mentah, mesh EthanGlasses mentah, dan akhirnya definisi avatar Ethan (yang digunakan untuk rigging).



Gambar 8.7 Menemukan model Ethan.

Preview Mesh

Jika Anda mengklik model Ethan atau Kacamata di baki, Anda akan melihat jendela preview kecil di bagian bawah Inspector. (Jika tidak, drag ke atas untuk menunjukkannya.) Di sini Anda dapat memutar submodel itu untuk melihatnya dari semua sudut. Setelah selesai melihat komponen, tutup baki Ethan.fbx dengan mengklik panah di sebelah kanan aset.



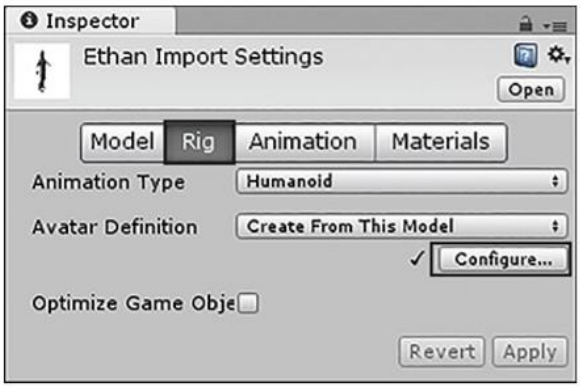
Gambar 8.8 Tampilan Inspector model.

Mengonfigurasi Aset Anda

Sekarang setelah Anda mengimpor model dan animasi (yang disertakan dengan aset lainnya), Anda perlu mengonfigurasinya. Proses untuk mengonfigurasi animasi cukup identik dengan mengonfigurasi model. Dengan model yang dipilih, Anda dapat melihat pengaturan impor yang tercantum dalam tampilan Inspector. Tab Model adalah rumah bagi semua pengaturan yang menentukan bagaimana model diimpor ke Unity. Barang-barang ini dapat diabaikan dengan aman untuk keperluan jam ini. Tab yang Anda perhatikan untuk saat ini adalah tab Rig. (Tab Animasi akan dibahas beberapa saat kemudian pada jam ini.)

Persiapan Rig

Anda mengonfigurasi rig model dalam pengaturan impor, di bawah tab Rig di tampilan Inspector. Properti yang paling Anda perhatikan di sini adalah Jenis Animasi (lihat Gambar 8.9). Saat ini ada empat jenis yang tersedia di dropdown: Tidak ada, Legacy, Generic, dan Humanoid. Menyetel properti ini ke None menyebabkan Unity mengabaikan rig model ini. Legacy adalah untuk sistem animasi lama Unity dan tidak boleh digunakan. Generik adalah untuk semua model nonhumanoid (model sederhana, kendaraan, bangunan, hewan, dan sebagainya), dan semua model yang diimpor ke Unity default ke jenis animasi ini. Terakhir, Humanoid (yang akan Anda gunakan) adalah untuk semua karakter humanoid. Pengaturan ini memungkinkan Unity untuk menargetkan ulang animasi untuk Anda. Seperti yang Anda lihat, Ethan sudah diatur dengan benar sebagai humanoid. Saat Anda menetapkan model sebagai humanoid, Unity secara otomatis melakukan proses pemetaan rig untuk Anda. Jika Anda ingin melihat betapa mudahnya ini, Anda cukup mengubah Jenis Animasi ke Generik, klik Terapkan, lalu ubah kembali (seperti itulah model ini awalnya disiapkan untuk Anda; tidak ada pekerjaan tambahan yang disembunyikan). Untuk melihat pekerjaan yang dilakukan Unity untuk Anda, Anda dapat masuk ke alat rigging dengan mengklik tombol Configure



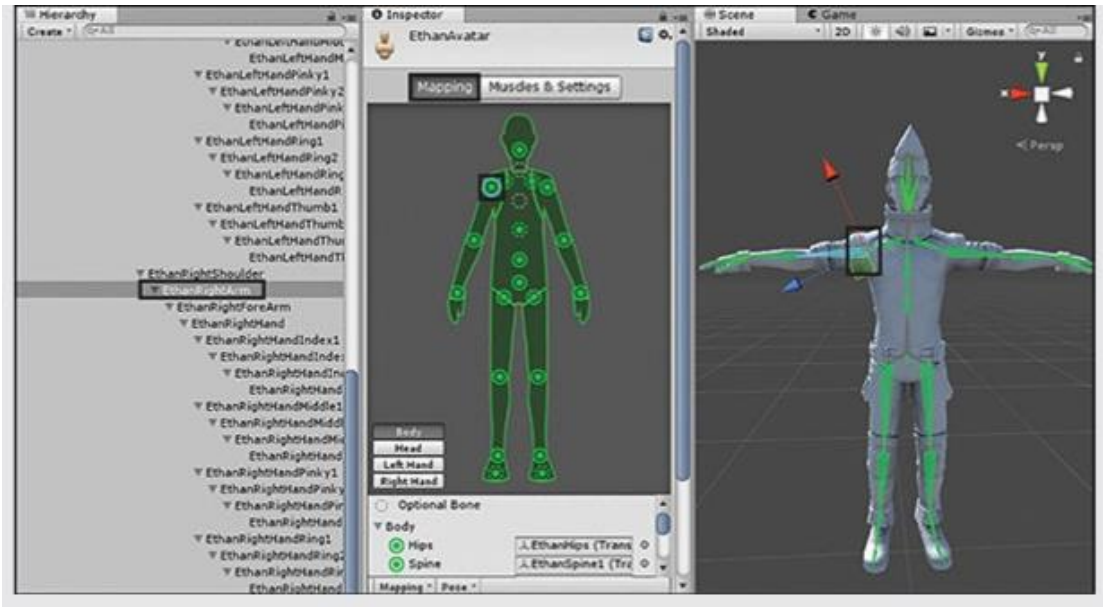
Gambar 8.9 Pengaturan rig.

Menjelajahi Bagaimana Ethan Dicurangi

Dalam latihan ini, Anda akan melihat bagaimana Ethan dicurangi. Ini akan memberi Anda ide yang jauh lebih praktis tentang bagaimana model yang dicurangi dirakit. Ikuti langkah-langkah berikut:

1. Jika Anda belum melakukannya, buat new project dan impor aset karakter dari Aset Standar. Temukan aset Ethan.fbx dan pilih untuk menampilkan pengaturan impornya di tampilan Inspector, seperti yang dijelaskan sebelumnya dalam jam ini.
2. Klik Konfigurasi pada tab Rig. Melakukannya akan meluncurkan Anda ke scene baru, jadi simpan yang lama jika diminta.

3. Atur ulang antarmuka Anda sehingga Anda dapat melihat terutama tampilan Hierarki dan Inspector. (Jam 1, "Pengantar Kesatuan," membahas cara menutup dan memindahkan tab.) Anda mungkin ingin menyimpan tata letak ini. Anda selalu dapat kembali ke Default nanti.
4. Dengan memilih tab Mapping, klik berbagai lingkaran hijau (lihat Gambar 8.10). Perhatikan bagaimana ini menyoroti anak yang sesuai dari EthanSkeleton dalam tampilan Hierarchy dan menempatkan lingkaran biru di sekitar titik kerangka yang sesuai di bawah garis besar. Perhatikan semua poin ekstra dari rig dalam tampilan Hierarchy. Potongan-potongan itu tidak penting untuk humanoids dan karenanya tidak ditargetkan ulang. Namun, jangan khawatir: Mereka tetap berperan dalam memastikan bahwa model terlihat benar saat bergerak.



Gambar 8.10 Tampilan Rigging dengan lengan kanan dipilih.

6. Lanjutkan menjelajahi bagian tubuh lainnya dengan mengklik Body, Head, Left Hand, dan seterusnya. Ini semua adalah sendi, yang dapat sepenuhnya ditargetkan ulang untuk humanoid apa pun.
7. Klik Selesai jika sudah selesai. Perhatikan bahwa Ethan(Clone) sementara menghilang dari tampilan Hierarchy.

8.5 PERSIAPAN ANIMASI

Anda dapat menggunakan animasi yang disertakan dengan Ethan, tetapi itu akan membosankan dan tidak akan menggambarkan fleksibilitas sistem Mecanim. Sebagai gantinya, Anda akan menggunakan beberapa animasi lain yang disediakan dalam file buku untuk Hour 18. Setiap animasi memiliki opsi yang mengontrol bagaimana animasi berperilaku yang harus dikonfigurasi secara khusus seperti yang Anda inginkan. Misalnya, Anda perlu memastikan bahwa animasi berjalan berputar dengan benar sehingga transisi tidak memiliki jahitan yang jelas. Bagian ini memandu Anda mempersiapkan setiap animasi. Mulailah dengan menyeret folder Animations dari aset buku ke editor Unity. Anda akan bekerja dengan empat animasi: Idle, WalkForwardStraight, WalkForwardTurnRight, dan WalkForwardTurnLeft (meskipun folder Animations hanya berisi tiga file; lebih lanjut tentang itu segera). Masing-masing

animasi ini perlu diatur secara unik. Jika Anda melihat di folder Animations, Anda akan melihat bahwa animasi tersebut sebenarnya adalah file .fbx. Ini karena animasi itu sendiri terletak di dalam model default mereka. Namun, jangan khawatir: Anda akan dapat memodifikasi dan mengekstraknya di dalam Unity.

Animasi Idle

Untuk mengatur animasi Idle, ikuti langkah-langkah berikut (lihat Tabel 18.1 untuk penjelasan pengaturan):

1. Pilih file Idles.fbx di folder Animations. Di Inspector, pilih tab Rig. Ubah jenis animasi menjadi Humanoid dan klik Terapkan. Ini memberi tahu Unity bahwa animasi itu untuk humanoid.
2. Saat rig dikonfigurasi, klik tab Animations di Inspector. Atur bingkai awal ke 128 dan centang kotak di sebelah Waktu Loop dan Pose Loop. Selain itu, centang kotak Bake into Pose untuk semua properti Root Transform. Pastikan pengaturan Anda cocok dengan yang ada di Gambar 18.6 lalu klik Terapkan.
3. Untuk memastikan bahwa animasi sekarang dikonfigurasi dengan benar, perluas file Idles.fbx. Pastikan untuk mengingat cara mengakses animasi itu. (Modelnya tidak relevan. Ini adalah animasi yang Anda inginkan.)

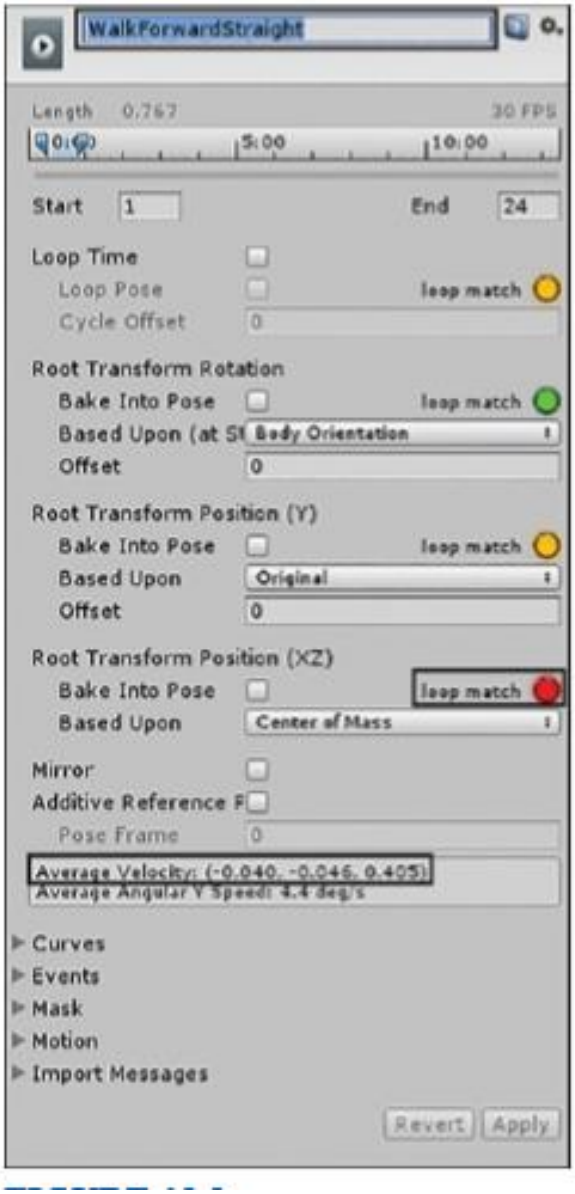
Lampu Merah, Lampu Hijau

Anda mungkin telah memperhatikan lingkaran hijau yang ada di pengaturan animasi. Itu adalah alat kecil yang bagus yang dapat Anda gunakan untuk menentukan apakah animasi Anda berbaris. Fakta bahwa lingkaran berwarna hijau berarti lingkaran itu akan berputar dengan mulus. Lingkaran berwarna kuning menunjukkan bahwa animasi mendekati loop dengan mulus, tetapi ada perbedaan kecil yang akan membuat sedikit jahitan. Lingkaran merah menunjukkan bahwa awal dan akhir animasi tidak sejajar sama sekali, dan jahitannya akan sangat terlihat. Jika Anda memiliki animasi yang tidak sejajar, Anda dapat mengubah properti Mulai dan Akhir untuk menemukan segmen animasi yang sesuai.

Animasi WalkForwardStraight

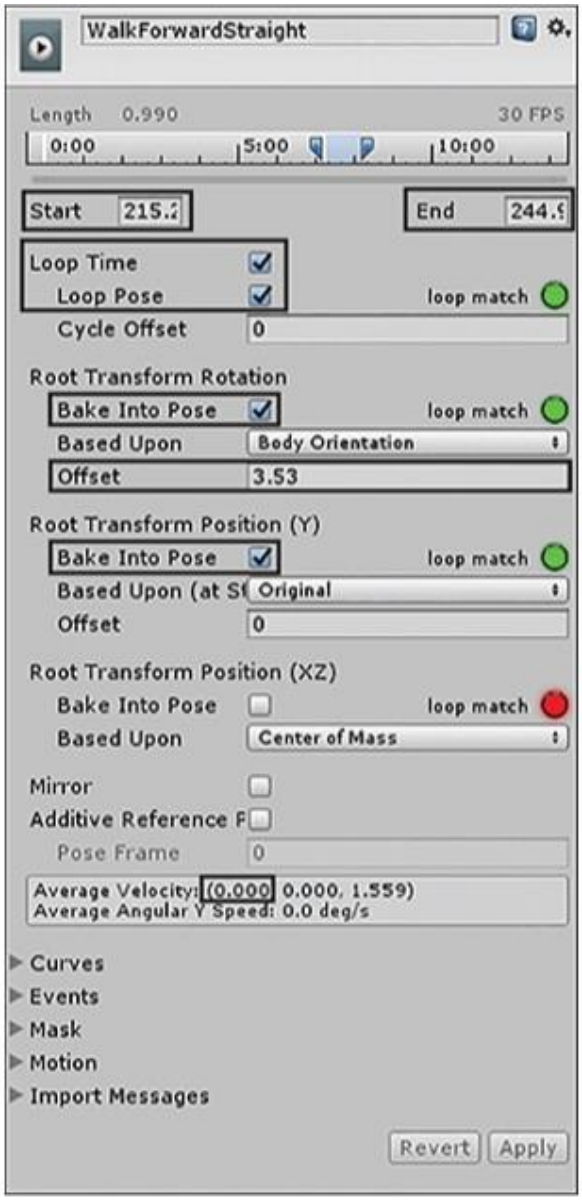
Untuk mengatur animasi WalkForwardStraight, ikuti langkah-langkah berikut:

1. Pilih file WalkForward.fbx di folder Animations dan selesaikan pemasangan dengan cara yang sama seperti yang Anda lakukan untuk animasi Idle.
2. Ubah nama klip yang saat ini sangat umum (Ambil 001) menjadi WalkForwardStraight dengan mengklik nama dan mengubahnya.
3. Pada tab Animations, pastikan pengaturan Anda sama dengan yang ditunjukkan pada Gambar 8.11. Anda harus mencatat dua hal. Pertama, Root Transform Position (XZ) memiliki lingkaran merah di sebelahnya. Ini bagus; artinya di akhir animasi, model berada pada posisi sumbu x dan z yang berbeda. Karena ini adalah animasi berjalan, itulah perilaku yang Anda inginkan. Hal lain yang harus Anda perhatikan adalah indikator Average Velocity. Anda harus memperhatikan kecepatan bukan nol pada sumbu x dan sumbu z. Kecepatan sumbu z bagus karena Anda ingin model bergerak maju, tetapi kecepatan sumbu x menjadi masalah karena akan menyebabkan model melayang ke samping saat berjalan. Anda akan menyesuaikan pengaturan ini di langkah



Gambar 8.11 Pengaturan animasi WalkForwardStraight

4. Untuk menyesuaikan kecepatan sumbu x, centang kotak Bake into Pose untuk properti Root Transform Rotation dan Root Transform Position (Y). Ubah juga Root Transform Rotation Offset sehingga nilai sumbu x dari Average Velocity menjadi 0.
5. Atur End frame menjadi 244.9 dan frame Start menjadi 215.2 (dalam urutan itu) sehingga animasi hanya berisi frame berjalan.
6. Terakhir, centang kotak Loop Time dan Loop Pose.
7. Pastikan bahwa pengaturan Anda sesuai dengan pengaturan akhir yang ditunjukkan pada Gambar 8.12 dan klik tombol Terapkan.

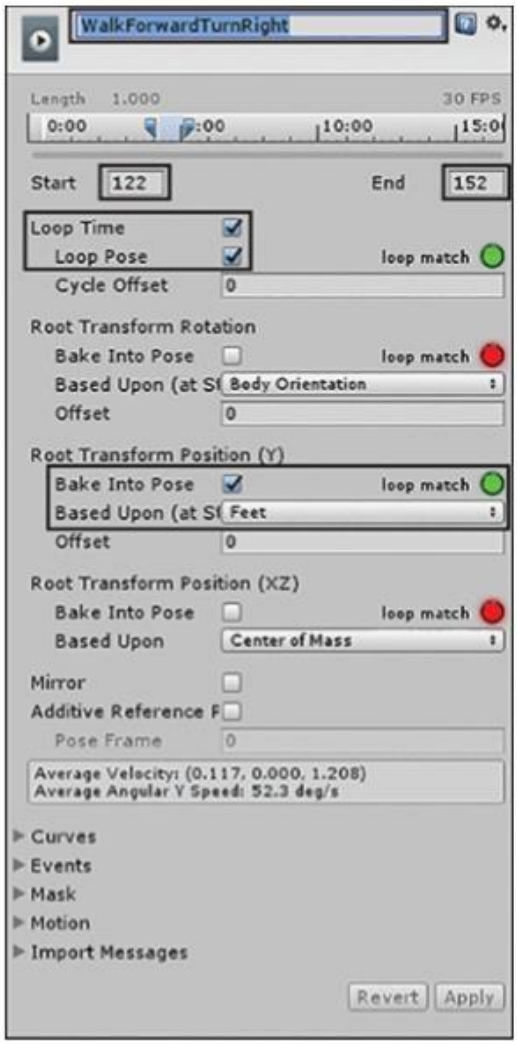


Gambar 8.12 Pengaturan animasi WalkForwardStraight tetap.

Animasi WalkForwardTurnRight

Animasi WalkForwardTurnRight memungkinkan model untuk mengubah arah dengan mulus saat berjalan ke depan. Yang ini sedikit berbeda dari dua yang telah Anda buat karena Anda perlu membuat dua animasi dari satu rekaman animasi. Ini terdengar lebih rumit daripada yang sebenarnya. Ikuti langkah ini:

1. Pilih file WalkForwardTurns.fbx di folder Animations dan selesaikan pemasangan dengan cara yang sama seperti yang Anda lakukan untuk animasi Idle.
2. Secara default, akan ada animasi panjang dengan nama 7a_U1_M_P_WalkForwardTurnRight. Ganti namanya dengan mengetik WalkForwardTurnRight ke dalam bidang teks Clip Name dan menekan tombol Enter.
3. Dengan klip WalkForwardTurnRight dipilih. Waktu Mulai dan Akhir yang lebih pendek akan memotong klip dan memastikan bahwa klip hanya berisi model yang bergerak dalam lingkaran ke kanan. (Pastikan untuk mempreviewnya untuk melihat seperti apa.) Setelah Anda melakukannya, klik Terapkan.



Gambar 8.13 Pengaturan WalkForwardTurnRight.

4. Buat klip animasi WalkForwardTurnLeft dengan mengklik ikon + di daftar Clips. Properti untuk klip WalkForwardTurnLeft akan sama persis dengan klip WalkForwardTurnRight kecuali bahwa Anda perlu memberi tanda centang pada properti Mirror. Ingatlah untuk mengklik Terapkan setelah Anda selesai dengan pengaturan. Pada titik ini, semua animasi sudah diatur dan siap digunakan. Sekarang yang tersisa untuk dilakukan adalah membangun animator.

Gambar 8.14 Mirroring Animasi

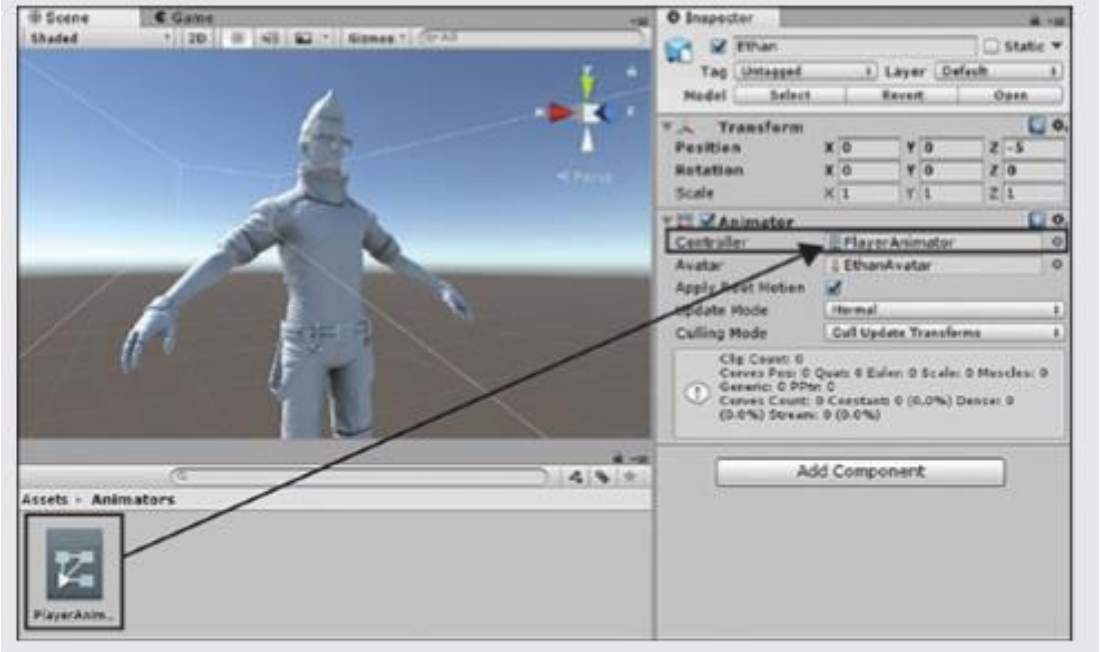
Membuat Animator Animator di Unity adalah aset. Ini berarti mereka adalah bagian dari proyek dan ada di luar satu scene. Ini bagus karena memungkinkan untuk digunakan kembali dengan mudah berulang kali. Untuk menambahkan animator ke proyek Anda, dalam tampilan Proyek Anda cukup klik kanan folder dan pilih Create > Animator Controller (tapi jangan lakukan itu dulu).

Menyiapkan Scene

Dalam latihan ini, Anda akan mengatur scene dan bersiap untuk sisa jam ini. Pastikan untuk menyimpan scene yang dibuat di sini karena Anda akan membutuhkannya nanti. Ikuti langkah ini:

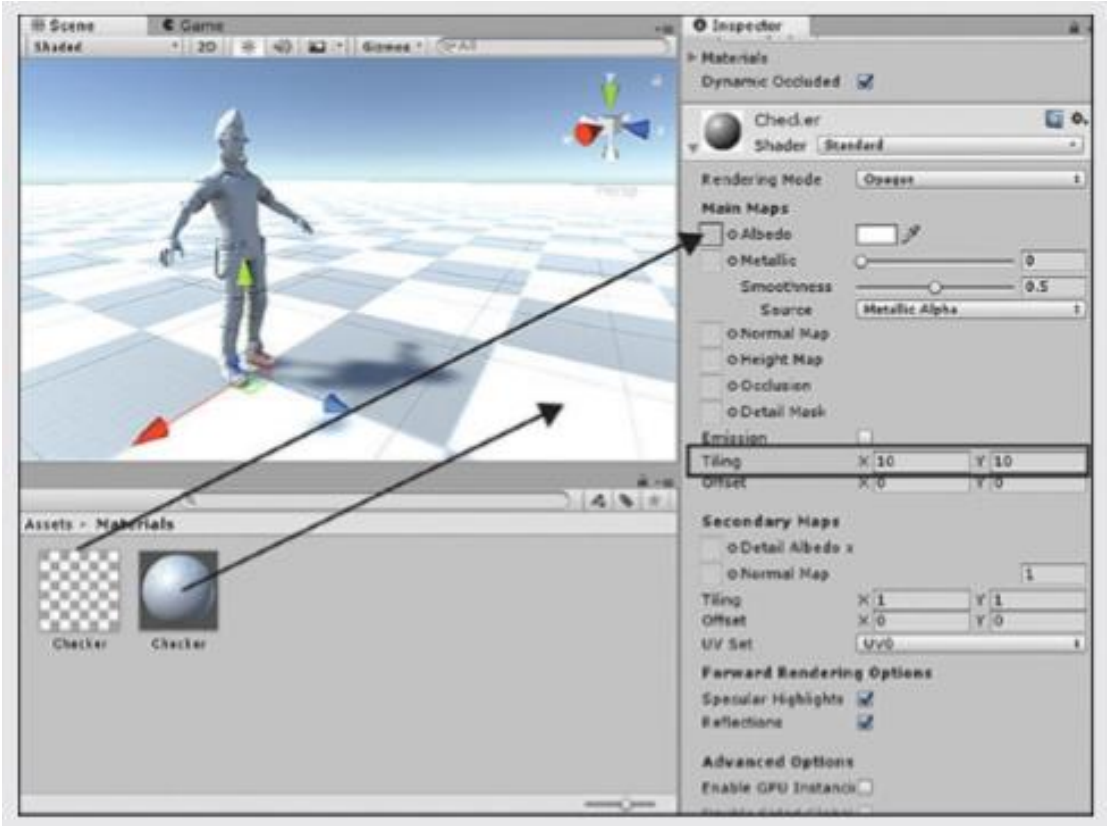
1. Jika Anda belum melakukannya, buat new project dan selesaikan langkah-langkah persiapan model dan animasi di bagian sebelumnya.
2. Drag model Ethan ke dalam scene Anda (dari Assets\Standard Assets\Characters \ThirdPersonCharacter\Models) dan berikan posisinya (0, 0, -5).
3. Letakkan Kamera Utama di bawah Ethan (dengan menyeret Kamera Utama ke objek game Ethan dalam tampilan Hierarchy) dan posisikan kamera pada (0, 1.5, -1.5) dengan rotasi (20, 0, 0).
4. Dalam tampilan Proyek Anda, buat folder baru bernama Animator. Klik kanan folder baru dan pilih Create > Animator Controller. Beri nama animator PlayerAnimator.

Dengan Ethan dipilih dalam scene, drag animator ke properti Controller dari komponen Animator di Inspector



Gambar 8.15 Menambahkan animator ke model.

5. Tambahkan pesawat ke scene Anda. Posisikan bidang pada (0, 0, -5) dengan scale (10, 1, 10).
6. Cari file Checker.tga di aset buku untuk Jam 18 dan impor ke proyek Anda. Buat materi baru bernama Checker dan atur Checker.tga sebagai albedo untuk materi

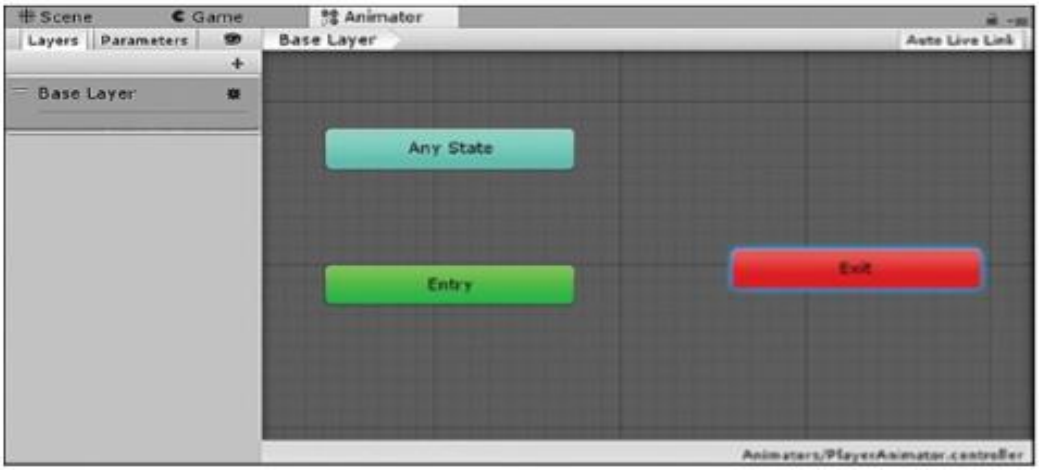


Gambar 8.16 Mengatur ubin tekstur checker.

8. Atur properti X dan Y Tiling ke 10 dan terapkan material ke bidang. (Pesawat tidak terlalu berguna sekarang, tetapi akan menjadi penting nanti di jam ini.)

Tampilan Animator

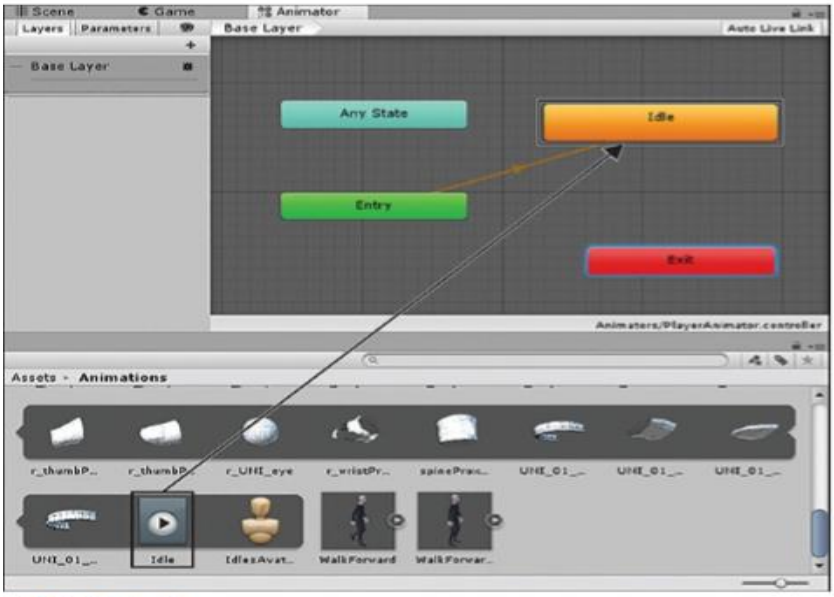
Mengklik dua kali seorang animator akan menampilkan tampilan Animator (yang juga dapat Anda buka dengan memilih Window > Animator). Tampilan ini berfungsi seperti grafik aliran, memungkinkan Anda membuat jalur dan pencampuran animasi secara visual. Ini adalah kekuatan sebenarnya dari sistem Mecanim. Anda dapat bergerak di sekitar tampilan Animator dengan menyeret dengan menekan tombol tengah mouse, dan Anda dapat memperbesar dengan roda gulir. Animator baru ini sangat sederhana: hanya memiliki layer dasar, tidak ada parameter, node Masuk dan Keluar, dan node Any State. (Komponen ini dibahas secara lebih rinci nanti di jam ini.)



Gambar 8.17 Tampilan Animator.

Animasi Idle

Animasi pertama yang ingin Anda terapkan ke Ethan adalah animasi Idle. Anda telah menyelesaikan proses penyiapan yang panjang sebelumnya, dan sekarang, menambahkan animasi ini menjadi mudah. Anda perlu menemukan klip animasi Idle, yang disimpan di dalam file Idles.fbx (lihat Gambar 8.18, di awal jam), dan drag ke animator dalam tampilan Animator

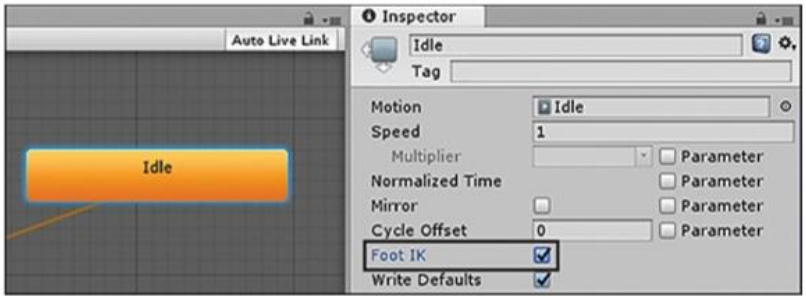


Gambar 8.18 Menerapkan animasi Idle.

Anda sekarang harus dapat menjalankan scene Anda dan melihat model Ethan berulang melalui animasi Idle.

Slip dan Slide

Saat Anda menjalankan scene untuk melihat model Ethan memainkan animasi idle, Anda mungkin melihat kaki model meluncur di tanah. Ini karena bagaimana animasi khusus ini dibuat. Dalam animasi ini, karakter menentukan gerakannya berdasarkan pinggulnya sebagai lawan dari kakinya (menyebabkannya berputar sedikit seperti kincir kincir). Anda dapat memperbaikinya di tampilan Animator. Cukup pilih status animasi Idle, dan dalam tampilan Inspector, pilih kotak centang Foot IK (lihat Gambar 8.19). Model sekarang mencoba untuk melacak kakinya ke tanah. Ini menyebabkan karakter bernyawa dengan benar, dengan kaki yang tertanam kuat. Omong-omong, IK adalah singkatan dari Inverse Kinematika; rincian seperti itu sebagian besar berada di luar cakupan teks ini.



Gambar 8.19 Memilih Kaki IK.

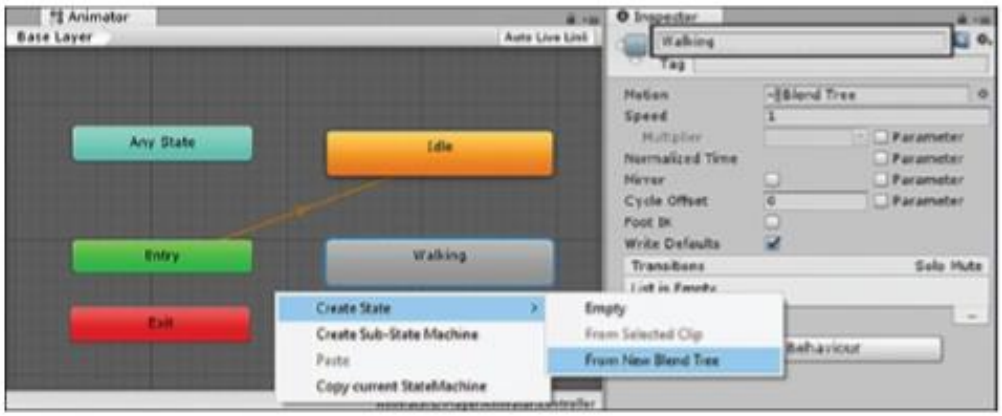
Parameter

Parameter seperti variabel untuk seorang animator. Anda mengaturnya di tampilan Animator dan kemudian memanipulasinya dengan skrip. Parameter ini mengontrol kapan animasi ditransisikan dan digabungkan. Untuk membuat parameter, cukup klik + di tab Parameter di tampilan Animator.

8.6 STATES AND BLEND TREES

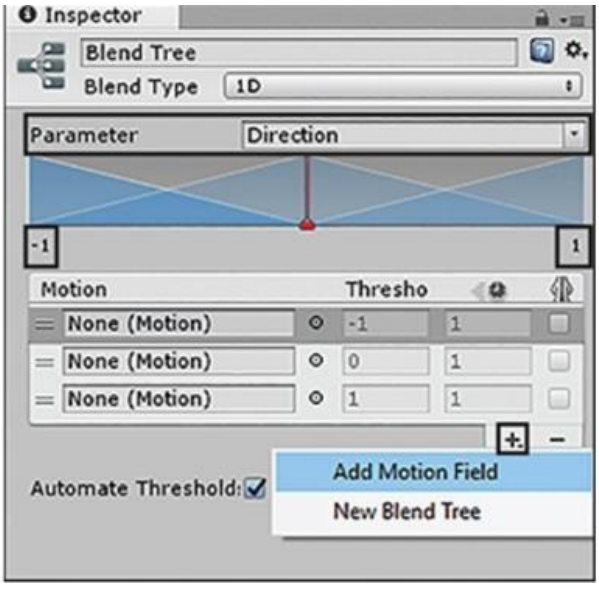
Langkah Anda selanjutnya adalah membuat status baru. Status pada dasarnya adalah status model saat ini yang mendefinisikan animasi apa yang sedang diputar. Anda membuat status sebelumnya, saat Anda menambahkan animasi Idle ke controller Animator. Model Ethan akan memiliki dua status: Idle dan Walking. Idle sudah ada di tempatnya. Karena status Berjalan dapat berupa salah satu dari tiga animasi, Anda ingin membuat status yang menggunakan pohon campuran, yang memadukan satu atau beberapa animasi dengan mulus, berdasarkan beberapa parameter. Untuk membuat status baru, ikuti langkah berikut:

1. Klik kanan tempat kosong di tampilan Animator dan pilih Create State > From New Blend Tree. Dalam tampilan Inspector, beri nama negara bagian baru Berjalan



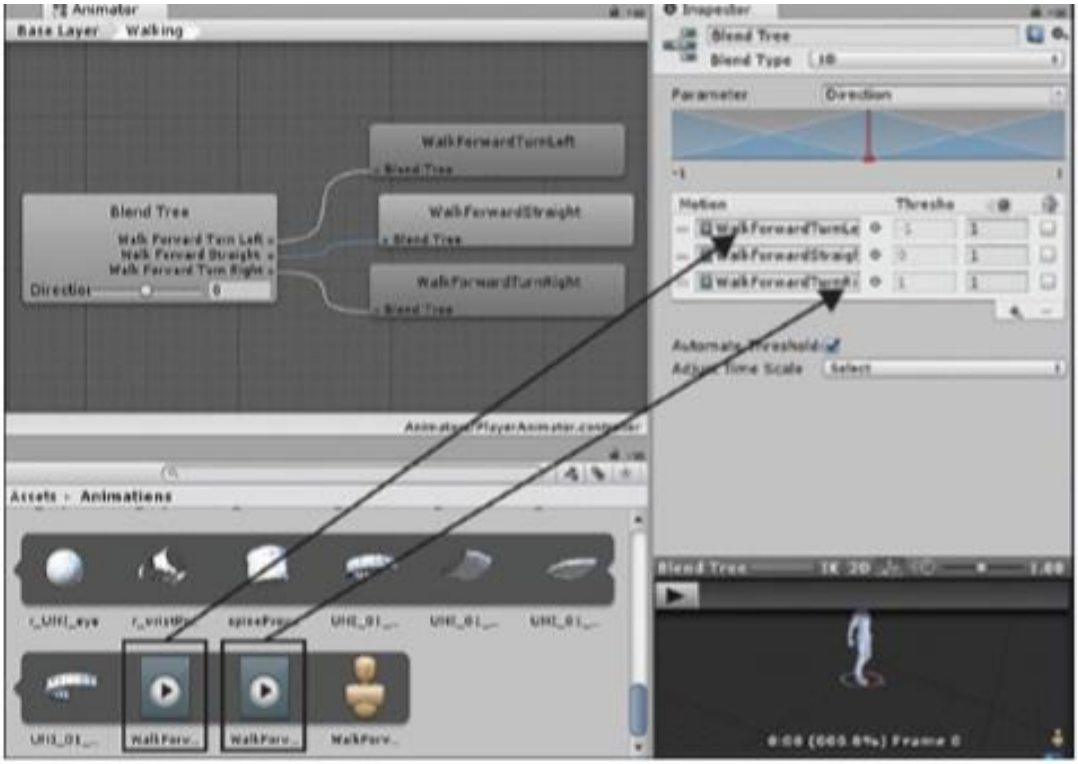
Gambar 8.20 Membuat dan menamai negara baru.

2. Klik dua kali status baru untuk memperluasnya dan pilih status Blend Tree baru. Di Inspector, klik drop-down properti Parameter dan ubah ke Direction. Kemudian tambahkan tiga gerakan dengan mengklik + di bawah gerakan dan pilih Tambahkan Bidang Gerakan. Di bawah grafik, atur nilai minimum ke -1 dan nilai maksimum ke 1



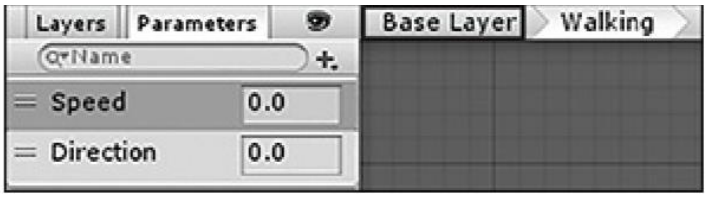
Gambar 8.21 Menambahkan bidang gerak.

3. Drag masing-masing dari tiga animasi berjalan ke salah satu dari tiga bidang gerak, dalam urutan ini: WalkForwardTurnLeft, WalkForwardStraight, WalkForwardTurnRight (lihat Gambar 8.22). Ingat bahwa klip animasi berputar terletak di bawah WalkForwardTurns.fbx, dan animasi berjalan lurus berada di bawah WalkForward.fbx.



Gambar 8.22 Mengubah nilai minimum dan menambahkan animasi ke pohon campuran.

Animasi berjalan Anda sekarang siap untuk berbaur, berdasarkan parameter Arah. Pada dasarnya, Anda memberi tahu keadaan campuran untuk mengevaluasi parameter Arah. Berdasarkan nilai parameter tersebut, pohon campuran akan memilih beberapa jumlah persentase dari setiap animasi untuk dipadukan untuk memberi Anda animasi unik akhir. Misalnya, jika Arah sama dengan -1, pohon campuran memainkan 100% animasi WalkForwardTurnLeft. Jika Arah sama dengan .5, pohon campuran memainkan 50% animasi WalkForwardStraight yang dipadukan dengan 50% animasi WalkForwardTurnRight. Anda dapat dengan mudah melihat betapa kuatnya pohon campuran! Untuk keluar dari tampilan yang diperluas, klik remah roti Layer Dasar di bagian atas tampilan Animator.



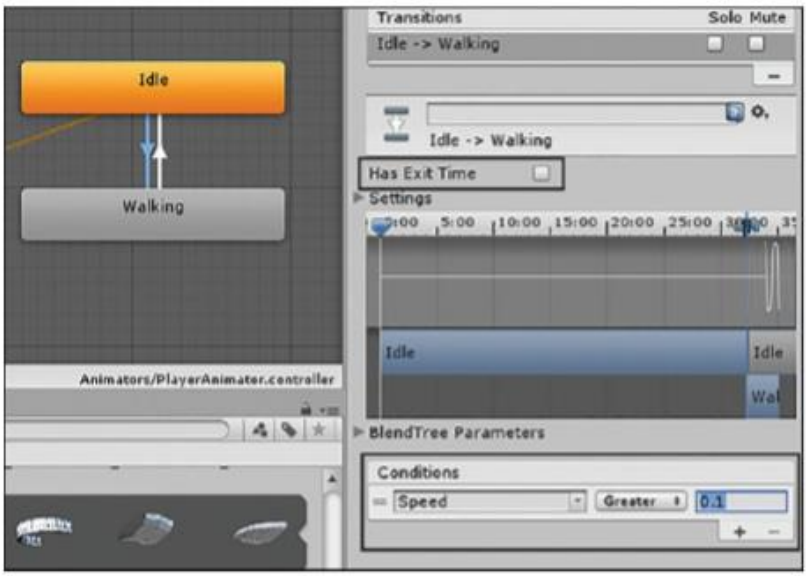
Gambar 8.23 Menavigasi tampilan Animator.

Transisi

Hal terakhir yang perlu Anda lakukan untuk memastikan bahwa animator Anda telah selesai adalah memberi tahu animator cara melakukan transisi antara animasi Idle dan Walking. Anda perlu menyiapkan dua transisi. Salah satunya mentransisikan animator dari idle ke berjalan, dan transisi lainnya kembali. Untuk membuat transisi, ikuti langkah-langkah berikut:

1. Klik kanan status Idle dan pilih Make Transition untuk membuat garis putih yang mengikuti mouse Anda. Klik status Berjalan untuk menghubungkannya ke status Idle.
2. Ulangi langkah 1, kecuali kali ini hubungkan status Berjalan ke status Idle.

3. Edit transisi Idle to Walking dengan mengklik panah putih di atasnya. Tambahkan kondisi dan atur menjadi Kecepatan Lebih Besar dari nilai .1 (lihat Gambar 8.24). Lakukan hal yang sama untuk transisi Walking to Idle, kecuali atur kondisinya ke Speed Less Than nilai .1.
4. Hapus centang pada kotak Has Exit Time untuk memungkinkan animasi Idle diinterupsi saat tombol berjalan ditekan.



Gambar 8.24 Memodifikasi transisi.

Animator selesai. Anda mungkin memperhatikan bahwa saat menjalankan scene, tidak ada animasi gerakan yang berfungsi. Ini karena parameter kecepatan dan arah tidak pernah berubah. Di bagian selanjutnya, Anda akan mempelajari cara mengubahnya melalui skrip.

Scripting Animator

Sekarang semuanya telah diatur dengan model, rigging, animasi, animator, transisi, dan pohon campuran, akhirnya saatnya untuk membuat semuanya menjadi interaktif. Untungnya, komponen skrip yang sebenarnya sederhana. Sebagian besar kerja keras sudah dilakukan di editor. Pada titik ini, yang perlu Anda lakukan hanyalah memanipulasi parameter yang Anda buat di animator untuk mengaktifkan dan menjalankan Ethan. Karena parameter yang Anda atur bertipe float, Anda perlu memanggil metode animator:

8.7 TIMELINE

Dalam jam ini, Anda akan melihat alat pengurutan yang sangat kuat di Unity: Timeline. Anda akan mulai dengan melihat struktur dan konsep timeline dan sutradara. Dari sana Anda akan menjelajahi konsep klip dan bagaimana klip tersebut dapat diurutkan pada timeline. Terakhir, Anda akan menyelesaikan jam ini dengan melihat lebih banyak cara Anda dapat menggunakan Timeline untuk menghadirkan kecanggihan dan kerumitan pada proyek Anda.

Misterinya Dilampaui Hanya dengan Kekuatannya

Sepanjang jam ini, Anda akan melihat berbagai bagian sistem Timeline di Unity. Salah satu hal yang perlu diperhatikan tentang Timeline adalah bahwa itu adalah alat yang sangat mumpuni dan sangat kompleks. Sebenarnya, jam ini hanya menggores permukaan dari apa yang mungkin dengan sequencer ini. Seluruh sistem dibangun dari potongan-potongan sederhana namun modular, yang memungkinkannya untuk digunakan dalam banyak cara

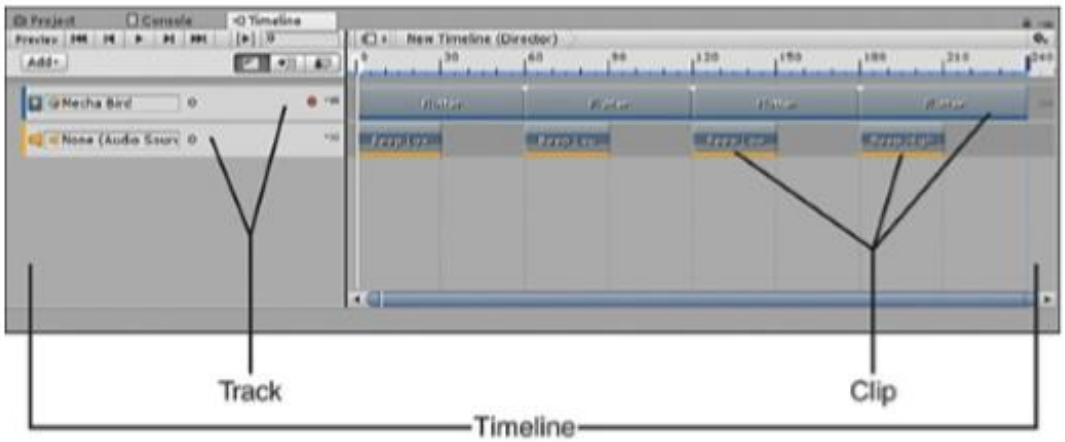
khusus dan menarik (seperti mainan konstruksi plastik yang saling terkait yang tidak boleh disebutkan namanya). Inti dari catatan ini adalah untuk memberi tahu Anda bahwa jika Anda pernah bertanya-tanya, "Hmm, saya ingin tahu apakah saya bisa melakukan ini dengan Timeline," jawabannya mungkin ya. Ini mungkin tidak selalu mudah, tetapi mungkin saja!

Dasar-dasar Timeline

Pada intinya, Timeline adalah alat pengurutan. Ini berarti dapat digunakan untuk menyebabkan hal-hal terjadi pada waktu tertentu dalam kaitannya satu sama lain. Apa hal-hal itu benar-benar terserah Anda. Di satu sisi, timeline sangat mirip dengan controller Animator (lihat Jam 18, “Animator,” untuk penyegaran). Kontroler Animator digunakan untuk mengurutkan dan mengontrol animasi mana yang dimainkan pada suatu objek. Batasannya, bagaimanapun, adalah bahwa controller Animator hanya dapat mengontrol dirinya sendiri atau objek anak. Itu tidak dapat digunakan, misalnya, untuk membuat sinematik di mana dua penjaga berbicara satu sama lain sementara seorang pencuri menyelinap di belakang mereka dalam bayang-bayang. Ini persis jenis pekerjaan yang Timeline ingin lakukan. Anda dapat menggunakannya untuk mengurutkan banyak objek berbeda yang melakukan banyak hal berbeda pada banyak waktu berbeda.

Anatomi Timeline

Elemen inti dari item yang diurutkan disebut klip. Meskipun ini akan menunjukkan bahwa Timeline digunakan untuk animasi, kenyataannya adalah bahwa klip dapat berupa apa saja mulai dari klip audio, hingga trek kontrol, hingga peristiwa data khusus, atau bahkan mengaktifkan dan menonaktifkan objek game. Anda bahkan dapat memprogram klip Anda sendiri dan membuatnya melakukan apa pun yang Anda inginkan. Anda menempatkan klip pada satu atau lebih trek (lihat Gambar 8.25). Trek menentukan jenis klip apa yang dapat ditempatkan di atasnya dan objek apa yang dikontrolnya. Untuk menganimasikan dua karakter secara berurutan, misalnya, Anda memerlukan dua trek animasi (satu untuk setiap karakter), dengan satu atau lebih animasi di masing-masingnya.



Gambar 8.25 Anatomi timeline.

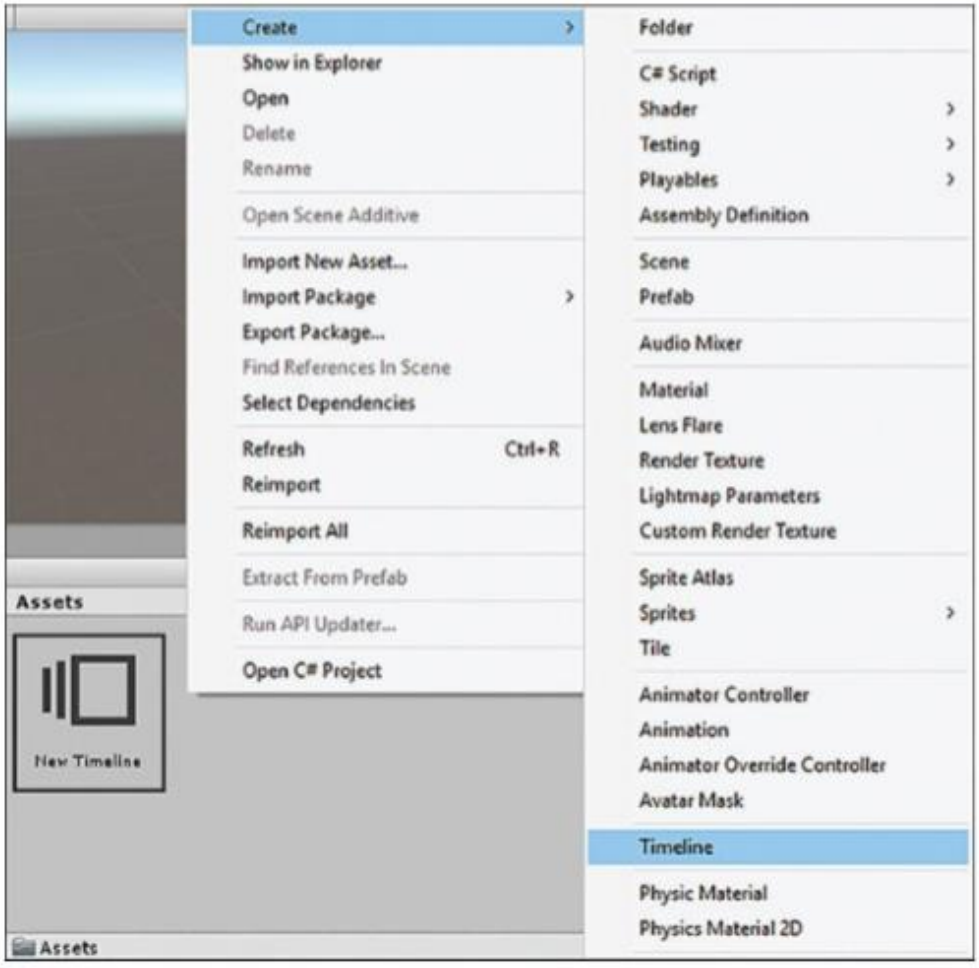
Baik klip maupun trek berada pada aset timeline. Aset ini dapat digunakan kembali di antara scene atau bahkan dapat memiliki beberapa instans dalam satu scene. Timeline melacak objek yang dikontrolnya melalui binding. Binding ini dapat diatur melalui kode, tetapi biasanya Anda hanya menyeret objek yang ingin Anda kontrol ke timeline. Terakhir, sistem Timeline menggunakan komponen Playable Director untuk mengontrol timeline. Komponen

ini ditambahkan ke objek game dalam sebuah scene dan menentukan kapan timeline diputar, kapan berhenti, dan apa yang terjadi saat selesai.

Baik klip maupun trek berada pada aset timeline (digambarkan pada Gambar 19.1). Aset ini dapat digunakan kembali di antara scene atau bahkan dapat memiliki beberapa instans dalam satu scene. Timeline melacak objek yang dikontrolnya melalui binding. Binding ini dapat diatur melalui kode, tetapi biasanya Anda hanya menyeret objek yang ingin Anda kontrol ke timeline. Terakhir, sistem Timeline menggunakan komponen Playable Director untuk mengontrol timeline. Komponen ini ditambahkan ke objek game dalam sebuah scene dan menentukan kapan timeline diputar, kapan berhenti, dan apa yang terjadi saat selesai.

Membuat Timeline

Seperti yang disebutkan sebelumnya dalam jam ini, aset timeline adalah bagian yang cocok dengan semua bagian lainnya. Jadi, langkah pertama dalam bekerja dengan Timeline adalah membuat aset timeline. Untuk melakukan ini, klik kanan pada tampilan Proyek dan pilih Buat > Timeline



Gambar 8.26 Membuat Timeline

Setelah timeline dibuat, itu perlu dikontrol oleh komponen Direktur yang Dapat Diputar, yang ditambahkan ke objek game di scene. Untuk menambahkan komponen Direktur yang Dapat Diputar ke objek game, Anda cukup memilihnya lalu klik Tambahkan Komponen > Dapat Dimainkan > Direktur yang Dapat Dimainkan. Kemudian Anda perlu mengatur aset timeline sebagai properti Playable dari komponen Playable Director. Cara yang lebih mudah

untuk menyelesaikan kedua langkah ini sekaligus adalah dengan menyeret aset timeline dari tampilan Proyek ke objek permainan dalam tampilan Hierarki yang ingin Anda kendalikan timeline. Menambahkan komponen Direktur yang Dapat Diputar ke objek game.

Membuat Aset Timeline

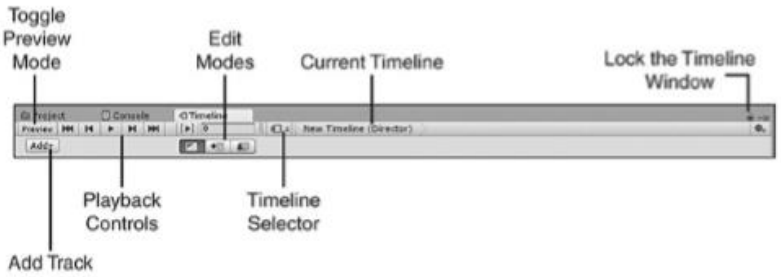
Luangkan waktu sejenak untuk membuat aset timeline dan menambahkan komponen Direktur yang Dapat Diputar ke objek game. Pastikan untuk menyimpan proyek dan scene yang Anda buat dalam latihan ini sehingga Anda dapat menggunakannya nanti di jam ini. Ikuti langkah-langkah berikut: 1. Buat proyek atau scene baru. Tambahkan folder baru bernama Timelines ke proyek Anda. 2. Dalam folder Timelines, buat timeline baru dengan mengklik kanan dan memilih Create > Timeline. 3. Tambahkan objek game baru ke scene Anda (dengan memilih GameObject > Create > Create Empty). Beri nama objek baru Director. 4. Drag aset timeline baru Anda dari tampilan Proyek ke objek permainan Direktur dalam tampilan Hierarki. (Ini tidak akan berfungsi jika Anda mencoba menyeretnya ke tampilan Scene atau tampilan Inspector.) 5. Pilih objek game Director dan pastikan bahwa komponen Playable Director muncul di tampilan Inspector.

Bekerja dengan Timeline

Membuat timeline adalah proses yang cukup sederhana, tetapi timeline tidak benar-benar mencapai apa pun dengan sendirinya. Untuk menggunakan timeline, Anda perlu membuat trek dan klip yang mengontrol dan mengurutkan objek dalam scene Anda. Semua pekerjaan ini dilakukan melalui jendela Timeline, yang sangat mirip dengan jendela Animation yang dieksplorasi di Hour 17, "Animasi."

Jendela Timeline

Untuk melihat dan bekerja dengan timeline, Anda perlu membuka jendela Timeline. Anda dapat melakukannya dengan mengklik Window > Timeline atau cukup mengklik dua kali aset timeline di tampilan Project. Jendela ini menampilkan kontrol untuk preview dan pemutaran, kontrol mode, dan area yang luas untuk bekerja dengan trek dan klip



Gambar 8.27 Jendela Timeline.

Tidak ada gunanya memeriksa jendela Timeline sekarang, karena agak kosong. Sebagai gantinya, mari kita lanjutkan, dan Anda dapat mempelajari lebih lanjut tentang jendela Timeline saat itu berlaku.

Mengunci Jendela Timeline

Saat bekerja di dalam jendela Timeline, Anda akan sering mengklik objek lain di jendela Scene atau Project. Namun, melakukannya, membatalkan pilihan objek game Director Anda dan menyebabkan jendela Timeline menjadi kosong. Ini bisa sangat membuat frustrasi. Cara yang lebih baik untuk bekerja dengan jendela Timeline adalah dengan menguncinya (lihat Gambar 8.27). Ini mencegah pilihan Anda saat ini mengubah apa yang menjadi fokus jendela

Timeline. Kemudian Anda dapat memilih timeline mana yang ingin Anda ubah dengan menggunakan pemilih timeline di jendela Timeline. Jauh lebih mudah!

Mode Preview Jendela

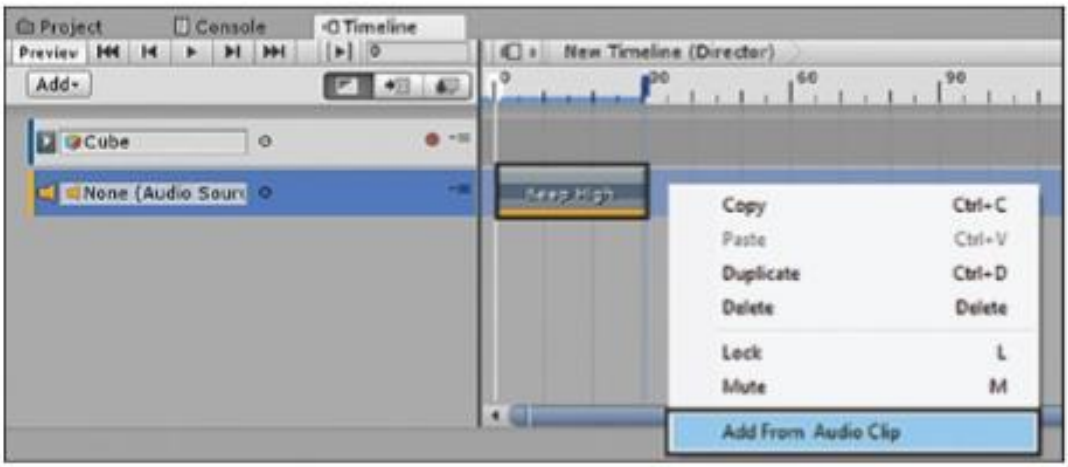
Timeline memiliki mode Preview, Saat mode ini diaktifkan, Anda dapat melihat bagaimana timeline akan memengaruhi objek dalam scene Anda. Ini sangat penting untuk mengurutkan dan memastikan bahwa semuanya berbaris. Untungnya, bekerja di jendela Timeline secara otomatis menempatkan Anda dalam mode Preview. Namun, saat dalam mode Preview, beberapa fungsi dinonaktifkan. Salah satu contohnya adalah Anda tidak dapat menerapkan perubahan Prefab apa pun. Jika Anda melihat ada yang tidak beres, coba keluar dari mode Preview untuk memastikan bahwa jendela Timeline tidak mempermainkan Anda.

Trek Timeline

Trek pada timeline menentukan hal-hal apa yang dapat dilakukan, objek apa yang melakukan hal-hal itu, dan hal-hal apa yang dilakukan objek. Intinya, trek timeline melakukan banyak pekerjaan untuk Anda. Untuk menambahkan trek ke timeline, klik tombol Add di jendela Timeline (lihat Gambar 8.28). Tabel 8.1 mencantumkan jenis trek bawaan dan menjelaskan apa yang mereka lakukan.

Klip Timeline

Setelah Anda memiliki trek di timeline Anda, Anda perlu menambahkan klip. Cara klip berperilaku sedikit berbeda tergantung pada jenis trek yang Anda miliki, tetapi fungsi umum semua klip adalah sama. Untuk menambahkan klip ke trek, cukup klik kanan trek dan pilih Tambah Dari <jenis klip>. Jadi, misalnya, jika Anda menambahkan klip ke trek audio, menu akan mengatakan Tambahkan dari Klip Audio. Menu muncul, memungkinkan Anda memilih klip untuk ditambahkan ke trek. Atau, Anda dapat menambahkan klip ke trek dengan menyeret aset yang relevan dari tampilan Proyek ke trek di jendela Timeline.



Gambar 8.28 Menambahkan klip yang disebut Beep High ke trek audio.

Setelah ada klip di trek, Anda dapat memindahkannya untuk mengontrol saat diputar. Anda juga dapat mengubah ukuran klip untuk mengubah durasi pemutarannya. Anda bahkan dapat menggunakan durasi klip untuk memotong atau mengulang animasi selama periode waktu tertentu. Perhatikan bahwa perilaku yang tepat dari durasi klip bergantung pada jenis trek. Selain menyeret klip di sekitar trek untuk menyesuaikan pengaturan pemutarannya, Anda juga dapat memilih klip dan kemudian mengubah pengaturan dalam tampilan Inspector.

Ini memberi Anda tingkat kontrol dan penyetelan yang lebih baik saat bekerja dengan klip (meskipun tidak secepat hanya dengan mengklik dan menyeret).

8.8 FAST ANIMATIONS

Animation track bekerja sangat baik dengan animasi yang diimpor dan animasi yang dibuat di Unity menggunakan jendela Animation. Namun, jika Anda ingin menganimasikan objek dengan cepat untuk digunakan dengan timeline, ada cara yang lebih mudah. Dengan objek game yang terikat pada trek animasi, Anda dapat mengklik tombol Rekam secara langsung pada trek itu sendiri (lihat Gambar 8.29). Ini berfungsi identik dengan mode Rekam dari jendela Animasi (terlihat pada Jam 17). Satu-satunya perbedaan adalah bahwa dalam kasus ini, alih-alih membuat klip animasi baru yang terpisah, data animasi disimpan langsung di aset timeline. Dengan menggunakan metode ini, Anda dapat dengan cepat menghasilkan animasi sederhana yang menambahkan banyak kedalaman pada sinematik Anda.

Gambar 8.29 Merekam dalam jendela Timeline

Mematikan dan Mengunci Setiap trek di jendela

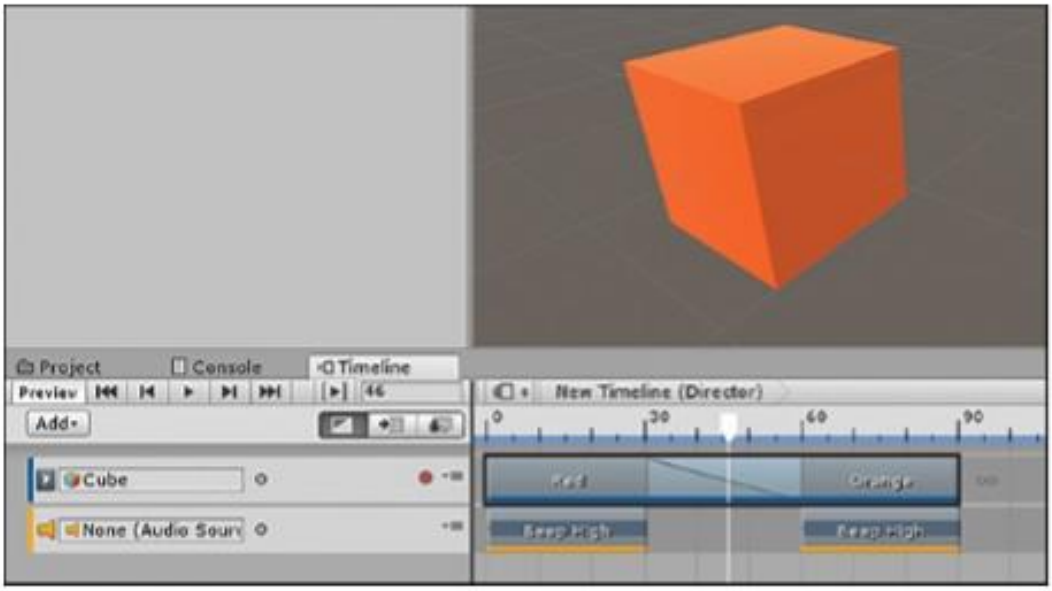
Timeline dapat dikunci untuk mencegah perubahan yang tidak disengaja. Cukup pilih trek dan tekan L atau klik kanan dan pilih Kunci. Selain itu, trek dapat dibungkam sehingga tidak diputar saat timeline diputar. Untuk membisukan trek, pilih trek dan tekan M atau klik kanan dan pilih Bungkam.

Melampaui Kontrol Sederhana

Anda baru saja mulai menggores permukaan dari apa yang dapat Anda lakukan dengan sistem Timeline. Jelas, sistem ini sangat bagus untuk sinematik, tetapi juga dapat digunakan untuk membuat banyak jenis perilaku yang kaya. Beberapa contoh adalah mengontrol gerakan penjaga, membuat kerumunan terlihat lebih hidup, atau menerapkan serangkaian efek layar yang kompleks saat karakter menerima kerusakan.

Blending Klip pada Trek

Sejauh ini, Anda telah melihat bekerja dengan klip sebagai item individual di trek. Itu tidak harus terjadi, meskipun. Anda benar-benar dapat menggunakan timeline untuk memadukan dua klip berbeda untuk mendapatkan hasil gabungan yang baru. Melakukannya hanya mengharuskan Anda menyeret satu klip ke klip lainnya. Gambar 8.9 menunjukkan hasil pencampuran klip Merah dan Oranye dari “Sequencing Clips” Coba Sendiri.



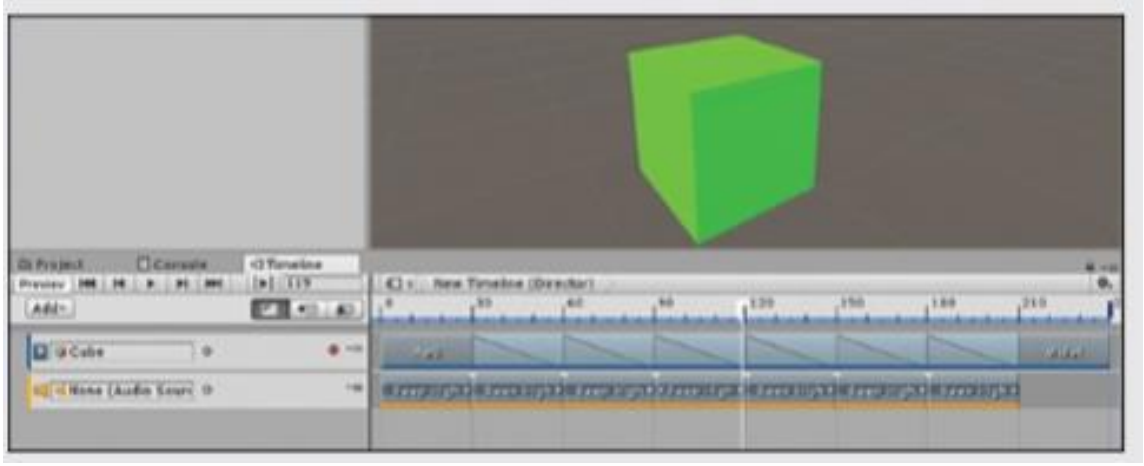
Gambar 8.30 Blending klip di jendela Timeline.

Pencampuran berfungsi untuk lebih dari sekadar klip animasi. Anda juga dapat memadukan trek audio dan banyak trek khusus yang dibuat oleh komunitas Unity. Mampu memadukan klip memungkinkan tingkat kontrol dan "tweening" tindakan halus di antara bingkai utama.

Blending Clip

Langkah-langkah berikut menunjukkan kepada Anda bagaimana memadukan klip yang Anda tambahkan di "Sequencing Clips" Coba Sendiri untuk menciptakan transisi warna yang mulus melintasi pelangi. Pastikan untuk menyimpan scene ini lagi karena Anda akan terus menggunakannya nanti di jam ini. Ikuti langkah-langkah ini:

1. Buka scene yang Anda buat di "Sequencing Clips" Coba Sendiri. Pastikan jendela Timeline terbuka dan objek game Director dipilih.
2. Di jendela Timeline, drag klip Oranye sehingga menutupi separuh klip Merah. Dalam tampilan Inspector, klip Oranye harus dimulai pada bingkai 30 dan berakhir pada bingkai 90.
3. Lanjutkan memadukan klip warna ke kiri hingga timeline terdiri dari campuran yang hampir berkesinambungan



Gambar 8.31 Memadukan semua klip warna.

4. Geser klip audio sehingga masing-masing cocok dengan awal animasi warna. Isyarat suara ini akan membantu mengidentifikasi setiap warna saat diterapkan.

Scripting dengan Timeline

Untuk sebagian besar, kode yang diperlukan untuk membangun trek dan klip yang dapat diputar kustom cukup kompleks dan di luar cakupan teks ini. Namun, sesuatu yang sederhana yang dapat Anda lakukan adalah memberi tahu timeline kapan harus dijalankan alih-alih membiarkannya berjalan secara otomatis saat scene pertama kali dimulai. Ini memungkinkan Anda untuk memicu sinematik atau event dalam game. Untuk menulis kode yang berfungsi dengan sistem Timeline, Anda perlu memberi tahu Unity untuk menggunakan perpustakaan Playables:

```
Using UnityEngine.Playables;
```

Kemudian Anda dapat membuat variabel tipe PlayableDirector dan menggunakannya untuk mengontrol timeline. Dua metode utama yang digunakan adalah Play() dan Stop():

```
PlayableDirector director = GetComponent<PlayableDirector>();
director.Play(); // Start a timeline
director.Stop(); // Stop a timeline
```

Hal lain yang dapat Anda lakukan adalah memberi tahu sutradara apa yang dapat dimainkan untuk dimainkan saat runtime. Salah satu kasus penggunaan potensial untuk ini adalah memiliki kumpulan timeline yang dipilih secara acak atau untuk menentukan timeline apa yang akan dimainkan tergantung pada hasil permainan:

```
public PlayableAsset newTimeline;
void SomeMethod()
{
director.Play(newTimeline);
}
```

Dengan menggunakan pemanggilan metode sederhana ini, Anda dapat menyediakan sebagian besar fungsionalitas yang Anda butuhkan dari sistem Timeline saat runtime.

BAB 9
MEMUTAR AUDIO: SOUND EFFECT DAN MUSIK

Meskipun grafik mendapatkan sebagian besar perhatian dalam hal konten dalam video game, audio juga sangat penting. Sebagian besar game memutar musik latar dan memiliki sound effect. Dengan demikian, Unity memiliki fungsionalitas audio sehingga Anda dapat memasukkan sound effect dan musik ke dalam game Anda. Unity dapat mengimpor dan memutar berbagai format file audio, menyesuaikan volume suara, dan bahkan menangani suara yang diputar dari posisi tertentu dalam scene.

Bab ini dimulai dengan sound effect daripada musik. Sound effect adalah klip pendek yang diputar bersama dengan tindakan dalam game (seperti tembakan yang diputar saat player menembak), sedangkan klip suara untuk musik lebih panjang (sering kali berlangsung dalam hitungan menit) dan pemutaran tidak terkait langsung dengan event di permainan. Pada akhirnya, keduanya bermuara pada jenis file audio dan kode pemutaran yang sama, tetapi fakta sederhana bahwa file suara untuk musik biasanya jauh lebih besar daripada klip pendek yang digunakan untuk sound effect (memang, file untuk musik sering kali merupakan file terbesar di permainan!) manfaat membahasnya di bagian terpisah.

Peta jalan lengkap untuk bab ini adalah mengambil game tanpa suara dan melakukan hal berikut:

1. Impor file audio untuk sound effect.
2. Mainkan sound effect untuk musuh dan untuk menembak.
3. Program pengelola audio untuk mengontrol volume.
4. Optimalkan pemuatan musik.
5. Kontrol volume musik secara terpisah dari sound effect, termasuk trek crossfading.

Bab ini sebagian besar tidak tergantung pada proyek yang Anda bangun; itu hanya menambahkan kemampuan audio di atas demo game yang ada. Semua contoh dalam bab ini dibangun di atas FPS dan Anda dapat mengunduh proyek sampel itu, tetapi Anda bebas menggunakan demo game apa pun yang Anda inginkan. Setelah Anda memiliki demo game yang sudah disalin untuk digunakan pada bab ini, Anda dapat melakukan langkah pertama: mengimpor sound effect.

9.1 MENGIMPOR SOUND EFFECT

Sebelum Anda dapat memainkan suara apa pun, Anda jelas perlu mengimpor file suara ke dalam proyek Unity Anda. Pertama Anda akan mengumpulkan klip suara dalam format file yang diinginkan, dan kemudian Anda akan membawa file ke Unity dan menyesuaikannya untuk tujuan Anda.

Format file yang didukung

Unity mendukung berbagai format audio dengan pro dan kontra yang berbeda. Tabel dibawah ini mencantumkan format file audio yang didukung Unity. Pertimbangan utama yang membedakan file audio adalah kompresi yang diterapkan. Kompresi mengurangi ukuran file tetapi menyelesaikannya dengan membuang sedikit informasi dalam file. Kompresi audio pintar hanya membuang informasi yang paling tidak penting sehingga suara terkompresi tetap terdengar bagus. Namun demikian, ini adalah penurunan kualitas yang kecil, jadi Anda harus

memilih audio yang tidak terkompresi saat klip suara pendek dan dengan demikian tidak akan menjadi file besar. Klip suara yang lebih panjang (terutama musik) harus menggunakan audio terkompresi, karena klip audio akan menjadi sangat besar jika tidak. Unity menambahkan sedikit kerutan pada keputusan ini, meskipun.

Meskipun musik harus dikompresi di game terakhir, Unity dapat mengompresi audio setelah Anda mengimpor file. Jadi, saat mengembangkan game di Unity, Anda biasanya ingin menggunakan format file yang tidak terkompresi bahkan untuk musik yang panjang, bukan mengimpor audio terkompresi.

9.2 CARA KERJA AUDIO DIGITAL

Secara umum, file audio menyimpan bentuk gelombang yang akan dibuat di speaker saat suara diputar. Suara adalah serangkaian gelombang yang merambat di udara, dan suara yang berbeda dibuat dengan ukuran dan frecookiesnsi gelombang suara yang berbeda. File audio merekam gelombang ini dengan mengambil sampel gelombang berulang kali pada interval waktu yang singkat dan menyimpan status gelombang pada setiap sampel.

Rekaman yang mengambil sampel gelombang lebih sering mendapatkan rekaman yang lebih akurat tentang perubahan gelombang dari waktu ke waktu—ada jarak yang lebih kecil di antara perubahan. Tetapi sampel yang lebih sering berarti lebih banyak data untuk disimpan, menghasilkan file yang lebih besar. File suara terkompresi mengurangi ukuran file melalui sejumlah trik, termasuk membuang data pada frecookiesnsi suara yang tidak terlihat oleh pendengar.

Pelacak musik adalah jenis khusus software sequencer yang digunakan untuk membuat musik. Sedangkan file audio tradisional menyimpan bentuk gelombang mentah untuk suara, sequencer menyimpan sesuatu yang lebih mirip dengan lembaran musik: file tracker adalah urutan catatan, dengan informasi seperti intensitas dan nada disimpan dengan setiap catatan. "Catatan" ini terdiri dari bentuk gelombang kecil, tetapi jumlah total data yang disimpan berkurang karena catatan yang sama digunakan berulang kali di seluruh urutan. Musik yang dikomposisikan dengan cara ini bisa menjadi efisien, tetapi ini adalah jenis audio yang cukup khusus.

Karena Unity akan memampatkan audio setelah diimpor, Anda harus selalu memilih format file WAV atau AIF. Anda mungkin perlu menyesuaikan pengaturan impor secara berbeda untuk sound effect pendek dan musik yang lebih panjang (khususnya, untuk memberi tahu Unity kapan harus menerapkan kompresi), tetapi file asli harus selalu tidak terkompresi. Ada berbagai cara untuk membuat file suara (misalnya, lampiran B menyebutkan alat seperti Audacity yang dapat merekam suara dari mikrofon), tetapi untuk tujuan kami, kami akan mengunduh beberapa suara dari salah satu dari banyak situs web suara gratis. Kami akan menggunakan sejumlah klip yang diunduh dari www.freesound.org dan mendapatkan klip dalam format file WAV.

Suara "Gratis" ditawarkan di bawah berbagai skema lisensi, jadi selalu pastikan bahwa Anda diizinkan untuk menggunakan klip suara sesuai keinginan Anda. Misalnya, banyak suara gratis hanya untuk penggunaan nonkomersial.

Proyek sampel menggunakan sound effect domain publik berikut (tentu saja, Anda dapat memilih untuk mengunduh suara Anda sendiri; cari lisensi 0 yang tercantum di samping):

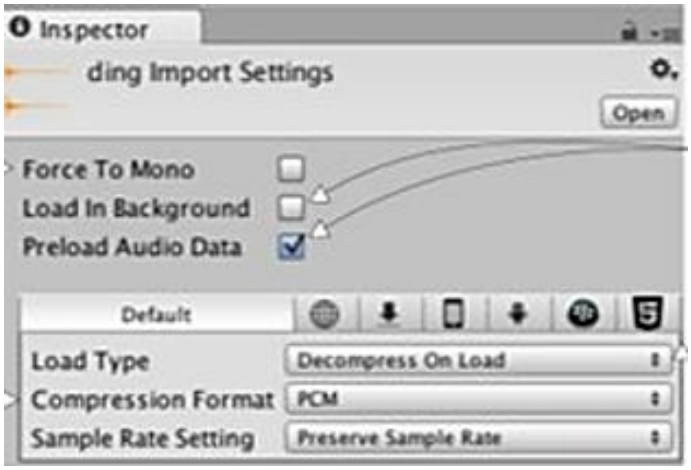
- “thump” oleh hy96
- “ding” oleh Daphne_in_Wonderland

- “swish bamboo pole” oleh ra_gun
- “fireplace” oleh leosalom

Setelah Anda memiliki file suara untuk digunakan dalam gim Anda, langkah selanjutnya adalah mengimpor suara ke Unity.

Mengimpor file audio

Setelah mengumpulkan beberapa file audio, Anda perlu membawanya ke Unity. Seperti yang Anda lakukan dengan aset seni di bab 4, Anda harus mengimpor aset audio ke dalam proyek sebelum dapat digunakan dalam game. Mekanisme sebenarnya dari mengimpor file sederhana dan sama dengan aset lainnya: drag file dari lokasinya di komputer ke tampilan Proyek dalam Unity (buat folder bernama Sound FX untuk menyeret file ke dalamnya). Nah, itu mudah! Tapi seperti aset lainnya, ada pengaturan impor (ditunjukkan pada gambar 10.1) untuk menyesuaikan di Inspector.



Gambar 9.1 Impor pengaturan untuk file audio

Biarkan Force To Mono tidak dicentang. Itu mengacu pada suara mono versus stereo; seringkali suara direkam dalam stereo, di mana sebenarnya ada dua bentuk gelombang dalam file, masing-masing untuk telinga/speaker kiri dan kanan. Untuk menghemat ukuran file, Anda mungkin ingin membagi dua informasi audio sehingga bentuk gelombang yang sama dikirim ke kedua speaker daripada gelombang terpisah yang dikirim ke speaker kiri dan kanan.

Berikutnya adalah kotak centang untuk Load In Background dan Preload Audio Data. Pengaturan pramuat berkaitan dengan menyeimbangkan kinerja pemutaran dan penggunaan memori; audio pramuat akan menghabiskan memori sementara suara menunggu untuk digunakan tetapi akan menghindari keharusan menunggu untuk memuat. Memuat audio di latar belakang program akan memungkinkan program tetap berjalan saat audio sedang dimuat; ini biasanya merupakan ide yang baik untuk klip musik yang panjang sehingga program tidak akan berhenti. Tapi ini berarti audio tidak akan langsung diputar; biasanya Anda ingin menonaktifkan pengaturan ini untuk klip suara pendek untuk memastikan bahwa klip tersebut dimuat sepenuhnya sebelum diputar. Karena klip yang diimpor adalah sound effect pendek, Anda harus menonaktifkan Load In Background.

Terakhir, pengaturan yang paling penting adalah Load Type dan Compression Format. Format Kompresi mengontrol pemformatan data audio yang disimpan. Seperti yang dibahas di bagian sebelumnya, musik harus dikompresi; pilih Vorbis (itu nama format audio terkompresi) dalam kasus itu. Klip suara pendek tidak perlu dikompres, jadi pilih PCM (Pulse Code Modulation, istilah teknis untuk gelombang suara mentah yang disampel) untuk klip ini.

Pengaturan ketiga, ADPCM, adalah variasi pada PCM dan terkadang menghasilkan kualitas suara yang sedikit lebih baik.

Load Type mengontrol bagaimana data dari file akan dimuat oleh komputer. Karena komputer memiliki memori terbatas dan file audio bisa berukuran besar, terkadang Anda ingin audio diputar saat streaming ke memori, sehingga komputer tidak perlu memuat seluruh file sekaligus. Namun ada sedikit overhead komputasi saat streaming audio seperti ini, jadi audio diputar paling cepat saat dimuat ke memori terlebih dahulu. Bahkan kemudian Anda dapat memilih apakah data audio yang dimuat akan dalam bentuk terkompresi atau jika akan didekompresi untuk pemutaran yang lebih cepat. Karena klip suara ini pendek, mereka tidak perlu streaming dan dapat diatur ke Dekompresi Saat Muat. Pada titik ini, semua sound effect diimpor dan siap digunakan.

Memutar sound effect

Sekarang setelah Anda memiliki beberapa file suara yang ditambahkan ke proyek, Anda tentu ingin memutar suara. Kode untuk memicu sound effect tidak terlalu sulit untuk dipahami, tetapi sistem audio di Unity memang memiliki sejumlah bagian berbeda yang harus bekerja bersama.

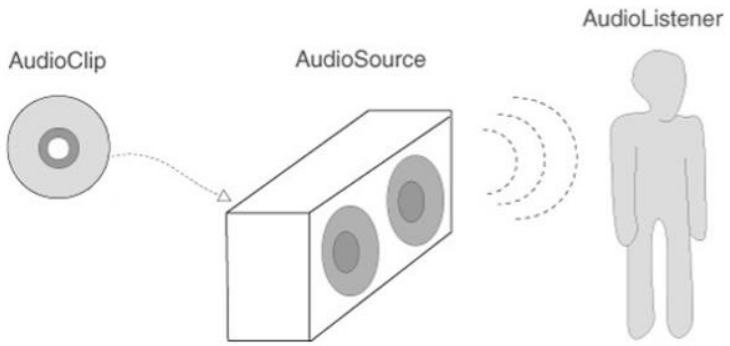
Menjelaskan apa yang terlibat: klip audio vs. sumber vs. pendengar

Meskipun Anda mungkin mengira bahwa memutar suara hanyalah masalah memberi tahu Unity klip mana yang akan diputar, ternyata Anda harus menentukan tiga bagian berbeda untuk memutar suara di Unity: AudioClip, AudioSource, dan AudioListener. Alasan untuk memecah sistem suara menjadi beberapa komponen berkaitan dengan dukungan Unity untuk suara 3D: komponen yang berbeda memberi tahu informasi posisi Unity yang digunakan untuk memanipulasi suara 3D.

9.3 SUARA 2D VS. 3D

Suara dalam game dapat berupa 2D atau 3D. Suara 2D adalah yang sudah Anda kenal: audio standar yang diputar secara normal. Moniker "suara 2D" sebagian besar berarti "bukan suara 3D." Suara 3D khusus untuk simulasi 3D dan mungkin belum familiar bagi Anda; ini adalah suara yang memiliki lokasi tertentu dalam simulasi. Volume dan nada mereka dipengaruhi oleh gerakan pendengar. Misalnya, sound effect yang dipicu di kejauhan akan terdengar sangat redup. Unity mendukung kedua jenis audio, dan Anda memutuskan apakah sumber audio harus memutar audio sebagai suara 2D atau suara 3D. Hal-hal seperti musik harus berupa suara 2D, tetapi menggunakan suara 3D untuk sebagian besar sound effect akan menciptakan audio yang imersif dalam scene.

Sebagai analogi, bayangkan sebuah ruangan di dunia nyata. Kamar memiliki stereo yang memutar CD. Jika seorang pria masuk ke ruangan, dia mendengarnya dengan jelas. Ketika dia meninggalkan ruangan, dia mendengarnya dengan lebih pelan, dan akhirnya tidak mendengarnya sama sekali. Demikian pula, jika kita menggerakkan stereo di sekitar ruangan, dia akan mendengar musik berubah volume saat bergerak. Seperti yang diilustrasikan pada Gambar 10.2, dalam analogi ini CD adalah AudioClip, stereo adalah AudioSource, dan pria adalah AudioListener.



Gambar 9.2 Diagram dari tiga hal yang Anda kendalikan dalam sistem audio Unity

Yang pertama dari tiga bagian yang berbeda adalah Audio Clip. Itu mengacu pada file suara aktual yang kami impor di bagian terakhir. Data bentuk gelombang mentah ini adalah dasar untuk semua hal lain yang dilakukan sistem audio, tetapi klip audio tidak melakukan apa pun sendiri. Jenis objek berikutnya adalah Sumber Audio. Ini adalah objek yang memutar klip audio. Ini adalah abstraksi dari apa yang sebenarnya dilakukan sistem audio, tetapi ini adalah abstraksi yang berguna yang membuat suara 3D lebih mudah dipahami. Suara 3D yang diputar dari sumber audio tertentu terletak di posisi sumber audio tersebut; Suara 2D juga harus diputar dari sumber audio, tetapi lokasi tidak masalah.

Jenis objek ketiga yang terlibat dalam sistem audio Unity adalah Audio Listener. Sesuai dengan namanya, ini adalah objek yang mendengar suara yang diproyeksikan dari sumber audio. Ini adalah abstraksi lain di atas apa yang dilakukan sistem audio (jelas pendengar yang sebenarnya adalah player dari permainan!), tetapi—seperti bagaimana posisi sumber audio memberikan posisi dari mana suara diproyeksikan—posisi pendengar audio memberikan posisi dari mana suara itu terdengar.

Kontrol suara tingkat lanjut menggunakan Audio Mixers

Audio Mixer adalah fitur baru yang ditambahkan di Unity 5. Daripada memutar klip audio secara langsung, mixer audio memungkinkan Anda memproses sinyal audio dan menerapkan berbagai efek ke klip Anda. Pelajari lebih lanjut tentang AudioMixer dalam dokumentasi Unity; misalnya, tonton video tutorial ini: https://unity3d.com/learn/tutorials/modules/beginner/5-pre-orderbeta/audiomixer-and-audiomixer-groups. Meskipun klip audio dan komponen AudioSource harus ditetapkan, komponen AudioListener sudah ada di kamera default saat Anda membuat scene baru. Biasanya Anda ingin suara 3D bereaksi terhadap posisi penampil.

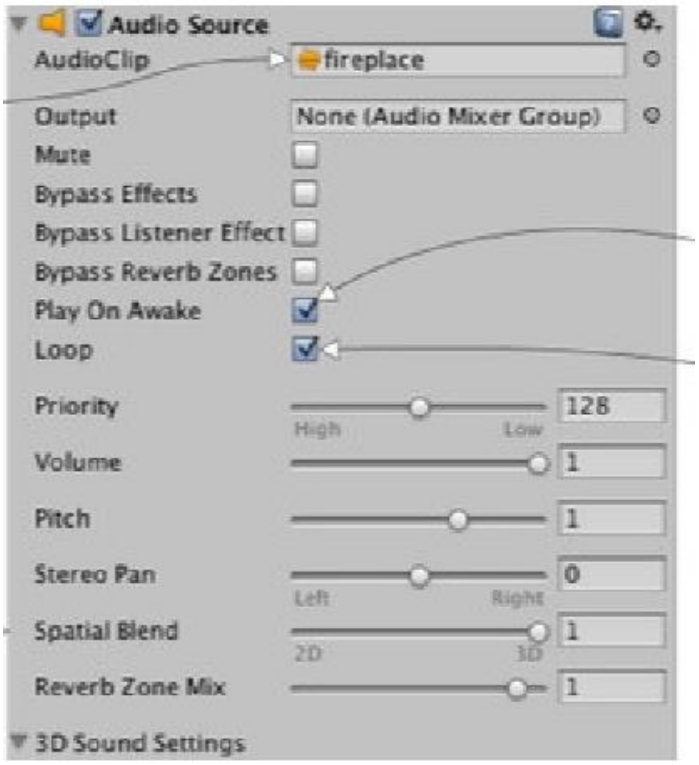
Menetapkan looping sound

Baiklah, sekarang mari kita atur suara pertama kita di Unity! Klip audio sudah diimpor, dan kamera default memiliki komponen AudioListener, jadi kita hanya perlu menetapkan komponen AudioSource. Kami akan menempatkan suara api berderak pada Prefab Musuh, karakter musuh yang berkeliaran.

Karena musuh akan terdengar seperti sedang terbakar, Anda mungkin ingin memberinya sistem partikel agar terlihat seperti sedang terbakar. Anda dapat menyalin sistem partikel yang dibuat di bab 4 dengan membuat objek partikel menjadi Prefab dan kemudian memilih Paket Ekspor dari menu Aset. Sebagai alternatif, Anda dapat mengulangi langkah-

langkah dari bab 4 di sini untuk membuat objek partikel baru dari awal (drag prefab Musuh ke dalam scene untuk mengeditnya lalu pilih GameObject > Apply Changes To Prefab).

Biasanya Anda perlu menyeret prefab ke dalam scene untuk mengeditnya, tetapi Anda dapat mengedit aset prefab secara langsung saat Anda hanya menambahkan komponen ke objek. Pilih Prefab Musuh sehingga propertinya muncul di Inspector. Sekarang tambahkan komponen baru; pilih Audio > Sumber Audio. Komponen AudioSource akan muncul di Inspector. Beri tahu sumber audio klip suara apa yang akan diputar. Drag file audio dari tampilan Proyek ke slot Klip Audio di Inspector; kita akan menggunakan sound effect "perapian" untuk contoh ini.



Gambar 9.3 Pengaturan untuk komponen AudioSource

Lewati sedikit di pengaturan dan pilih Play On Awake dan Looping (tentu saja, pastikan Mute tidak dicentang). Play On Awake memberi tahu sumber audio untuk mulai diputar segera setelah scene dimulai (di bagian berikutnya Anda akan mempelajari cara memicu suara secara manual saat scene sedang berjalan). Looping memberi tahu sumber audio untuk terus memutar, mengulangi klip audio saat pemutaran selesai.

Anda ingin sumber audio ini memproyeksikan suara 3D. Seperti yang dijelaskan sebelumnya, suara 3D memiliki posisi berbeda di dalam scene. Aspek sumber audio tersebut disesuaikan menggunakan pengaturan Spatial Blend. Pengaturan itu adalah penggeser antara 2D dan 3D; atur ke 3D untuk sumber audio ini. Sekarang mainkan gamenya dan pastikan speaker Anda dihidupkan. Anda dapat mendengar suara tembakan dari musuh, dan suara menjadi samar jika Anda menjauh karena Anda menggunakan sumber audio 3D.

Memicu sound effect dari kode

Mengatur komponen AudioSource untuk diputar secara otomatis berguna untuk beberapa looping sound, tetapi untuk sebagian besar sound effect, Anda ingin memicu suara dengan perintah kode. Pendekatan itu masih memerlukan komponen AudioSource, tetapi

sekarang sumber audio hanya akan memutar klip suara ketika diperintahkan oleh program, bukan secara otomatis sepanjang waktu. Tambahkan komponen AudioSource ke objek pemutar (bukan objek kamera). Anda tidak perlu menautkan dalam klip audio tertentu karena klip audio akan ditentukan dalam kode. Anda dapat mematikan Play On Awake karena suara dari sumber ini akan dipicu dalam kode. Juga, sesuaikan Spatial Blend ke 3D karena suara ini berada di scene. Sekarang buat tambahan yang ditunjukkan dalam daftar berikutnya ke RayShooter, skrip yang menangani pemotretan.

Sound effect ditambahkan dalam skrip RayShooter

```
...
[SerializeField] private AudioSource soundSource;
[SerializeField] private AudioClip hitWallSound;
[SerializeField] private AudioClip hitEnemySound;
...

if (target != null) {
    target.ReactToHit();
    soundSource.PlayOneShot(hitEnemySound);
} else {
    StartCoroutine(SphereIndicator(hit.point));
    soundSource.PlayOneShot(hitWallSound);
}
...
```

Baiklah, mainkan gamenya dan tembak-tembakan. Anda sekarang memiliki beberapa sound effect yang berbeda dalam permainan. Langkah-langkah dasar yang sama ini dapat digunakan untuk menambahkan segala macam sound effect. Sistem suara yang kuat dalam game membutuhkan lebih dari sekadar sekumpulan suara yang terputus; minimal, semua game harus menawarkan kontrol volume. Anda akan menerapkan kontrol itu selanjutnya melalui modul audio pusat.

Antarmuka Kontrol Audio

Melanjutkan arsitektur kode yang dibuat di bab sebelumnya, Anda akan membuat AudioManager. Ingatlah bahwa objek Manajer memiliki daftar induk dari berbagai modul kode yang digunakan oleh game, seperti manajer untuk inventaris player. Kali ini Anda akan membuat pengelola audio untuk dimasukkan ke dalam daftar. Modul audio pusat ini akan memungkinkan Anda untuk memodulasi volume audio dalam game dan bahkan membisukannya. Awalnya Anda hanya akan khawatir tentang sound effect, tetapi di bagian selanjutnya Anda akan memperluas AudioManager untuk menangani musik juga.

9.4 MENYIAPKAN AUDIO MANAGER PUSAT

Langkah pertama dalam menyiapkan AudioManager adalah menempatkan kerangka kerja kode Manajer. Dari proyek bab sebelumnya, salin melalui IGameManager, ManagerStatus, dan NetworkService; kami tidak akan mengubahnya. (Ingat bahwa IGameManager adalah antarmuka yang harus diterapkan oleh semua manajer, sedangkan ManagerStatus adalah enum yang digunakan IGameManager. NetworkService menyediakan panggilan ke internet dan tidak akan digunakan dalam bab ini.)

Unity mungkin akan mengeluarkan peringatan karena NetworkService ditetapkan tetapi tidak digunakan. Anda bisa mengabaikan peringatan Unity; kami ingin mengaktifkan

kerangka kode untuk mengakses internet, meskipun kami tidak menggunakan fungsi itu dalam bab ini. Salin juga file Manajer, yang akan disesuaikan untuk AudioManager baru. Biarkan saja untuk saat ini (atau komentari bagian yang salah jika melihat kesalahan kompiler membuat Anda gila!). Buat skrip baru bernama AudioManager yang dapat dirujuk oleh kode Manajer (lihat daftar berikut).

Kode kerangka untuk AudioManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class AudioManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   private NetworkService _network;

   // Add volume controls here (listing 10.4)

   public void Startup(NetworkService service) {
      Debug.Log("Audio manager starting...");

      _network = service;

      // Initialize music sources here (listing 10.10)

      status = ManagerStatus.Started;
   }
}
```

Kode awal ini terlihat seperti manajer dari bab sebelumnya; ini adalah jumlah minimum yang diperlukan oleh IGameManager untuk diimplementasikan oleh kelas. Skrip Manajer sekarang dapat disesuaikan dengan manajer baru (lihat daftar berikutnya).

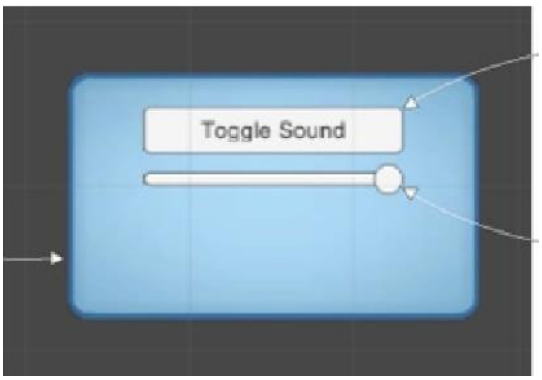
Skrip manajer disesuaikan dengan AudioManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

[RequireComponent(typeof(AudioManager))]

public class Managers : MonoBehaviour {
   public static AudioManager Audio {get; private set;}

   private List<IGameManager> _startSequence;
   void Awake() {
      Audio = GetComponent<AudioManager>();

      _startSequence = new List<IGameManager>();
      _startSequence.Add(Audio);

      StartCoroutine(StartupManagers());
   }

   private IEnumerator StartupManagers() {
      NetworkService network = new NetworkService();

   foreach (IGameManager manager in _startSequence) {
      manager.Startup(network);
   }

   yield return null;

   int numModules = _startSequence.Count;
   int numReady = 0;

   while (numReady < numModules) {
      int lastReady = numReady;
      numReady = 0;

      foreach (IGameManager manager in _startSequence) {
         if (manager.status == ManagerStatus.Started) {
            numReady++;
         }
      }

      if (numReady > lastReady)
         Debug.Log("Progress: " + numReady + "/" + numModules);

      yield return null;
   }

   Debug.Log("All managers started up");
  }
}
```

Seperti yang telah Anda lakukan di bab sebelumnya, buat objek Game Managers di scene dan kemudian lampirkan Managers dan AudioManager ke objek kosong. Memainkan game akan menampilkan pesan startup manajer di konsol, tetapi manajer audio belum melakukan apa-apa.

9.5 UI KONTROL VOLUME

Dengan pengaturan AudioManager yang sederhana, inilah saatnya untuk memberikan fungsionalitas kontrol volume. Metode kontrol volume ini kemudian akan digunakan oleh tampilan UI untuk menonaktifkan sound effect atau menyesuaikan volume. Anda akan menggunakan alat UI baru yang menjadi fokus bab 6. Secara khusus, Anda akan membuat jendela pop-up dengan tombol dan penggeser untuk mengontrol pengaturan volume (lihat gambar 9.4). Saya akan mencantumkan langkah-langkah yang terlibat tanpa merinci; jika Anda membutuhkan penyegaran, lihat kembali bab 6:



Gambar 9.4 Tampilan UI untuk bisu dan kontrol volume

1. Impor popup.png sebagai sprite (atur Jenis Tekstur ke Sprite).
2. Di Editor Sprite, atur batas 12-piksel di semua sisi (ingat untuk menerapkan perubahan).
3. Buat kanvas di scene (GameObject > UI > Canvas).
4. Aktifkan pengaturan Pixel Perfect untuk kanvas.
5. (Opsional) Beri nama objek HUD Canvas dan alihkan ke mode tampilan 2D.
6. Buat gambar yang terhubung ke kanvas itu (GameObject > UI > Image).
7. Beri nama objek baru Pengaturan Popup.
8. Tetapkan sprite popup ke Gambar Sumber gambar.
9. Atur Jenis Gambar ke Irisan dan nyalakan Pusat Isi.
10. Posisikan gambar pop-up pada 0, 0 untuk memusatkannya.
11. Scale pop-up hingga 250 lebar dan 150 tinggi.
12. Buat tombol (GameObject > UI > Button).
13. Induk tombol ke pop-up (yaitu, drag di Hierarki).
14. Posisikan tombol pada 0, 40.
15. Perluas hierarki tombol untuk memilih label teksnya.
16. Ubah teks menjadi Toggle Sound.
17. Buat penggeser (GameObject > UI > Slider).
18. Induk penggeser ke pop-up dan posisikan di 0, 15.

Itu semua langkah-langkah untuk membuat pop-up pengaturan! Sekarang pop-up telah dibuat, mari tulis kode yang akan digunakan. Ini akan melibatkan skrip pada objek pop-up itu sendiri, serta fungsionalitas kontrol volume yang dipanggil oleh skrip popup. Pertama-tama sesuaikan kode di AudioManager sesuai dengan daftar berikutnya.

Kontrol volume ditambahkan ke AudioManager

```
...
public float soundVolume {
   get {return AudioListener.volume;}
   set {AudioListener.volume = value;}
}
public bool soundMute {
   get {return AudioListener.pause;}
   set {AudioListener.pause = value;}
}

public void Startup(NetworkService service) {
   Debug.Log("Audio manager starting...");

   _network = service;

   soundVolume = 1f;

   status = ManagerStatus.Started;
}
...
```

Properti untuk soundVolume dan soundMute ditambahkan ke AudioManager. Untuk kedua properti, fungsi get dan set diimplementasikan menggunakan nilai global di AudioListener. Kelas AudioListener dapat memodulasi volume semua suara yang diterima oleh semua instans AudioListener. Menyetel properti soundVolume AudioManager memiliki efek yang sama seperti menyetel volume pada AudioListener. Keuntungannya di sini adalah enkapsulasi: segala sesuatu yang berkaitan dengan audio ditangani dalam satu manajer, tanpa kode di luar manajer yang perlu mengetahui detail implementasinya. Dengan metode tersebut ditambahkan ke AudioManager, Anda sekarang dapat menulis skrip untuk pop-up. Buat skrip bernama Settings Popup dan tambahkan konten daftar berikut.

Pengaturan Skrip popup dengan kontrol untuk menyesuaikan volume

```
using UnityEngine;
using System.Collections;

public class SettingsPopup : MonoBehaviour {

   public void OnSoundToggle() {
      Managers.Audio.soundMute = !Managers.Audio.soundMute;
   }

   public void OnSoundValue(float volume) {
      Managers.Audio.soundVolume = volume;
   }
}
```

Skrip ini memiliki dua metode yang memengaruhi properti AudioManager: OnSoundToggle() menyetel properti soundMute, dan OnSoundValue() menyetel properti soundVolume. Seperti biasa, tautkan di skrip SettingsPopup dengan menyeretnya ke objek Settings Popup di UI. Kemudian, untuk memanggil fungsi dari tombol dan penggeser, tautkan objek pop-up ke peristiwa interaksi di kontrol tersebut. Di tombol Inspector, cari panel berlabel OnClick. Klik tombol + untuk menambahkan entri baru ke event ini. Tarik Pengaturan Popup ke slot objek di entri baru dan kemudian cari PengaturanPopup di menu; pilih OnSoundToggle() untuk membuat tombol memanggil fungsi itu.

Metode yang digunakan untuk menautkan fungsi juga berlaku untuk penggeser. Pertama-tama cari event interaksi di panel pengaturan penggeser; dalam hal ini, panel disebut OnValueChanged. Klik tombol + untuk menambahkan entri baru lalu drag Popup Pengaturan

ke slot objek. Di menu fungsi, temukan skrip SettingsPopup lalu pilih OnSoundVolume() di bawah Dynamic Float. Ingatlah untuk memilih fungsi di bawah Float Dinamis dan bukan Parameter Statis! Meskipun metode muncul di kedua bagian daftar, dalam kasus terakhir ini hanya akan menerima satu nilai yang diketik sebelumnya.

Kontrol pengaturan sekarang berfungsi, tetapi ada satu skrip lagi yang kami perlukan untuk mengatasi fakta bahwa pop-up saat ini selalu menutupi layar. Perbaikan sederhana adalah membuat pop-up hanya terbuka ketika Anda menekan tombol M. Buat skrip baru bernama UIController, tautkan skrip itu ke objek Controller di scene, dan tulis kode yang ditampilkan di daftar berikutnya.

UIController yang mengaktifkan pop-up pengaturan

```
using UnityEngine;
using System.Collections;

public class UIController : MonoBehaviour {
    [SerializeField] private SettingsPopup popup;

    void Start() {
        popup.gameObject.SetActive(false);
    }

    void Update() {
        if (Input.GetKeyDown(KeyCode.M)) {
            bool isShowing = popup.gameObject.activeSelf;
            popup.gameObject.SetActive(!isShowing);

            if (isShowing) {
                Cursor.lockState = CursorLockMode.Locked;
                Cursor.visible = false;
            } else {
                Cursor.lockState = CursorLockMode.None;
                Cursor.visible = true;
            }
        }
    }
}
```

Untuk menyambungkan referensi objek ini, drag pop-up pengaturan ke slot pada skrip ini. Mainkan sekarang dan coba ubah penggeser (ingat untuk mengaktifkan UI dengan menekan M) sambil memotret untuk mendengar sound effect; Anda akan mendengar sound effect mengubah volume sesuai dengan penggeser.

Memutar suara UI

Anda akan membuat tambahan lain ke AudioManager sekarang untuk memungkinkan UI memutar suara saat tombol diklik. Tugas ini lebih terlibat daripada yang terlihat pada awalnya, karena kebutuhan Unity akan AudioSource. Saat sound effect dikeluarkan dari objek di scene, sudah cukup jelas di mana harus memasang AudioSource. Tapi sound effect UI bukan bagian dari scene, jadi Anda akan menyiapkan AudioSource khusus hanya untuk AudioManager untuk digunakan saat tidak ada sumber audio lain.

Buat GameObject kosong baru dan parent ke objek utama Game Managers; objek baru ini akan memiliki AudioSource yang digunakan oleh AudioManager, jadi panggil objek baru Audio. Tambahkan komponen AudioSource ke objek ini (tinggalkan pengaturan Spatial Blend pada 2D saat ini, karena UI tidak memiliki posisi tertentu dalam scene) lalu tambahkan kode yang ditampilkan di daftar berikutnya untuk menggunakan sumber ini di AudioManager.

Putar sound effect di AudioManager

```
...
[SerializeField] private AudioSource soundSource;
...
public void PlaySound(AudioClip clip) {
    soundSource.PlayOneShot(clip);
}
...
```

Sebuah slot variabel baru akan muncul di Inspector; drag objek Audio ke slot ini. Sekarang tambahkan sound effect UI ke skrip pop-up (lihat daftar berikut).

Menambahkan sound effect ke SettingsPopup

```
...
[SerializeField] private AudioClip sound;
...
public void OnSoundToggle() {
   Managers.Audio.soundMute = !Managers.Audio.soundMute;
   Managers.Audio.PlaySound(sound);
}
...
```

Drag sound effect UI ke slot variabel; Saya menggunakan suara 2D "thump." Saat Anda menekan tombol UI, sound effect itu diputar pada saat yang bersamaan (tentu saja, saat suara tidak dimatikan!). Meskipun UI tidak memiliki sumber audio itu sendiri, AudioManager memiliki sumber audio yang memainkan sound effect. Bagus, kami telah menyiapkan semua sound effect kami! Sekarang mari kita mengalihkan perhatian kita ke musik.

9.6 LATAR BELAKANG MUSIK

Anda akan menambahkan beberapa musik latar ke game, dan Anda akan melakukannya dengan menambahkan musik ke AudioManager. Seperti yang dijelaskan dalam pendahuluan bab, klip musik pada dasarnya tidak berbeda dari sound effect. Cara fungsi audio digital melalui bentuk gelombang adalah sama, dan perintah untuk memutar audio sebagian besar sama. Perbedaan utama adalah panjang audio, tetapi perbedaan itu menimbulkan sejumlah konsecookiesnsi.

Sebagai permulaan, trek musik cenderung menghabiskan banyak memori di komputer, dan konsumsi memori itu harus dioptimalkan. Anda harus berhati-hati terhadap dua area masalah memori: memasukkan musik ke dalam memori sebelum dibutuhkan, dan menghabiskan terlalu banyak memori saat dimuat.

Mengoptimalkan saat pemuatan musik dilakukan menggunakan perintah Resources.Load() yang diperkenalkan di bab 8. Seperti yang Anda pelajari, perintah ini memungkinkan Anda memuat aset berdasarkan nama; meskipun itu pasti salah satu fitur yang berguna, itu bukan satu-satunya alasan untuk memuat aset dari folder Resources. Pertimbangan utama lainnya adalah menunda pemuatan; biasanya Unity memuat semua aset dalam scene segera setelah scene dimuat, tetapi aset dari Sumber Daya tidak dimuat hingga kode mengambilnya secara manual. Dalam hal ini, kami ingin memuat klip audio untuk musik dengan lambat. Jika tidak, musik dapat menghabiskan banyak memori saat tidak digunakan.

Lazy-loading adalah ketika file tidak dimuat sebelumnya tetapi ditunda sampai dibutuhkan. Biasanya data merespons lebih cepat (misalnya, suara langsung diputar) jika dimuat sebelum digunakan, tetapi pemuatan lambat dapat menghemat banyak memori saat responsivitas tidak terlalu penting. Pertimbangan memori kedua ditangani dengan streaming musik dari disk.

Seperti yang dijelaskan di bagian 10.1.2, streaming audio menyelamatkan komputer dari keharusan memuat seluruh file sekaligus. Gaya pemuatan adalah pengaturan di Inspector klip audio yang diimpor. Pada akhirnya ada beberapa langkah yang harus dilalui untuk memutar musik latar, termasuk langkah-langkah untuk menutupi pengoptimalan memori ini.

Memutar musik loop

Proses memutar musik melibatkan serangkaian langkah yang sama seperti sound effect UI (musik latar juga merupakan suara 2D tanpa sumber di dalam scene), jadi kita akan melakukan semua langkah lagi:

1. Impor klip audio.
2. Siapkan AudioSource untuk digunakan AudioManager.
3. Tulis kode untuk memutar klip audio di AudioManager.
4. Tambahkan kontrol musik ke UI.

Setiap langkah akan dimodifikasi sedikit untuk bekerja dengan musik, bukan sound effect. Mari kita lihat langkah pertama.

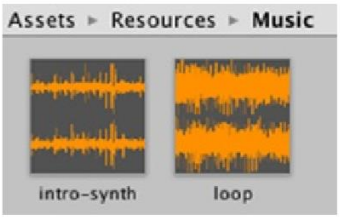
Langkah 1: Impor klip audio

Dapatkan musik dengan mengunduh atau merekam trek. Untuk proyek sampel saya pergi ke www.freesound.org dan mengunduh loop musik domain publik berikut:

- "loop" oleh Xythe/Ville Nousiainen
- "Intro Synth" oleh noirenex

Drag file ke Unity untuk mengimpornya, lalu sesuaikan pengaturan impornya di Inspector. Seperti yang dijelaskan sebelumnya, klip audio untuk musik umumnya memiliki pengaturan yang berbeda dari klip audio untuk sound effect. Pertama, format audio harus diatur ke Vorbis, untuk audio terkompresi. Ingat, audio terkompresi akan memiliki ukuran file yang jauh lebih kecil. Kompresi juga sedikit menurunkan kualitas audio, tetapi sedikit penurunan itu merupakan pertukaran yang dapat diterima untuk klip musik yang panjang; atur Kualitas menjadi 50% di bilah geser yang muncul.

Pengaturan impor berikutnya yang harus disesuaikan adalah Load Type. Sekali lagi, musik harus mengalir dari disk daripada dimuat sepenuhnya. Pilih Streaming dari menu Load Type. Demikian pula, aktifkan Muat Di Latar Belakang agar permainan tidak berhenti atau melambat saat musik sedang dimuat. Bahkan setelah Anda menyesuaikan semua pengaturan impor, file aset harus dipindahkan ke lokasi yang benar agar dapat dimuat dengan benar. Ingat bahwa perintah Resources.Load() mengharuskan aset berada di folder Resources. Buat folder baru bernama Resources, buat folder di dalamnya yang bernama Music, dan drag file audio ke folder Music (lihat gambar 9.5).



Gambar 9.5 Klip audio musik ditempatkan di dalam folder Resources

Langkah 2: Siapkan AudioSource untuk AudioManager untuk digunakan

Langkah 2 adalah membuat AudioSource baru untuk pemutaran musik. Buat GameObject kosong lainnya, beri nama objek ini Music 1 (bukan hanya Music karena kita akan

menambahkan Music 2 nanti di bab ini), dan parent ke objek Audio. Tambahkan komponen AudioSource ke Music 1 lalu sesuaikan pengaturan di komponen. Batalkan pilihan Play On Awake tetapi aktifkan opsi Loop kali ini; sedangkan sound effect biasanya hanya diputar sekali, musik diputar berulang-ulang. Biarkan pengaturan Spatial Blend pada 2D, karena musik tidak memiliki posisi tertentu dalam scene.

Anda mungkin ingin mengurangi nilai Prioritas juga. Untuk sound effect, nilai ini tidak masalah, jadi kami meninggalkan nilai pada default 128. Tapi untuk musik Anda mungkin ingin menurunkan nilai ini, jadi saya menyetel sumber musik ke 60. Nilai ini memberi tahu Unity suara mana yang paling penting saat melapisi banyak suara; agak berlawanan dengan intuisi, nilai yang lebih rendah adalah prioritas yang lebih tinggi. Jika terlalu banyak suara yang diputar secara bersamaan, sistem audio akan mulai membuang suara; dengan membuat musik lebih diprioritaskan daripada sound effect, Anda memastikan musik akan terus diputar ketika terlalu banyak sound effect dipicu pada saat yang bersamaan.

Langkah 3: Tulis kode untuk memutar klip audio di AudioManager

Sumber audio Musik telah disiapkan, jadi tambahkan kode yang ditampilkan di daftar berikutnya ke AudioManager.

Memutar musik di AudioManager

```
...
[SerializeField] private AudioSource music1Source;
[SerializeField] private string introBGMusic;
[SerializeField] private string levelBGMusic;
...
public void PlayIntroMusic() {
   PlayMusic(Resources.Load("Music/"+introBGMusic) as AudioClip);
}
public void PlayLevelMusic() {
   PlayMusic(Resources.Load("Music/"+levelBGMusic) as AudioClip);
}

private void PlayMusic(AudioClip clip) {
   music1Source.clip = clip;
   music1Source.Play();
}

public void StopMusic() {
   music1Source.Stop();
}
...
```

Seperti biasa, variabel serial baru akan terlihat di Inspector saat Anda memilih objek Game Managers. Drag Musik 1 ke dalam slot sumber audio. Kemudian ketik nama file musik di dua variabel string: introsynth dan loop. Sisa dari kode yang ditambahkan memanggil perintah untuk memuat dan memutar musik (atau, dalam metode terakhir yang ditambahkan, menghentikan musik). Perintah Resources.Load() memuat aset bernama dari folder Resources (dengan mempertimbangkan bahwa file ditempatkan di subfolder Music di dalam Resources). Objek generik dikembalikan oleh perintah itu, tetapi objek tersebut dapat dikonversi ke tipe yang lebih spesifik (dalam hal ini, AudioClip) menggunakan kata kunci as.

Klip audio yang dimuat kemudian diteruskan ke metode PlayMusic(). Fungsi ini mengatur klip di AudioSource dan kemudian memanggil Play(). Seperti yang saya jelaskan sebelumnya, sound effect lebih baik diimplementasikan menggunakan PlayOneShot(), tetapi

menyetel klip di AudioSource adalah pendekatan yang lebih kuat untuk musik, memungkinkan Anda untuk menghentikan atau menjeda musik yang sedang diputar.

Langkah 4: Tambahkan kontrol musik ke UI

Metode pemutaran musik baru di AudioManager tidak akan melakukan apa pun kecuali jika dipanggil dari tempat lain. Mari tambahkan lebih banyak tombol ke UI audio yang akan memutar musik berbeda saat ditekan. Berikut adalah langkah-langkah yang disebutkan dengan sedikit penjelasan (lihat kembali bab 6 jika diperlukan):

1. Ubah lebar pop-up menjadi 350 (agar lebih banyak tombol).
2. Buat tombol UI baru dan masukkan ke pop-up.
3. Atur lebar tombol menjadi 100 dan posisikan ke 0, -20.
4. Perluas hierarki tombol untuk memilih label teks dan atur ke Level Music.
5. Ulangi langkah ini dua kali lagi untuk membuat dua tombol tambahan.
6. Posisikan satu di -105, -20 dan yang lainnya di 105, -20 (sehingga mereka muncul di kedua sisi).
7. Ubah label teks pertama menjadi Musik Intro dan label teks terakhir menjadi Tanpa Musik.

Sekarang pop-up memiliki tiga tombol untuk memainkan musik yang berbeda. Tulis metode (ditampilkan dalam daftar berikut) di SettingsPopup yang akan ditautkan ke setiap tombol.
Menambahkan kontrol musik ke SettingsPopup

```
...
public void OnPlayMusic(int selector) {
    Managers.Audio.PlaySound(sound);

    switch (selector) {
    case 1:
        Managers.Audio.PlayIntroMusic();
        break;
    case 2:
        Managers.Audio.PlayLevelMusic();
        break;
    default:
        Managers.Audio.StopMusic();
        break;
    }
}
...
```

Perhatikan bahwa fungsi mengambil parameter int kali ini; biasanya metode tombol tidak memiliki parameter dan hanya dipicu oleh tombol. Dalam hal ini, kita perlu membedakan ketiga tombol tersebut, sehingga masing-masing tombol akan mengirimkan nomor yang berbeda. Ikuti langkah-langkah khas untuk menghubungkan tombol ke kode ini: tambahkan entri ke panel OnClick di Inspector, drag pop-up ke slot objek, dan pilih fungsi yang sesuai dari menu. Kali ini akan ada text box untuk mengetikkan sebuah angka, karena OnPlayMusic() mengambil sebuah angka untuk sebuah parameter. Ketik 1 untuk Intro Music, 2 untuk Level Music, dan yang lainnya untuk No Music (saya memilih 0). Pernyataan sakelar di OnMusic() memainkan musik intro atau musik level tergantung pada nomornya, atau menghentikan musik sebagai default jika nomornya bukan 1 atau 2.

Saat Anda menekan tombol musik saat game sedang dimainkan, Anda akan mendengar musiknya. Besar! Kode sedang memuat klip audio dari folder Resources. Musik diputar secara efisien, meskipun masih ada dua polesan yang akan kami tambahkan: kontrol volume musik terpisah dan pemudaran silang saat mengubah musik.

9.7 MENGONTROL VOLUME MUSIK SECARA TERPISAH

Gim ini sudah memiliki kontrol volume, dan saat ini juga memengaruhi musik. Namun, sebagian besar game memiliki kontrol volume terpisah untuk sound effect dan musik, jadi mari kita atasi itu sekarang. Langkah pertama adalah memberi tahu AudioSources musik untuk mengabaikan pengaturan di AudioListener. Kami ingin volume dan mute di AudioListener global terus memengaruhi semua sound effect, tetapi kami tidak ingin volume ini berlaku untuk musik. Daftar 10.10 menyertakan kode untuk memberi tahu sumber musik agar mengabaikan volume di AudioListener. Kode dalam daftar berikut juga menambahkan kontrol volume dan mute untuk musik, jadi tambahkan ke AudioManager.

Mengontrol volume musik secara terpisah di AudioManager

```
...
private float _musicVolume;
public float musicVolume {
   get {
      return _musicVolume;
   }
   set {
      _musicVolume = value;

      if (musicSource != null) {
         musicSource.volume = _musicVolume;
      }
   }
}
...
...
public bool musicMute {
   get {
      if (musicSource != null) {
         return musicSource.mute;
      }

      return false;
   }
   set {
      if (musicSource != null) {
         musicSource.mute = value;
      }
   }
}

public void Startup(NetworkService service) {
   Debug.Log("Audio manager starting...");

   _network = service;

   musicSource.ignoreListenerVolume = true;
   musicSource.ignoreListenerPause = true;

   soundVolume = 1f;
   musicVolume = 1f;

   status = ManagerStatus.Started;
}
...
```

Kunci dari kode ini adalah menyadari bahwa Anda dapat menyesuaikan volume AudioSource secara langsung, meskipun sumber audio tersebut mengabaikan volume global yang ditentukan di AudioListener. Ada properti untuk volume dan bisu yang memanipulasi sumber musik individu. Metode Startup() menginisialisasi sumber musik dengan baik mengabaikanListener-Volume dan mengabaikanListenerPause diaktifkan. Seperti namanya, properti tersebut menyebabkan sumber audio mengabaikan pengaturan volume global di AudioListener.

Anda dapat menekan rotate sekarang untuk memverifikasi bahwa musik tidak lagi terpengaruh oleh kontrol volume yang ada. Sekarang mari tambahkan kontrol UI kedua untuk volume musik; mulai dengan menyesuaikan SettingsPopup sesuai dengan daftar berikutnya.
Kontrol volume musik di SettingsPopup

```
...
public void OnMusicToggle() {
   Managers.Audio.musicMute = !Managers.Audio.musicMute;
   Managers.Audio.PlaySound(sound);
}

public void OnMusicValue(float volume) {
   Managers.Audio.musicVolume = volume;
}
...
```

Tidak banyak yang bisa dijelaskan tentang kode ini — sebagian besar mengulangi kontrol volume suara. Jelas properti AudioManager yang digunakan telah berubah dari soundMute/soundVolume menjadi musicMute/musicVolume.

Di editor, buat tombol dan penggeser seperti yang Anda lakukan sebelumnya. Berikut langkah-langkahnya lagi:

1. Ubah tinggi pop-up menjadi 225 (agar lebih banyak kontrol).
2. Buat tombol UI.
3. Induk tombol ke pop-up.
4. Posisikan tombol pada 0, -60.
5. Perluas hierarki tombol untuk memilih label teksnya.
6. Ubah teks menjadi Toggle Music.
7. Buat slider (dari menu UI yang sama).
8. Induk penggeser ke pop-up dan posisikan di 0, -85.

Tautkan kontrol UI ini ke kode di SettingsPopup. Temukan panel OnClick/OnValueChanged di pengaturan elemen UI, klik tombol + untuk menambahkan entri, drag objek pop-up ke slot objek, dan pilih fungsi dari menu. Fungsi yang perlu Anda pilih adalah OnMusicToggle() dan OnMusicValue() dari bagian Dynamic Float pada menu. Sekarang jalankan kode ini dan Anda akan melihat bahwa kontrol memengaruhi sound effect dan musik secara terpisah. Ini semakin canggih, tetapi ada satu lagi polesan yang tersisa: cross-fade di antara trek musik.

Memudar di antara lagu

Sebagai polesan terakhir, mari buat AudioManager memudar masuk dan keluar di antara nada latar yang berbeda. Saat ini pergantian antara trek musik yang berbeda cukup menggelegar, dengan suara tiba-tiba terputus dan berubah ke trek baru. Kita dapat memperlancar transisi itu dengan membuat volume trek sebelumnya berkurang dengan cepat sementara volume dengan cepat naik dari 0 di trek baru. Ini adalah kode sederhana namun

cerdas yang menggabungkan kedua metode kontrol volume yang baru saja Anda lihat, bersama dengan coroutine untuk mengubah volume secara bertahap seiring waktu.

Listing dibawah ini adalah untuk menambahkan banyak bit ke AudioManager, tetapi sebagian besar berkisar pada konsep sederhana: sekarang kita akan memiliki dua sumber audio terpisah, memutar trek musik terpisah pada sumber audio terpisah, dan secara bertahap meningkatkan volume satu sumber sekaligus mengurangi volume yang lain (seperti biasa, kode yang dicetak miring sudah ada di skrip dan ditampilkan di sini untuk referensi).

Pudar-silang antara musik di AudioManager

```
...
[SerializeField] private AudioSource music2Source;

private AudioSource _activeMusic;
private AudioSource _inactiveMusic;

public float crossFadeRate = 1.5f;
private bool _crossFading;
...
public float musicVolume {
   ...
   set {
      _musicVolume = value;

      if (music1Source != null && !_crossFading) {
         music1Source.volume = _musicVolume;
         music2Source.volume = _musicVolume;
      }
   }
}
...
public bool musicMute {
   ...
   set {
      if (music1Source != null) {
         music1Source.mute = value;
         music2Source.mute = value;
      }
   }
}

 public void Startup(NetworkService service) {
    Debug.Log("Audio manager starting...");

    _network = service;

    music1Source.ignoreListenerVolume = true;
    music2Source.ignoreListenerVolume = true;
    music1Source.ignoreListenerPause = true;
    music2Source.ignoreListenerPause = true;
    soundVolume = 1f;
    musicVolume = 1f;

    _activeMusic = music1Source;
    _inactiveMusic = music2Source;

    status = ManagerStatus.Started;
 }
 ...
 private void PlayMusic(AudioClip clip) {
    if (_crossFading) {return;}
    StartCoroutine(CrossFadeMusic(clip));
 }
 private IEnumerator CrossFadeMusic(AudioClip clip) {
    _crossFading = true;

    _inactiveMusic.clip = clip;
    _inactiveMusic.volume = 0;
    _inactiveMusic.Play();
```

```
        float scaledRate = crossFadeRate * _musicVolume;
        while (_activeMusic.volume > 0) {
            _activeMusic.volume -= scaledRate * Time.deltaTime;
            _inactiveMusic.volume += scaledRate * Time.deltaTime;

            yield return null;
        }

        AudioSource temp = _activeMusic;

        _activeMusic = _inactiveMusic;
        _activeMusic.volume = _musicVolume;

        _inactiveMusic = temp;
        _inactiveMusic.Stop();

        _crossFading = false;
    }

    public void StopMusic() {
        _activeMusic.Stop();
        _inactiveMusic.Stop();
    }
    ...
```

Penambahan pertama adalah variabel untuk sumber musik kedua. Sambil mempertahankan objek AudioSource pertama, duplikat objek itu (pastikan pengaturannya sama—pilih Loop) lalu drag objek baru ke dalam slot Inspector ini. Kode juga mendefinisikan variabel AudioSource aktif dan tidak aktif tetapi itu adalah variabel personal yang digunakan dalam kode dan tidak diekspos di Inspector. Secara khusus, variabel tersebut menentukan mana dari dua sumber audio yang dianggap "aktif" atau "tidak aktif" pada waktu tertentu.

Kode sekarang memanggil coroutine saat memutar musik baru. Coroutine ini menyetel musik baru yang diputar di satu AudioSource sementara musik lama terus diputar di sumber audio lama. Kemudian coroutine secara bertahap meningkatkan volume musik baru sambil secara bertahap mengurangi volume musik lama. Setelah cross-fading selesai (yaitu, volume telah sepenuhnya bertukar tempat), fungsi menukar sumber audio mana yang dianggap "aktif" dan "tidak aktif." Kita elah menyelesaikan musik latar untuk sistem audio game kami.

FMOD: alat untuk audio game

Sistem audio di Unity didukung oleh FMOD, perpustakaan pemrograman audio yang populer. Perpustakaan tersedia di www.fmod.org, tetapi sudah terintegrasi ke dalam Unity. Unity memiliki banyak fitur FMOD terintegrasi, meskipun tidak memiliki fitur perpustakaan yang paling canggih (Anda dapat mengunjungi situs web mereka untuk mempelajari tentang fitur tersebut).

Fitur audio canggih tersebut ditawarkan melalui FMOD Studio (plug-in yang menambahkan lebih banyak fungsionalitas ke Unity), tetapi contoh dalam bab ini akan tetap pada fungsionalitas yang ada di Unity. Fungsionalitas inti itu terdiri dari fitur paling penting untuk sistem audio game. Sebagian besar developer game memiliki kebutuhan audio yang dilayani dengan cukup baik oleh fungsi inti ini, tetapi plug-in berguna bagi mereka yang ingin lebih rumit dengan audio game mereka.

9.8 MENYATUKAN BAGIAN-BAGIAN MENJADI GAME YANG LENGKAP

Merakit aset dan kode dari beberapa proyek

Baiklah, modifikasi pertama adalah memperbarui kerangka kerja manajer dan membawa musuh yang dikendalikan komputer. Untuk tugas sebelumnya, ingat bahwa

pembaruan kerangka kerja dibuat di bab 9, yang berarti pembaruan tersebut tidak ada dalam proyek dari bab 8. Untuk tugas terakhir, ingatlah bahwa Anda memprogram musuh di bab 3.

Memperbarui kerangka kerja manajer

Memperbarui manajer adalah tugas yang cukup sederhana, jadi mari kita selesaikan dulu. Antarmuka IGameManager telah dimodifikasi di bab 9 (lihat daftar berikutnya).

IGameManager yang Disesuaikan

```
public interface IGameManager {
ManagerStatus status {get;}
void Startup(NetworkService service);
}
```

Kode dalam daftar ini menambahkan referensi ke NetworkService, jadi pastikan juga untuk menyalin skrip tambahan itu; drag file dari lokasinya di proyek bab 9 (ingat, proyek Unity adalah folder di disk Anda, jadi dapatkan file dari sana) dan letakkan di new project. Sekarang ubah Managers.cs agar berfungsi dengan antarmuka yang diubah (lihat daftar berikut).

Mengubah sedikit kode dalam skrip Manajer

```
...
private IEnumerator StartupManagers() {
   NetworkService network = new NetworkService();

   foreach (IGameManager manager in _startSequence)
      manager.Startup(network);
   }
```

Terakhir, sesuaikan InventoryManager dan PlayerManager untuk mencerminkan antarmuka yang diubah. Daftar berikutnya menunjukkan kode yang dimodifikasi dari InventoryManager; PlayerManager membutuhkan modifikasi kode yang sama tetapi dengan nama yang berbeda.

Menyesuaikan InventoryManager untuk mencerminkan IGameManager

```
...
private NetworkService _network;

public void Startup(NetworkService service) {
   Debug.Log("Inventory manager starting...");

   _network = service;

   _items = new Dictionary<string, int>();
   ...
```

Salinan pertama skrip ini (ingat, WanderingAI dan ReactiveTarget adalah perilaku untuk musuh AI, Fireball adalah proyektil yang ditembakkan, musuh menyerang komponen PlayerCharacter, dan SceneController menangani musuh yang bertelur):

- PlayerCharacter.cs
- SceneController.cs
- WanderingAI.cs
- ReactiveTarget.cs
- Fireball.cs

Demikian pula, dapatkan bahan Api, Prefab Bola Api, dan Prefab Musuh dengan menyeret file-file itu ke dalam. Jika Anda mendapatkan musuh dari bab 10, bukan 3, Anda juga memerlukan

bahan partikel api tambahan. Setelah menyalin semua aset yang diperlukan, tautan antar aset mungkin akan terputus, jadi Anda harus menautkan ulang semuanya agar dapat berfungsi. Secara khusus, skrip mungkin tidak terhubung dengan benar ke Prefab. Misalnya, prefab Musuh memiliki dua skrip yang hilang di Inspector, jadi klik tombol lingkaran (ditunjukkan pada gambar 9.6) untuk memilih WanderingAI dan ReactiveTarget dari daftar skrip. Demikian pula, periksa Prefab Fireball dan tautkan kembali skrip itu jika diperlukan. Setelah Anda selesai dengan skrip, periksa tautan ke bahan dan tekstur.

Gambar 9.6 Menautkan skrip ke komponen

Sekarang tambahkan SceneController.cs ke objek controller dan drag prefab Enemy ke slot Enemy komponen itu di Inspector. Anda mungkin perlu menyeret Prefab Fireball ke komponen skrip Musuh (pilih Prefab Musuh dan lihat WanderingAI di Inspector). Lampirkan juga PlayerCharacter.cs pada objek player sehingga musuh akan menyerang player. Mainkan gamenya dan Anda akan melihat musuh berkeliaran. Musuh menembakkan bola api ke player, meskipun itu tidak akan menimbulkan banyak kerusakan; pilih prefab Fireball dan atur nilai Damagenya menjadi 10.

Saat ini musuh tidak terlalu pandai melacak dan memukul player. Dalam hal ini, saya akan mulai dengan memberi musuh bidang pandang yang lebih luas. Namun, pada akhirnya, Anda akan menghabiskan banyak waktu untuk memoles permainan, dan itu termasuk mengulangi perilaku musuh. Memoles game agar lebih menyenangkan, meskipun penting untuk game yang akan dirilis, bukanlah sesuatu yang akan Anda lakukan dalam buku ini. Masalah lainnya adalah ketika Anda menulis kode ini di bab 3, kesehatan player adalah hal yang ad hoc untuk pengujian. Sekarang gim ini memiliki PlayerManager yang sebenarnya, jadi ubah PlayerCharacter sesuai dengan daftar berikutnya untuk bekerja dengan kesehatan di manajer itu.

Menyesuaikan PlayerCharacter untuk menggunakan kesehatan di PlayerManager

```
using UnityEngine;
using System.Collections;

public class PlayerCharacter : MonoBehaviour {
   public void Hurt(int damage) {
      Managers.Player.ChangeHealth(-damage);
   }
}
```

Pada titik ini Anda memiliki demo permainan dengan potongan-potongan yang dikumpulkan dari beberapa proyek sebelumnya. Karakter musuh telah ditambahkan ke scene, membuat permainan lebih mengancam. Tapi kontrol dan sudut pandangnya masih dari demo gerakan orang ketiga, jadi mari kita terapkan kontrol tunjuk dan klik untuk RPG aksi.

Memprogram kontrol titik-dan-klik: gerakan dan perangkat

Demo ini membutuhkan tampilan top-down dan kontrol mouse dari pergerakan player. Saat ini kamera merespons mouse, sedangkan player merespons keyboard (yaitu, apa yang diprogram dalam bab 7), yang merupakan kebalikan dari apa yang Anda inginkan dalam bab ini. Selain itu, Anda akan memodifikasi monitor yang berubah warna sehingga perangkat

dioperasikan dengan mengkliknya. Dalam kedua kasus, kode yang ada tidak terlalu jauh dari yang Anda butuhkan; Anda akan membuat penyesuaian pada skrip gerakan dan perangkat.

Scene dari atas ke bawah

Pertama, Anda akan menaikkan kamera ke 8 Y untuk memposisikannya agar terlihat di atas kepala. Anda juga akan menyesuaikan OrbitCamera untuk menghapus kontrol mouse dari kamera dan hanya menggunakan tombol panah (lihat daftar berikut).

Menyesuaikan OrbitCamera untuk menghapus kontrol mouse

```
...
void LateUpdate() {
    _rotY -= Input.GetAxis("Horizontal") * rotSpeed;
    Quaternion rotation = Quaternion.Euler(0, _rotY, 0),
    transform.position = target.position - (rotation * _offset);
    transform.LookAt(target);
}
...
```

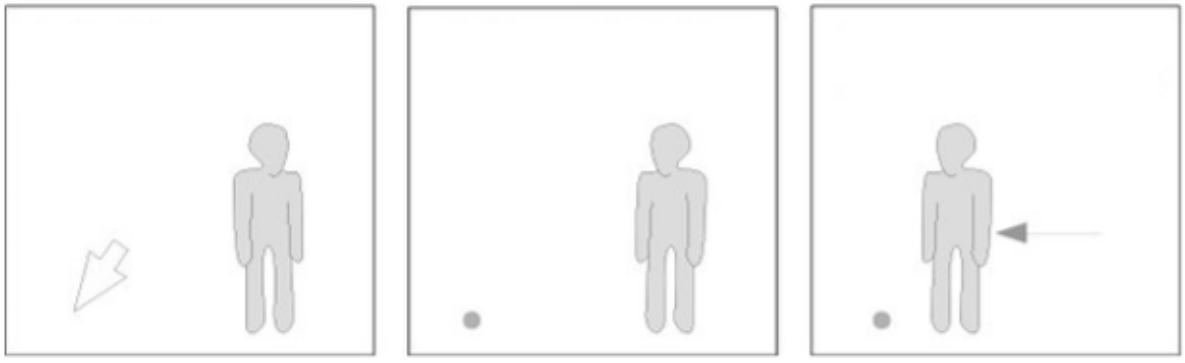
Bidang kliping Dekat/Jauh kamera

Selama Anda menyesuaikan kamera, saya ingin menunjukkan bidang kliping Dekat/Jauh. Pengaturan ini tidak pernah muncul sebelumnya karena defaultnya baik-baik saja, tetapi Anda mungkin perlu menyesuaikannya di beberapa proyek mendatang. Pilih kamera di scene dan cari bagian Clipping Planes di Inspector; baik Dekat dan Jauh adalah angka yang akan Anda ketik di sini. Nilai-nilai ini menentukan batas dekat dan jauh di mana mesh dirender: poligon lebih dekat dari bidang kliping Dekat atau lebih jauh dari bidang kliping Jauh tidak digambar. Anda ingin bidang kliping Dekat/Jauh sedekat mungkin sambil tetap cukup berjauhan untuk membuat semua yang ada di scene Anda. Ketika bidang-bidang itu terlalu berjauhan (Dekat terlalu dekat atau Jauh terlalu jauh), algoritme rendering tidak dapat lagi membedakan poligon mana yang lebih dekat. Ini menghasilkan kesalahan rendering karakteristik yang disebut z-fighting (seperti pada sumbu Z untuk kedalaman) di mana poligon berkedip di atas satu sama lain.

Dengan kamera yang dinaikkan lebih tinggi, tampilan saat Anda memainkan game akan menjadi top-down. Namun, saat ini, kontrol gerakan masih menggunakan keyboard, jadi mari kita menulis skrip untuk gerakan tunjuk dan klik.

Menulis kode gerakan

Gagasan umum untuk kode ini (diilustrasikan pada gambar 9.7) adalah untuk secara otomatis memindahkan player ke posisi targetnya. Posisi ini diatur dengan mengklik di scene. Dengan cara ini, kode yang menggerakkan player tidak langsung bereaksi terhadap mouse tetapi gerakan player dikendalikan secara tidak langsung dengan mengklik.



Gambar 9.7 Diagram cara kerja kontrol arahkan dan klik.

Algoritma gerakan ini juga berguna untuk karakter AI. Alih-alih menggunakan klik mouse, posisi target bisa berada di jalur yang diikuti karakter. Untuk menerapkan ini, buat skrip baru bernama PointClickMovement dan ganti komponen RelativeMovement pada pemutar. Mulai coding PointClickMovement dengan menempelkan keseluruhan RelativeMovement (karena Anda masih menginginkan sebagian besar skrip untuk menangani jatuh dan animasi). Kemudian sesuaikan kodenya dengan listing selanjutnya.

Kode gerakan baru dalam skrip PointClickMovement

```
...
public class PointClickMovement : MonoBehaviour {
...
public float deceleration = 20.0f;
public float targetBuffer = 1.5f;
private float _curSpeed = 0f;
private Vector3 _targetPos = Vector3.one;
...
void Update() {
   Vector3 movement = Vector3.zero;

   if (Input.GetMouseButton(0)) {
      Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
      RaycastHit mouseHit;
      if (Physics.Raycast(ray, out mouseHit)) {
         _targetPos = mouseHit.point;
         _curSpeed = moveSpeed;
      }
   }

   if (_targetPos != Vector3.one) {
      Vector3 adjustedPos = new Vector3(_targetPos.x,

         transform.position.y, _targetPos.z);
      Quaternion targetRot = Quaternion.LookRotation(
         adjustedPos - transform.position);
      transform.rotation = Quaternion.Slerp(transform.rotation,
         targetRot, rotSpeed * Time.deltaTime);

     movement = _curSpeed * Vector3.forward;
     movement = transform.TransformDirection(movement);
     if (Vector3.Distance(_targetPos, transform.position) < targetBuffer) {
         _curSpeed -= deceleration * Time.deltaTime;
         if (_curSpeed <= 0) {
            _targetPos = Vector3.one;
         }
     }
   }
   _animator.SetFloat("Speed", movement.sqrMagnitude);
   ...
```

Hampir semua yang ada di awal metode Update() dihilangkan, karena kode tersebut menangani pergerakan keyboard. Perhatikan bahwa kode baru ini memiliki dua pernyataan if utama: satu yang dijalankan saat mouse diklik, dan satu lagi yang dijalankan saat target ditetapkan.

Saat mouse diklik, atur target sesuai dengan tempat mouse diklik. Inilah kegunaan hebat lainnya untuk raycasting: untuk menentukan titik mana dalam scene yang berada di bawah kursor mouse. Posisi target diatur ke tempat mouse menyentuh.

Adapun syarat kedua, pertama rotate menghadap sasaran. Quaternion.Slerp() berputar dengan mulus untuk menghadap target, daripada langsung bergerak ke rotasi itu. Kemudian, ubah arah maju dari koordinat lokal player ke koordinat global (untuk bergerak maju). Terakhir, periksa jarak antara player dan target: jika player hampir mencapai target, kurangi kecepatan gerakan dan akhiri gerakan dengan menghapus posisi target.

Latihan: Matikan kontrol lompat

Saat ini skrip ini masih memiliki kontrol lompatan dari RelativeMovement. Player masih melompat ketika spasi ditekan, tetapi seharusnya tidak ada tombol lompat dengan gerakan tunjuk dan klik. Berikut petunjuknya: sesuaikan kode di dalam cabang kondisional 'if (hitGround)'. Ini menangani pemindahan player menggunakan kontrol mouse. Mainkan game untuk mengujinya. Selanjutnya mari kita buat perangkat beroperasi saat diklik.

Mengoperasikan perangkat menggunakan mouse

Dalam bab 8 (dan di sini sampai kita menyesuaikan kode), perangkat dioperasikan dengan menekan sebuah tombol. Sebaliknya, mereka harus beroperasi saat diklik. Untuk melakukan ini, pertama-tama Anda akan membuat skrip dasar yang akan diwarisi semua perangkat; skrip dasar akan memiliki kontrol mouse, dan perangkat akan mewarisinya. Buat skrip baru bernama BaseDevice dan tulis kode dari daftar berikut.

Daftar Skrip BaseDevice yang beroperasi saat diklik

```
using UnityEngine;
using System.Collections;

public class BaseDevice : MonoBehaviour {
   public float radius = 3.5f;
   void OnMouseDown() {
      Transform player = GameObject.FindWithTag("Player").transform;
      if (Vector3.Distance(player.position, transform.position) < radius) {
         Vector3 direction = transform.position - player.position;
         if (Vector3.Dot(player.forward, direction) > .5f) {
            Operate();
         }
      }
   }

   public virtual void Operate() {
      // behavior of the specific device
   }
}
```

Sebagian besar kode ini terjadi di dalam OnMouseDown() karena. MonoBehaviour memanggil metode itu ketika objek diklik. Pertama, ia memeriksa jarak ke pemutar, dan kemudian menggunakan produk titik untuk melihat apakah player menghadap perangkat. Operate() adalah shell kosong yang harus diisi oleh perangkat yang mewarisi skrip ini. Kode ini terlihat dalam scene untuk objek dengan tag Player, jadi tetapkan tag ini ke objek player. Tag adalah menu tarik-turun di bagian atas Inspector; Anda juga dapat menentukan tag khusus, tetapi beberapa tag ditentukan secara default, termasuk Player. Pilih objek pemutar untuk

diedit, lalu pilih tag Player. Sekarang BaseDevice diprogram, Anda dapat memodifikasi ColorChangeDevice untuk mewarisi dari skrip itu. Daftar berikut menunjukkan kode baru.

Menyesuaikan ColorChangeDevice untuk mewarisi dari BaseDevice

```
using UnityEngine;
using System.Collections;

public class ColorChangeDevice : BaseDevice {
   public override void Operate() {
      Color random = new Color(Random.Range(0f,1f),
         Random.Range(0f,1f), Random.Range(0f,1f));
      GetComponent<Renderer>().material.color = random;
   }
}
```

Karena skrip ini mewarisi dari BaseDevice alih-alih MonoBehaviour, skrip ini mendapatkan fungsionalitas kontrol mouse. Kemudian menimpa metode Operate() yang kosong untuk memprogram perilaku perubahan warna. Sekarang perangkat akan beroperasi ketika Anda mengkliknya. Hapus juga komponen skrip Operator-Perangkat pemutar, karena skrip tersebut mengoperasikan perangkat menggunakan tombol kontrol. Input perangkat baru ini memunculkan masalah dengan kontrol gerakan: saat ini target gerakan disetel setiap kali mouse diklik, tetapi Anda tidak ingin menetapkan target gerakan saat mengklik perangkat. Anda dapat memperbaiki masalah ini dengan menggunakan layer; mirip dengan bagaimana tag ditetapkan pada pemutar, objek dapat diatur ke layer yang berbeda dan kode dapat memeriksanya. Sesuaikan PointClickMovement untuk memeriksa layer objek (lihat daftar berikutnya).

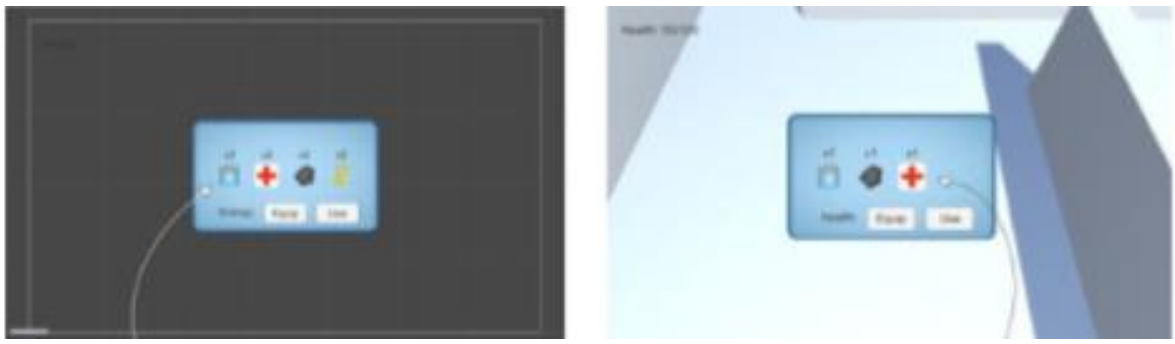
Menyesuaikan kode klik mouse di PointClickMovement

```
...
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit mouseHit;
if (Physics.Raycast(ray, out mouseHit)) {
   GameObject hitObject = mouseHit.transform.gameObject;
   if (hitObject.layer == LayerMask.NameToLayer("Ground")) {
      _targetPos = mouseHit.point;
      _curSpeed = moveSpeed;
   }
}
...
```

Daftar ini menambahkan kondisi di dalam kode klik mouse untuk melihat apakah objek yang diklik berada di layer Ground. Layer (seperti Tag) adalah menu tarik-turun di bagian atas Inspector; klik untuk melihat opsi. Juga seperti tag, beberapa layer sudah ditentukan secara default. Anda ingin membuat layer baru, jadi pilih Edit Layers di menu. Ketik Ground di slot layer kosong (mungkin slot 8; NameToLayer() dalam kode mengubah nama menjadi nomor layer sehingga Anda dapat menyebutkan nama alih-alih nomor). Sekarang layer Ground telah ditambahkan ke menu, atur objek ground ke layer Ground—itu berarti lantai bangunan, bersama dengan landai dan platform tempat player dapat berjalan. Pilih objek tersebut, lalu pilih Ground di menu Layers.

Mainkan game dan Anda tidak akan bergerak saat mengklik monitor yang berubah warna. Hebat, kontrol arahkan dan klik selesai! Satu hal lagi yang dibawa ke dalam proyek ini dari proyek-proyek sebelumnya adalah UI yang ditingkatkan.

Mengganti GUI lama dengan antarmuka baru



Gambar 9.8 Gambar UI untuk proyek bab ini

Pertama, Anda akan mengatur grafik UI. Setelah semua gambar UI ada di layar, Anda dapat melampirkan skrip ke objek UI. Saya akan mencantumkan langkah-langkah yang terlibat tanpa merinci; jika Anda membutuhkan penyegaran, lihat kembali bab 6:

1. Impor popup.png sebagai sprite (pilih Jenis Tekstur).
2. Di Editor Sprite, atur batas 12-piksel di semua sisi (ingat untuk menerapkan perubahan).
   3. Buat kanvas di scene (GameObject > UI > Canvas).
3. Pilih pengaturan Pixel Perfect dari kanvas.
4. Opsional: Beri nama objek HUD Canvas dan alihkan ke mode tampilan 2D.
5. Buat objek Teks yang terhubung ke kanvas itu (GameObject > UI > Text).
6. Atur jangkar objek Teks ke kiri atas dan posisikan 100, -40.
7. Ketik Kesehatan: seperti teks pada label.
8. Buat gambar yang terhubung ke kanvas itu (GameObject > UI > Image).
9. Beri nama objek baru Inventory Popup.
10. Tetapkan sprite pop-up ke Gambar Sumber gambar.
11. Atur Jenis Gambar ke Irisan dan pilih Isi Pusat.
12. Posisikan gambar pop-up pada 0, 0 dan scalekan pop-up ke 250 lebar 150 tinggi.

Ingat bagaimana beralih antara melihat scene 3D dan antarmuka 2D: beralih mode tampilan 2D dan klik dua kali Kanvas atau Gedung untuk memperbesar objek itu. Sekarang Anda memiliki label Kesehatan di sudut dan jendela pop-up biru besar di tengah. Mari kita programkan bagian-bagian ini terlebih dahulu sebelum masuk lebih dalam ke fungsionalitas UI. Kode antarmuka akan menggunakan sistem Messenger yang sama dari bab 6, jadi salin skrip Messenger. Kemudian buat skrip GameEvent (lihat daftar berikut).

Skrip GameEvent untuk digunakan dengan sistem Messenger ini

```
public static class GameEvent {
public const string HEALTH_UPDATED = "HEALTH_UPDATED";
}
```

Untuk saat ini hanya satu peristiwa yang didefinisikan; selama bab ini Anda akan menambahkan beberapa event lagi. Siarkan event ini dari PlayerManager.cs (ditampilkan di daftar berikutnya).

Menyiarkan event kesehatan dari PlayerManager.cs

```
...
public void ChangeHealth(int value) {
   health += value;
   if (health > maxHealth) {
      health = maxHealth;
   } else if (health < 0) {
      health = 0;
   }

   Messenger.Broadcast(GameEvent.HEALTH_UPDATED);
}
...
```

Event disiarkan setiap kali ChangeHealth() selesai untuk memberi tahu program lainnya bahwa kesehatan telah berubah. Anda ingin menyesuaikan label kesehatan dalam menanggapi peristiwa ini, jadi buat skrip UIController (lihat daftar berikutnya).

Skrip UIController, yang menangani antarmuka

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class UIController : MonoBehaviour {
   [SerializeField] private Text healthLabel;
   [SerializeField] private InventoryPopup popup;

   void Awake() {
      Messenger.AddListener(GameEvent.HEALTH_UPDATED, OnHealthUpdated);
   }
   void OnDestroy() {
      Messenger.RemoveListener(GameEvent.HEALTH_UPDATED, OnHealthUpdated);
   }

   void Start() {
      OnHealthUpdated();

      popup.gameObject.SetActive(false);
   }


   void Update() {
      if (Input.GetKeyDown(KeyCode.M)) {
         bool isShowing = popup.gameObject.activeSelf;
         popup.gameObject.SetActive(!isShowing);
         popup.Refresh();
      }
   }

   private void OnHealthUpdated() {
      string message = "Health: " + Managers.Player.health + "/" +
     Managers.Player.maxHealth;
      healthLabel.text = message;
   }
}
```

Lampirkan skrip ini ke objek Controller dan hapus BasicUI. Juga, buat skrip InventoryPopup (tambahkan metode Refresh() publik kosong untuk saat ini; sisanya akan diisi nanti) dan lampirkan ke jendela pop-up (objek Gambar). Sekarang Anda dapat menyeret pop-up ke slot referensi di komponen Controller; juga menautkan label kesehatan ke Controller.

Label kesehatan berubah saat Anda terluka atau menggunakan paket kesehatan, dan menekan M akan mengaktifkan jendela pop-up. Satu detail terakhir yang harus disesuaikan adalah bahwa saat ini mengklik jendela pop-up menyebabkan player bergerak; seperti halnya perangkat, Anda tidak ingin menetapkan posisi target saat UI telah diklik. Buat penyesuaian yang ditunjukkan pada daftar berikutnya ke PointClickMovement.

Memeriksa UI di PointClickMovement

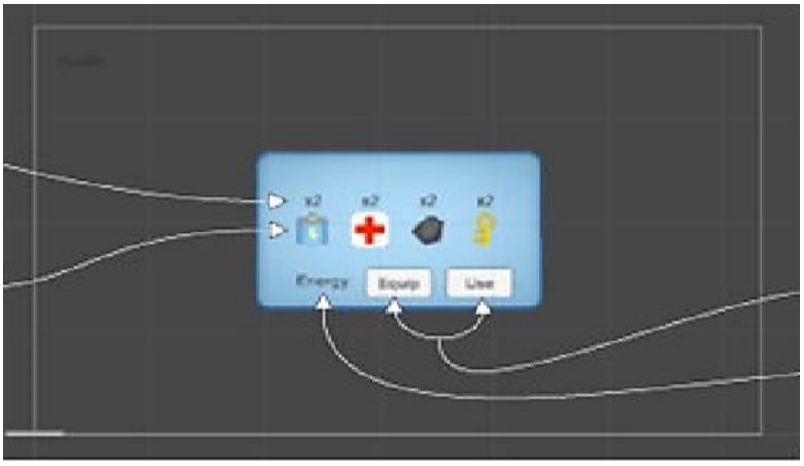
```
using UnityEngine.EventSystems;
...
void Update() {
Vector3 movement = Vector3.zero;
if (Input.GetMouseButton(0) &&
!EventSystem.current.IsPointerOverGameObject()) {
...
```

Perhatikan bahwa kondisi memeriksa apakah mouse ada di UI atau tidak. Itu melengkapi keseluruhan struktur antarmuka, jadi sekarang mari kita berurusan dengan pop-up inventaris secara khusus.

Menerapkan pop-up Inventaris

Jendela pop-up saat ini kosong tetapi seharusnya menampilkan inventaris player (digambarkan dalam gambar 9.9). Langkah-langkah ini akan membuat objek UI:

1. Buat empat gambar dan indukkan ke pop-up (yaitu, drag objek dalam Hierarki).
2. Buat empat label teks dan masukkan ke pop-up.
3. Posisikan semua gambar pada nilai 0 Y dan X -75, -25, 25, dan 75.
4. Posisikan label teks pada nilai 50 Y dan X -75, -25, 25, dan 75.
5. Atur teks (bukan jangkar!) ke Perataan tengah, Perataan vertikal bawah, dan Tinggi 60.
6. Di Sumber Daya, atur semua ikon inventaris sebagai Sprite (bukan Tekstur).
7. Drag sprite ini ke slot Gambar Sumber dari objek Gambar (juga atur Ukuran Asli).
8. Masukkan x2 untuk semua label teks.
9. Tambahkan label teks lain dan dua tombol, semuanya di-parent ke pop-up.
10. Posisikan label teks ini pada -120, -55 dan atur Right alignment.
11. Ketik Energi: untuk teks pada label ini
12. Atur kedua tombol ke Lebar 60, lalu Posisikan pada -50 Y dan nilai X 0 atau 70.
13. Ketik Equip pada satu tombol dan Use pada tombol lainnya.



Gambar 9.9 Diagram UI inventaris

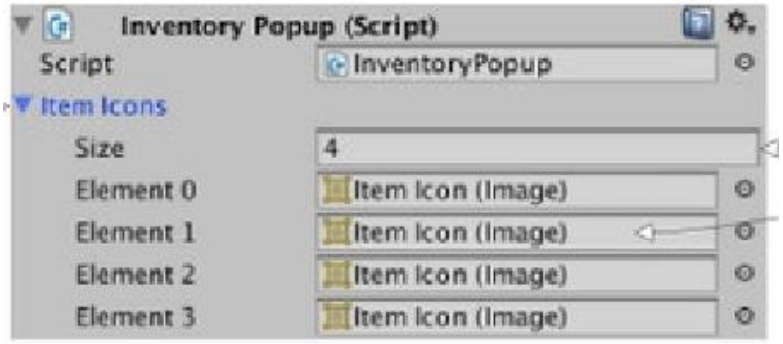
Ini adalah elemen visual untuk pop-up inventaris; selanjutnya adalah kode. Tulis isi daftar berikut ke dalam skrip InventoryPopup.

Skrip lengkap untuk InventoryPopup

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using System.Collections;
using System.Collections.Generic;

public class InventoryPopup : MonoBehaviour {
    [SerializeField] private Image[] itemIcons;
    [SerializeField] private Text[] itemLabels;

    [SerializeField] private Text curItemLabel;
    [SerializeField] private Button equipButton;
    [SerializeField] private Button useButton;

    private string _curItem;

    public void Refresh() {
        List<string> itemList = Managers.Inventory.GetItemList();

        int len = itemIcons.Length;
        for (int i = 0; i < len; i++) {
            if (i < itemList.Count) {
                itemIcons[i].gameObject.SetActive(true);
                itemLabels[i].gameObject.SetActive(true);

            string item = itemList[i];

            Sprite sprite = Resources.Load<Sprite>("Icons/"+item);
            itemIcons[i].sprite = sprite;
            itemIcons[i].SetNativeSize();

            int count = Managers.Inventory.GetItemCount(item);
            string message = "x" + count;
            if (item == Managers.Inventory.equippedItem) {
                message = "Equipped\n" + message;
            }
            itemLabels[i].text = message;
            EventTrigger.Entry entry = new EventTrigger.Entry();
            entry.eventID = EventTriggerType.PointerClick;
            entry.callback.AddListener((BaseEventData data) => {
                OnItem(item);
            });

            EventTrigger trigger = itemIcons[i].GetComponent<EventTrigger>();
            trigger.delegates.Clear();
            trigger.delegates.Add(entry);
        }
        else {
            itemIcons[i].gameObject.SetActive(false);
            itemLabels[i].gameObject.SetActive(false);
        }
    }
}
```

```
        if (!itemList.Contains(_curItem)) {
            _curItem = null;
        }
        if (_curItem == null) {
            curItemLabel.gameObject.SetActive(false);
            equipButton.gameObject.SetActive(false);
            useButton.gameObject.SetActive(false);
        }
        else {
            curItemLabel.gameObject.SetActive(true);
            equipButton.gameObject.SetActive(true);
            if (_curItem == "health") {
                useButton.gameObject.SetActive(true)
            } else {
                useButton.gameObject.SetActive(false);
            }

            curItemLabel.text = _curItem+":";
        }
    }

    public void OnItem(string item) {
        _curItem = item;
        Refresh();
    }

    public void OnEquip() {
        Managers.Inventory.EquipItem(_curItem);
        Refresh();
    }

    public void OnUse() {
        Managers.Inventory.ConsumeItem(_curItem);
        if (_curItem == "health") {
            Managers.Player.ChangeHealth(25);
        }
        Refresh();
    }
}
```

Wah, itu script yang panjang! Dengan ini diprogram, saatnya untuk menghubungkan semua yang ada di antarmuka. Komponen skrip sekarang memiliki berbagai referensi objek, termasuk dua larik; perluas kedua larik dan atur ke panjang 4 (lihat gambar 9.10). Drag empat gambar ke larik ikon, dan drag empat label teks ke larik label.



Gambar 9.10 Array ditampilkan di Inspector

Jika Anda tidak yakin objek mana yang ditautkan di mana (semuanya terlihat sama), klik slot di Inspector untuk melihat objek tersebut disorot dalam tampilan Hierarchy. Demikian pula, slot di komponen merujuk ke label teks dan tombol di bagian bawah pop-up. Setelah menautkan objek tersebut, Anda akan menambahkan pendengar OnClick untuk kedua tombol. Tautkan peristiwa ini ke objek pop-up, dan pilih OnEquip() atau OnUse() yang sesuai.

Terakhir, tambahkan komponen EventTrigger ke keempat gambar item. Skrip InventoryPopup memodifikasi komponen ini pada setiap ikon, jadi mereka lebih baik memiliki komponen ini! Anda akan menemukan EventTrigger di bawah Add Component > Event (mungkin lebih mudah untuk menyalin/menempelkan komponen dengan mengklik tombol roda gigi kecil di sudut atas komponen: pilih Salin Komponen dari satu objek lalu Tempel Sebagai Baru di objek lainnya). Tambahkan komponen ini tetapi jangan tetapkan pendengar event, karena itu dilakukan dalam kode InventoryPopup. Dan itu melengkapi UI inventaris! Mainkan game untuk melihat popup inventaris merespons saat Anda mengumpulkan item dan mengklik tombol. Kami sekarang telah selesai merakit bagian dari proyek sebelumnya; selanjutnya saya akan menjelaskan bagaimana membangun game yang lebih ekspansif dari awal ini.

9.9 MENGEMBANGKAN STRUKTUR PERMAINAN YANG MENYELURUH

Sekarang setelah Anda memiliki demo RPG aksi yang berfungsi, kami akan membangun struktur menyeluruh dari game ini. Maksud saya aliran keseluruhan permainan melalui berbagai level dan maju melalui permainan dengan mengalahkan level. Apa yang kami dapatkan dari proyek bab 8 adalah satu level, tetapi peta jalan untuk bab ini menetapkan tiga level.

Melakukan ini akan melibatkan pemisahan scene lebih jauh dari bagian belakang Manajer, jadi Anda akan menyiarkan pesan tentang manajer (seperti PlayerManager menyiarkan pembaruan kesehatan). Buat skrip baru bernama StartupEvent; tentukan event ini dalam skrip terpisah karena event ini berjalan dengan sistem Manajer yang dapat digunakan kembali, sedangkan GameEvent khusus untuk game.

Daftar Skrip Event Startup

```
public static class StartupEvent {
public const string MANAGERS_STARTED=
"MANAGERS_STARTED"; public const string
MANAGERS_PROGRESS = "MANAGERS_PROGRESS";
}
```

Sekarang saatnya untuk mulai menyesuaikan Manajer, termasuk menyiarkan event baru ini!

Mengontrol aliran misi dan berbagai level

Saat ini proyek hanya memiliki satu scene, dan objek Manajer Game ada di scene itu. Masalahnya adalah bahwa setiap scene akan memiliki set manajer permainannya sendiri, sedangkan Anda sebenarnya ingin satu set manajer permainan dibagikan oleh semua scene. Untuk melakukan itu, Anda akan membuat scene Startup terpisah yang menginisialisasi manajer dan kemudian membagikan objek itu dengan scene game lainnya.

Kami juga akan membutuhkan manajer baru untuk menangani kemajuan melalui permainan. Buat skrip baru bernama MissionManager (seperti yang ditunjukkan pada daftar berikutnya).

Daftar Manajer Misi

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class MissionManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   public int curLevel {get; private set;}
   public int maxLevel {get; private set;}

   private NetworkService _network;

   public void Startup(NetworkService service) {
      Debug.Log("Mission manager starting...");

      _network = service;

      curLevel = 0;
      maxLevel = 1;


      status = ManagerStatus.Started;
   }

   public void GoToNext() {
      if (curLevel < maxLevel) {
         curLevel++;
         string name = "Level" + curLevel;
         Debug.Log("Loading " + name);
         Application.LoadLevel(name);
      } else {
         Debug.Log("Last level");
      }
   }
}
```

Untuk sebagian besar, tidak ada yang tidak biasa terjadi dalam daftar ini, tetapi perhatikan metode LoadLevel() di dekat bagian akhir; meskipun saya menyebutkan metode itu sebelumnya (di bab 5), itu tidak penting sampai sekarang. Itulah metode Unity untuk memuat file scene; di bab 5 Anda menggunakannya untuk memuat ulang satu scene dalam game, tetapi Anda dapat memuat scene apa pun dengan memasukkan nama file scene. Lampirkan skrip ini ke objek Game Managers di scene. Juga tambahkan komponen baru ke skrip Manajer (lihat daftar berikut).

Daftar Menambahkan komponen baru ke skrip Manajer

```
...
[RequireComponent(typeof(MissionManager))]

public class Managers : MonoBehaviour {
   public static PlayerManager Player {get; private set;}
   public static InventoryManager Inventory {get; private set;}
   public static MissionManager Mission {get; private set;}
   ...
   void Awake() {
      DontDestroyOnLoad(gameObject);

      Player = GetComponent<PlayerManager>();
      Inventory = GetComponent<InventoryManager>();
      Mission = GetComponent<MissionManager>();

      _startSequence = new List<IGameManager>();
      _startSequence.Add(Player);
      _startSequence.Add(Inventory);
      _startSequence.Add(Mission);


      StartCoroutine(StartupManagers());
   }

   private IEnumerator StartupManagers() {
      ...
        if (numReady > lastReady) {
          Debug.Log("Progress: " + numReady + "/" + numModules);
          Messenger<int, int>.Broadcast(
            StartupEvent.MANAGERS_PROGRESS, numReady, numModules);
        }

        yield return null;
      }

      Debug.Log("All managers started up");
      Messenger.Broadcast(StartupEvent.MANAGERS_STARTED);
   }
   ...
```

Sebagian besar kode ini seharusnya sudah Anda kenal (menambahkan MissionManager sama seperti menambahkan manajer lain), tetapi ada dua bagian baru. Salah satunya adalah kejadian yang mengirimkan dua nilai integer; Anda melihat peristiwa dan pesan generik yang tidak bernilai dengan satu nomor sebelumnya, tetapi Anda dapat mengirim sejumlah nilai arbitrer dengan sintaks yang sama.

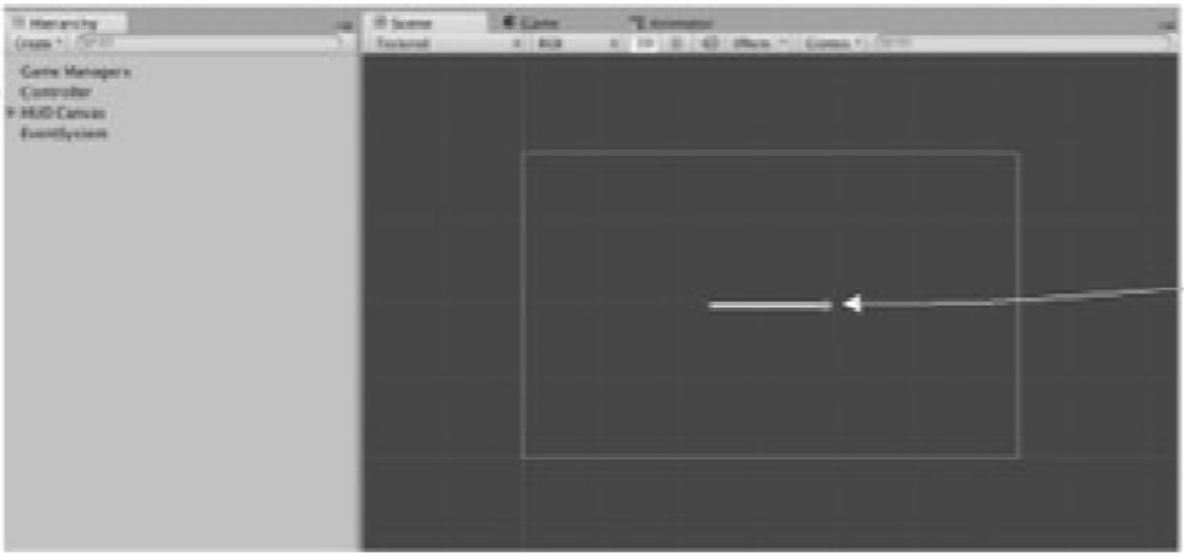
Bit kode baru lainnya adalah metode DontDestroyOnLoad() . Ini adalah metode yang disediakan oleh Unity untuk mempertahankan objek di antara scene. Biasanya semua objek dalam sebuah scene dibersihkan saat scene baru dimuat, tetapi dengan menggunakan DontDestroyOnLoad() pada objek, Anda memastikan bahwa objek itu akan tetap ada di scene baru.

Scene terpisah untuk startup dan level

Karena objek Game Managers akan tetap ada di semua scene, Anda harus memisahkan manajer dari level individual game. Dalam tampilan Proyek, duplikat file scene (Edit > Duplikat) lalu ganti nama kedua file dengan tepat: satu Startup dan yang lainnya Level1. Buka Level1 dan hapus objek Game Managers (akan disediakan oleh Startup). Buka Startup dan hapus semuanya selain Game Managers, Controller, HUD Canvas, dan EventSystem. Hapus

komponen skrip di Controller, dan hapus objek UI (label kesehatan dan InventoryPopup) yang di-parent ke Canvas.

UI saat ini kosong, jadi buat penggeser baru (lihat gambar 9.11) lalu matikan pengaturan Interactable-nya. Objek Controller juga tidak memiliki komponen skrip lagi, jadi buat skrip StartupController baru dan lampirkan ke objek Controller (lihat daftar berikut).



Gambar 9.11 Scene Startup dengan semua yang tidak perlu dihapus

Skrip StartupController baru

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class StartupController : MonoBehaviour {
   [SerializeField] private Slider progressBar;

   void Awake() {
      Messenger<int, int>.AddListener(StartupEvent.MANAGERS_PROGRESS,
         OnManagersProgress);
      Messenger.AddListener(StartupEvent.MANAGERS_STARTED,
         OnManagersStarted);
   }
   void OnDestroy() {
      Messenger<int, int>.RemoveListener(StartupEvent.MANAGERS_PROGRESS,
         OnManagersProgress);
      Messenger.RemoveListener(StartupEvent.MANAGERS_STARTED,
         OnManagersStarted);
   }

    private void OnManagersProgress(int numReady, int numModules) {
       float progress = (float)numReady / numModules;
       progressBar.value = progress;
    }

    private void OnManagersStarted() {
       Managers.Mission.GoToNext();
    }
}
```

Selanjutnya, tautkan objek Slider ke slot di Inspector. Satu hal terakhir yang harus dilakukan dalam persiapan adalah menambahkan dua scene ke Build Settings. Membangun

aplikasi akan menjadi topik bab berikutnya, jadi untuk saat ini pilih File > Build Settings untuk melihat dan menyesuaikan daftar scene. Klik tombol Add Current untuk menambahkan scene ke daftar (muat kedua scene dan lakukan ini untuk masing-masing).

Anda perlu menambahkan scene ke Pengaturan Bangun agar dapat dimuat. Jika tidak, Unity tidak akan tahu scene apa yang tersedia. Anda tidak perlu melakukan ini karena Anda tidak benar-benar berpindah level—Anda sedang memuat ulang scene saat ini. Sekarang Anda dapat meluncurkan game dengan menekan Play dari scene Startup. Objek Manajer Game akan dibagikan di kedua scene. Karena manajer dimuat di scene Startup, Anda selalu perlu meluncurkan game dari scene itu. Anda dapat mengingat untuk selalu membuka scene itu sebelum menekan Putar, tetapi ada skrip di wiki Unify yang akan secara otomatis beralih ke scene yang ditetapkan saat Anda mengklik Putar: http://wiki.unity3d.com/index.php/SceneAutoLoader. Perubahan struktural ini menangani pembagian manajer game di antara scene yang berbeda, tetapi Anda masih belum memiliki kondisi sukses atau gagal dalam level tersebut.

Menyelesaikan level dengan mencapai pintu keluar

Untuk menangani penyelesaian level, Anda akan meletakkan objek di scene untuk disentuh player, dan objek itu akan memberi tahu MissionManager saat player mencapai tujuannya. Ini akan melibatkan UI yang menanggapi pesan tentang penyelesaian level, jadi tambahkan GameEvent lain (lihat daftar berikut).

Daftar Level Selesai ditambahkan ke GameEvent.cs

```
Public static class GameEvent {
public const string HEALTH_UPDATED = "HEALTH_UPDATED";
public const string LEVEL_COMPLETE =
"LEVEL_COMPLETE";
}
```

Sekarang tambahkan metode baru ke MissionManager untuk melacak tujuan misi dan menyiarkan pesan event baru (lihat daftar berikutnya).

Daftar Metode objektif di MissionManager

```
...
public void ReachObjective() {
// could have logic to handle multiple objectives
Messenger.Broadcast(GameEvent.LEVEL_COMPLETE); }
...
```

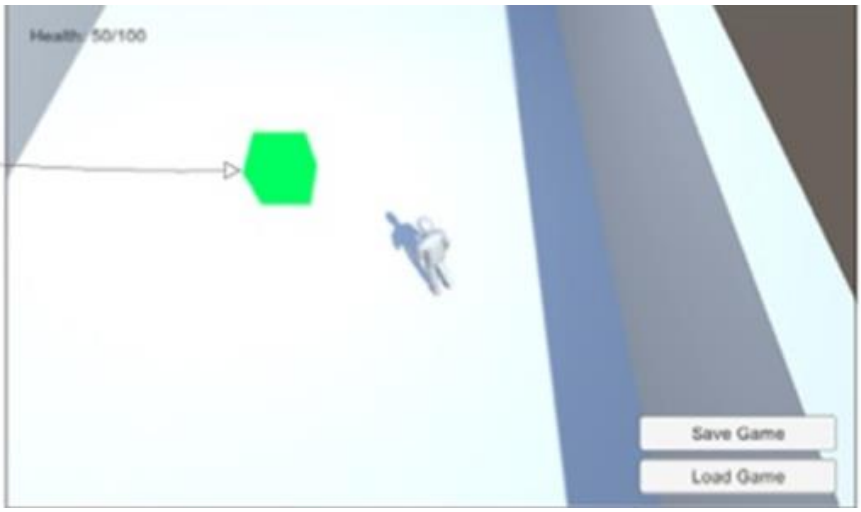
Sekarang tambahkan metode baru ke MissionManager untuk melacak tujuan misi dan menyiarkan pesan event baru (lihat daftar berikutnya).

Daftar Metode objektif di MissionManager

```
...
[SerializeField] private Text levelEnding;
...
void Awake() {
   Messenger.AddListener(GameEvent.HEALTH_UPDATED, OnHealthUpdated);
   Messenger.AddListener(GameEvent.LEVEL_COMPLETE, OnLevelComplete);
}
void OnDestroy() {
   Messenger.RemoveListener(GameEvent.HEALTH_UPDATED, OnHealthUpdated);
   Messenger.RemoveListener(GameEvent.LEVEL_COMPLETE, OnLevelComplete);
}
...
void Start() {
   OnHealthUpdated();

   levelEnding.gameObject.SetActive(false);
   popup.gameObject.SetActive(false);
}
...

private void OnLevelComplete() {
   StartCoroutine(CompleteLevel());
}
private IEnumerator CompleteLevel() {
   levelEnding.gameObject.SetActive(true);
   levelEnding.text = "Level Complete!";

   yield return new WaitForSeconds(2);

   Managers.Mission.GoToNext();
}
...
```

Anda akan melihat bahwa cantuman ini memiliki referensi ke label teks. Buka scene Level1 untuk mengeditnya, dan buat objek teks UI baru. Label ini akan menjadi pesan penyelesaian level yang muncul di tengah layar. Atur teks ini ke Width 240, Height 60, Center untuk Align dan Vertical-align, dan Font Size 22. Type Level Complete! di area teks dan kemudian tautkan objek teks ini ke referensi levelEnding dari UIController.

Terakhir, kita akan membuat objek yang disentuh player untuk menyelesaikan level. Ini akan mirip dengan barang koleksi: membutuhkan bahan dan skrip, dan Anda akan membuat semuanya menjadi Prefab.



Gambar 9.12 Objek objektif yang disentuh player untuk menyelesaikan level

Buat objek kubus di Posisi 18, 1, 0. Pilih opsi Is Trigger dari Box Collider, matikan Cast/Receive Shadows di Mesh Renderer, dan atur objek ke layer Ignore Raycast. Buat materi baru yang disebut tujuan; buat hijau terang dan atur shader ke Unlit > Color untuk tampilan datar dan cerah. Selanjutnya, buat skrip ObjectiveTrigger (ditunjukkan dalam daftar berikutnya) dan lampirkan skrip itu ke objek objektif.

Daftar Kode untuk ObjectiveTrigger untuk diletakkan di objek objektif

```
using UnityEngine;
using System.Collections;

public class ObjectiveTrigger : MonoBehaviour {
   void OnTriggerEnter(Collider other) {
      Managers.Mission.ReachObjective();
   }
}
```

Drag objek ini dari Hierarchy ke tampilan Project untuk mengubahnya menjadi prefab; di tingkat masa depan, Anda bisa menempatkan Prefab di scene. Sekarang mainkan gamenya dan raih tujuannya. Pesan penyelesaian muncul saat Anda mengalahkan level. Selanjutnya mari kita tampilkan pesan kegagalan saat Anda kalah.

Kehilangan level saat ditangkap oleh musuh

Kondisi gagal adalah ketika player kehabisan kesehatan (karena musuh menyerang). Pertama tambahkan GameEvent lain:

Public const string LEVEL_FAILED = "LEVEL_FAILED";

Sekarang sesuaikan PlayerManager untuk menyiarkan pesan ini ketika kesehatan turun ke 0 (seperti yang ditunjukkan pada daftar berikutnya).

Daftar Level Siaran Gagal dari PlayerManager

```
...
public void Startup(NetworkService service) {
   Debug.Log("Player manager starting...");

   _network = service;

   UpdateData(50, 100);

   status = ManagerStatus.Started;
}

public void UpdateData(int health, int maxHealth) {
   this.health = health;
   this.maxHealth = maxHealth;
}

public void ChangeHealth(int value) {
   health += value;
   if (health > maxHealth) {
      health = maxHealth;
   } else if (health < 0) {
      health = 0;
   }


   if (health == 0) {
      Messenger.Broadcast(GameEvent.LEVEL_FAILED);
   }
   Messenger.Broadcast(GameEvent.HEALTH_UPDATED);
}

public void Respawn() {
   UpdateData(50, 100);
}
...
```

Tambahkan metode kecil ke MissionManager untuk memulai ulang level (lihat daftar berikutnya).

Daftar MissionManager, yang dapat memulai kembali level saat ini

```
...
public void RestartCurrent() {
string name = "Level"+ curLevel;
Debug.Log("Loading " + name);
Application.LoadLevel(name); }
...
```

Dengan itu, tambahkan pendengar event lain ke UIController (ditampilkan dalam daftar berikut).

Daftar Menanggapi Level Gagal di UIController

```
...
Messenger.AddListener(GameEvent.LEVEL_FAILED, OnLevelFailed);
...
Messenger.RemoveListener(GameEvent.LEVEL_FAILED, OnLevelFailed);
...
private void OnLevelFailed() {
   StartCoroutine(FailLevel());
}
private IEnumerator FailLevel() {
   levelEnding.gameObject.SetActive(true);
   levelEnding.text = "Level Failed";

   yield return new WaitForSeconds(2);

   Managers.Player.Respawn();
   Managers.Mission.RestartCurrent();
}
...
```

Mainkan game dan biarkan musuh menembak Anda beberapa kali; pesan tingkat kegagalan akan muncul. Kerja bagus—player sekarang dapat menyelesaikan dan gagal level! Membangun itu, permainan harus melacak kemajuan player.

Menangani perkembangan player melalui permainan

Saat ini level individu beroperasi secara independen, tanpa ada hubungannya dengan permainan secara keseluruhan. Anda akan menambahkan dua hal yang akan membuat kemajuan permainan terasa lebih lengkap: menyimpan kemajuan player dan mendeteksi saat permainan (bukan hanya level) selesai.

Menyimpan dan memuat kemajuan player

Menyimpan dan memuat game adalah bagian penting dari sebagian besar game. Unity dan Mono menyediakan fungsionalitas I/O yang dapat Anda gunakan untuk tujuan ini. Namun, sebelum Anda dapat mulai menggunakannya, Anda harus menambahkan UpdateData() untuk MissionManager dan InventoryManager. Metode itu akan berfungsi seperti halnya di PlayerManager dan akan mengaktifkan kode di luar manajer untuk memperbarui data di dalam manajer. Listing dibawah ini menunjukkan pergantian manajer.

Daftar UpdateData() metode di MissionManager

```
...
public void Startup(NetworkService service) {
   Debug.Log("Mission manager starting...");

   _network = service;

   UpdateData(0, 1);
   status = ManagerStatus.Started;
}

public void UpdateData(int curLevel, int maxLevel) {
   this.curLevel = curLevel;
   this.maxLevel = maxLevel;
}
...
```

Daftar Metode UpdateData() di InventoryManager

```
...
public void Startup(NetworkService service) {
   Debug.Log("Inventory manager starting...");

   _network = service;

   UpdateData(new Dictionary<string, int>());

   status = ManagerStatus.Started;
}

public void UpdateData(Dictionary<string, int> items) {
   _items = items;
}

public Dictionary<string, int> GetData() {
   return _items;
}
...
```

Sekarang berbagai manajer semuanya memiliki metode UpdateData(), data dapat disimpan dari modul kode baru. Menyimpan data akan melibatkan prosedur yang disebut sebagai serialisasi data.

Serialize berarti mengkodekan kumpulan data ke dalam bentuk yang dapat disimpan. Anda akan menyimpan game sebagai data biner, tetapi perhatikan bahwa C# juga sepenuhnya mampu menyimpan file teks. Misalnya, string JSON yang Anda kerjakan di bab 9 adalah data yang diserialisasikan sebagai teks. Bab sebelumnya menggunakan PlayerPrefs tetapi dalam proyek ini Anda akan menyimpan file lokal (PlayerPrefs terbatas pada satu megabyte dan hanya dimaksudkan untuk menyimpan beberapa nilai). Buat skrip DataManager (lihat daftar berikutnya).

Anda tidak dapat mengakses sistem file dalam game web. Ini adalah fitur keamanan yang berarti game web tidak dapat menyimpan file lokal. Untuk menyimpan data untuk game web, posting data ke server Anda.

Daftar Skrip baru untuk DataManager

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class DataManager : MonoBehaviour, IGameManager {
   public ManagerStatus status {get; private set;}

   private string _filename;

   private NetworkService _network;

   public void Startup(NetworkService service) {
      Debug.Log("Data manager starting...");

      _network = service;

      _filename = Path.Combine(
            Application.persistentDataPath, "game.dat");

      status = ManagerStatus.Started;
```

Selama Startup() jalur file lengkap dibangun menggunakan Application.persistentDataPath, lokasi yang disediakan Unity untuk menyimpan data. Jalur file yang tepat berbeda pada platform yang berbeda, tetapi Unity mengabstraksikannya di belakang variabel statis ini (kebetulan, jalur ini menyertakan kedua Nama Perusahaan dan Nama Produk dari Pengaturan Player, jadi sesuaikan jika perlu). Metode File.Create() akan membuat file biner; panggil File.CreateText() jika Anda menginginkan file teks. Saat membangun jalur file, pemisah jalur berbeda pada platform komputer yang berbeda. C# memiliki Path.DirectorySeparatorChar untuk memperhitungkan ini. Buka scene Startup untuk menemukan Manajer Game. Tambahkan komponen skrip DataManager ke objek Game Managers, lalu tambahkan manajer baru ke skrip Managers.

Daftar Menambahkan DataManager ke Managers.cs

```
   public void SaveGameState() {
      Dictionary<string, object> gamestate = new Dictionary<string,
     object>();
      gamestate.Add("inventory", Managers.Inventory.GetData());
      gamestate.Add("health", Managers.Player.health);
      gamestate.Add("maxHealth", Managers.Player.maxHealth);
      gamestate.Add("curLevel", Managers.Mission.curLevel);
      gamestate.Add("maxLevel", Managers.Mission.maxLevel);

      FileStream stream = File.Create(_filename);
      BinaryFormatter formatter = new BinaryFormatter();
      formatter.Serialize(stream, gamestate);
      stream.Close();
   }

   public void LoadGameState() {
      if (!File.Exists(_filename)) {
         Debug.Log("No saved game");
         return;
      }

      Dictionary<string, object> gamestate;

      BinaryFormatter formatter = new BinaryFormatter();
      FileStream stream = File.Open(_filename, FileMode.Open);
      gamestate = formatter.Deserialize(stream) as Dictionary<strin
     object>;
      stream.Close();
```

```
            Managers.Inventory.UpdateData((Dictionary<string,
        int>)gamestate["inventory"]);
            Managers.Player.UpdateData((int)gamestate["health"]
        (int)gamestate["maxHealth"]);
            Managers.Mission.UpdateData((int)gamestate["curLevel"],
        (int)gamestate["maxLevel"]);
            Managers.Mission.RestartCurrent();
        }
    }
```

Karena DataManager menggunakan manajer lain (untuk memperbaruinya), Anda harus memastikan bahwa manajer lain muncul lebih awal di urutan startup. Terakhir, tambahkan tombol untuk menggunakan fungsi di DataManager (gambar 9.13 menunjukkan tombol). Buat dua tombol yang di-parenting ke HUD Canvas (bukan di pop-up Inventory). Panggil mereka (atur objek teks terlampir) Simpan Game dan Muat Game, atur Anchor ke kanan bawah, dan posisikan di -100,65 dan -100,30.

Gambar 9.13 Tombol Simpan dan Muat di kanan bawah layar

Tombol-tombol ini akan ditautkan ke fungsi di UIController, jadi tulis metode tersebut (seperti yang ditunjukkan dalam daftar berikut).

Daftar Simpan dan Muat metode di UIController

```
...
public void SaveGame() {
Managers.Data.SaveGameState();
}
public void LoadGame() {
Managers.Data.LoadGameState();
}
...
```

Tautkan fungsi ini ke pendengar OnClick di tombol (tambahkan daftar di pengaturan OnClick, drag objek UIController, dan pilih fungsi dari menu). Sekarang mainkan gamenya, ambil beberapa item, gunakan paket kesehatan untuk meningkatkan kesehatan Anda, lalu save game. Mulai ulang game dan periksa inventaris Anda untuk memverifikasi bahwa itu kosong. Tekan Muat; Anda sekarang memiliki kesehatan dan item yang Anda miliki ketika Anda menyimpan permainan!

Mengalahkan permainan dengan menyelesaikan tiga level

Seperti yang tersirat dari penyelamatan kemajuan player kami, game ini dapat memiliki beberapa level, bukan hanya satu level yang telah Anda uji. Untuk menangani beberapa level dengan benar, gim harus mendeteksi tidak hanya penyelesaian satu level, tetapi juga penyelesaian seluruh gim. Pertama tambahkan GameEvent lain:

```
Public const string GAME_COMPLETE = "GAME_COMPLETE";
```

Sekarang ubah MissionManager untuk menyiarkan pesan itu setelah level terakhir (lihat daftar berikutnya).

Daftar Game Penyiaran Lengkap dari MissionManager

```
...
public void GoToNext() {
...
} else {
Debug.Log("Last level");
Messenger.Broadcast(GameEvent.GAME_COMPLETE);
}
}
```

Tanggapi pesan itu di UIController (seperti yang ditunjukkan dalam daftar berikut).

Daftar Menambahkan pendengar event ke UIController

```
...
Messenger.AddListener(GameEvent.GAME_COMPLETE, OnGameComplete);
...
Messenger.RemoveListener(GameEvent.GAME_COMPLETE, OnGameComplete);
...
private void OnGameComplete() {
levelEnding.gameObject.SetActive(true);
levelEnding.text = "You Finished the Game!";
}
...
```

Coba selesaikan level untuk melihat apa yang terjadi: pindahkan player ke tujuan level untuk menyelesaikan level seperti sebelumnya. Pertama-tama Anda akan melihat pesan Level Complete, tetapi setelah beberapa detik pesan tersebut akan berubah menjadi pesan tentang menyelesaikan game.

Menambahkan lebih banyak level

Pada titik ini Anda dapat menambahkan sejumlah level tambahan, dan MissionManager akan mengawasi level terakhir. Hal terakhir yang akan Anda lakukan dalam bab ini adalah menambahkan beberapa level lagi ke proyek untuk mendemonstrasikan kemajuan permainan melalui berbagai level.

Gandakan file scene Level1 dua kali (Unity akan secara otomatis menambah angka ke Level2 dan Level3) dan menambahkan level baru ke Pengaturan Bangun (sehingga dapat dimuat selama bermain game). Ubah setiap scene sehingga Anda dapat membedakan antara level; jangan ragu untuk mengatur ulang sebagian besar scene, tetapi ada beberapa elemen permainan penting yang harus Anda simpan: objek player yang diberi tag Player, objek lantai yang disetel ke layer Ground, dan objek objektif, Controller, HUD Canvas, dan EventSystem.

Secara default, sistem pencahayaan membuat ulang peta cahaya saat level dimuat. Tapi ini hanya berfungsi saat Anda mengedit level; lightmaps tidak akan dihasilkan saat memuat level saat game sedang berjalan. Seperti yang Anda lakukan di bab 9, Anda dapat mematikan Continuous Baking di jendela pencahayaan (Window > Lighting) lalu klik Build to bake lightmaps (ingat, jangan sentuh folder Scene yang sudah dibuat). Anda juga perlu menyesuaikan MissionManager untuk memuat level baru. Ubah maxLevel menjadi 3 dengan mengubah panggilan UpdateData(0, 1); ke UpdateData(0, 3);.

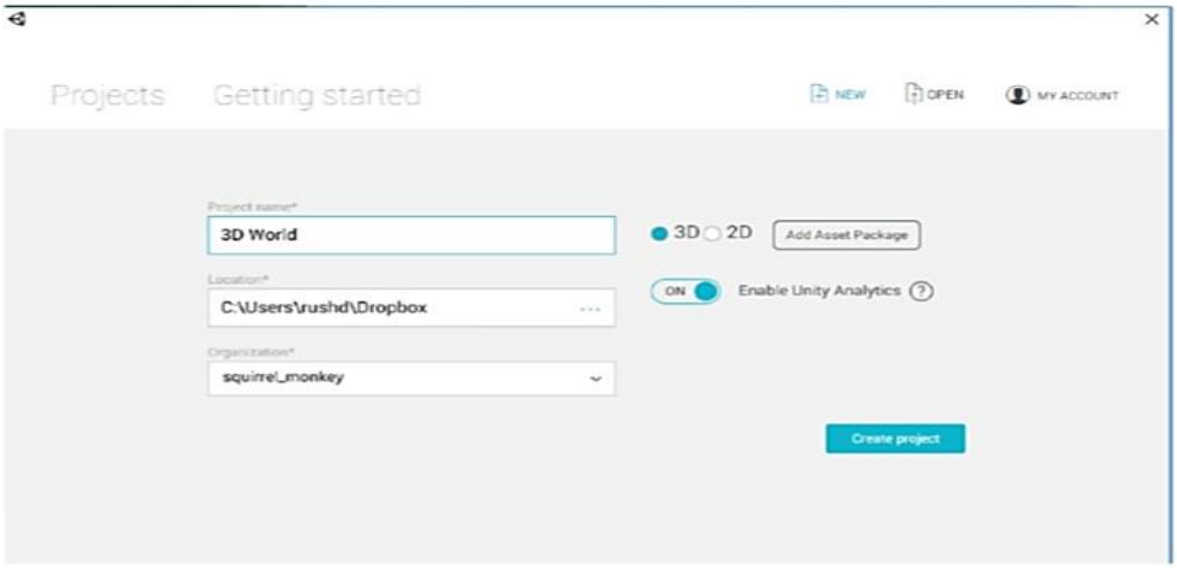
Sekarang mainkan gamenya dan Anda akan mulai di Level1 pada awalnya; mencapai tujuan level dan Anda akan melanjutkan ke level berikutnya! Kebetulan, Anda juga dapat menyimpan di level selanjutnya untuk melihat bahwa game akan memulihkan kemajuan itu. Anda sekarang tahu cara membuat game lengkap dengan berbagai level. Tugas berikutnya yang jelas adalah bab terakhir: membawa game Anda ke tangan player.

Pengantar Developeran Game 3D dan Realitas Virtual

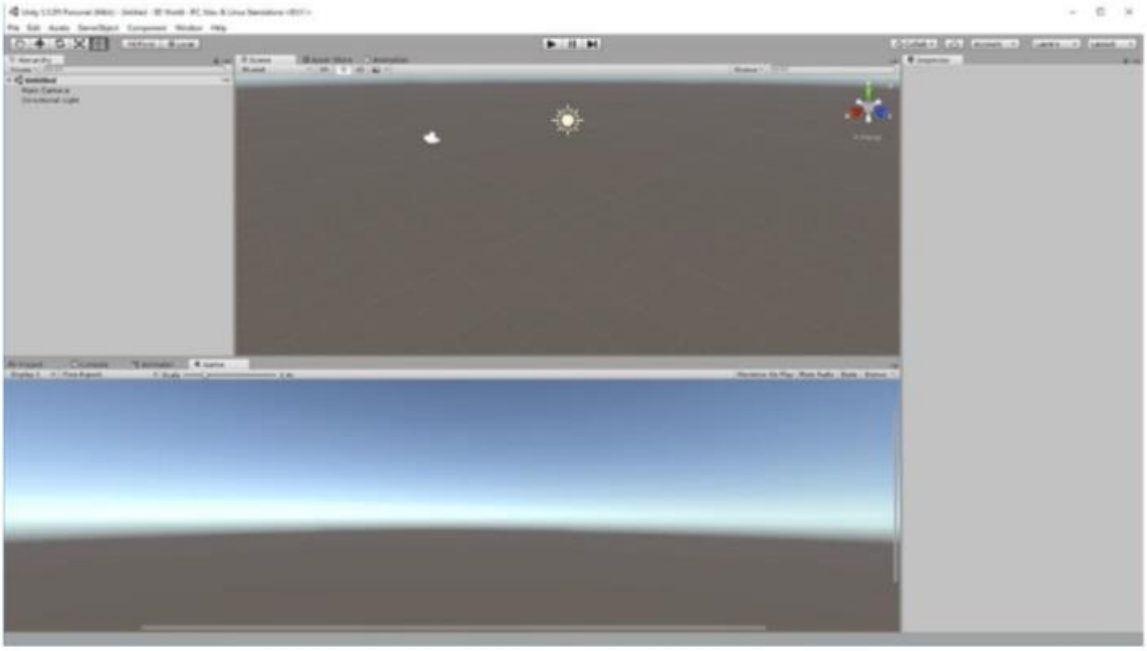
Tapi ada kategori game tambahan yang belum kami tangani dan itu akan membutuhkan pendekatan yang agak berbeda: game 3D. Meskipun kontrol 3D terkadang sulit dilakukan dengan layar sentuh kecil, ada banyak contoh game di luar sana yang berhasil mengatasi batasan ini; beberapa telah menjadi sangat populer. Judul 3D hit termasuk orang-orang seperti N.O.V.A. seri, Asphalt 8: Airborne, Geometry Wars 3, dan Minecraft: Pocket Edition untuk menyebutkan beberapa saja. Yang lebih menarik lagi adalah setelah Anda mempelajari keterampilan yang diperlukan untuk membuat game 3D, Anda akan siap untuk mulai menciptakan pengalaman realitas virtual Anda sendiri untuk Galaxy Gear VR dan Google Daydream Headset.

Membuat Dunia 3D

Untuk memulai, hal pertama yang harus Anda lakukan adalah membuat lingkungan 3D agar karakter Anda dapat mulai menjelajah. Mungkin terasa aneh mengucapkan selamat tinggal kepada Kevin, tetapi inilah saatnya untuk melanjutkan. Klik File - New Project tetapi kali ini pilih untuk memulai proyek 3D (Gambar 9.13). Anda kemudian akan disajikan dengan pengaturan yang serupa dengan yang biasa Anda lakukan, kecuali perspektifnya akan menjadi 3D. Perubahan perspektif itu juga memengaruhi kisi, yang sekarang akan dilihat dari atas dengan sudut tertentu (Gambar 9.14).

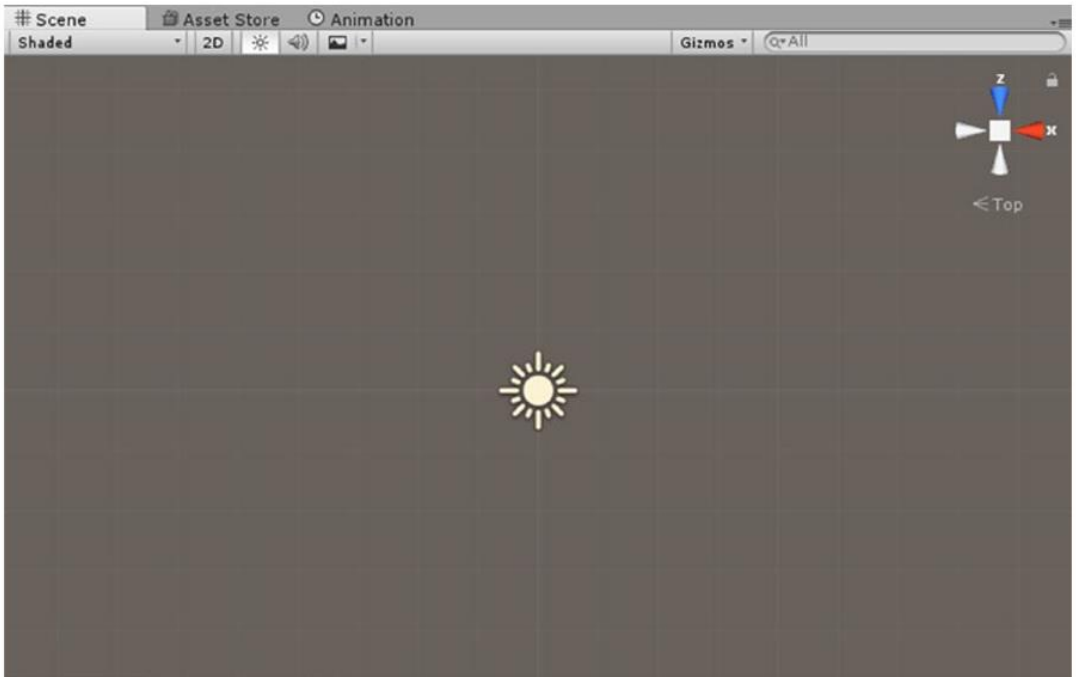


Gambar 9.13 Buat proyek 3D baru Anda



Gambar 9.14 Unity yang Anda kenal dan cintai, sekarang dalam 3D

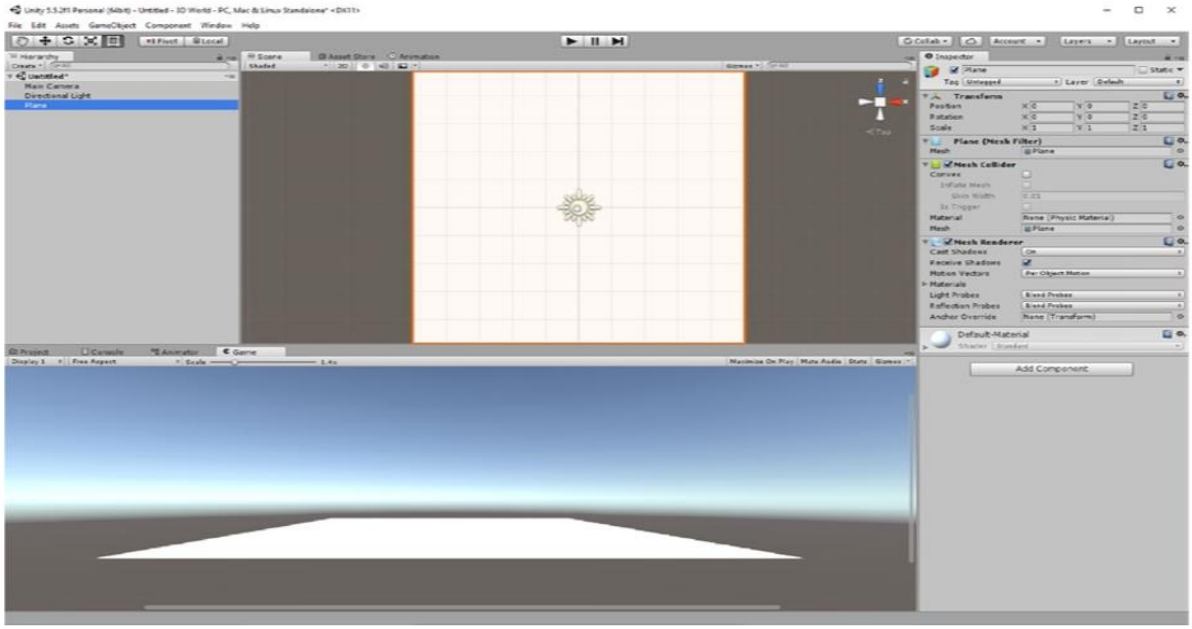
UI pada dasarnya berperilaku sama seperti sebelumnya. Perbedaan utama adalah penyertaan widget perspektif di kanan atas jendela Scene Anda. Klik ini dan Anda dapat mengubah sudut tampilan menjadi top-down (Gambar 9.15), side-on, dan seterusnya. Bekerja dengan objek 3D dapat menjadi rumit pada awalnya, dan Anda dapat menemukan diri Anda terus-menerus berjuang dengan perspektif. Tip singkatnya adalah tidak ada yang bisa menghentikan Anda untuk membuka beberapa jendela Scene: klik kanan tab mana saja lalu pilih Add Tab - Scene. Dengan cara ini Anda dapat memiliki banyak jendela, masing-masing dengan perspektif yang berbeda (atas-bawah, samping-atas, dan seterusnya). Tetapi terserah Anda untuk memutuskan pengaturan apa yang paling cocok untuk alur kerja Anda, seperti biasa.



Gambar 9.15 Sudut ini dapat berguna untuk menyelaraskan ubin tanah misalnya

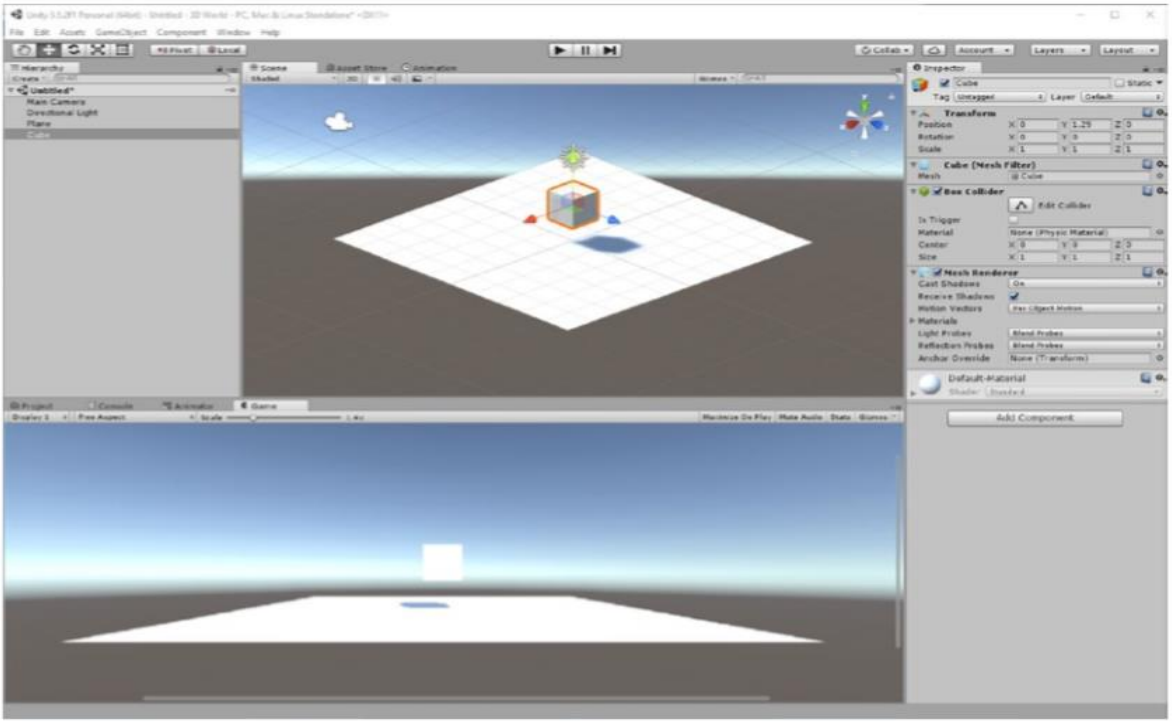
Jika tidak, Anda masih dapat menyeret tampilan dengan alat tangan, atau memperbesar dengan roda gulir seperti sebelumnya.

Namun, hal pertama yang akan kita lakukan adalah menjatuhkan objek 3D ke dunia ini. Arahkan ke GameObject➤3DObject➤Plane. Ini akan menjatuhkan—Anda dapat menebaknya—pesawat 3D ke dalam scene. Kami kemudian akan dapat mengubah ukuran pesawat tersebut, memindahkannya, atau mengubah atributnya melalui Inspector, seperti yang kami bisa dengan sprite 2D kami sebelumnya.



Gambar 9.16 Biarkan ada pesawat

Sekarang masukkan elemen lain ke dalam game. Bagaimana dengan kubus kali ini? Klik GameObject➤ 3DObject➤Cube, dan sebuah kubus akan muncul di layar Anda

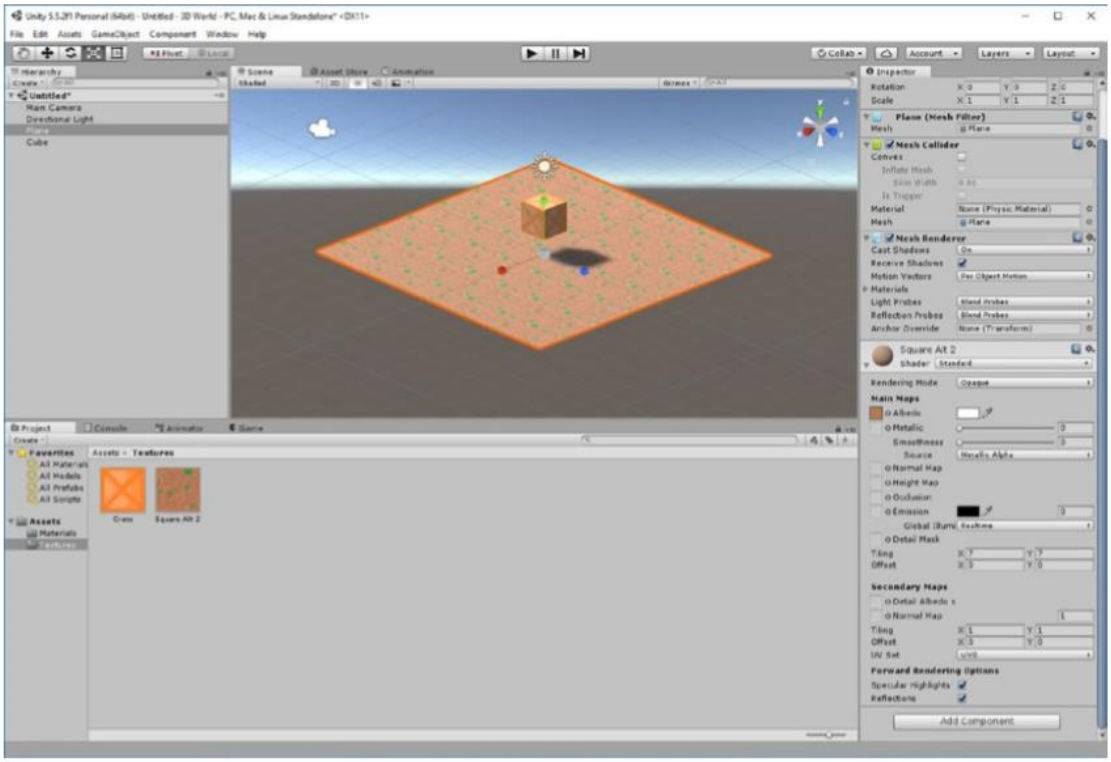


Gambar 9.17 Dan sekarang kubus

Sejauh ini, sangat sederhana. Tapi tekan play dan Anda akan melihat bahwa kubus hanya melayang di udara tanpa batas. Seperti sebelumnya, kita perlu menerapkan beberapa fisika. Jadi pilih kubus lalu Tambahkan Komponen➤Fisika➤Rigidbody. Perhatikan tidak ada sufiks 2D kali ini. Unity awalnya dikembangkan khusus untuk developeran 3D, jadi skrip dan objek "default" semuanya 3D tanpa perlu menyatakan fakta secara eksplisit. Segera setelah objek permainan ditambahkan, mereka sudah memiliki penumbuk mesh yang terpasang, jadi jika Anda menekan rotate sekarang, kubus akan jatuh ke pesawat dan berhenti di jalurnya. Luar biasa.

Sprite dan Skybox

Untuk membuat dunia terlihat sedikit lebih bagus, Anda dapat membuat subfolder di direktori Aset Anda yang disebut Tekstur dan memasukkan beberapa sprite. Sekarang drag sprite dari sana ke GameObjects Anda dan perluas menu Shader di Inspector. Di sini Anda dapat mengatur seberapa banyak Anda ingin tekstur berulang (Ubin) serta reflektifitas (Metalik), kehalusan, dan banyak lagi. Pada Gambar saya telah melakukannya menggunakan sprite dari versi 2D game tersebut.

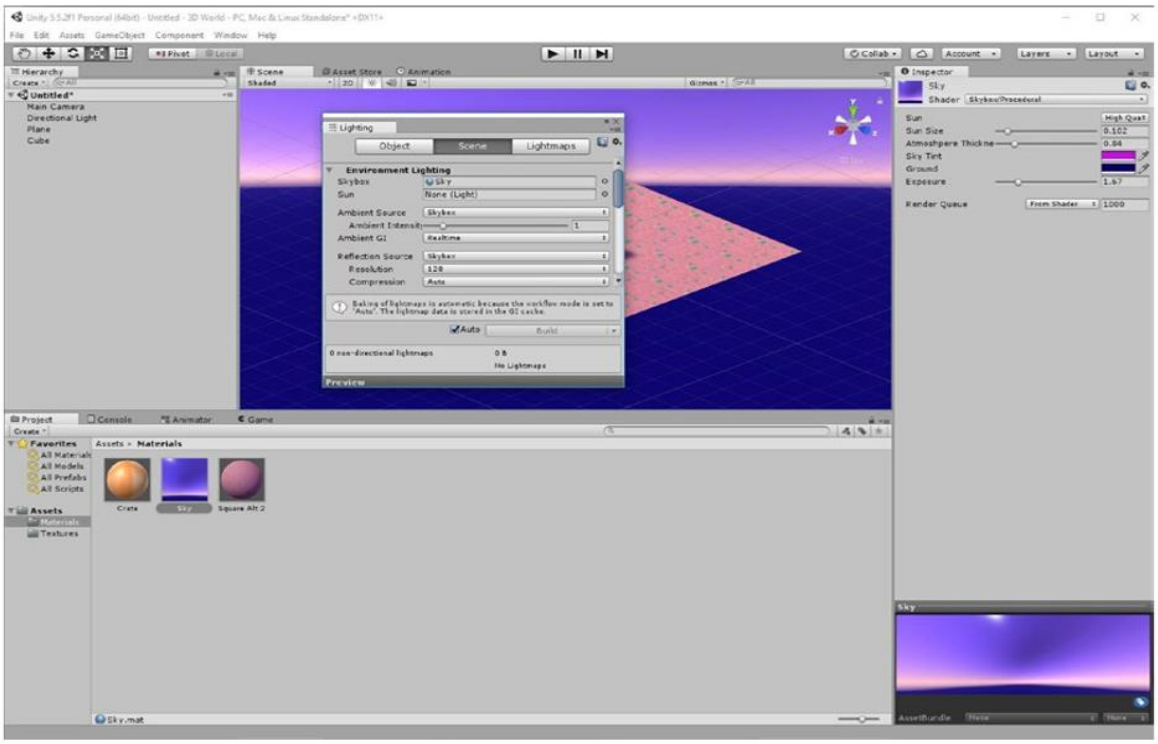


Gambar 9.18 Sebuah peti di atas beberapa kotoran

Sekarang kami ingin menambahkan latar belakang yang lebih baik untuk dunia kami. Untuk melakukan itu kita perlu membuat lightbox. Temukan folder Material di proyek Anda dan klik kanan di mana saja di sana untuk membuat material baru bernama Sky. Di Inspector, atur Shader ke Skybox menggunakan menu tarik-turun lalu pilih Prosedural. Ini akan; ubah warna dan tampilan langit di latar belakang, jadi atur Ketebalan Atmosfer, Eksposur, Ukuran Matahari, dan sebagainya, sesuai keinginan Anda. Saya akan mengatur scene saya di lokasi tipe pantai saat senja.

Sekarang pilih Window➤Lighting dan gunakan tab Scene untuk mengatur Skybox ke materi Sky yang baru saja Anda buat. Anda akan langsung melihat tampilan game Anda

berubah. Perhatikan bahwa jika Anda memilih 6 Sisi untuk skybox Anda, Anda dapat menggunakan tekstur apa pun yang Anda inginkan untuk latar belakang Anda.

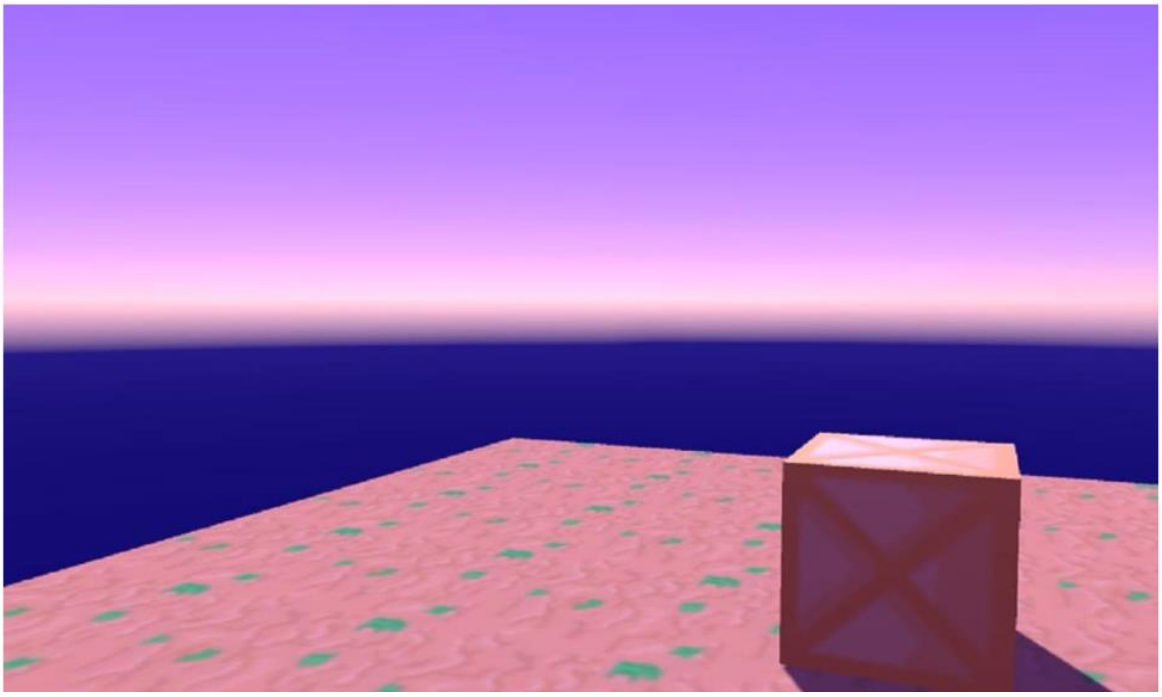


Gambar 9.19 Menyiapkan beberapa pencahayaan suasana hati

Menambahkan Player

Bersiaplah untuk terkesan dengan Unity lagi: menambahkan player ke dunia game kami sangat mudah karena ada aset siap pakai lain yang tersedia untuk membantu kami melakukannya: FPSController. Ini menangani kontrol, fisika, dan lainnya untuk player orang pertama. Klik kanan folder Aset dan pilih Impor Paket➤Karakter. Biarkan semuanya dipilih apa adanya dan tekan Impor. Ini akan memakan waktu beberapa detik, tetapi setelah selesai, Anda akan memiliki folder baru di proyek Anda yang disebut Aset Standar, dan di dalamnya akan ada berbagai subdirektori. Yang kami minati untuk saat ini adalah FirstPersonCharacter. Kemudian, untuk mengurangi ukuran proyek Anda, Anda dapat memilih untuk hanya mengimpor elemen yang Anda butuhkan.

Temukan Aset Standar➤KarakterKarakter Orang Pertama➤Prefab, lalu pilih FPSController. Anda tahu apa itu prefab sekarang, jadi Anda mungkin sudah menebak bahwa ini pada dasarnya adalah karakter FPS siap pakai yang bisa kita masukkan ke dalam scene kita. Lakukan itu, pastikan karakter berada di atas tanah, lalu hapus GameObject Kamera Utama yang berlebihan. Kemudian tekan mainkan.

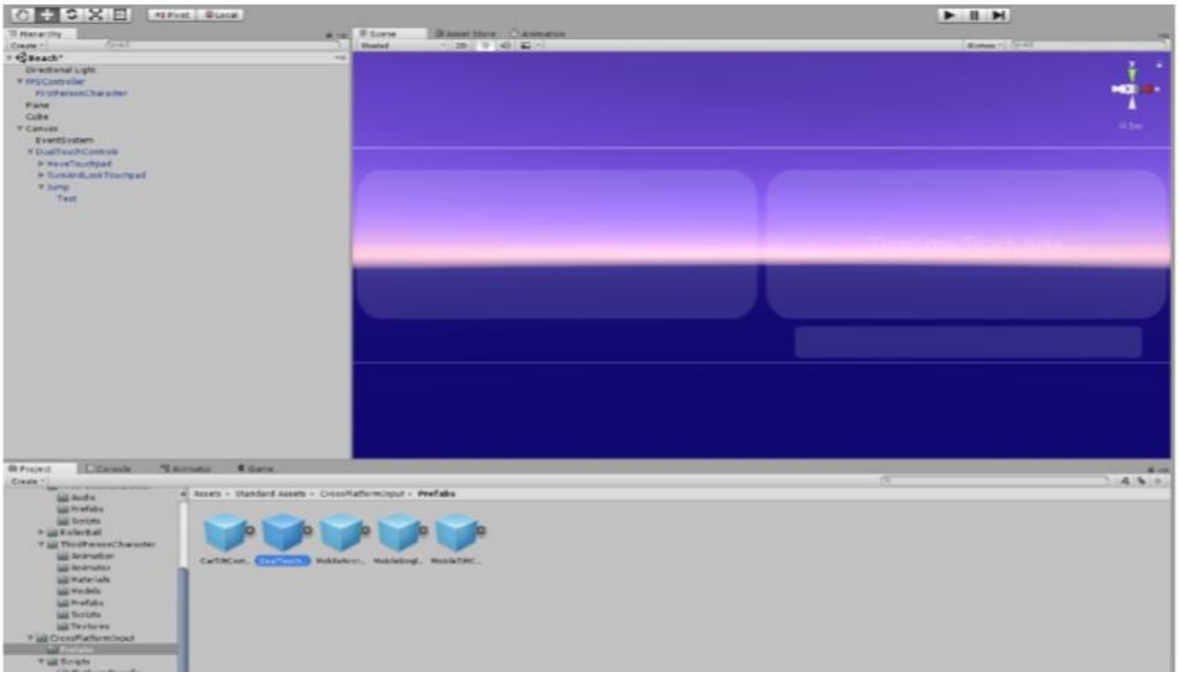


Gambar 9.20 Scene yang anehnya tenang…

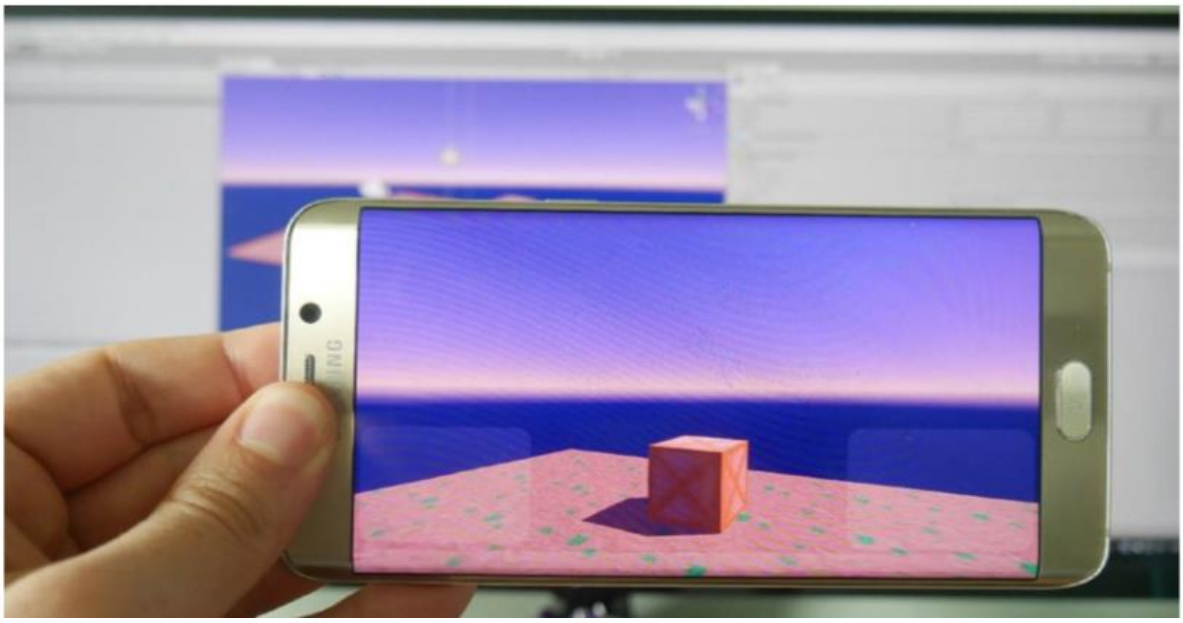
Ini keajaiban Natal! Itu mudah, kami memiliki game FPS dan berjalan. Anda dapat melihat-lihat dengan mouse, berkeliling dengan tombol W, A, S, dan D, dan melompat dengan spasi. Bahkan ada sound effect berjalan, dan Anda dapat berinteraksi dengan kotak untuk mendorongnya. Untuk keluar tekan Esc, maka Anda bisa menggerakkan pointer mouse ke atas sampai tombol stop. Jika Anda ingin bereksperimen, kembalikan Kamera Utama, hapus FPSController, dan masukkan ThirdPersonController. Ini adalah objek 3D yang jauh lebih detail dengan animasi kompleks tetapi tanpa shader, dan dapat dipindahkan dengan kontrol yang sama sehingga Anda dapat merasakan bagaimana rasanya membuat platformer 3D. Kombinasi aset ini membuat game terlihat cukup aneh. Untuk saat ini, kami akan tetap menggunakan FPSController.

Gambar 9.21 Jika Damien Hirst membuat game komputer…

Kontrol Sentuh Bersiaplah untuk terkesan lagi: menambahkan kontrol sentuh sama efisiennya di Unity. Sekali lagi, kami memiliki Prefab siap pakai; yang ini disebut DualTouchControls. Itu ada di CrossPlatformInput➤Prefabs, dan yang perlu Anda lakukan hanyalah memasukkannya ke dalam scene Anda dan menambahkan Sistem Event. Anda juga harus mengalihkan platform ke Android di Pengaturan Bangun agar ini berfungsi. Anda dapat mengatur area sentuh ini di layar sesuka Anda dengan menggunakan kanvas dan menempelkan ke layar, seperti yang Anda lakukan sebelumnya. Setelah siap untuk digunakan, coba buat dan jalankan aplikasi di ponsel cerdas untuk melihat seperti apa sebenarnya penggunaannya. Jika semuanya berjalan sesuai rencana, Anda dapat berjalan-jalan dengan menyeret di sekitar bagian kiri layar dan melihat pada saat yang sama menggunakan touchpad tak terlihat di sisi kanan layar. Bilah di sepanjang bagian bawah adalah untuk melompat.



Gambar 9.22 Atur kontrol sentuh di kanvas sesuai keinginan Anda



Gambar 9.23 Kontrol yang tidak terlalu mudah, tetapi tetap mengontrol

Perhatikan bahwa segera setelah Anda menyetel platform ke Android, game akan berhenti merespons input keyboard dan mouse Anda. Jadi, Anda mungkin ingin mengembalikan ini saat Anda berkembang.

Menggunakan Model 3D

Ketika kami sedang membangun platformer 2D kami, kami akan mengambil jeda dari pengkodean sesekali untuk membuat semacam sprite. Inilah yang menghidupkan dunia game kami, dan tentu saja kami tidak ingin membatasi upaya 3D kami hanya dengan kubus dan bola. Alih-alih sprite, sekarang kita akan menambahkan model 3D ke dunia game kita, yang akan membuka peluang tak terbatas untuk apa yang bisa kita buat. Free3D.com adalah situs yang menawarkan banyak model 3D bebas lisensi untuk Anda unduh dan masukkan ke dalam gim Anda, termasuk furnitur, monster, margasatwa, dan sebagainya. Anda juga dapat menemukan satu ton di Asset Store, tentu saja, dan banyak di antaranya gratis dari Unity itu sendiri. Pastikan bahwa apa pun yang Anda gunakan, Anda yakin bahwa lisensi tersebut berlaku untuk penggunaan komersial—jika Anda berencana untuk menjual kreasi akhir Anda. Menggunakan aset dari Free3D.com, saya telah membuat ulang ruang tamu saya dalam 3D.

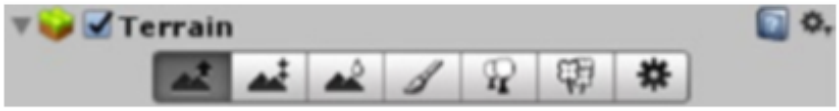
Gambar 9.24 ruang tamu saya, atau cukup dekat. Saya seorang SIM rupanya

Maafkan organisasi saya yang buruk di Hierarchy di sana. Ini hanya sedikit menyenangkan. Jadikan diri Anda di rumah! Ingat, sangat penting untuk selalu memastikan bahwa Anda diizinkan secara hukum untuk menggunakan, mendistribusikan, dan mengambil untung dari karya yang berpotensi memiliki hak cipta seperti model 3D—sebelum Anda menyelesaikan aplikasi. Ingin membuat model 3D Anda sendiri? Cara terbaik untuk melakukannya adalah dengan menggunakan software gratis bernama Blender. Agak sulit untuk dipahami, tetapi begitu Anda tahu apa yang Anda lakukan, hampir semua hal mungkin terjadi, dan Anda bahkan dapat mulai membuat animasi. Sebelum Anda menyadarinya, Anda akan bekerja untuk Pixar

Gambar 9.25 Tapi yang mana dunia nyata?

Medan Baru lainnya

Atau, cara lain Anda dapat menciptakan dunia yang sedikit berbeda dan lebih menarik adalah dengan menggunakan beberapa fitur bawaan Unity untuk menciptakan lanskap alam yang rimbun. Mulai scene baru atau new project dan kali ini pilih GameObject➤3D Object➤Terrain. Anda akan disambut dengan dataran lain, meskipun yang ini jauh lebih besar dari upaya sebelumnya. Apa yang benar-benar rapi tentang medan, adalah bahwa itu akan memungkinkan Anda memasukkan gunung, bukit, pohon, dan banyak lagi untuk menciptakan sesuatu yang terlihat jauh lebih organik dan alami. Anda harus mengimpor paket lain untuk ini. Klik Aset➤Impor Paket➤Lingkungan. Di sini Anda akan menemukan pepohonan, rumput, dan segala jenis objek lingkungan lainnya untuk Anda mainkan. Sekarang pilih objek medan di scene Anda dan Anda akan melihat beberapa ikon menarik di Inspector. Ini menampilkan gunung, kuas, pohon, dan banyak lagi. Coba klik ikon dengan pegunungan dan panah mengarah ke atas (lihat Gambar 9.26), lalu drag penunjuk mouse Anda di sekitar medan—gunung yang tampak alami akan mulai muncul, bergantung pada kecepatan dan tekanan yang Anda terapkan.



Gambar 9.26 Ini menyenangkan

Anda juga dapat mencoba melukis pohon ke lanskap, pertama dengan memilih jenis pohon yang ingin Anda tambahkan (di sinilah aset tersebut berguna) dan kemudian dengan mengecatnya dalam kepadatan yang berbeda. Anda juga dapat melukis tekstur dengan alat kuas, sekali lagi setelah terlebih dahulu memilih sesuatu yang terlihat bagian dari aset. Cobalah sedikit dan Anda akan segera mendapatkan gambaran tentang seberapa cepat Anda dapat menciptakan dunia yang hanya ingin dijelajahi.

Gambar 9.27 Breath of the Wild, makanlah hatimu

Menambahkan Pistol

Hampir setiap permainan orang pertama melibatkan penembakan beberapa deskripsi, jadi bagaimana Anda bisa menambahkan senjata? Ini sebenarnya sangat sederhana dan pada dasarnya melibatkan keterampilan yang sama yang sudah Anda kenal dari bekerja dengan 2D. Hanya ada beberapa perubahan yang harus Anda ketahui. Kita bisa mulai dengan membuat/menempatkan model pistol 3D dan kemudian menjadikannya anak dari karakter Player. Kami akan menyelaraskannya sehingga berada di posisi yang benar, seolah-olah karakter yang memegangnya. Dari sana, kami akan menambahkan skrip ke objek senjata itu sehingga akan merespons klik mouse. Ini akan sesuai dengan ketukan di mana saja di layar (atau tombol samping pada Gear VR). Kemudian kita akan membuat peluru di dalam pistol pada sudut yang sama:

```
if (Input.GetKeyDown(KeyCode.Mouse0))
        {
        Instantiate(blast, gameObject.transform.position, gameObject.transform.rotation);
        }
```

Peluru itu akan memiliki skripnya sendiri untuk memastikan peluru itu menghancurkan dirinya sendiri setelah jangka waktu tertentu dan membuatnya tetap bergerak maju. Satu-satunya perbedaan adalah peluru akan bergerak dalam tiga dimensi sekarang. Kami akan menggunakan transform.forward, dan dengan cara ini peluru akan bergerak maju pada sudut apa pun yang dihadapinya saat ini (yang pada gilirannya sama dengan pistol):

```
public class Forward : MonoBehaviour {
        private float timetodestroy;
        void Start () {
                timetodestroy = 3;
                }
                void Update () {
                timetodestroy = timetodestroy - Time.deltaTime;
                gameObject.transform.position += transform.forward * Time.deltaTime * 30;
```

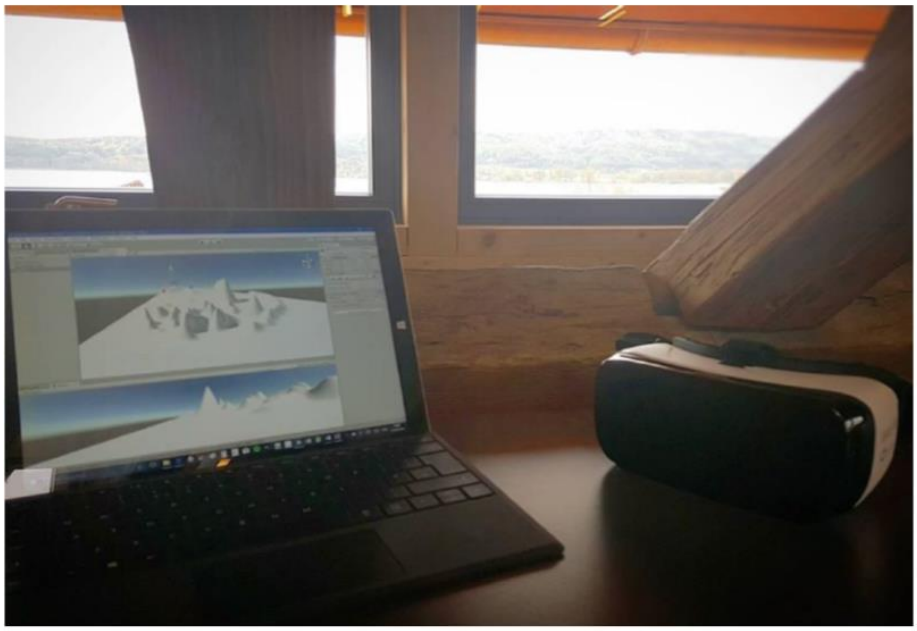
```
            if (timetodestroy < 0)
            {
            (gameObject);
            }
        }
}
```

Dari sana, kita dapat menggunakan onTriggerEnter atau onCollisionEnter seperti biasanya (minus 2D) untuk membuat peluru kita meledakkan item, membalik sakelar, dan melakukan hal lain. Seperti yang Anda lihat, ini adalah masalah yang cukup sederhana untuk mengambil pengaturan dasar ini dan membangunnya menjadi permainan penuh seperti yang Anda lakukan terakhir kali. Anda dapat memperkirakan dari instruksi ini untuk membuat AI, pegas, dan hal lainnya.

Melangkah ke Virtual Reality

Tahukah Anda, Sniper pertama (juga dikenal sebagai FPS) tidak begitu menyenangkan di ponsel menggunakan input sentuh. Jadi apa gunanya membiasakan diri dengan 3D sama sekali? Tentu saja, Anda mungkin menemukan ada jenis game 3D lain yang dapat Anda buat. Mungkin Anda tertarik untuk membuat game pinball atau game balap menggunakan kontrol kemiringan. Atau mungkin Anda hanya keras kepala dan Anda tetap membuat FPS. N.O.V.A. seri tampaknya baik-baik saja setelah semua.

Tapi apa yang juga bisa Anda buat—dan yang lebih menarik—adalah aplikasi realitas virtual



Gambar 9.28 Developeran untuk Gear VR di perpustakaan di Radolfzell, Jerman. Waktu yang baik!

Berkat Samsung Gear VR dan Google Daydream Headset, realitas virtual menjadi masalah besar di perangkat seluler. Sebenarnya, saya lebih suka memprediksi bahwa mungkin inilah masa depan VR. Adopsi adalah yang terbesar di seluler (karena biaya yang mahal dan tantangan teknis yang terkait dengan PC VR), tetapi yang lebih menarik adalah bahwa seluler

sekarang mampu memecahkan salah satu tantangan terbesar yang dihadapi VR: pelacakan posisi.

Gambar 9.29 Apakah VR seluler masa depan?

Pelacakan posisi mengacu pada kemampuan untuk tidak hanya melihat-lihat tetapi benar-benar bangun dan berjalan-jalan di ruang 3D. Untuk melompat, menunduk, berlari, dan memiringkan. Oculus Rift dan HTC Vive menyiasatinya dengan menggunakan solusi "luar ke dalam" yang rumit yang melibatkan sensor yang dipasang di sekitar ruangan. Gear VR dan Daydream saat ini hanya menawarkan pelacakan kepala. Namun pada konferensi Google I/O terbaru, HTC meluncurkan headset "mandiri" yang akan menggunakan sesuatu yang disebut WorldSense untuk menawarkan solusi pelacakan posisi yang sepenuhnya tidak terikat tanpa pengaturan dan tanpa sensor eksternal. Ini adalah pelacakan dari dalam ke luar. Perangkat akan ditautkan dengan pengalaman Daydream, jadi kami dapat memperkirakan bahwa perangkat tersebut kemungkinan akan menjalankan Android atau setidaknya sangat mirip dengan pengaturan saat ini. Perangkat ini belum keluar, dan itu bukan teknologi yang sepenuhnya baru. Kemungkinan besar ini adalah evolusi dari Project Tango Google—sebuah upaya untuk membuat ponsel dengan sensor bawaan yang cukup untuk dapat melihat dunia yang kita lihat. Teknologi ini sudah tersedia di Lenovo Phab Phone 2 dan akan datang ke lebih banyak perangkat dalam waktu dekat (mungkin sudah ada saat Anda membaca ini). Singkatnya, ponsel akan segera memiliki kemampuan untuk menggunakan "visi komputer" untuk merasakan apa yang ada di depan mereka dan bagaimana Anda bergerak melalui ruang—sehingga dapat melacak gerakan di VR dan memastikan bahwa player tidak menempatkan diri mereka sendiri. dalam bahaya apa pun. Tapi siapa tahu, mungkin VR seluler tidak akan menjadi hal besar berikutnya. Mungkin VR tidak akan lepas sama sekali. Apapun yang terjadi, kita semua pasti bisa sepakat bahwa membuat dunia maya lalu melangkah ke dalamnya itu keren.

Membuat Aplikasi Gear VR/Google Daydream Ready

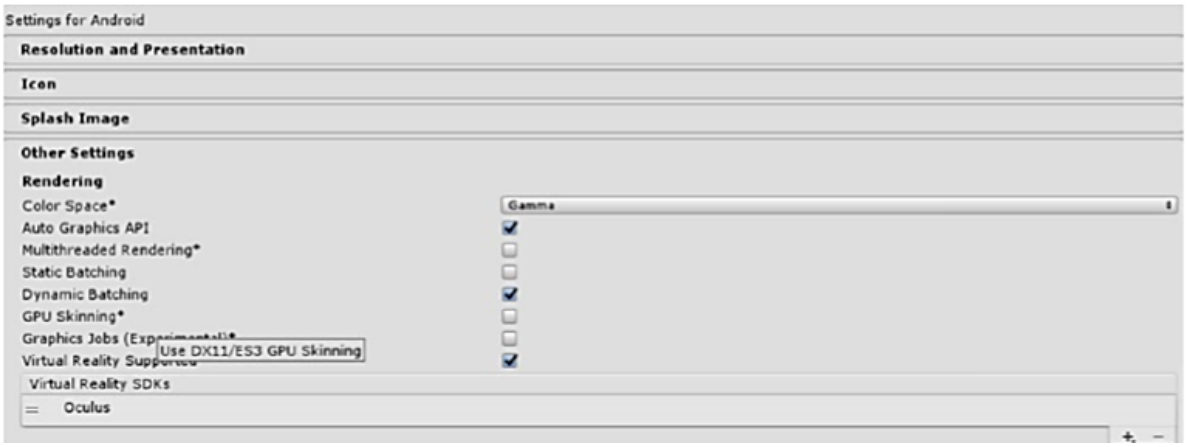
Jadi, bagaimana cara mengubah salah satu lanskap 3D yang baru saja kita buat menjadi aplikasi VR yang akan berjalan di Gear VR (lihat Gambar 9.30) atau Google Daydream?

Gambar 9.30 Masukkan kepalamu di sini

Ini sebenarnya tidak bisa lebih sederhana — selama itu berhasil. Sebenarnya, pada saat penulisan, Unity memiliki beberapa bug untuk diperbaiki, jadi semuanya tidak sepenuhnya mulus. Mencoba membuat aplikasi untuk Gear VR menyebabkan masalah saat file AndroidManifest tidak dapat digabungkan (ini adalah file yang berisi informasi tentang aplikasi Anda, seperti versi, nama, dan sebagainya — lihat Gambar 11-19) . Tim berjanji untuk memperbaikinya, jadi semoga semuanya sudah aktif dan berjalan saat Anda membaca ini. Untuk Google Daydream atau Cardboard, tidak ada masalah seperti itu. Saya membagikan ini kepada Anda karena ini adalah realitas pengkodean. Terkadang ada solusi, tetapi di lain waktu Anda terpaksa menunggu

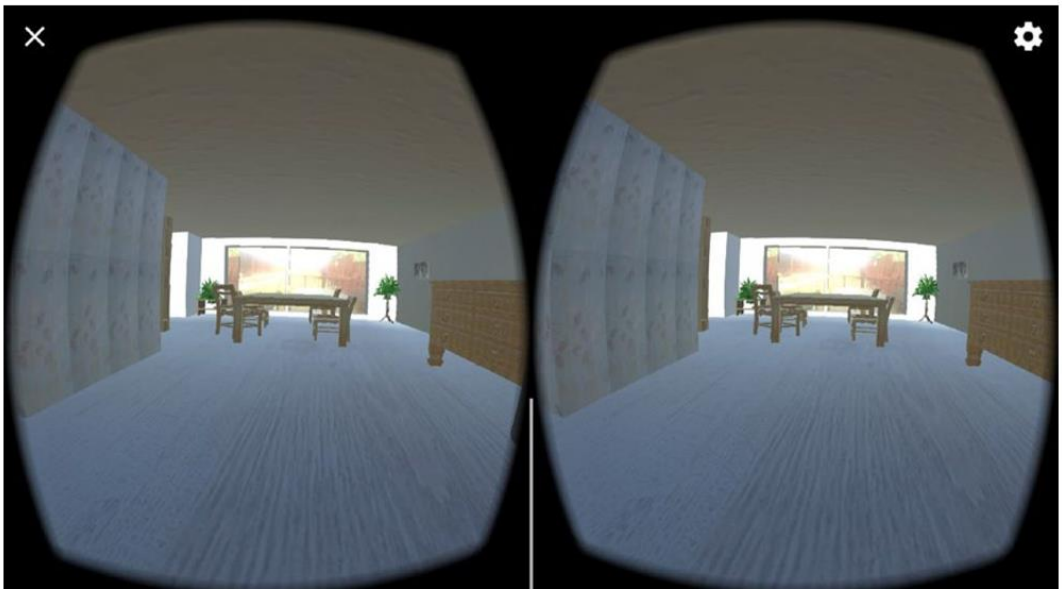
Gambar 9.31 Unity memperbaiki beberapa bug yang terjadi ketika mencoba membuat aplikasi untuk Gear VR

Dengan asumsi Unity bermain bagus, Anda hanya perlu membuat beberapa perubahan di Preferensi Player Anda. Secara khusus, Anda ingin mencentang kotak yang bertuliskan VR Supported, lalu pilih Oculus atau Daydream sebagai SDK Anda (Gambar 11-20). Tentu saja, Anda juga dapat mencoba yang lain, termasuk Cardboard, jika Anda mau. Unity akan tahu apa yang perlu diunduh dan ditambahkan ke plugin agar semuanya berfungsi.



Gambar 9.32 Perubahan prefensi player

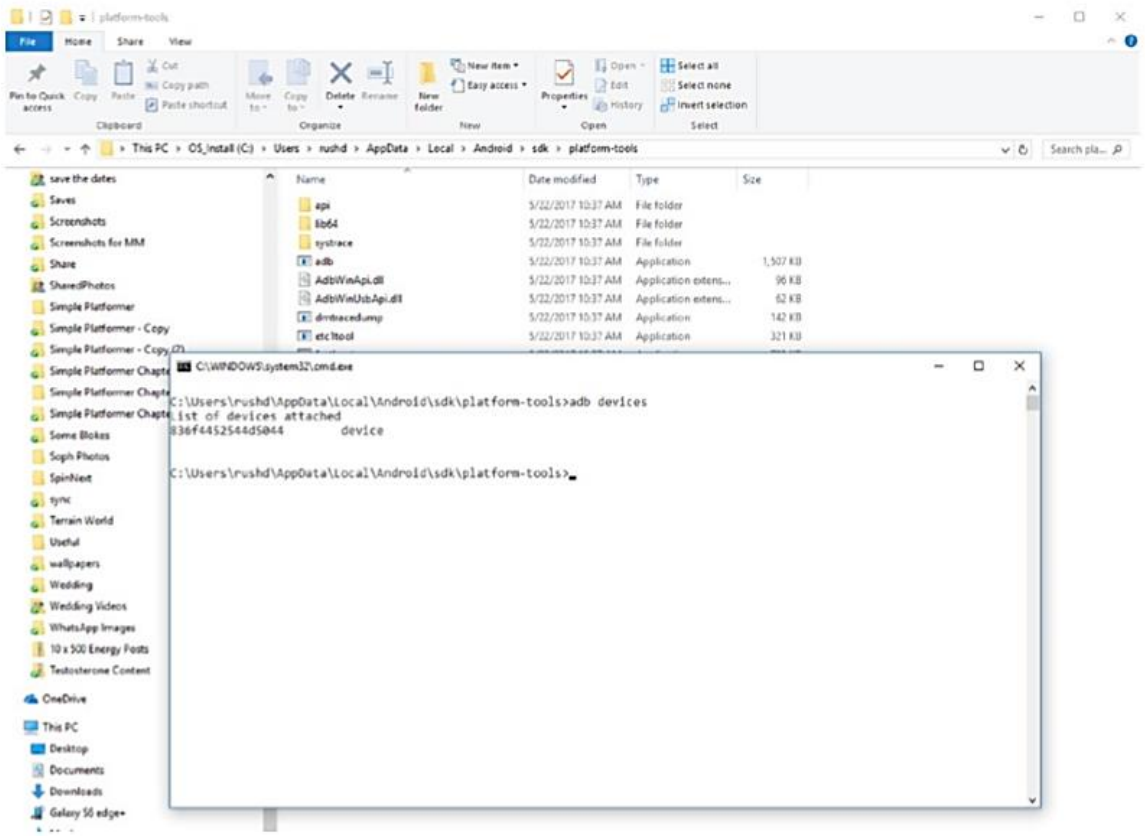
Ini benar-benar semua yang perlu Anda lakukan agar aplikasi Anda berjalan di hardware VR dan mencoba menjelajahi lanskap Anda dalam realitas virtual. Tekan Build and Run lalu colokkan.



Gambar 9.33 Versi VR lebih rapi

Mendapatkan File Tanda Tangan Oculus Anda Hal-hal hanya sedikit lebih rumit jika Anda mengembangkan untuk Gear VR. Itu karena Oculus sedikit protektif terhadap platformnya dan tidak ingin orang mendistribusikan aplikasi mereka sendiri melalui saluran selain Play Store. Dengan demikian, ia telah memperkenalkan sistem untuk melarang berbagi aplikasi secara tidak sengaja, yang merupakan "file tanda tangan" mereka. Pada dasarnya, setiap APK hanya dapat bekerja pada satu perangkat, yang akan ditentukan oleh file yang Anda tambahkan pada waktu pembuatan. Untuk mendapatkan file tanda tangan Oculus, Anda harus terlebih dahulu mendapatkan ID Perangkat Anda. Ini adalah pengidentifikasi untuk hardware khusus Anda, jadi setiap kali Anda ingin menguji aplikasi Anda di perangkat baru, Anda harus melalui proses ini lagi. Untuk melakukannya, navigasikan ke folder platform-tools dari instalasi Android SDK Anda di PC Anda. Di sini, Anda akan menemukan executable bernama adb.exe, yang merupakan akronim untuk Android Debug Bridge. Tahan Shift, klik kanan di mana saja di folder ini, dan pilih Open Command Line Here. Di shell yang terbuka,

Anda sekarang akan mengetik adb devices dengan perangkat Android Anda terpasang. Ini akan mencantumkan ID Perangkat dari semua hardware yang terhubung.

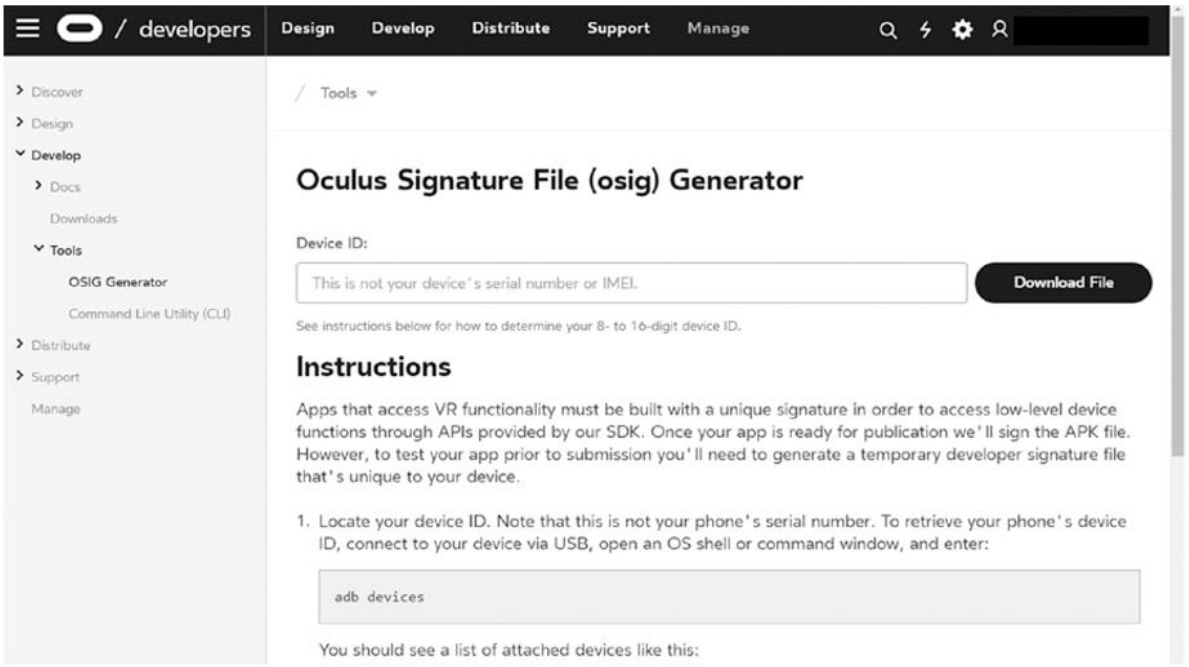


Gambar 9.34 ID perangkat saya

Bagaimanapun, ADB adalah hal yang berguna untuk membiasakan diri Anda, karena memiliki sejumlah kegunaan lain. Sekarang buka developer.oculus.com dan temukan Generator Oculus Signature File (osig). Pada saat penulisan, ini ada di https://dashboard.oculus.com/tools/osig-generator/. Masukkan nomor ID Perangkat Anda ke dalam kotak seperti yang ditunjukkan, lalu klik Unduh File. Sekarang Anda hanya perlu menempatkan file ini ke direktori tertentu dalam proyek Anda:

Assets > Plugins > Android > assets

Ya, ini peka huruf besar-kecil, dan ya, itu berarti Anda memerlukan satu folder Aset dengan huruf besar dan satu versi huruf kecil. Dan tidak, folder ini belum ada, jadi Anda harus membuatnya. Ini merepotkan tetapi setelah selesai, Anda akan dapat mulai menguji aplikasi VR baru Anda di hardware Anda sendiri dan Anda tidak perlu khawatir lagi untuk sementara waktu.



Gambar 9.35 Anda harus membuat akun baru terlebih dahulu

Kemungkinan Tanpa Batas

Ada seni untuk membuat konten VR, dan ini adalah bentuk seni yang masih tumbuh dan berkembang. Hal-hal yang berfungsi dengan baik di layar belum tentu ditranslate dengan baik ke VR dan sebaliknya. Lalu ada masalah seperti input dan mual. Tetapi fakta bahwa ruang ini sangat belum dijelajahi adalah yang membuatnya sangat menarik. Kemungkinan di sini tidak terbatas, dan ada setiap kesempatan bagi Anda untuk menemukan sesuatu yang mengubah permainan. Dan dengan itu, repertoar keterampilan Anda cukup banyak di tempat yang seharusnya. Anda akan belajar sambil melakukan dan Anda akan berkembang menjadi kemampuan Anda sebagai developer seiring berjalannya waktu. Tapi saya merasa Anda siap untuk menghadapi tantangan itu dan mulai menemukan jalan Anda sendiri. Hanya ada satu hal lagi yang perlu kami lakukan: siapkan dan jalankan kreasi Anda di Google Play Store sehingga orang lain dapat menikmatinya.

BAB 10
PUBLIKASI DAN PROMOSI GAME ANDA

Sepanjang buku ini, Anda telah mempelajari cara memprogram berbagai game dalam Unity, tetapi langkah terakhir yang penting telah hilang: menerapkan game tersebut ke player. Hingga game dapat dimainkan di luar editor Unity, game tersebut tidak menarik bagi siapa pun selain developer. Unity bersinar pada langkah terakhir ini, dengan kemampuan untuk membangun aplikasi untuk berbagai macam platform game. Bab terakhir ini akan membahas cara membuat game untuk berbagai platform ini.

Ketika saya berbicara tentang membangun untuk sebuah platform, saya mengacu pada menghasilkan paket aplikasi yang akan berjalan di platform itu. Pada setiap platform (Windows, iOS, dan sebagainya) bentuk yang tepat dari aplikasi yang dibangun berbeda, tetapi setelah executable dibuat, paket aplikasi tersebut dapat dimainkan tanpa Unity dan dapat didistribusikan ke player. Satu proyek Unity dapat di-deploy ke platform apa pun tanpa perlu dibuat ulang untuk masing-masingnya.

Kemampuan "bangun sekali, sebarkan di mana saja" ini berlaku untuk sebagian besar fitur dalam game Anda, tetapi tidak untuk semuanya. Saya memperkirakan bahwa 95% dari kode yang ditulis dalam Unity (misalnya, hampir semua yang telah kami lakukan sejauh ini dalam buku ini) adalah platform-agnostik dan akan bekerja dengan baik di semua platform. Tetapi ada beberapa tugas khusus yang berbeda untuk platform yang berbeda, jadi kami akan membahas area developeran khusus platform tersebut.

Secara total, versi dasar gratis dari Unity mampu membangun aplikasi untuk platform berikut:

- Windows PC
- Mac OS X
- Linux Web (kedua web player dan WebGL)
- iOS
- Android
- Blackberry 10

Selain itu, melalui modul berlisensi khusus, Unity dapat membuat aplikasi untuk hal-hal berikut:

- XBox 360
- XBox One
- PlayStation 3
- PlayStation 4
- PS Vita
- Wii U
- Windows Phone 8

Wah, daftar lengkapnya sangat panjang! Terus terang, itu hampir lucu, jauh lebih banyak daripada platform yang didukung dari hampir semua alat developeran game lain di luar sana. Bab ini akan fokus pada enam platform pertama yang terdaftar karena platform tersebut merupakan minat utama sebagian besar orang yang menjelajahi Unity, tetapi perlu diingat berapa banyak opsi yang tersedia untuk Anda.

Untuk melihat semua platform ini, buka jendela Build Settings. Itu adalah jendela yang Anda gunakan di bab sebelumnya untuk menambahkan scene yang akan dimuat; untuk mengaksesnya, pilih File > Build Settings. Anda akan melihat banyak ruang yang digunakan oleh daftar platform; platform yang sedang aktif ditunjukkan dengan ikon Unity. Pilih platform dalam daftar ini dan kemudian klik tombol Switch Platform.

Gambar 10.1 Jendela Pengaturan Bangun

Saat dalam proyek besar, berpindah platform sering kali membutuhkan waktu yang cukup lama untuk diselesaikan; pastikan Anda siap menunggu. Ini karena Unity mengompresi ulang semua aset (seperti tekstur) secara optimal untuk setiap platform. Juga di bagian bawah jendela ini terdapat tombol Pengaturan Player dan Bangun. Klik Pengaturan Player untuk melihat pengaturan aplikasi di Inspector, seperti nama dan ikon untuk aplikasi. Mengklik Build meluncurkan proses build.

Build And Run melakukan hal yang sama seperti Build, ditambah lagi secara otomatis menjalankan aplikasi yang dibangun. Saya biasanya ingin melakukan bagian itu secara manual, jadi saya jarang menggunakan Build And Run. Saat Anda mengklik Build, hal pertama yang muncul adalah pemilih file sehingga Anda dapat memberi tahu Unity tempat membuat paket aplikasi. Setelah Anda memilih lokasi file, proses pembuatan dimulai. Unity membuat paket aplikasi yang dapat dieksekusi untuk platform yang saat ini aktif; mari kita bahas proses pembuatan untuk platform paling populer: desktop, web, dan seluler.

Mulailah dengan membangun untuk desktop: Windows, Mac, dan Linux

Tempat paling sederhana untuk memulai saat pertama kali belajar membuat game Unity adalah dengan menerapkannya ke komputer desktop—PC Windows, Mac OS X, atau Linux. Karena Unity berjalan di komputer desktop, itu berarti Anda akan membuat aplikasi untuk komputer yang sudah Anda gunakan. Buka proyek apa pun untuk dikerjakan di bagian ini. Serius, proyek Unity apa pun akan berhasil; sebenarnya, saya sangat menyarankan

menggunakan proyek yang berbeda di setiap bagian untuk menunjukkan fakta bahwa Unity dapat membangun proyek apa pun ke platform apa pun!

10.1 MEMBANGUN APLIKASI

Pertama pilih File > Build Settings untuk membuka jendela Build Settings. Secara default, platform saat ini akan disetel ke PC, Mac, dan Linux, tetapi jika itu tidak terkini, pilih platform yang benar dari daftar dan klik Switch Platform. Di sisi kanan jendela Anda akan melihat menu Target Platform. Menu ini memungkinkan Anda memilih antara Windows PC, Mac OS X, dan Linux. Ketiganya diperlakukan sebagai satu platform dalam daftar di sisi kiri, tetapi ini adalah platform yang sangat berbeda, jadi pilihlah yang benar.

Setelah Anda memilih platform desktop Anda, klik Build. Dialog file muncul, memungkinkan Anda memilih ke mana aplikasi yang dibangun akan pergi. Ubah ke lokasi yang aman jika perlu (lokasi default biasanya berada di dalam proyek Unity, yang bukan merupakan tempat yang tepat untuk meletakkan build). Kemudian proses membangun dimulai; ini bisa memakan waktu cukup lama untuk proyek besar, tetapi proses pembuatannya harus cepat untuk demo kecil yang telah kami buat.

Skrip postbuild khusus

Meskipun proses build dasar berfungsi dengan baik di sebagian besar situasi, Anda mungkin ingin mengambil serangkaian langkah (seperti memindahkan file bantuan ke direktori yang sama dengan aplikasi) setiap kali Anda membangun game. Anda dapat dengan mudah mengotomatiskan tugas-tugas tersebut dengan memprogramnya dalam skrip yang akan dijalankan setelah proses pembuatan selesai.

Pertama, buat folder baru di tampilan Proyek dan beri nama folder itu Editor; skrip apa pun yang memengaruhi editor Unity (dan itu termasuk proses pembuatan) harus masuk ke folder Editor. Buat skrip baru di folder itu, beri nama TestPostBuild, dan tulis kode berikut di dalamnya:

```
using UnityEngine; using UnityEditor;
using UnityEditor.Callbacks;
public static class TestPostBuild {
    [PostProcessBuild] public static void OnPostprocessBuild(BuildTarget target,
        string
    pathToBuiltProject) {
    Debug.Log("build location: " + pathToBuiltProject);
    }
}
```

Arahan [PostProcessBuild] memberi tahu skrip untuk menjalankan fungsi yang segera setelahnya. Fungsi itu akan menerima lokasi aplikasi yang dibangun; Anda kemudian dapat menggunakan lokasi itu dengan berbagai perintah sistem file yang disediakan oleh C#. Untuk saat ini jalur file sedang dicetak ke konsol untuk menguji apakah skrip berfungsi.

Aplikasi akan muncul di lokasi yang Anda pilih; klik dua kali untuk menjalankannya, seperti program lainnya. Selamat, itu mudah! Membangun aplikasi sangatlah mudah, tetapi prosesnya dapat disesuaikan dengan beberapa cara; mari kita lihat cara menyesuaikan build.

Menyesuaikan Pengaturan Player: mengatur nama dan ikon game

Kembali ke jendela Build Settings, tapi kali ini klik Player Settings, bukan Build. Daftar besar pengaturan akan muncul di Inspector (lihat gambar 10.2); pengaturan ini mengontrol sejumlah aspek dari aplikasi yang dibangun.

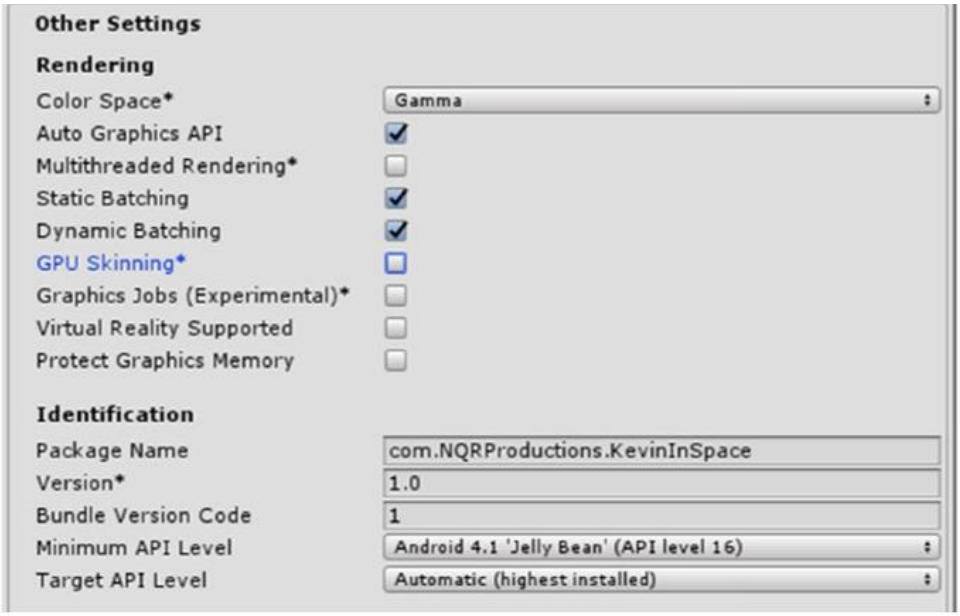
Gambar 10.2 Pengaturan pemutar ditampilkan di Inspector

Karena banyaknya pengaturan, Anda mungkin ingin mencarinya di manual Unity; halaman dokumen yang relevan adalah http://docs.unity3d.com/Manual/classPlayerSettings.html. Tiga pengaturan pertama di atas paling mudah dipahami: Nama Perusahaan, Nama Produk, dan Ikon Default. Ketik nilai untuk dua yang pertama. Nama Perusahaan adalah nama untuk studio developeran Anda, dan Nama Produk adalah nama dari game khusus ini. Kemudian drag gambar dari tampilan Proyek (impor gambar ke dalam proyek jika perlu) untuk mengatur gambar itu sebagai ikon; saat aplikasi dibangun, gambar ini akan muncul sebagai ikon aplikasi.

Membuat APK yang Ditandatangani

Setelah Anda menyelesaikan permainan dan Anda senang dengan itu, hal pertama yang perlu Anda lakukan adalah membuat APK yang telah ditandatangani. APK adalah file paket yang digunakan Android untuk menginstal aplikasi Anda di ponsel orang lain, dan Anda sudah membuat beberapa saat menguji game Anda. Namun, hingga saat ini, jika Anda telah menggunakan opsi Bangun dan Jalankan dan sebagian besar membiarkan Pengaturan Player sebagai default, Anda akan menjalankan versi debug aplikasi. Untuk benar-benar mempublikasikan kreasi Anda, Anda harus kembali ke Build Settings Player Settings untuk "menandatangani" agar siap untuk waktu yang lama (bersama dengan melakukan beberapa perubahan lainnya). Pertama, tambahkan beberapa ikon untuk digunakan bersama aplikasi Anda. Kemudian pastikan Anda memiliki nama paket yang Anda puas. Nama paket adalah nama file internal yang akan dilihat oleh aplikasi Android lainnya (seperti peluncur layar beranda), tetapi sebagian besar pengguna Anda tidak akan melihatnya, kecuali mereka

mendapatkan informasi teknis. Namun demikian, ini harus menjadi sesuatu yang masuk akal, karena Anda tidak akan dapat mengubahnya nanti. Ini semua dapat ditemukan di bawah judul Pengaturan Lainnya di Inspector, dan format yang biasa adalah menyertakan nama perusahaan Anda dan kemudian nama aplikasi.



Gambar 10.3 Menetapkan nama paket untuk platformer

Nama Produk adalah apa yang akan dilihat pengguna Anda (juga disebut label). Di sinilah branding Anda akan pergi, dan bagian ini dapat diubah nanti. Dengan kata lain, ini adalah judul untuk game Anda, tetapi Anda tidak perlu khawatir jika Anda berubah pikiran nanti. Kode Versi Anda adalah untuk Anda dan pengguna Anda. Ini adalah nomor versi yang akan mereka lihat, dan Anda mungkin ingin memperbarui ini secara bertahap setiap kali Anda meluncurkan versi yang lebih baru. Perbaikan bug dan tweak kecil mungkin menghasilkan tweak inkremental (1.0.1), tetapi perubahan besar yang menambahkan fitur dan level baru mungkin menghasilkan peningkatan versi secara keseluruhan (2.0). Perhatikan bahwa produk alfa dan beta harus memiliki kode versi yang lebih rendah dari 1.

Bundle Version Code, di sisi lain, adalah penghitung internal yang melacak versi aplikasi Anda. Setiap kali Anda mengunggah versi baru APK Anda, ini harus naik 1. Ini memungkinkan Google untuk melacak dan memastikan bahwa pengguna Anda didorong ke versi terbaru. Dengan demikian, pengguna Anda mungkin melihat versi 1.1.4, tetapi kode versi bundel Anda mungkin 8. Anda juga perlu memikirkan versi Android yang ingin Anda targetkan. Beberapa plugin dan fitur mengharuskan Anda menargetkan versi Android yang lebih tinggi atau menyetel API minimum yang lebih tinggi. Ini berlaku untuk aplikasi VR, misalnya. Tetapi mengizinkan sebanyak mungkin orang untuk menginstal dan menggunakan aplikasi Anda baik untuk bisnis jika memungkinkan. Secara mengejutkan, sejumlah besar pengguna masih menggunakan versi Android yang lebih lama, jadi untuk memaksimalkan jangkauan Anda, disarankan untuk menggunakan API yang rendah. Anda dapat melihat statistiknya di https://developer.android.com/about/dashboards/index.html. Coba instal APK baru di perangkat Anda dan Anda akan melihat bahwa sekarang memiliki nama dan ikon yang benar di laci aplikasi Anda. Kita semakin dekat sekarang—aku bisa merasakannya!

Membuat Keystore Di bawah Publishing Settings, Anda akan melihat opsi untuk membuat keystore. Ini adalah jenis sertifikat digital khusus yang akan mengidentifikasi dan mengotorisasi file APK Anda. Dengan kata lain, itu akan menjadi file dengan kata sandi dan nama pengguna yang harus Anda sertakan di setiap versi baru aplikasi Anda untuk membuktikan bahwa itu benar-benar aplikasi Anda. Jika Anda kehilangan file ini, Anda tidak akan pernah dapat memublikasikan pembaruan di masa mendatang—jadi simpan di tempat yang sangat aman! (Ada opsi untuk membiarkan Google menyimpan ini di cloud, yang akan saya bahas nanti.)

Meskipun skema keystore mungkin terdengar ketat, ini merupakan ukuran keamanan yang penting bagi pengguna dan developer. Jika tidak, seseorang yang memiliki sandi akun developer Anda berpotensi mengunggah "versi" baru aplikasi Anda yang hanya menggantikannya dengan software perusak. Itu dapat membahayakan pengguna Anda, merusak reputasi Anda, dan menghancurkan bisnis Anda. Jadi itu membuat frustrasi, tetapi pastikan Anda ingat di mana Anda menyimpan file dan Anda seharusnya tidak memiliki masalah apa pun. Untuk membuat keystore Anda, cukup pilih Create New Keystore, masukkan nama pengguna dan sandi.

Gambar 10.4 Jangan pernah kehilangan ini!

10.2 PENGATURAN KUALITAS

Aplikasi yang dibangun juga dipengaruhi oleh pengaturan proyek yang terletak di bawah menu Edit. Secara khusus, kualitas visual dari aplikasi akhir dapat disetel di sini. Buka Pengaturan Proyek di menu Edit dan kemudian pilih Kualitas dari menu tarik-turun. Pengaturan kualitas muncul di Inspector, dan pengaturan yang paling penting adalah kotak tanda centang di bagian atas. Berbagai platform yang dapat ditargetkan Unity terdaftar sebagai ikon di bagian atas, dan kemungkinan pengaturan kualitas tercantum di samping. Kotak dicentang untuk pengaturan kualitas yang tersedia untuk platform itu, dan kotak centang disorot hijau untuk pengaturan yang digunakan. Sebagian besar waktu pengaturan ini default ke Tercepat (yang merupakan kualitas terendah) tetapi Anda dapat mengubah ke kualitas Fantastis jika keadaan terlihat buruk; jika Anda mengklik panah bawah di bawah kolom platform, menu pop-up akan muncul. Tampaknya agak berlebihan bahwa UI ini memiliki kotak centang dan menu Default, tetapi begitulah. Platform yang berbeda sering kali memiliki kemampuan grafis yang berbeda, sehingga Unity memungkinkan Anda menetapkan tingkat kualitas yang berbeda untuk target build yang berbeda (seperti kualitas tertinggi di desktop dan kualitas lebih rendah di seluler).

Gambar 10.5 Tampilan pengaturan kualitas

Menyesuaikan ikon dan nama aplikasi penting untuk memberikan tampilan akhir. Cara lain yang berguna untuk menyesuaikan perilaku aplikasi yang dibangun adalah dengan kode yang bergantung pada platform.

Kompilasi yang bergantung pada platform

Secara default, semua kode yang Anda tulis akan berjalan dengan cara yang sama di semua platform. Tetapi Unity menyediakan sejumlah arahan kompiler (dikenal sebagai definisi platform) yang menyebabkan kode yang berbeda berjalan pada platform yang berbeda. Anda akan menemukan daftar lengkap definisi platform di halaman manual ini: http://docs.unity3d.com/Manual/PlatformDependent-Compilation.html.

Seperti yang ditunjukkan halaman itu, ada arahan untuk setiap platform yang didukung Unity, yang memungkinkan Anda menjalankan kode terpisah di setiap platform. Biasanya sebagian besar kode Anda tidak harus berada di dalam arahan platform, tetapi terkadang sebagian kecil kode perlu dijalankan secara berbeda pada platform yang berbeda. Beberapa kumpulan kode hanya ada di satu platform (misalnya, bab 11 menyebutkan bahwa akses sistem file tidak tersedia di pemutar web), jadi Anda perlu memiliki arahan compiler platform di sekitar perintah tersebut. Daftar berikut menunjukkan cara menulis kode tersebut.

Daftar Skrip PlatformTest menunjukkan cara menulis kode yang bergantung pada platform

```
using UnityEngine;
using System.Collections;

public class PlatformTest : MonoBehaviour {
   void OnGUI() {
#if UNITY_EDITOR
      GUI.Label(new Rect(10, 10, 200, 20), "Running in Editor");
#elif UNITY_STANDALONE
      GUI.Label(new Rect(10, 10, 200, 20), "Running on Desktop");
#else
      GUI.Label(new Rect(10, 10, 200, 20), "Running on other platform");
#endif
   }
}
```

Buat skrip bernama PlatformTest dan tulis kode dari daftar ini di dalamnya. Lampirkan skrip itu ke objek di scene (objek apa pun akan dilakukan untuk pengujian), dan pesan kecil akan muncul di kiri atas layar. Saat Anda memainkan game dalam editor Unity, pesannya akan mengatakan "Berjalan di Editor", tetapi jika Anda membuat game dan menjalankan aplikasi

yang dibangun, pesannya akan mengatakan "Berjalan di Desktop." Kode yang berbeda sedang dijalankan dalam setiap kasus!

Untuk pengujian ini kami menggunakan definisi platform yang memperlakukan semua platform desktop sebagai satu, tetapi seperti yang ditunjukkan pada halaman dokumen itu, definisi platform terpisah tersedia untuk Windows, Mac, dan Linux. Sebenarnya, ada definisi platform untuk semua platform yang didukung oleh Unity sehingga Anda dapat menjalankan kode yang berbeda pada masing-masing platform. Mari beralih ke platform penting berikutnya: web.

Membangun untuk web

Meskipun platform desktop adalah target paling dasar untuk dibangun, platform penting lainnya untuk game Unity adalah penyebaran ke web. Ini mengacu pada game yang berjalan dalam browser web dan dengan demikian dapat dimainkan melalui internet.

10.3 UNITY PLAYER VS. HTML5/WEBGL

Sebelumnya, Unity harus men-deploy web build dalam bentuk yang diputar dalam plug-in browser kustom. Ini telah lama diperlukan karena grafik 3D tidak ada di dalam browser web. Namun, dalam beberapa tahun terakhir, standar telah muncul untuk grafik 3D di web yang disebut WebGL. Secara teknis, WebGL terpisah dari HTML5, meskipun kedua istilah tersebut terkait dan sering digunakan secara bergantian. Unity 5 telah menambahkan WebGL ke daftar platform jendela Build, dan versi mendatang bahkan dapat menjadikannya jalan utama baru untuk membuat web build. Sebagian, perubahan dalam pembuatan web Unity ini didorong oleh keputusan strategis yang dibuat dalam Unity (perusahaan). Perubahan ini juga didorong oleh dorongan dari pembuat browser, yang beralih dari plugin khusus dan menggunakan HTML5/WebGL sebagai cara untuk melakukan aplikasi web interaktif, termasuk game.

Terlepas dari bentuk akhir aplikasi yang dibuat, proses pembuatan web hampir sama persis untuk pemutar web dan WebGL. Bagian berikut akan menjelaskan proses untuk pemutar web, jadi Anda juga harus menggunakan platform itu. Teks akan menyebutkan tempat di mana kode yang Anda tulis sedikit berbeda untuk WebGL.

10.4 MEMBANGUN FILE UNITY DAN HALAMAN WEB PENGUJIAN

Buka proyek yang berbeda (sekali lagi, ini untuk menekankan cara kerja proyek apa pun) dan buka jendela Pengaturan Bangun. Alihkan platform ke Web Player lalu klik tombol Build. Sebuah pemilih file akan muncul; ketikkan nama WebTest untuk aplikasi ini, dan ubah ke lokasi yang aman jika perlu (yaitu, lokasi yang tidak berada dalam proyek Unity). Proses pembuatan sekarang akan membuat dua file: game Unity yang sebenarnya akan memiliki ekstensi .unity3d, dan akan ada halaman web sederhana untuk memainkan game itu. Buka halaman web ini dan game harus disematkan di tengah halaman yang kosong.

Tidak ada yang istimewa dari halaman ini; itu hanya contoh untuk menguji permainan Anda. Dimungkinkan untuk menyesuaikan kode pada halaman itu, atau bahkan menyediakan halaman web Anda sendiri (dengan kode Unity yang disalin). Salah satu penyesuaian terpenting yang harus dilakukan adalah mengaktifkan komunikasi antara Unity dan browser, jadi mari kita bahas selanjutnya.

10.5 BERKOMUNIKASI DENGAN JAVASCRIPT DI BROWSER

Game web Unity dapat berkomunikasi dengan browser (atau lebih tepatnya dengan JavaScript yang berjalan di browser), dan pesan ini dapat dikirim ke dua arah: dari Unity ke browser, dan dari browser ke Unity. Mengirim pesan ke browser sangatlah mudah: Unity memiliki beberapa perintah khusus yang langsung menjalankan kode di browser. ntuk pesan dari browser, metodologinya sedikit lebih terlibat: JavaScript di browser mengidentifikasi objek dengan nama, dan kemudian Unity meneruskan pesan ke objek bernama di scene. Dengan demikian Anda harus memiliki objek dalam scene yang akan menerima komunikasi dari browser.

Untuk mendemonstrasikan tugas ini, buat skrip baru di Unity bernama WebTestObject. Juga buat objek kosong di scene aktif yang disebut Listener (objek di scene harus memiliki nama yang tepat, karena itulah nama yang digunakan dalam kode). Lampirkan skrip baru ke objek itu, lalu tulis kode dari daftar berikutnya.

Daftar Skrip WebTestObject untuk menguji komunikasi dengan browser

```
using UnityEngine;
using System.Collections;

public class WebTestObject : MonoBehaviour {
    private string _message;

    void Start() {
        _message = "No message yet";
    }

    void Update() {
        if (Input.GetMouseButtonDown(0)) {
            Application.ExternalCall("ShowAlert", "Hello out there!");
        }
    }

    void OnGUI() {
        GUI.Label(new Rect(10, 10, 200, 20), _message);
    }
    public void RespondToBrowser(string message) {
        _message = message;
    }
}
```

Sekarang buat web lagi untuk memperbarui game dengan kode baru ini. Pembuatan web Unity sudah siap sekarang, tetapi halaman webnya juga perlu disesuaikan. Anda perlu menambahkan beberapa fungsi ke JavaScript di halaman, serta menambahkan tombol ke HTML. Tambahkan kode JavaScript dan tag HTML dalam daftar berikut; fungsi JavaScript berada di akhir tag <script>, dan tombol HTML berada di akhir <body> halaman.

Daftar JavaScript dan HTML yang memungkinkan komunikasi browser–Unity

```
...
function ShowAlert(arg) {
    alert(arg);
}
function SendToUnity() {
    u.getUnity().SendMessage("Listener", "RespondToBrowser", "Hello from the
      browser!");
}
-->
</script>
...
<input type="button" value="Send to Unity" onclick="SendToUnity();" /> #C
</body>
</html>
```

Buka halaman web untuk menguji kode ini. Untuk menguji komunikasi dari Unity ke browser, skrip WebTestObject di Unity akan memanggil fungsi di browser saat Anda mengklik

dalam Unity; coba klik beberapa kali dan Anda akan melihat kotak peringatan muncul di browser. Metode Application.ExternalCall() akan menjalankan fungsi JavaScript bernama. Unity juga memiliki Application.ExternalEval() untuk mengirim pesan ke browser; dalam hal ini, potongan JavaScript sewenang-wenang dijalankan di browser, daripada memanggil fungsi yang ditentukan. Sebagian besar waktu lebih baik untuk memanggil fungsi (untuk menjaga JavaScript dan Unity terkotak-kotak), tetapi terkadang berguna untuk menjalankan cuplikan arbitrer, seperti kode ini untuk memuat ulang halaman:

```
Application.ExternalEval("location.reload();");
```

JavaScript di halaman web juga dapat mengirim pesan ke Unity; klik tombol di halaman web dan Anda akan melihat pesan yang diubah ditampilkan di Unity. Tag HTML tombol menautkan ke fungsi JavaScript, dan fungsi itu memanggil SendMessage() pada instance Unity. Metode ini memanggil fungsi bernama pada objek bernama dalam Unity; parameter pertama adalah nama objek, parameter kedua adalah nama metode, dan parameter ketiga adalah string untuk dilewatkan saat memanggil metode. Listing diatas tersebut memanggil RespondToBrowser() dari skrip WebTestObject.

Pembuatan WebGL juga dapat berkomunikasi dengan JavaScript di halaman web, dan kode untuk melakukannya hampir sama persis. Memang, itu persis sama untuk berkomunikasi dari Unity ke halaman. Sedangkan untuk arah lain—dari halaman ke Unity—metode SendMessage() memiliki parameter yang sama tetapi tidak lagi memerlukan awalan u.getUnity(). Itu membungkus komunikasi browser untuk pembuatan web; ada satu platform lagi (atau lebih tepatnya, kumpulan platform) untuk membahas pembuatan aplikasi untuk: aplikasi seluler.

10.6 MEMBANGUN UNTUK APLIKASI SELULER: IOS DAN ANDROID

Aplikasi seluler adalah target build penting lainnya untuk Unity. Kesan saya (sama sekali tidak ilmiah) adalah bahwa game seluler adalah game komersial dengan jumlah terbesar yang dibuat menggunakan Unity. Mobile mengacu pada perangkat komputasi genggam yang dibawa orang. Penunjukan dimulai dengan smartphone tetapi sekarang termasuk tablet. Dua platform komputasi seluler yang paling banyak digunakan adalah iOS (dari Apple) dan Android (dari Google). Menyiapkan proses pembuatan untuk aplikasi seluler lebih rumit daripada versi desktop atau web, jadi ini adalah bagian opsional—opsional karena hanya membacanya dan tidak benar-benar melakukan langkah-langkahnya; Saya masih akan menulis seolah-olah Anda sedang bekerja bersama, tetapi Anda harus membeli lisensi developer untuk iOS dan menginstal semua alat developer untuk Android.

Perangkat seluler mengalami begitu banyak inovasi sehingga proses pembuatan yang tepat kemungkinan akan sedikit berbeda pada saat Anda membaca ini. Konsep tingkat tinggi mungkin masih benar, tetapi Anda harus melihat dokumentasi online terbaru untuk ikhtisar yang tepat tentang perintah yang harus dijalankan dan tombol yang harus ditekan. Sebagai permulaan, berikut adalah halaman dokumen dari Apple dan Google: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html.
http://developer.android.com/tool/bangunan/index.html

Input sentuhan

Input pada perangkat seluler bekerja secara berbeda dari pada desktop atau web. Input seluler dilakukan dengan menyentuh layar, bukan dengan mouse dan keyboard. Unity

memiliki fungsionalitas input untuk menangani sentuhan, termasuk kode seperti Input.touchCount dan Input.GetTouch().

Anda mungkin ingin menggunakan perintah ini untuk menulis kode khusus platform di perangkat seluler. Namun, menangani input seperti itu bisa merepotkan, jadi sejumlah kerangka kerja kode tersedia untuk merampingkan penggunaan input sentuh. Sebagai contoh, saya menggunakan FingerGestures (http://fingergestures.fatalfrog.com/). Baiklah, dengan peringatan itu, saya akan menjelaskan keseluruhan proses pembuatan untuk iOS dan Android. Ingatlah bahwa platform ini terkadang mengubah detail proses pembuatan.

Menyiapkan build tool

Semua perangkat seluler terpisah dari komputer yang Anda kembangkan, dan keterpisahan itu membuat proses pembuatan dan penerapan ke perangkat sedikit lebih rumit. Anda harus menyiapkan berbagai alat khusus sebelum dapat mengeklik Bangun.

Menyiapkan build tool iOS

Pada tingkat tinggi, proses penerapan game Unity di iOS membutuhkan terlebih dahulu membangun proyek Xcode dari Unity dan kemudian membangun proyek Xcode menjadi IPA (Paket Aplikasi iOS) menggunakan Xcode. Unity tidak dapat membangun IPA final secara langsung karena semua aplikasi iOS harus melalui alat build Apple. Itu berarti Anda perlu menginstal Xcode (IDE pemrograman Apple), termasuk iOS SDK. Itu berarti Anda harus bekerja di Mac—Xcode hanya berjalan di OS X. Mengembangkan game dalam Unity dapat dilakukan di Windows atau Mac, tetapi membangun aplikasi iOS harus dilakukan di Mac.

Dapatkan Xcode dari situs web Apple, di bagian developer: https://developer.apple.com/xcode/downloads/. Jika Anda belum menjadi developer berlisensi, Anda harus mengikuti semua langkah ini dan membuat aplikasi iOS. Biaya Program Developer iOS $99/tahun; mendaftar di https://developer.apple.com/programs/ios/.

Setelah Xcode diinstal, kembali ke Unity dan beralih ke iOS. Anda perlu menyesuaikan pengaturan Player untuk aplikasi iOS (ingat, buka Pengaturan Bangun dan klik Pengaturan Player). Anda seharusnya sudah berada di tab iOS pada pengaturan Player, tetapi klik tab ikon iPhone jika diperlukan. Gulir ke bawah ke Pengaturan Lain dan kemudian cari Identifikasi. Bundle Identifier perlu disesuaikan agar Apple dapat mengidentifikasi aplikasi dengan benar. Baik iOS dan Android menggunakan Bundle Identifier dengan cara yang sama, sehingga pengaturan itu penting di kedua platform. Pengidentifikasi harus mengikuti konvensi yang sama seperti untuk paket kode apa pun: semua huruf kecil dalam bentuk com.companyname.productname. Pengaturan penting lainnya yang berlaku untuk iOS dan Android adalah Versi Bundel (ini adalah nomor versi aplikasi). Namun, sebagian besar pengaturan di luar itu adalah khusus platform; misalnya, baru-baru ini iOS menambahkan nomor versi pendek yang akan terlihat oleh player, terpisah dari versi bundel utama. Ada juga pengaturan untuk Scripting Backend; Mono selalu digunakan sebelumnya, tetapi bagian belakang IL2CPP yang baru dapat mendukung pembaruan platform, seperti binari 64-bit.

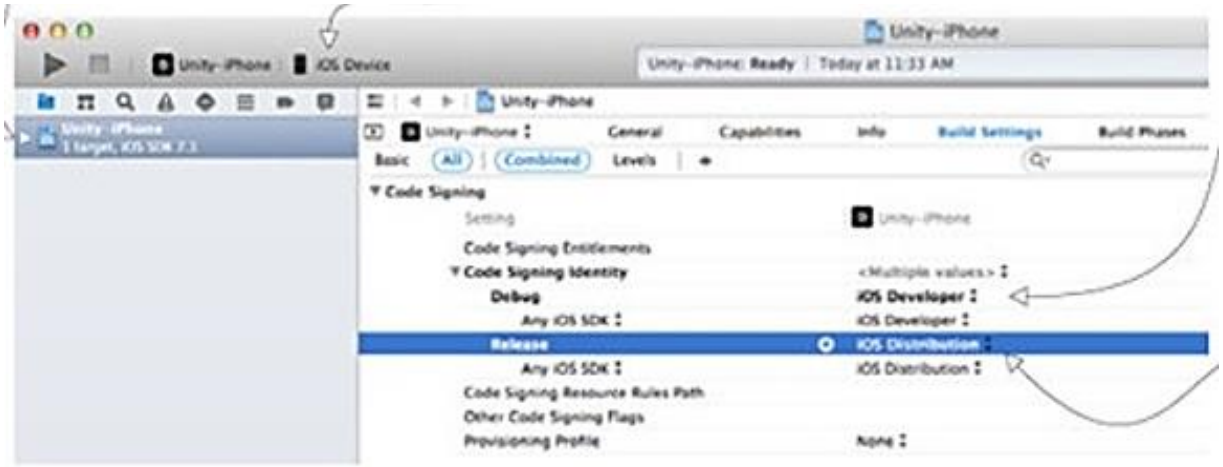
Sekarang klik Bangun di Unity. Pilih lokasi untuk file yang dibuat, dan itu akan menghasilkan proyek Xcode di lokasi itu. Proyek Xcode yang dihasilkan dapat dimodifikasi secara langsung jika Anda mau (beberapa modifikasi sederhana dapat menjadi bagian dari skrip postbuild). Apapun, buka proyek Xcode; folder yang dibangun memiliki banyak file, tetapi klik dua kali file .xcodeproj (memiliki ikon cetak biru). Xcode akan terbuka dengan proyek ini dimuat; Unity sudah menangani sebagian besar pengaturan yang diperlukan dalam proyek, tetapi Anda perlu menyesuaikan profil penyediaan yang digunakan.

Profil penyediaan iOS

Dari semua aspek developeran iOS, profil penyediaan paling sering berubah dan paling tidak biasa. Singkatnya, ini adalah file yang digunakan untuk identifikasi dan otorisasi. Apple dengan ketat mengontrol aplikasi apa yang dapat berjalan di perangkat apa; aplikasi yang dikirimkan ke Apple untuk persetujuan menggunakan profil penyediaan khusus yang memungkinkannya bekerja melalui App Store, sedangkan aplikasi dalam developeran menggunakan profil penyediaan yang khusus untuk perangkat terdaftar. Anda harus menambahkan UDID iPhone Anda (ID khusus untuk perangkat Anda) dan ID aplikasi (Pengidentifikasi Bundel dalam Unity) ke panel kontrol di iOS Dev Center, situs web Apple untuk developer iOS. Untuk penjelasan lengkap tentang proses ini, kunjungi https://developer.apple.com/devcenter/ios/index.action.

Gambar 10.6 iOS Dev Center

Pilih aplikasi Anda di daftar proyek di sisi kiri Xcode. Beberapa tab yang relevan dengan proyek yang dipilih akan muncul; buka Pengaturan Bangun dan gulir ke bawah ke Penandatanganan Kode untuk mengatur profil penyediaan (lihat gambar 10.7). Pastikan juga Tujuan Skema di atas diatur ke Perangkat iOS dan bukan simulator (beberapa opsi build berwarna abu-abu jika ini salah).



Gambar 10.7 Menyediakan pengaturan profil di Xcode

Setelah profil penyediaan diatur, Anda siap untuk membangun aplikasi. Dari menu Produk, pilih Jalankan atau Arsipkan. Ada banyak opsi di menu Produk, termasuk Build yang menggiurkan, tetapi untuk tujuan kami, dua opsi yang berguna adalah Jalankan atau Arsipkan.

Build menghasilkan file yang dapat dieksekusi tetapi tidak memaketkannya untuk iOS, sedangkan

- Jalankan akan menguji aplikasi pada iPhone yang terhubung ke komputer dengan kabel USB.
- Arsip akan membuat paket aplikasi yang dapat dikirim ke perangkat lain yang terdaftar (yang disebut Apple sebagai "distribusi ad-hoc").

Arsip tidak membuat paket aplikasi secara langsung melainkan membuat bundel dalam tahap peralihan antara file kode mentah dan IPA. Arsip yang dibuat akan terdaftar di jendela Organizer Xcode; di jendela itu, klik tombol Distribusikan untuk menghasilkan file IPA dari arsip. Gambar dibawah ini menunjukkan proses ini; setelah Anda mengklik Distribusikan, Anda akan ditanya apakah Anda ingin mendistribusikan aplikasi di toko atau ad hoc.

Gambar 10.8 Distribusikan aplikasi iOS yang diarsipkan dari jendela Organizer.

Jika Anda memilih distribusi ad hoc, Anda akan mendapatkan file IPA yang dapat dikirim ke penguji. Anda dapat mengirim file secara langsung untuk mereka instal melalui iTunes, tetapi lebih nyaman menggunakan TestFlight (https://developer.apple.com/testflight/) untuk menangani pendistribusian dan penginstalan build ad hoc.

Menyiapkan build tool Android

Tidak seperti aplikasi iOS, Unity dapat menghasilkan APK (Paket Aplikasi Android) secara langsung. Ini memerlukan penunjuk Unity ke Android SDK, yang menyertakan kompiler yang diperlukan. Unduh Android SDK dari situs web Android; lalu pilih lokasi file ini di preferensi Unity (lihat gambar 10.9). Anda dapat mengunduh SDK di sini: http://developer.android.com/sdk/index.html.

Gambar 10.9 Pengaturan preferensi kesatuan untuk menunjuk ke Android SDK

Setelah menyetel Android SDK di preferensi Unity, Anda perlu menentukan Bundle Identifier seperti yang Anda lakukan untuk iOS. Anda akan menemukan Bundle Identifier di Pengaturan Player; atur ke com.companyname.productname. Kemudian klik Build untuk memulai proses. Seperti semua build, pertama-tama akan menanyakan di mana menyimpan file. Maka itu akan membuat file APK di lokasi itu.

Sekarang setelah Anda memiliki paket aplikasi, Anda harus menginstalnya di perangkat. Anda bisa mendapatkan file APK ke ponsel Android dengan mengunduh file dari web atau dengan mentransfer file melalui kabel USB yang terhubung ke komputer Anda (pendekatan yang disebut sebagai sideloading). Detail tentang cara mentransfer file ke ponsel Anda berbeda-beda untuk setiap perangkat, tetapi begitu ada di sana, file tersebut dapat diinstal menggunakan aplikasi pengelola file. Saya tidak tahu mengapa pengelola file tidak terpasang di Android, tetapi Anda dapat menginstalnya secara gratis dari Play Store. Arahkan ke file APK Anda di dalam pengelola file dan kemudian instal aplikasi.

Seperti yang Anda lihat, proses pembuatan dasar untuk Android jauh lebih sederhana daripada proses pembuatan untuk iOS. Sayangnya, proses penyesuaian build dan implementasi plugin lebih rumit dibandingkan dengan iOS.

Kompresi tekstur

Aset dapat menambahkan banyak ukuran file ke aplikasi, dan ini tentu saja termasuk tekstur. Untuk mengurangi ukuran file mereka, Anda dapat mengompresi aset dengan beberapa cara; aplikasi seluler khususnya harus berhati-hati dalam menggunakan terlalu banyak ruang, sehingga aplikasi ini menerapkan kompresi pada teksturnya. Ada berbagai metode untuk mengompresi gambar, dengan pro dan kontra yang berbeda untuk setiap metode. Karena pro dan kontra ini, Anda mungkin perlu menyesuaikan cara Unity mengompresi tekstur.

Sangat penting untuk mengelola kompresi tekstur pada perangkat seluler, tetapi secara teknis tekstur sering dikompresi pada platform lain juga. Tetapi Anda tidak perlu terlalu memperhatikan kompresi pada platform lain karena berbagai alasan (alasan utamanya adalah bahwa platform tersebut lebih matang secara teknologi). Pada perangkat seluler, Anda perlu lebih memperhatikan kompresi tekstur karena perangkat lebih sensitif tentang detail ini. Unity mengompres tekstur untuk Anda; di sebagian besar alat developeran, Anda perlu mengompresi gambar sendiri, tetapi di Unity Anda biasanya mengimpor gambar yang tidak dikompresi, dan kemudian Unity menerapkan kompresi gambar dalam pengaturan impor untuk gambar.

Gambar 10.10 Pengaturan kompresi tekstur di Inspector

Pengaturan kompresi ini adalah default, dan Anda mungkin perlu menyesuaikannya untuk gambar tertentu. Secara khusus, kompresi gambar lebih rumit di Android. Ini sebagian besar disebabkan oleh fragmentasi perangkat Android: karena semua perangkat iOS menggunakan hardware video yang hampir sama, aplikasi iOS dapat memiliki kompresi tekstur yang dioptimalkan untuk chip grafisnya (GPU). Aplikasi Android tidak menikmati keseragaman hardware yang sama, sehingga kompresi teksturnya harus ditujukan untuk penyebut umum terendah.

Untuk lebih spesifik, semua perangkat iOS menggunakan GPU PowerVR; dengan demikian, aplikasi iOS dapat menggunakan kompresi tekstur PVR yang dioptimalkan. Beberapa perangkat Android juga menggunakan chip PowerVR, tetapi mereka juga sering menggunakan chip Adreno dari Qualcomm, GPU Mali dari ARM, atau opsi lainnya. Akibatnya, aplikasi Android umumnya mengandalkan Ericsson Texture Compression (ETC), algoritme kompresi yang didukung oleh semua perangkat Android. Sayangnya, ETC (ETC1, bagaimanapun; penerus yang sedang dikembangkan adalah ETC2) tidak mendukung transparansi alfa, jadi gambar dengan transparansi alfa tidak dapat dikompres menggunakan algoritme itu.

Unity mengompresi ulang gambar saat Anda berpindah platform. Di Android, Unity mengatasi batasan gambar transparan dengan mengonversi gambar dengan transparansi ke 16-bit alih-alih mengompresinya. Mengonversi gambar ke 16-bit memang menurunkan ukuran file, tetapi hal itu dilakukan dengan mengorbankan kualitas gambar. Karena itu, di Android terkadang Anda perlu mengatur ulang kompresi gambar individual secara manual, menentukan gambar demi gambar mana yang membutuhkan transparansi versus mana yang dapat memiliki ETC (kualitas gambar lebih baik dan tidak ada transparansi), dan memutuskan gambar transparan mana yang perlu dikurangi ukuran filenya. versus mana yang dapat dibuat tidak terkompresi.

Jika Anda perlu menyesuaikan kompresi pada tekstur, sesuaikan pengaturan yang ditunjukkan pada gambar 10.10. Ubah Jenis Tekstur ke Tingkat Lanjut untuk mengakses pengaturan tersebut, dan atur Android (klik tab ikon Android) untuk mengganti kompresi default. Menyesuaikan kompresi tekstur adalah detail pengoptimalan yang penting di Android. Topik bagian berikutnya penting untuk iOS dan Android: mengembangkan plugin asli.

Mengembangkan plugin

Unity memiliki sejumlah besar fungsionalitas bawaan, tetapi fungsionalitas itu sebagian besar terbatas pada fitur-fitur umum di semua platform. Memanfaatkan toolkit khusus platform (seperti Layanan Play Game di Android) sering kali membutuhkan plugin tambahan untuk Unity. Berbagai plugin seluler siap pakai tersedia untuk fitur khusus iOS dan Android; lampiran D mencantumkan beberapa tempat untuk mendapatkan plugin seluler. Plugin ini beroperasi dengan cara yang dijelaskan di bagian ini, kecuali bahwa kode plug-in sudah ditulis untuk Anda.

Proses untuk berkomunikasi bolak-balik dengan plugin asli mirip dengan proses untuk berkomunikasi dengan browser. Di sisi Unity, ada perintah khusus yang memanggil fungsi di dalam plugin. Di sisi plugin, plugin dapat menggunakan SendMessage() untuk mengirim pesan ke objek dalam scene Unity. Kode persisnya terlihat berbeda pada platform yang berbeda, tetapi ide umumnya selalu sama.

Sama seperti proses pembuatan awal, proses developeran plugin seluler cenderung sering berubah—bukan akhir Unity dari proses, tetapi bagian kode asli. Saya akan membahas hal-hal di tingkat tinggi, tetapi Anda harus mencari dokumentasi terbaru secara online.

Juga, plugin untuk kedua platform diletakkan di tempat yang sama di dalam Unity. Buat folder di tampilan Proyek bernama Plugin; seperti halnya folder seperti Editor, Unity menangani folder Plugin dengan cara khusus. Dalam hal ini, Unity mencari file plug-in di dalam folder Plugins. Kemudian, di dalam Plugin buat dua folder untuk Android dan iOS; Unity menyalin konten folder tersebut saat melakukan build.

Plugin iOS

"Plugin" sebenarnya hanyalah beberapa kode asli yang dipanggil oleh Unity. Pertama buat skrip di Unity untuk menangani kode asli; panggil skrip ini TestPlugin (lihat daftar berikutnya).

Daftar Skrip TestPlugin yang memanggil kode asli iOS dari Unity

```
using UnityEngine;
using System;
using System.Collections;
using System.Runtime.InteropServices;

public class TestPlugin : MonoBehaviour {
   private static TestPlugin _instance;

   public static void Initialize() {
      if (_instance != null) {
         Debug.Log("TestPlugin instance was found. Already initialized");
         return;
      }
      Debug.Log("TestPlugin instance not found. Initializing...");

      GameObject owner = new GameObject("TestPlugin_instance");
      _instance = owner.AddComponent<TestPlugin>();
      DontDestroyOnLoad(_instance);
   }

   #region iOS
   [DllImport("__Internal")]
   private static extern float _TestNumber();
```

```
    [DllImport("__Internal")]
    private static extern string _TestString(string test);
    #endregion iOS

    public static float TestNumber() {
        float val = 0f;
        if (Application.platform == RuntimePlatform.IPhonePlayer)
            val = _TestNumber();
        return val;
    }

    public static string TestString(string test) {
        string val = "";
        if (Application.platform == RuntimePlatform.IPhonePlayer)
            val = _TestString(test);
        return val;
    }
}
```

Pertama, perhatikan bahwa fungsi static Initialize() membuat objek permanen dalam scene sehingga Anda tidak perlu melakukannya secara manual di editor. Anda belum pernah melihat kode untuk membuat objek dari awal karena lebih mudah menggunakan Prefab dalam banyak kasus, tetapi dalam kasus ini lebih bersih untuk membuat objek dalam kode (sehingga Anda dapat menggunakan skrip plug-in tanpa mengedit scene).

Keajaiban utama yang terjadi di sini melibatkan perintah DLLImport dan static extern. Perintah tersebut memberi tahu Unity untuk menautkan ke fungsi dalam kode asli yang Anda berikan. Kemudian Anda dapat menggunakan fungsi-fungsi yang direferensikan dalam metode skrip ini (dengan centang untuk memastikan kode berjalan di iPhone/iOS).

Selanjutnya Anda akan menggunakan fungsi plug-in ini untuk mengujinya. Buat skrip baru bernama MobileTestObject, buat objek kosong di scene, lalu lampirkan skrip (lihat daftar berikutnya) ke objek.

Daftar Menggunakan plug-in dari MobileTestObject

```
using UnityEngine;
using System.Collections;

public class MobileTestObject : MonoBehaviour {
    private string _message;

    void Awake() {
        TestPlugin.Initialize();
    }

    // Use this for initialization
    void Start() {
        _message = "START: " + TestPlugin.TestString("ThIs Is A tEsT");
    }

    // Update is called once per frame
    void Update() {


        // Make sure the user touched the screen
        if (Input.touchCount==0){return;}

        Touch touch = Input.GetTouch(0);
        if (touch.phase == TouchPhase.Began) {
            _message = "TOUCH: " + TestPlugin.TestNumber();
        }
    }

    void OnGUI() {
        GUI.Label(new Rect(10, 10, 200, 20), _message);
    }
}
```

Skrip dalam daftar ini menginisialisasi objek plug-in dan kemudian memanggil metode plug-in sebagai respons terhadap input sentuh. Setelah ini berjalan di perangkat, Anda akan melihat pesan pengujian di sudut berubah setiap kali Anda mengetuk layar. Hal terakhir yang harus dilakukan adalah menulis kode asli yang dirujuk oleh TestPlugin. Kode pada perangkat iOS ditulis menggunakan Objective C dan/atau C, jadi kita memerlukan file header .h dan file implementasi .mm. Seperti yang dijelaskan sebelumnya, mereka harus masuk ke folder Plugins/iOS/ di tampilan Project. Buat TestPlugin.h dan TestPlugin.mm di sana; dalam file .h tulis kode dari daftar berikut.

Daftar Header TestPlugin.h untuk kode iOS

```
#import <Foundation/Foundation.h>
@interface TestObject : NSObject {
NSString* status;
}
@end
```

Cari penjelasan tentang pemrograman iOS untuk memahami apa yang dilakukan header ini; menjelaskan pemrograman iOS berada di luar buku pengantar ini. Tulis kode dari daftar berikutnya di file .mm.

Daftar Implementasi TestPlugin.mm

```
#import "TestPlugin.h"
@implementation TestObject
@end
NSString* CreateNSString (const char* string)
{
if (string)
return [NSString stringWithUTF8String:
string];
else return [NSString stringWithUTF8String: ""];
}
char* MakeStringCopy (const char* string)
{
if (string == NULL) return NULL;
char* res = (char*)malloc(strlen(string) + 1);
strcpy(res, string);
return res; }
extern "C" {
const char*
        TestString(const char* string) {
        NSString* oldString = CreateNSString(string);
        NSString* newString = [oldString uppercaseString];
        return MakeStringCopy([newString UTF8String]);
        }

        float TestNumber() {
        return (arc4random()% 100)/100.0f;
        }
}
```

Sekali lagi, penjelasan rinci tentang kode ini sedikit di luar buku ini. Perhatikan bahwa banyak fungsi string yang ada untuk mengkonversi antara bagaimana Unity mewakili data string dan apa yang digunakan kode asli. Sampel ini hanya berkomunikasi dalam satu arah, dari Unity ke plug-in. Tetapi kode asli juga dapat berkomunikasi dengan Unity dengan menggunakan metode UnitySendMessage(). Anda dapat mengirim pesan ke objek bernama di scene; selama inisialisasi, plug-in membuat TestPlugin_instance untuk mengirim pesan. Dengan kode asli di tempat, Anda dapat membangun aplikasi iOS dan mengujinya di perangkat. Sangat keren! Itulah cara membuat plug-in iOS, jadi mari kita lihat Android juga.

Plugin Android

Untuk membuat plug-in Android, sisi Unity hampir persis sama. Kami tidak perlu mengubah MobileTestObject sama sekali. Buat tambahan yang ditunjukkan dalam daftar berikut di TestPlugin.

Daftar Memodifikasi TestPlugin untuk menggunakan plug-in Android

```
...
    #region iOS
    [DllImport("__Internal")]
    private static extern float _TestNumber();

    [DllImport("__Internal")]
    private static extern string _TestString(string test);
    #endregion iOS

#if UNITY_ANDROID
    private static Exception _pluginError;
    private static AndroidJavaClass _pluginClass;
    private static AndroidJavaClass GetPluginClass() {
        if (_pluginClass == null && _pluginError == null) {
            AndroidJNI.AttachCurrentThread();
            try {
                _pluginClass = new
      AndroidJavaClass("com.companyname.testplugin.TestPlugin");
            } catch (Exception e) {
                _pluginError = e;
            }
        }
        return _pluginClass;
    }

        private static AndroidJavaObject _unityActivity;
        private static AndroidJavaObject GetUnityActivity() {
            if (_unityActivity == null) {
                AndroidJavaClass unityPlayer = new
          AndroidJavaClass("com.unity3d.player.UnityPlayer");
                _unityActivity =
            unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
            }
            return _unityActivity;
        }
    #endif

        public static float TestNumber() {
            float val = 0f;
            if (Application.platform == RuntimePlatform.IPhonePlayer)
                val = _TestNumber();
    #if UNITY_ANDROID
            if (!Application.isEditor && _pluginError == null)
                val = GetPluginClass().CallStatic<int>("getNumber");
    #endif
            return val;
        }

        public static string TestString(string test) {
            string val = "";
            if (Application.platform == RuntimePlatform.IPhonePlayer)
                val = _TestString(test);
    #if UNITY_ANDROID
            if (!Application.isEditor && _pluginError == null)
                val = GetPluginClass().CallStatic<string>("getString", test);
    #endif
            return val;
        }
    }
```

Anda akan melihat sebagian besar penambahan terjadi di dalam definisi platform UNITY_ANDROID; seperti yang dijelaskan sebelumnya dalam bab ini, arahan kompiler ini menyebabkan kode hanya berlaku untuk platform tertentu dan dihilangkan pada platform lain. Sedangkan kode iOS tidak melakukan apa pun yang akan merusak platform lain (tidak akan melakukan apa-apa, tetapi juga tidak akan menyebabkan kesalahan), kode untuk plugin Android hanya akan dikompilasi ketika Unity diatur ke platform Android .

Secara khusus, perhatikan panggilan ke AndroidJNI. Itulah sistem dalam Unity untuk menghubungkan ke Android asli. Kata lain yang mungkin membingungkan yang muncul adalah Aktivitas; di aplikasi Android, aktivitas adalah proses aplikasi. Unity adalah aktivitas aplikasi Android, jadi kode plug-in memerlukan akses ke aktivitas itu untuk menyebarkannya saat dibutuhkan.

Terakhir, Anda memerlukan kode Android asli. Sedangkan kode iOS ditulis dalam bahasa seperti Objective C dan C, Android diprogram dalam Java. Tapi kami tidak bisa begitu saja memberikan kode Java mentah untuk plug-in; plug-in harus berupa JAR yang dikemas dari kode Java. Di sini sekali lagi, detail pemrograman Android berada di luar cakupan untuk intro Unity, tetapi untuk referensi, daftar berikut menunjukkan file build Ant (ganti jalur dengan lokasi di komputer Anda; terutama perhatikan class.jar Unity untuk digunakan saat membuat plugin Android) dan daftar 12.10 menunjukkan kode Java untuk plug-in yang digunakan.

Daftar Script build.xml yang menghasilkan JAR dari kode Java

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="TestPluginJava">
<!-- Change this in order to match your configuration -->
<property name="sdk.dir"
value="LOCATION OF ANDROID SDK">
<property name="target" value="android-18">
<property name="unity.androidplayer.jarfile"
value="ApplicationsUnity/Unity.app/Contents/PlaybackEngines/
AndroidPlayer/development/bin/classes.jar">
<!-- Source directory -->
<property name="source.dir" value="LOCATION OF THIS
PROJECTAssets/Plugins/ Android/TestPlugin" >
<!-- Output directory for .class files-->
<property name="output.dir" value="LOCATION OF THIS
PROJECTAssets/Plugins/ Android/TestPlugin/classes">
<!-- Name of the jar to be created. Please note that the
name should match the name of the
class and the name placed in the
AndroidManifest.xml-->
<property name="output.jarfile" value="..TestPlugin.jar">
<!-- Creates the output
directories if they don't exist yet. -->
<targetname="-dirs"
depends="message">
<echo>Creating output directory: ${output.dir} <echo>
<mkdir dir="${output.dir}" >
<target>
<!-- Compiles this project's .java files into .class files. -->
<targetname="compile" depends="-dirs"
```

```
                description="Compiles project's .java files into
    .class files">
<javac encoding="ascii" target="1.6" debug="true"
destdir="${output.dir}" verbose="${verbose}"
includeantruntime="false">
<src path="${source.dir}" >
<classpath>
<pathelement
location="${sdk.dir}\platforms\${target}ndroid.jar">
<pathelement location="${unity.androidplayer.jarfile}">
<classpath>
</javac>
</target>
<targetname="build-jar" depends="compile">
<zip zipfile="${output.jarfile}" basedir="${output.dir}" >
<target>
<targetname="clean-post-jar">
<echo>Removing postbuild-jar-clean</echo>
<delete dir="${output.dir}">
<target>
<targetname="clean" description="Removes output files
    created by other targets.">
<delete dir="${output.dir}" verbose="${verbose}" >
<target>
<targetname="message">
<echo>Android Ant Build for Unity Android Plugin</echo>

<echo> message: Displays this
    message.
</echo>
</echo>
<echo> compile: Compiles .java files into
    .class files.
</echo>
<echo>build-jar: Compiles .class files into .jar file.
</echo>
</target> </project>
Daftar TestPlugin.java yang dikompilasi menjadi JAR
package com.companyname.testplugin;
public class TestPlugin {
private static int number = 0;
public static int getNumber() {
number++; return number;
}
public static String getString(String message) {
return message.toLowerCase();
}
}
```

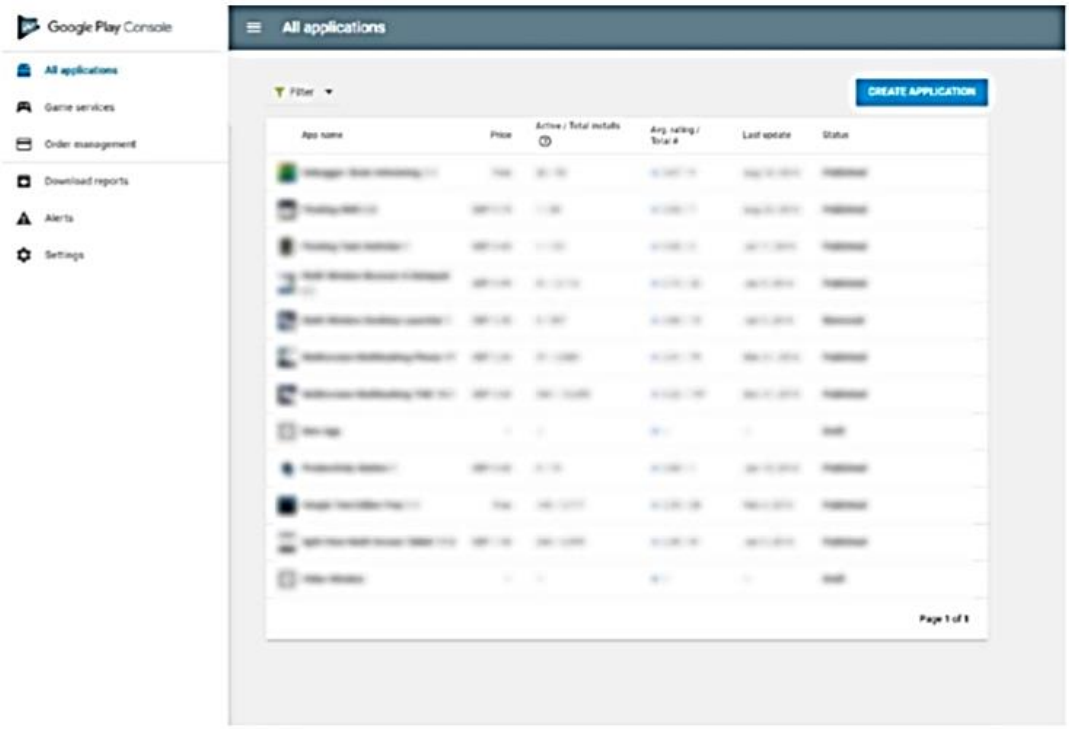
Folder Manifes Dan Sumber Daya Android

Itu tidak diperlukan untuk plug-in pengujian sederhana ini, tetapi plugin Android sering kali harus mengedit file manifes. Semua aplikasi Android dikendalikan oleh file konfigurasi utama yang disebut AndroidManifest.xml; Unity membuat file manifes dasar jika Anda tidak menyediakannya, tetapi Anda dapat menyediakannya secara manual dengan meletakkannya di Plugin/Android/di samping JAR plug-in.

Saat aplikasi Android dibuat, Unity menempatkan file manifes yang dihasilkan di folder Temp di StagingArea/AndroidManifest.xml; salin file itu untuk mengeditnya secara manual (kode yang diunduh menyertakan contoh file manifes). Demikian pula, ada folder bernama res tempat Anda dapat meletakkan sumber daya seperti ikon khusus; Anda dapat membuat res di folder plugin Android. Demikian pula, ada folder bernama res tempat Anda dapat meletakkan sumber daya seperti ikon khusus; Anda dapat membuat res di folder plugin Android.

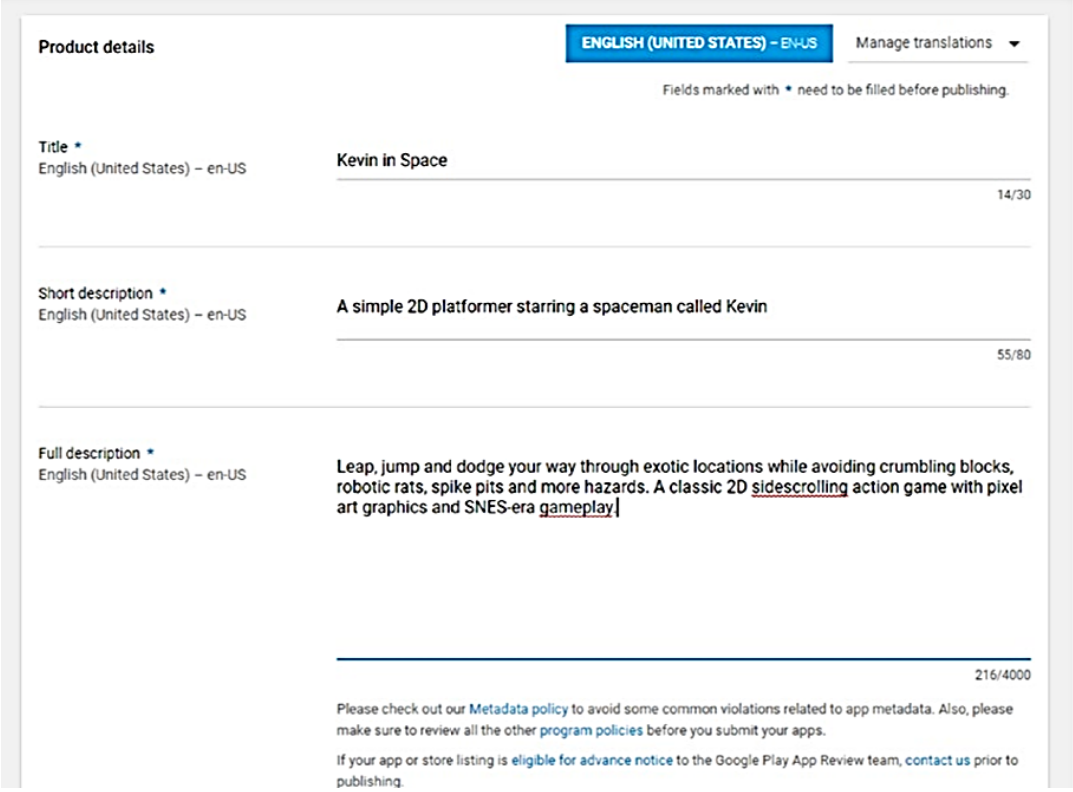
10.7 MENGUNGGAH APLIKASI ANDA

Setelah Anda memiliki APK dan ditandatangani dengan benar, Anda dapat mengunggahnya ke Play Store. Untuk melakukannya, buka https://play.google.com/apps/publish atau telusuri Konsol Developer Google Play. Jika Anda belum mendaftar, Anda harus melakukannya. Untungnya, Anda dapat menggunakan akun Google Anda yang akan membuat segalanya menjadi menyenangkan dan mudah. Saat ini ada biaya pendaftaran sebesar $25, tetapi ini adalah biaya satu kali dan tentu saja lebih baik daripada biaya tahunan yang lebih mahal yang harus ditanggung oleh developer iOS. Mudah-mudahan, ini akan menjadi salah satu dari $25 terbaik yang pernah Anda belanjakan dalam hal laba atas investasi Anda. Bahkan untuk proyek kesombongan, $25 tidak akan merusak bank. Konsol developer (Gambar 10.11) adalah tempat Anda dapat melihat aplikasi dan statistiknya setelah Anda memiliki beberapa aplikasi langsung. Anda akan dapat memeriksa ulasan pengguna, menerapkan pembaruan, dan memeriksa laporan pendapatan dan bug. Anda belum memiliki aplikasi apa pun, jadi ini akan cukup kosong untuk saat ini.



Gambar 10.11 Konsol developer saya

Membuat Cantuman Toko Anda Untuk memulai, temukan dan klik Buat Aplikasi. Anda akan diminta untuk memilih bahasa dan memberinya judul dan kemudian Anda akan dibawa ke halaman lain di mana Anda dapat memasukkan lebih banyak detail, termasuk deskripsi, terjemahan, dan berbagai aset grafis. Kami akan mengisi ini terlebih dahulu.



Gambar 10.12 Tulis deskripsi menarik yang akan membantu game Anda menonjol

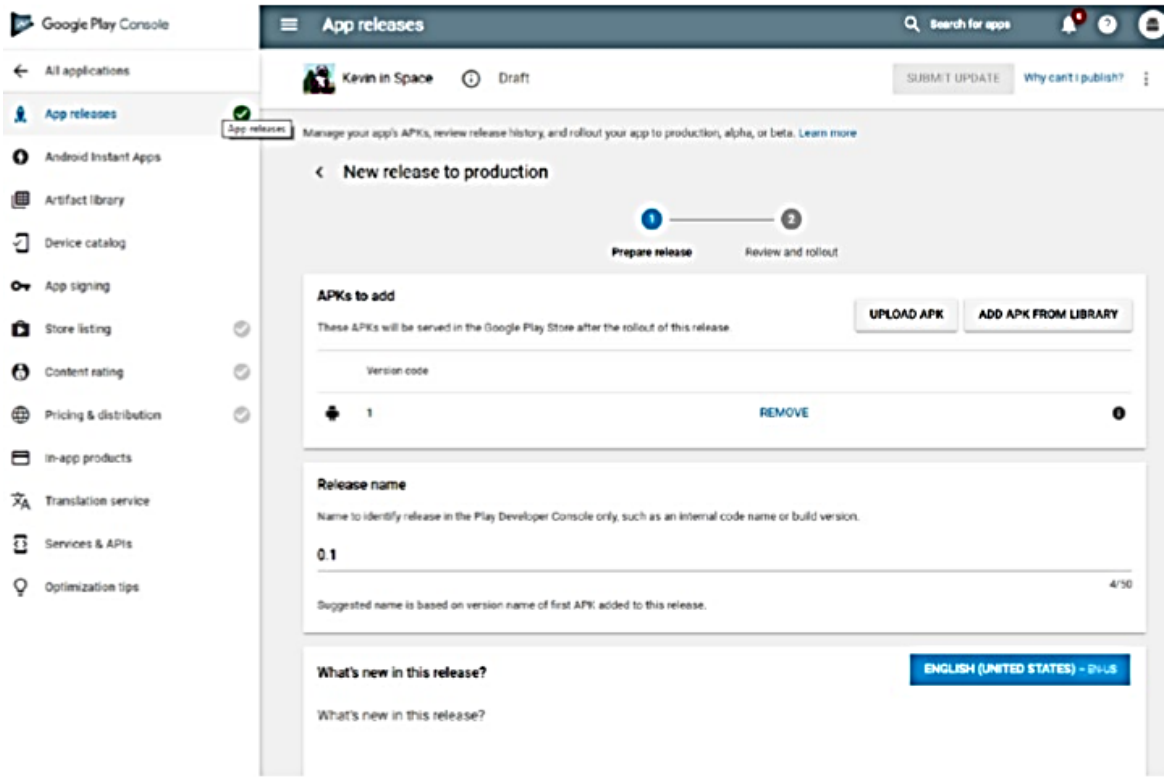
Anda dapat memasukkan judul, deskripsi, dan lainnya seperti yang Anda harapkan. Kami akan membahas lebih detail tentang apa yang harus diletakkan di sini untuk hasil terbaik sebentar lagi. Untuk gambar, ini termasuk ikon beresolusi tinggi (512 x 512) yang akan ditampilkan di samping cantuman aplikasi Anda di Play Store, grafik fitur (1024 x 500) yang akan ditampilkan di bagian atas laman, dan promo grafik (180 x 120). Grafik promo terutama digunakan untuk versi Android yang lebih lama dan tidak diperlukan untuk pengiriman, tetapi ada baiknya menghabiskan waktu untuk membuat gambar untuk setiap kategori. Anda menghabiskan waktu selama ini untuk membangun aplikasi Anda—jangan jatuh pada rintangan terakhir. Jika Anda juga menggunakan video promo (yang layak dilakukan), maka grafik fitur Anda akan memiliki ikon rotate di atasnya. Bagi mereka yang membuat aplikasi Daydream, Anda harus membuat video 360 derajat stereoskopik. Semoga berhasil! Aplikasi Android TV memerlukan gambar spanduk 1280 x 720. Anda juga akan melihat bahwa Anda dapat menambahkan screenshoot game Anda dan Anda dapat memilih yang berbeda untuk ponsel, tablet, TV, dan Android Wear. Di sinilah Anda akan menempatkan bidikan permainan Anda dalam aksi.

Lanjutkan menggulir ke bawah halaman dan Anda akan dapat memilih jenis aplikasi (Game) kategori (mungkin 'Aksi' dalam kasus ini), peringkat konten dan detail kontak Anda. Anda harus kembali ke peringkat konten nanti dan untuk mendapatkan sertifikat, Anda harus mengisi berbagai pertanyaan. Masukkan detail kontak Anda selanjutnya. Saya sarankan untuk menyiapkan alamat email baru jika menurut Anda ada kemungkinan game Anda menjadi hit besar. Siapa pun dapat menghubungi Anda melalui sini, dan Anda dapat mengharapkan banyak korespondensi jika aplikasi Anda diluncurkan. Itu termasuk pujian yang bagus serta

beberapa kritik yang sama sekali tidak masuk akal dan banyak omong kosong. Jadi berhati-hatilah saat memberikan detail personal Anda. Anda kemudian dapat memilih untuk mengirimkan kebijakan privasi jika Anda mau, atau biarkan kosong untuk saat ini. Jika Anda membuat game seluler yang tidak mengumpulkan data personal apa pun dari pengguna, Anda mungkin tidak perlu melalui langkah ini.

Mengunggah APK

Untuk mengunggah APK Anda, buka Apps Releas (di sebelah kiri) dan klik Kelola Produksi. Tentu saja, ini dapat berubah, tetapi dengan satu atau lain cara Anda harus memiliki opsi untuk Membuat Rilis. Di sini Anda dapat mengklik Unggah APK, dan jika semuanya berjalan sesuai rencana, itu harus lulus tes dan diunggah ke toko (belum hidup dulu, jangan khawatir). Anda seharusnya dapat melihat kode versi dan nama rilis yang benar, sesuai dengan apa yang Anda masukkan ke dalam Pengaturan Player. Judul aplikasi Anda harus berada di kiri atas halaman, di samping ikon yang Anda pilih. Anda akan kembali ke sini saat ingin menambahkan pembaruan dan Anda dapat menambahkan detail "Apa yang baru dalam rilis ini?" untuk memberi tahu pengguna Anda apa yang telah berubah. Jika ini tidak berhasil karena alasan apa pun, masalahnya mungkin karena keystore Anda, jadi lihat lagi bagian pertama bab ini.



Gambar 10.13 Sejauh ini baik

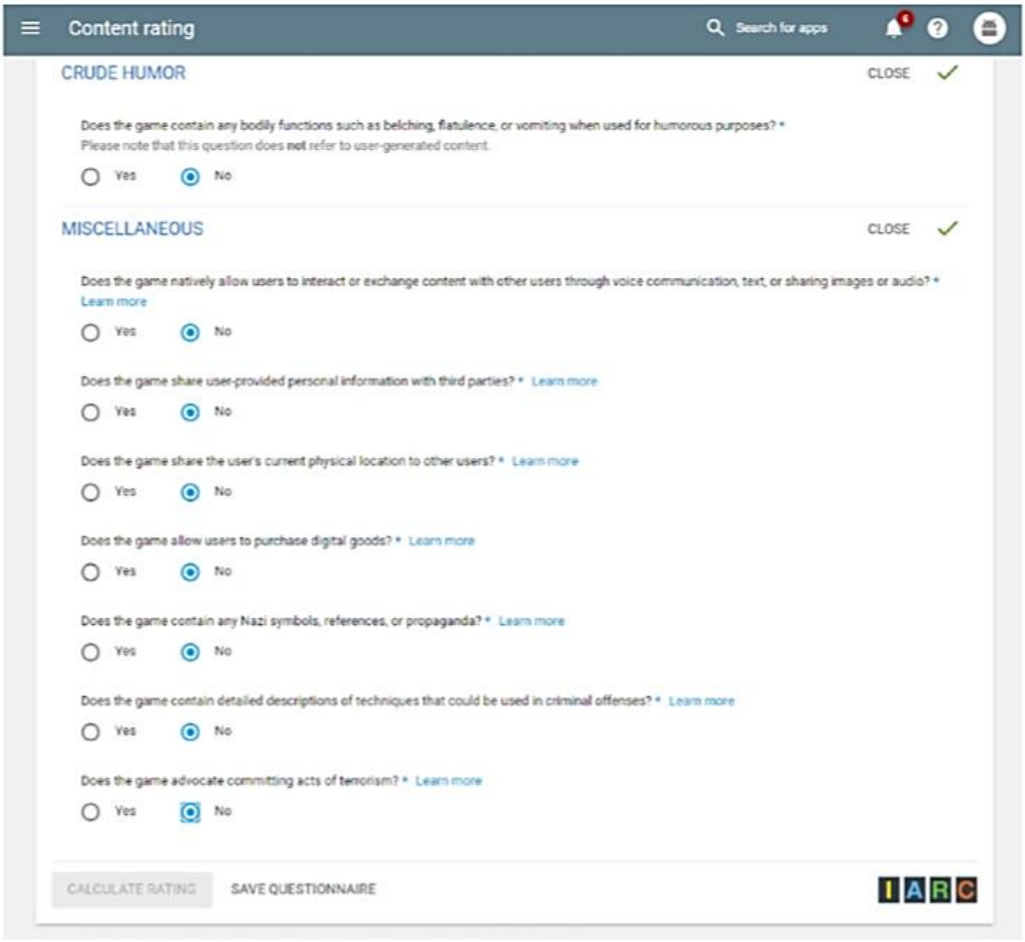
Untuk saat ini, Anda hanya akan menyimpan ini sebagai draf. Anda akan dapat kembali saat tiba waktunya untuk melakukan siaran langsung.

Pengaturan Lainnya

Ada banyak pengaturan lain yang perlu dipertimbangkan di Konsol juga, baik wajib maupun opsional. Kami akan segera membahas beberapa di antaranya di bagian ini.

Rating Konten

Untuk membuat game Anda tersedia secara luas, Anda harus menyelesaikan proses rating konten. Klik Rating Konten di sebelah kiri, pilih Lanjutkan, dan jawab pertanyaannya. Mereka cukup jelas.

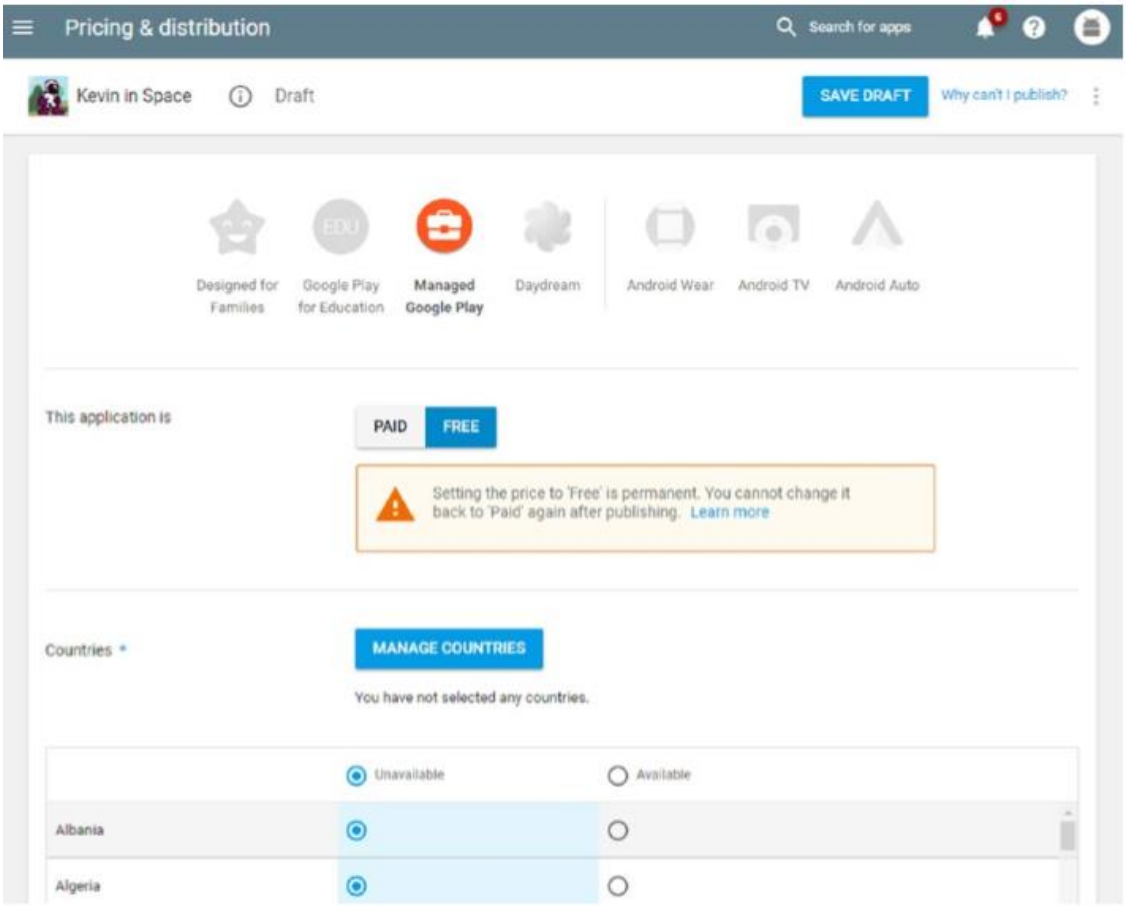


Gambar 10.14 Anda mungkin ingin mempertimbangkan kembali referensi Nazi itu

Klik Simpan Cookiessioner, lalu Hitung Peringkat untuk mendapatkan klasifikasi Anda di berbagai wilayah, dan terakhir Terapkan Peringkat.

Harga dan Distribusi

Anda juga perlu memutuskan berapa banyak yang ingin Anda kenakan untuk aplikasi Anda, atau apakah Anda ingin membuatnya gratis. Perhatikan bahwa jika aplikasi Anda dijual untuk mendapatkan uang, Anda selalu dapat membuatnya gratis nanti. Namun, setelah Anda membuatnya gratis, tidak ada jalan untuk kembali kecuali Anda mengupload listingan toko yang sama sekali baru.



Gambar 10.15 Pertimbangkan model bisnis terbaik untuk aplikasi Anda jika Anda ingin mendapat untung darinya

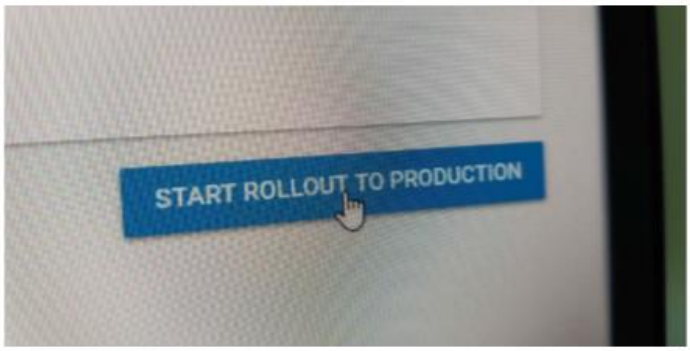
Bagian lain dari halaman ini memungkinkan Anda untuk memilih negara yang Anda inginkan untuk menyediakan aplikasi Anda. Anda mungkin ingin membuatnya tersedia di mana-mana, karena dengan cara itu jumlah maksimum orang yang dapat menikmati kreasi Anda (dan membayar Anda). Anda juga harus menjawab beberapa pertanyaan lagi di bawah layar itu, seperti apakah aplikasi berisi iklan atau tidak dan apakah aplikasi memenuhi Pedoman Konten Android dan undang-undang ekspor AS. Sekali lagi, ini semua harus cukup jelas.

Setelah Anda menyelesaikan bagian Rilis Aplikasi, Cantuman Toko, Rating Konten, dan Penetapan Harga dan Distribusi, aplikasi Anda akan siap dipublikasikan. Tetapi jika Anda bertanya-tanya untuk apa bagian lainnya, baca terus. Katalog Perangkat menunjukkan perangkat mana yang dapat menjalankan aplikasi Anda, dan Anda akan memiliki opsi untuk memfilter beberapa jika Anda lebih suka itu tidak tersedia di perangkat tersebut. Layanan Terjemahan memungkinkan Anda translate aplikasi Anda untuk wilayah lain. Layanan dan API adalah untuk mengakses alat eksternal seperti Firebase ("layanan backend" Google—jangan khawatir tentang itu), dan Tips Pengoptimalan hanya membagikan saran berdasarkan cantuman Anda saat ini (layak dibaca). Produk Dalam Aplikasi adalah tempat Anda dapat mengelola pembelian dalam aplikasi, tetapi jika aplikasi Anda tidak memilikinya, tidak ada yang dapat dilakukan di sini. Perpustakaan Artefak memungkinkan Anda mengunduh APK dan bit lain yang mungkin telah Anda unggah ke Play Store

Aplikasi Instan Android adalah fitur yang relatif baru yang memungkinkan pengguna menjalankan aplikasi tanpa harus mengunduh dan menginstalnya secara permanen, tetapi itu tidak akan berlaku di sini. Tidak hanya fitur tersebut belum tersedia untuk semua orang, tetapi

ukuran file besar yang terlibat dengan sebagian besar game (belum lagi sifatnya) berarti bahwa itu tidak akan sepenuhnya sesuai untuk sebagian besar developer. Terakhir, Penandatanganan Aplikasi adalah tempat Anda dapat mendaftar di skema Penandatanganan Aplikasi Google untuk menyimpan keystore Anda di cloud sehingga tidak pernah hilang. Pilihan yang praktis, tetapi tidak wajib, jadi terserah Anda apakah Anda ingin mengambil langkah ekstra ini. Anda selalu dapat memutuskan untuk melakukan ini nanti jika Anda mau. Namun, setelah Anda menempuh rute ini, Anda tidak akan dapat kembali. Saat aplikasi Anda aktif, Anda dapat kembali ke Konsol kapan saja untuk melihat semua proyek yang Anda publikasikan. Mengklik satu akan memungkinkan Anda melihat seluruh host data, termasuk pendapatan, jumlah unduhan, dan banyak lagi.

Dengan ini, Anda siap untuk mempublikasikan. Ini adalah kesempatan yang penting, terutama jika Anda telah mengerjakan aplikasi Anda selama berbulan-bulan atau bertahun-tahun, jadi tuangkan diri Anda sendiri. Kembali ke Rilis Aplikasi Edit Rilis, gulir ke bawah ke bawah, dan tekan Mulai Peluncuran ke Produksi.



Gambar 10.16 Tombol konfirmasi untuk memulai merilis aplikasi

Konfirmasikan, dan dengan itu ... aplikasi Anda dirilis! Atau, hampir.... Sebenarnya, aplikasi Anda sekarang akan ditinjau, dan Anda akan melihat tulisan "Pending Publication" di bagian atas halaman. Proses publikasinya otomatis, artinya ditangani oleh algoritme, bukan oleh kurator manusia yang sebenarnya (tidak seperti App Store Apple). Ini kabar baik karena artinya aplikasi Anda akan aktif dalam beberapa jam ke depan, siap untuk diunduh dan ditinjau oleh orang-orang. Sangat jarang ada sesuatu yang harus ditolak pada tahap ini. Tentu saja, itu bukan lisensi untuk mengabaikan persyaratan dan kesepakatan

Membuat Lebih Banyak Unduhan

Setelah aplikasi Anda aktif, pekerjaan Anda masih jauh dari selesai. Bukan hanya kewajiban moral Anda untuk terus memperbarui dan meningkatkan aplikasi saat Anda menggunakannya, tetapi Anda juga perlu memastikan bahwa Anda secara aktif mempromosikan kreasi Anda dan mendorong unduhan. Ini bukan kasus "membangunnya dan mereka akan datang." Saat ini ada terlalu banyak aplikasi hebat di Android, dan pasar sangat jenuh. Sebaliknya, terserah Anda untuk menyebarkan berita dan memastikan orang-orang tertarik untuk melihat kreasi Anda. Anda berhutang pada diri sendiri. Jadi, bagaimana Anda bisa memastikan orang benar-benar menemukan dan mengunduh aplikasi Anda? Bagian ini berbicara tentang beberapa tips pemasaran yang mungkin membantu.

Pikirkan Tentang SEO SEO adalah singkatan dari Search Engine Optimization. Ini adalah masalah besar dalam dunia pemasaran Internet, tetapi juga memiliki peran untuk dimainkan di antara para developer. Itu karena Google Play Store pada dasarnya adalah mesin pencari itu sendiri, dan orang akan sering menemukan aplikasi baru dengan mencari.

Gambar 10.17 Kevin in Space terbuka untuk bisnis

Konsep kunci yang harus dipahami di sini adalah penggunaan kata kunci. Kata kunci adalah kata atau frasa yang mungkin ditelusuri seseorang untuk menemukan aplikasi Anda. Dalam kasus kami, contoh yang baik mungkin termasuk yang berikut:

- Platformer 2D
- Sidescrolling Game
- Pixel Art
- Retro Game

Untuk meningkatkan peluang kami ditemukan, kami mungkin bertujuan untuk mencoba dan memasukkan istilah-istilah itu beberapa kali dalam deskripsi. Jangan berlebihan, karena itu terlihat seperti spam dan bisa membuat aplikasi Anda dihapus. Cobalah untuk memasukkan beberapa penyebutan secara alami. Semakin banyak Anda menulis deskripsi lengkap, semakin mudah, dan menambahkan deskripsi yang lebih panjang juga merupakan praktik yang didorong oleh Google. Benar-benar jual aplikasi Anda dan beri tahu orang-orang mengapa mereka harus tertarik. Juga pertimbangkan bahwa istilah yang terlalu populer akan lebih sulit untuk diperingkat, karena banyaknya persaingan. Carilah sweet spot itu: istilah yang diminati tetapi cukup kabur sehingga belum banyak konten yang tersedia.

Pilih Nama Anda dengan Bijak

Dan tentu saja, salah satu cara terbaik untuk memastikan Anda mendapat peringkat tinggi untuk istilah kunci adalah dengan memberi nama aplikasi Anda menggunakan frasa itu. Misalnya, Anda benar-benar dapat memanggil game Anda Retro 2D Platformer atau semacamnya. Namun, berhati-hatilah: melakukan ini juga berarti aplikasi Anda tidak akan memiliki kepersonalan yang sama atau merek yang kuat untuk Anda promosikan. Ini juga dapat membuat pengguna tidak nyaman dan dapat mempersulit Anda untuk berpromosi di luar Play Store. Nama yang mengomunikasikan sesuatu tentang game Anda bukanlah ide yang

buruk, tetapi cobalah untuk menjadi kreatif dan menarik. Pilih nama yang menyampaikan emosi (seperti Limbo atau Angry Birds) dan yang memancing minat (VVVVVV atau Thomas Was Alone). Idealnya, seseorang harus segera mengetahui tentang apa permainan Anda atau ingin mempelajari lebih lanjut segera setelah mereka mendengar apa namanya.

Temukan Rute Menuju Pasar

Pemasaran juga berarti sesekali keluar dan meneriakkan game baru Anda yang luar biasa. Cara terbaik untuk melakukannya adalah menemukan rute ke pasar, yang pada dasarnya adalah tempat di mana orang-orang yang mungkin tertarik berkumpul. Misalnya, ini mungkin grup Facebook atau Subreddit yang didedikasikan untuk genre permainan tertentu (coba juga www.reddit.com/r/playmygame/).

Gambar 10.18 Game Anda sekarang tersedia untuk diunduh jutaan orang

Rute ke pasar juga merupakan sesuatu yang perlu dipikirkan pada tahap desain awal game Anda. Tentu saja, Anda harus membuat game yang ingin Anda buat dan itu membuat Anda bersemangat, tetapi pikirkan juga peluang pemasaran apa yang akan ada untuk game Anda. Dengan menargetkan ceruk tertentu, Anda dapat menghindari menjadi ikan kecil di kolam besar dan menghindari persaingan. Lebih penting lagi, dengan menargetkan pengguna tertentu (disebut persona), Anda dapat memberikan diri Anda rute yang lebih spesifik ke pasar. Karena itu, Kevin in Space agak sulit dipasarkan karena tidak menonjol dan tidak menarik bagi siapa pun secara khusus. Tetapi jika ini adalah game tentang pahlawan yang berlari bebas, kami dapat mempostingnya di forum yang ditujukan untuk pelari gratis. Jika itu adalah permainan dengan soundtrack synthwave yang mengagumkan, kami dapat mencoba dan mendapatkan publisitas dari situs web synthwave. Demikian juga, pikirkan tentang kontak dan sumber daya yang saat ini tersedia untuk Anda. Tentu saja, Anda harus mencoba dan meminta teman Anda untuk mengunduh aplikasi Anda (dan memberikan ulasan yang bagus!), tetapi mungkin Anda mengenal seseorang yang menulis untuk situs web besar, misalnya?

Dapatkan Ulasan Bagus

SEO di Play Store sedikit berbeda dengan SEO di Google.com karena memperhitungkan berbagai faktor tambahan. Di antaranya adalah ulasan yang ditinggalkan oleh pengguna Anda, dan lebih banyak ulasan positif = lebih banyak unduhan. Mendapatkan ulasan bagus adalah masalah membuat game hebat yang Anda banggakan, tetapi juga merupakan ide bagus untuk meminta pengguna Anda untuk meninjaunya dengan pop-up sesekali. Jelaskan bagaimana itu

benar-benar akan membantu Anda, tetapi jangan mencoba taktik curang apa pun untuk memaksakan ulasan positif—Google tidak menyukai hal itu. Jika Anda mendapatkan ulasan yang buruk, selalu merupakan ide yang baik untuk merespons dengan cepat. Hal itu tidak hanya akan menunjukkan bahwa Anda adalah developer penuh perhatian yang benar-benar peduli dengan pengguna Anda, tetapi jika Anda menawarkan solusi, Anda bahkan mungkin menemukan bahwa pengguna mengubah skor mereka. Jangan pernah membayar untuk ulasan. Praktik ini dapat membuat aplikasi Anda dihapus dan hanya akan menyebabkan pengguna yang tidak senang.

Perbarui Secara Teratur

Memperbarui aplikasi Anda secara teratur juga penting. Tidak hanya mendorong lebih banyak ulasan positif, itu juga memberi aplikasi Anda momen di bawah sinar matahari di bawah bagian Game Baru + Diperbarui di Play Store. Lebih banyak eksposur seperti itu berarti lebih banyak kesempatan bagi orang-orang untuk menemukan aplikasi Anda dan mencobanya.

Pilih Gambar dan Teks yang Tepat

Jika seseorang tiba-tiba menemukan aplikasi Anda, mereka akan memiliki opsi untuk mengklik dan membaca lebih banyak atau hanya melewati Anda tanpa berhenti untuk membaca. Faktor terbesar yang mempengaruhi keputusan ini kemungkinan adalah ikon yang Anda pilih, jadi tidak perlu dikatakan bahwa ini harus bagus. Tujuannya adalah untuk menonjol dari keramaian dan menarik minat sambil juga mengomunikasikan dengan tepat untuk siapa game Anda. Jangan mencoba untuk menarik semua orang tetapi merangkul genre, niche, dan gaya yang Anda pilih. Pikirkan tentang kebiasaan Anda sendiri: gambar seperti apa yang akan menarik minat Anda saat menelusuri sesuatu yang baru untuk dimainkan? Bagi saya, itu pasti akan menjadi game yang terlihat futuristik, penuh aksi, dan indie. Saya menjauhi hal-hal seperti Clash of Clans atau game freemium lain yang tampak halus yang jelas ditujukan untuk pasar kasual. Itu hanya saya—tetapi dengan mengetahui apa yang dicari pengguna Anda, Anda dapat mengetahui apa yang harus dikomunikasikan melalui gambar Anda. Tentu saja, Anda kemudian perlu merobohkannya dengan gambar fitur dan screenshoot Anda juga. Hal yang sama berlaku untuk deskripsi Anda. Menggunakan beberapa kata kunci bisa menjadi strategi yang baik, tetapi menulis untuk pengguna jauh lebih penting. Melakukannya berarti menarik perhatian dengan cepat dengan pernyataan pembuka yang kuat, menggunakan poin-poin untuk menjual fitur utama game Anda, dan menggunakan bahasa yang emosional untuk mencoba dan mendorong klik cepat. Bacalah tulisan persuasif karena itu bisa membuat perbedaan nyata. Dan lagi: kenali pengguna Anda dan arahkan promosi Anda ke mereka. Menarik bagi semua orang berarti tidak menarik bagi siapa pun.

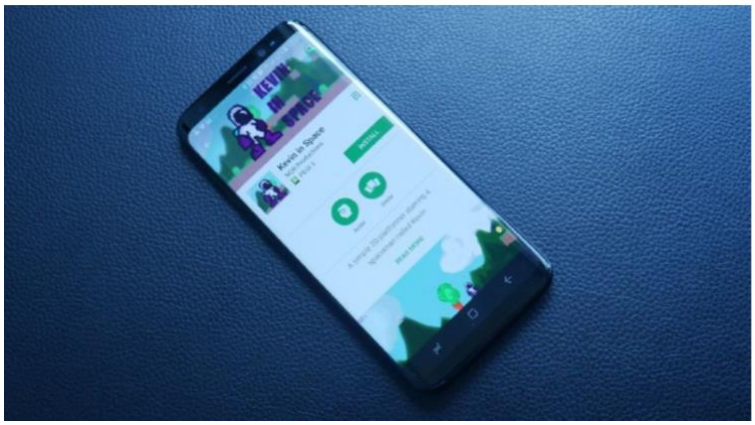
Buat Buzz

Terakhir, coba buat buzz untuk aplikasi Anda dengan mengirimkan siaran pers ke situs web game dan saluran Android/seluler. Coba tawarkan APK gratis kepada tokoh YouTube sebagai imbalan atas ulasan atau video "ayo main". Bidik kepersonalan yang lebih kecil dan saluran yang lebih mungkin untuk merespons, dan jika permainan Anda bagus, saluran yang lebih besar akan memperhatikan. Pertimbangkan untuk membuat situs web untuk game Anda sendiri yang dapat Anda promosikan secara terpisah. Ditto untuk halaman media sosial. Membuat situs web sangat mudah akhir-akhir ini menggunakan WordPress (www.wordpress.com). Juga pikirkan tentang membuat buzz sebelum rilis Anda juga. Anda dapat melakukannya dengan membuat blog developer dan membicarakan pembuatan aplikasi Anda, atau dengan merilis berita menggoda ke situs dan saluran kecil. Jika Anda benar-benar ambisius, coba jalankan kampanye di Kickstarter atau Indiegogo, situs crowdfunding di mana orang-orang akan memberikan dukungan finansial untuk membantu Anda menyelesaikan permainan Anda. Crowdfunding tidak hanya dapat membayar proyek Anda,

tetapi juga memastikan ada komunitas yang besar, aktif, dan terlibat di sekitar peluncuran game Anda. Plus itu membuatnya lebih mudah untuk diperhatikan oleh media game. Ingat, situs dan majalah tidak ingin menawarkan promosi gratis kepada Anda, tetapi mereka ingin meliput cerita yang menarik. Namun, sekali lagi, agar ini berhasil, Anda memerlukan USP yang menarik—apakah itu berarti menciptakan penerus spiritual untuk waralaba tercinta, menghidupkan kembali genre yang terlupakan, atau mencoba sesuatu yang sangat baru dan menarik. Beri orang alasan untuk mendukung permainan Anda dengan menjadikannya sebuah gerakan, bukan sekadar produk lain. - buatlah sesuatu yang dapat mereka percayai dan buat mereka bersemangat!

Menutup Komentar

Dan dengan itu, Anda sendirian. Saya telah membawa Anda sejauh yang saya bisa, dan sisanya terserah Anda. Saya percaya bahwa Anda akan menemukan jalan Anda sendiri dan menciptakan sesuatu yang Anda banggakan dan yang mendapat perhatian dan penghargaan yang layak.



Gambar 10.19 Sisanya terserah padamu

Ingatlah untuk memulai dari yang kecil dan membangun dari sana. Ciptakan sesuatu yang berbeda dan unik. Dan yang terpenting, bersenang-senanglah melakukannya. Jika Anda benar-benar menikmati proses penciptaan dan jika Anda mencurahkan hati dan jiwa Anda ke dalam proyek gairah Anda sendiri, itu akan terlihat dalam produk akhir. Buatlah game yang ingin Anda mainkan dan jangan takut untuk merangkul identitas Anda sendiri. Dan jika game Anda tidak sukses besar? Lanjut ke produk selanjutnya. Ada banyak peluang yang masuk ke membuat sukses besar juga. Saya berharap Anda beruntung dengan proyek Anda dan berharap Anda benar-benar menjatuhkannya dari taman. Ingat saya ketika Anda kaya dan terkenal!

Ringkasan

Desain Game

Seluruh seri Kredit Ekstra melampaui segelintir video yang disponsori oleh Unity ini. Ini mencakup banyak hal tetapi sebagian besar berfokus pada disiplin desain game. Desain game adalah proses mendefinisikan game dengan menciptakan tujuan, aturan, dan tantangannya. Desain game jangan disamakan dengan desain visual, yaitu mendesain tampilan, bukan fungsi; ini adalah kesalahan umum karena rata-rata orang paling akrab dengan "desain" dalam konteks "desain grafis."

Salah satu bagian terpenting dari desain game adalah membuat mekanika game; ini adalah tindakan individu (atau sistem tindakan) dalam permainan. Mekanik dalam permainan sering kali diatur oleh aturannya, sedangkan tantangan dalam permainan umumnya berasal dari penerapan mekanisme pada situasi tertentu. Misalnya, berjalan di sekitar permainan adalah mekanik, sedangkan labirin adalah semacam tantangan berdasarkan mekanik itu.

Memikirkan desain game bisa jadi rumit bagi pendatang baru dalam developeran game. Di satu sisi, game yang paling sukses (dan memuaskan untuk dibuat!) dibangun dengan mekanisme permainan yang menarik dan inovatif. Di sisi lain, terlalu mengkhawatirkan desain game pertama Anda dapat mengalihkan perhatian Anda dari aspek developeran game lainnya, seperti mempelajari cara memprogram game. Anda lebih baik memulai dengan meniru desain game yang ada (ingat, saya hanya berbicara tentang memulai; mengkloning game yang sudah ada bagus untuk latihan awal, tetapi pada akhirnya Anda akan memiliki keterampilan dan pengalaman yang cukup untuk berkembang lebih jauh) .

Konon, setiap developer game yang sukses harus penasaran dengan desain game. Ada banyak cara untuk mempelajari lebih lanjut tentang desain game—Anda sudah tahu tentang video Kredit Ekstra, tetapi berikut adalah beberapa situs web lain:

www.gamasutra.com www.lostgarden.com www.sloperama.com

Ada juga sejumlah buku bagus tentang masalah ini, seperti berikut ini:

- Workshop Game Design, Edisi Ketiga, oleh Tracy Fullerton (AK Peters/CRC Press, 2014)
- A Theory of Fun for Game Design, Edisi Kedua, oleh Raph Koster (O'Reilly Media, 2013)
- The Art of Game Design, Edisi Kedua , oleh Jesse Schell (AK Peters/CRC Press, 2014)

Memasarkan Game Anda

Dalam video Kredit Ekstra, video keempat adalah tentang memasarkan game Anda. Terkadang developer game menunda memikirkan pemasaran. Mereka hanya ingin berpikir tentang membangun permainan dan tidak memasarkannya, tetapi sikap itu mungkin akan menghasilkan permainan yang gagal. Game terbaik di dunia tetap tidak akan berhasil jika tidak ada yang mengetahuinya! Kata pemasaran sering kali membangkitkan pemikiran tentang iklan, dan jika Anda memiliki anggaran, menjalankan iklan untuk game Anda tentu saja merupakan salah satu cara untuk memasarkannya. Tetapi ada banyak cara murah atau bahkan gratis untuk menyebarkan berita tentang game Anda. Spesifik cenderung berubah dari waktu ke waktu, tetapi strategi keseluruhan yang disebutkan dalam video itu termasuk men-tweet tentang game Anda (atau memposting di media sosial secara umum, bukan hanya Twitter) dan membuat video cuplikan untuk dibagikan di YouTube dengan pengulas, blogger, dan sebagainya. Jadilah gigih dan menjadi kreatif!

Lampiran 1 Alat eksternal yang digunakan bersama Unity

Mengembangkan game menggunakan Unity bergantung pada berbagai alat software eksternal untuk menangani berbagai tugas. Dalam bab 1 kita telah membahas satu alat eksternal; MonoDevelop secara teknis adalah aplikasi terpisah, meskipun dibundel bersama dengan Unity. Dengan cara yang sama, developer mengandalkan serangkaian alat eksternal untuk melakukan pekerjaan yang tidak bersifat internal untuk Unity.

Ini bukan untuk mengatakan bahwa Unity tidak memiliki kemampuan yang seharusnya dimiliki. Sebaliknya, proses developeran game begitu kompleks dan beragam sehingga setiap software yang dirancang dengan baik dengan fokus yang jelas dan pemisahan yang bersih dari masalah pasti akan membatasi dirinya untuk menjadi baik pada subset proses yang terbatas. Dalam hal ini, Unity berkonsentrasi untuk menjadi perekat dan mesin yang menyatukan semua konten game dan membuatnya berfungsi. Membuat semua konten itu dilakukan dengan alat lain; mari kita lihat beberapa kategori software yang mungkin berguna bagi Anda.

B.1. Alat pemrograman

Kami telah melihat MonoDevelop, alat pemrograman paling signifikan yang digunakan bersama Unity. Tetapi ada beberapa alat pemrograman lain yang harus diperhatikan, seperti yang akan Anda lihat di bagian ini.

Visual Studio

Seperti yang disebutkan dalam bab 1, meskipun Unity hadir dengan MonoDevelop dan Anda dapat menggunakan IDE tersebut pada Windows dan Mac, pada Windows Anda juga dapat memilih untuk menggunakan Visual Studio. Baru-baru ini Microsoft mengakuisisi SyntaxTree, sebuah perusahaan yang telah meningkatkan integrasi Visual Studio:
http://unityvs.com

Xcode

Xcode adalah lingkungan pemrograman yang disediakan oleh Apple (khususnya IDE, tetapi juga termasuk SDK untuk platform Apple). Meskipun Anda masih akan melakukan sebagian besar pekerjaan dalam Unity, Anda perlu menggunakan Xcode untuk menyebarkan game ke iOS. Pekerjaan itu sering kali melibatkan proses debug atau pembuatan profil aplikasi Anda menggunakan alat di Xcode:
https://developer.apple.com/xcode/

Android SDK

Mirip dengan bagaimana Anda perlu menginstal Xcode untuk menyebarkan ke iOS, Anda perlu mengunduh Android SDK untuk menyebarkan ke Android. Tidak seperti saat membuat game iOS, Anda tidak perlu menjalankan alat developeran apa pun di luar Unity — Anda hanya perlu mengatur preferensi di Unity yang mengarah ke Android SDK:
http://developer.android.com/sdk/index.html

SVN, Git, atau Mercurial

Setiap proyek developeran software berukuran layak akan melibatkan banyak revisi kompleks pada file kode, sehingga pemrogram telah mengembangkan kelas software yang disebut VCS (sistem kontrol versi) untuk menangani masalah ini. Tiga dari sistem yang paling populer adalah Subversion (juga dikenal sebagai SVN), Git, dan Mercurial; jika Anda belum menggunakan VCS, saya sangat menyarankan untuk mulai menggunakannya. Unity mengisi folder proyek dengan file temp dan pengaturan ruang kerja, tetapi hanya dua folder yang perlu dalam kontrol versi adalah Aset (pastikan kontrol versi Anda mengambil file meta yang dihasilkan oleh Unity) dan Pengaturan Proyek:

- http://subversion.apache.org/
- http://git-scm.com/
- http://mercurial.selenic.com/wiki/Mercurial

Aplikasi seni 3D

Meskipun Unity sangat mampu menangani grafik 2D (dan bab 5 dan 6 berfokus pada grafik 2D), Unity berasal dari game engine 3D dan terus memiliki fitur grafik 3D yang kuat. Banyak seniman 3D bekerja dengan setidaknya satu paket software yang dijelaskan di bagian ini.

Maya

Maya adalah paket seni dan animasi 3D dengan akar yang dalam dalam pembuatan film. Kumpulan fitur Maya mencakup hampir setiap tugas yang muncul untuk seniman 3D, mulai dari membuat animasi sinematik yang indah hingga membuat model siap-game yang efisien. Animasi 3D yang dilakukan di Maya (seperti karakter berjalan) dapat diekspor ke Unity:
www.autodesk.com/products/autodesk-maya/overview

3ds Maks

Paket seni dan animasi 3D lainnya yang banyak digunakan, 3ds Max menawarkan rangkaian fitur yang hampir identik dan cukup sebanding dalam alur kerja dengan Maya. 3ds Max hanya berjalan di Windows (sedangkan alat lain, termasuk Maya, bersifat lintas platform), tetapi alat ini sering digunakan di industri game:
www.autodesk.com/products/autodesk-3ds-max/overview

Blender

Meskipun tidak biasa digunakan dalam industri game seperti 3ds Max atau Maya, Blender juga sebanding dengan aplikasi lain tersebut. Blender juga mencakup hampir semua tugas seni 3D, dan yang terbaik, Blender adalah open source. Mengingat tersedia secara gratis di semua platform, Blender adalah satu-satunya aplikasi seni 3D yang diasumsikan tersedia oleh buku ini:
www.blender.org

Editor Gambar 2D

Gambar 2D sangat penting untuk semua game, baik yang ditampilkan langsung untuk game 2D atau sebagai tekstur pada permukaan model 3D. Beberapa alat grafis 2D sering muncul dalam developeran game, seperti yang akan Anda lihat di bagian ini.

Photoshop

Photoshop adalah aplikasi gambar 2D yang paling banyak digunakan. Alat-alat di Photoshop dapat digunakan untuk menyempurnakan gambar yang ada, menerapkan filter gambar, atau bahkan melukis gambar dari awal. Photoshop mendukung lusinan format file yang berbeda, termasuk semua format gambar yang digunakan di Unity:
www.photoshop.com

GIMP

Singkatan dari GNU Image Manipulation Program, ini adalah aplikasi grafis 2D open source yang paling terkenal. GIMP membuntuti Photoshop dalam fitur dan kegunaan, tetapi ini masih merupakan editor gambar yang berguna, dan Anda tidak dapat mengalahkan harganya!
www.gimp.org

TexturePacker

Sedangkan alat yang disebutkan sebelumnya semuanya digunakan di luar bidang developeran game, TexturePacker hanya berguna untuk developeran game. Tapi itu sangat bagus untuk tugas yang dirancang untuknya: merakit lembar sprite untuk digunakan dalam game 2D. Jika Anda mengembangkan game 2D, Anda mungkin ingin mencoba TexturePacker:
www.codeandweb.com/texturepacker

Software Audio

Serangkaian alat produksi audio yang memusingkan tersedia, termasuk editor suara (yang bekerja dengan bentuk gelombang mentah) dan sequencer (yang membuat musik menggunakan urutan nada). Untuk memberikan gambaran tentang software audio yang tersedia, bagian ini membahas dua alat pengeditan suara utama (contoh lain di luar daftar ini termasuk Logika, Ableton, dan Reason).

Pro Tools

Software audio ini menawarkan banyak fitur berguna dan dianggap sebagai standar industri oleh banyak produser musik dan teknisi audio. Ini sering digunakan untuk semua jenis pekerjaan audio profesional, termasuk developeran game:
www.avid.com/US/products/family/Pro-Tools

Audacity

Meskipun sama sekali tidak berguna untuk pekerjaan audio profesional, Audacity adalah editor suara yang berguna untuk pekerjaan audio scale kecil, seperti menyiapkan file suara pendek untuk digunakan sebagai sound effect dalam game. Ini adalah pilihan populer bagi mereka yang mencari software pengedit suara open source:
http://audacity.sourceforge.net/

Lampiran 2

Sumber belajar online

Buku ini dirancang untuk menjadi pengantar lengkap untuk developeran game di Unity, tetapi masih banyak lagi yang harus dipelajari di luar pengantar ini. Ada banyak sumber daya online yang bagus yang dapat Anda gunakan untuk melangkah lebih jauh setelah menyelesaikan buku ini.

Tutorial tambahan

Ada banyak situs yang menyediakan informasi terarah tentang berbagai topik dalam Unity. Beberapa di antaranya bahkan disediakan secara resmi oleh perusahaan di belakang Unity.

Manual Unity

Ini adalah panduan pengguna komprehensif yang disediakan oleh Unity. Manual ini tidak hanya berguna untuk mencari informasi, tetapi daftar topik itu sendiri berguna untuk memberi pengguna gambaran lengkap tentang kemampuan Unity:
http://docs.unity3d.com/Documentation/Manual/index.html

Referensi skrip

Pemrogram Unity akhirnya membaca sumber ini lebih dari yang lain (setidaknya, saya tahu!). Manual pengguna mencakup kemampuan mesin dan penggunaan editor, tetapi referensi skrip adalah referensi menyeluruh ke seluruh API Unity (antarmuka pemrograman aplikasi). Setiap perintah Unity tercantum di sini:
http://docs.unity3d.com/Documentation/ScriptReference/index.html

Unity3D Student

Situs ini menyediakan perpustakaan besar tutorial yang mencakup berbagai topik. Yang terpenting, tutorialnya semua video. Ini mungkin baik atau buruk tergantung pada perspektif Anda; jika Anda adalah seseorang yang suka menonton video tutorial, maka ini adalah situs yang bagus untuk dikunjungi:
www.unity3dstudent.com

Pelajari Unity3D

Bagian dari keluarga situs web yang sama dengan Unity 3D Student, situs Learn Unity 3D memiliki tujuan yang serupa tetapi memberikan informasi yang sedikit berbeda dalam format yang sangat berbeda (lebih merupakan situs berita dengan artikel yang menarik bagi pelajar). Ini adalah situs bagus lainnya untuk menelusuri tutorial: http://learnunity3d.com/

Developeran game di StackExchange

Ini adalah situs informasi hebat lainnya dengan format berbeda dari yang sebelumnya terdaftar. Daripada serangkaian tutorial mandiri, StackExchange menyajikan sebagian besar QA teks yang mendorong pencarian. StackExchange memiliki bagian tentang beragam topik; ini adalah area situs yang berfokus pada developeran game. Untuk apa nilainya, saya mencari informasi Unity di sini hampir sesering saya menggunakan referensi skrip:
http://gamedev.stackexchange.com/

Panduan Maya LT

Seperti yang dijelaskan sebelumnya dalam lampiran B, aplikasi seni eksternal adalah bagian penting dalam menciptakan game yang memukau secara visual. Banyak sumber tersedia yang mengajarkan tentang Maya, 3ds Max, Blender, atau aplikasi seni 3D lainnya di luar sana. Lampiran C menawarkan tutorial tentang Blender. Sementara itu, inilah satu panduan online tentang menggunakan Maya LT (yang merupakan versi Maya yang lebih murah dan berorientasi developeran game):
http://steamcommunity.com/sharedfiles/filedetails/?id=242847724

Code Library

Meskipun sumber yang terdaftar sebelumnya menyediakan tutorial dan/atau informasi pembelajaran tentang Unity, situs di bagian ini menyediakan kode yang dapat digunakan dalam proyek Anda. Pustaka dan plugin adalah jenis sumber daya lain yang dapat berguna bagi developer baru, baik untuk digunakan secara langsung tetapi juga untuk belajar dari (dengan membaca kode mereka).

Unity Komunitas Wiki

Wiki ini adalah basis data pusat dari kontribusi kode dari banyak developer, dan skrip yang dihosting di sini mencakup berbagai fungsi. Di sepanjang buku ini, saya terkadang mengarahkan Anda ke skrip tertentu yang dihosting di sini (sistem event dan parser JSON, misalnya). Tentu saja ada banyak skrip yang lebih berguna yang dapat Anda temukan di sini:
http://wiki.unity3d.com/index.php/Scripts

Pola kesatuan

Pustaka skrip di sini tidak seluas di wiki Unify, tetapi ada beberapa kode yang berguna untuk dilihat, bersama dengan beberapa tutorial yang mencerahkan: http://unitypatterns.com/

iTween

Seperti disebutkan secara singkat dalam bab 3 dan 8, semacam efek gerakan yang biasa digunakan dalam permainan disebut sebagai tween. Ini adalah jenis gerakan di mana satu perintah kode dapat mengatur objek bergerak ke target selama jangka waktu tertentu. Fungsionalitas tweening dapat ditambahkan ke Unity melalui sejumlah pustaka, dan inilah satu opsi yang bagus: http://itween.pixelplacement.com/index.php

prima[31]

Unity menyediakan penerapan ke platform seluler seperti iOS dan Android, tetapi fitur khusus platform yang sebenarnya terbatas pada fitur inti. Anda dapat menambahkan banyak fitur yang lebih spesifik melalui plugin, dan prime[31] memiliki banyak plugin seperti itu: https://prime31.com/

Layanan Play Game dari Google

Di iOS, Unity memiliki integrasi GameCenter bawaan sehingga game Anda dapat memiliki papan peringkat dan pencapaian asli platform. Sistem yang setara di Android disebut Google Play Games; meskipun ini tidak dibangun ke dalam Unity, Google mengelola plug-in: https://github.com/playgameservices/playgames-plugin-for-unity

Studio FMOD

Fungsionalitas audio yang dibangun ke dalam Unity berfungsi dengan baik hanya untuk memutar ulang rekaman tetapi dapat dibatasi untuk pekerjaan desain suara tingkat lanjut. FMOD Studio adalah alat desain suara canggih yang memiliki plug-in Unity gratis untuk digunakan (tetapi tidak harus dipublikasikan). Gulir ke bawah untuk menemukannya di halaman Unduhan mereka: www.fmod.org/download/

ProBuilder dan Prototipe

ProBuilder dan Prototype adalah add-on yang memungkinkan pengeditan level yang kuat dalam Unity. ProBuilder membutuhkan biaya untuk fitur profesional seperti tekstur fleksibel,

tetapi Prototipe gratis dan ideal untuk digunakan dalam alur kerja tinju putih dari bab 4: www.protoolsforunity3d.com/prototype/

Kontrol FPS

Kontrol FPS adalah seperangkat alat dan kode yang dirancang untuk memudahkan pembuatan game FPS (first-person shooter). Kerangka kerja ini berkembang dari serangkaian tutorial video populer: www.fpscontrol.com/features

DAFTAR PUSTAKA

Agus, Ida ,Mahendra, “Optimasi Lintasan Game Mekepung 3d Pada Engine Unity3d”, Jurnal Ilmu Komputer Udayana, vol 8 no 2 September 2015

Asmiatun, Siti. Astrid Novita Putri. 2017. Belajar membuat Game 2D dan 3D menggunakan unity. Cv Budi utama. Yogyakarta.

Beck, J.C., Wade, M. (2007). Gamer Juga Bisa Sukses. Jakarta: PT Grasindo.

Blackman, S. (2011). Beginning 3D Game Development with Unity: All-in-one, multi-platform game development. New York: Apress.

Ekasari, Yeti, “Merancang Game Petualangan “Binggo” menggunakan Unity 3D Game Engine”, Yogyakarta ,Sekolah Tinggi Manajemen Informatika dan Komputer AMIKOM, 2012

Finnegan, T. (2013). Unity Android Game Development by Example Beginner's Guide. Birmingham UK: Packt.

Karamian, Vahe. 2016. Introduction to Game Programming: Using C# dan Unity 3D. Los Angeles: Noorcon Inc.

Khaerudin, M., Warta, J., Srisulistiowati, D. B. (2020). Manajemen Pengetahuan Sala h Sebagai Satu Jalan Dalam Pengembangan Lembaga Pendidikan Unggul Pada Tk Bina Mulia Cibitung. Journal of Informatics and Information Security, 159-172. doi:https://doi.org/10.31599/jiforty.v1i2.390

Koriaty, Sri., Agustani, M.D. (2016). PENGEMBANGAN MODEL PEMBELAJARAN GAME EDUKASI UNTUK MENINGKATKAN MINAT SISWA KELAS X TKJ SMK NEGERI 7 PONTIANAK. Jurnal Edukasi, 14 (2), 277-288. doi:http://dx.doi.org/10.31571/edukasi.v14i2.360

Miftah, Suyatno, Maharani, “Penerapan Metode Finite State Machine Pada Game The Relationship”, Jurnal Informatika Mulawarman, Vol 11, no 1 Feb 2016

Moore, D. (2006). E-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning. Educational Technology Research and Development, 54, 197-200. doi:https://doi.org/10.1007/s11423-006-8254-8

Murray, Jeff. 2014. C# Game Programming Cookbook forUnity 3D. Florida: CRC Press

Patah, Trisna, “Rancang Bangun Game 3d “Ena Burena” Dengan Algoritma A* Dan Collision Detection Menggunakan Unity 3d Berbasis Desktop Dan Android”, Jurnal Informasi Vol VIII no 1/ Feb/2016

Putra, Febriyanto Pratama, Husni Tamrin dan Dedy Ary Prasetya, "Pembuatan Game Animasi 3D Role Playing Game Untuk Pendidikan Budaya dengan Unity dan Bahasa Pemograman C#", Surakarta, Universitas Muhammadiyah, 2012.

Rahman, S., Munawar, W., Berman, E. T. (2014). PEMANFAATAN MEDIA PEMBELAJARAN BERBASIS WEBSITE PADA PROSES PEMBELAJARAN PRODUKTIF DI SMK. Journal of Mechanical Engineering Education, 1 (1), 137-145. doi:https://doi.org/10.17509/jmee.v1i1.3746

Refi, Meisadri. et al. Pembangunan Game First Person Shooter 3D Alien Hunter. Jurnal Ilmiah Komputer dan Informatika (KOMPUTA), Bandung, vol. 2, No.1, ISSN:2089-9033, Maret 2013.

Rizali, M., Warhat, Z., Zebua, E. (2019). PENGARUH ELEMEN-ELEMEN DESAIN KOMUNIKASI VISUAL (DKV) BOX ART. Gorga Jurnal Seni Rupa, 8 (2), 295-302. doi:https://doi.org/10.24114/gr.v8i2.14700

Roedavan, Rickman, "UNITY Tutorial Game Engine", Bandung, Informatika, 2014.

Rosa, A. S. (2014). Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek. Bandung: Informatika.

Ryan Mahendra Kusuma Putra, "Perancangan Game First Person Shooter 3D 'Zombie Hunter' dengan Menggunakan Metode A* ", JIntech, vol 3 no 1 2015

Tafonao, T. (2018). PERANAN MEDIA PEMBELAJARAN DALAM MENINGKATKAN MINAT BELAJAR MAHASISWA. Jurnal Komunikasi Pendidikan, 2 (2), 103-114. doi:https://doi.org/10.32585/jkp.v2i2.113

Unity, "Unity Answer", http://answer.unity3d.com. Tanggal akses 3 Februari 2016.

Unity, "Unity Manual", http://docs.unity3d.com/Manual/index.html. Tanggal akses 5 Februari 2016.

Widyastuti, Reni., Puspita, L. S. (2020). Pengembangan Media Pembelajaran Berbasis Game Edukasi Pada MatPel IPA Tematik Kebersihan Lingkungan. Jurnal Paradigma, 22 (1), 95-100. doi:https://doi.org/10.31294/p.v22i1.7084

MEMBUAT VIDEO GAME dengan 3D Unity

Dr. Mars Caroline Wibowo, ST, M.Mm.Tech.

Bio Data Penulis

Penulis lahir di Semarang pada tanggal 1 Maret 1983. Penulis menempuh pendidikan Sarjana Teknik Elektro di Universitas Kristen Satya Wacana (UKSW), lulus tahun 2004, kemudian tahun 2005 melanjutkan studi pada Magister Desain pada Fakultas Seni Rupa dan Desain, Institut Teknologi Bandung (ITB), dan melanjutkan studi pada program studi Teknologi Multimedia pada Swinburne University of Technology Australia.

Penulis sejak tahun 2010, menjadi dosen pada program studi Desain Grafis Universitas Sains dan Teknologi Komputer (Universitas STEKOM), memiliki jabatan fungsional Lektor 300 dan sedang proses mengajukan kenaikan jabatan fungsional menjadi Lektor Kepala. Penulis juga seorang wirausaha di bidang toko online yang berhasil di kota Semarang dan juga aktif sebagai freelancer dalam bidang fotografi, web design dan multimedia.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-5734-60-6 (PDF)
9 786235 734606

MEMBUAT VIDEO GAME dengan 3D Unity

Dr. Mars Caroline Wibowo, ST, M.Mm.Tech.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id