Mike Geig

Full Color

Sams Teach Yourself

Unity® Game Development in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Mike Geig

Sams Teach Yourself

Unity® Game Development in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Unity® Game Development in 24 Hours

Copyright © 2014 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Unity is a trademark of Unity technologies.

Kinect is a trademark of Microsoft®.

PlayStation and PlayStation Move are trademarks of Sony®.

Wii is a trademark of Nintendo®.

ISBN-13: 978-0-672-33696-6
ISBN-10: 0-672-33696-6

Library of Congress Control Number: 2013950040

Printed in the United States of America

First Printing November 2013

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearsoned.com

Editor-in-Chief
Mark Taub

Executive Editor
Laura Lewin

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Keith Cline

Indexer
Brad Herriman

Proofreader
Sheri Cain

Technical Editors
Tim Harrington
Valerie Shipbaugh
Jeff Somers

Publishing Coordinator
Olivia Basegio

Interior Designer
Gary Adair

Cover Designer
Mark Shirar

Compositor
Gloria Schurick

Contents at a Glance

Preface ... xi

HOUR 1 Introduction to Unity ... 1

HOUR 2 Game Objects ... 23

HOUR 3 Models, Materials, and Textures ... 37

HOUR 4 Terrain ... 51

HOUR 5 Environments ... 65

HOUR 6 Lights and Cameras ... 81

HOUR 7 Game 1: *Amazing Racer* ... 101

HOUR 8 Scripting Part 1 ... 117

HOUR 9 Scripting Part 2 ... 137

HOUR 10 Collision ... 155

HOUR 11 Game 2: *Chaos Ball* ... 167

HOUR 12 Prefabs ... 185

HOUR 13 Graphical User Interfaces ... 197

HOUR 14 Character Controllers ... 213

HOUR 15 Game 3: *Captain Blaster* ... 227

HOUR 16 Particle Systems ... 245

HOUR 17 Animations ... 261

HOUR 18 Animators ... 277

HOUR 19 Game 4: *Gauntlet Runner* ... 297

HOUR 20 Audio ... 317

HOUR 21 Mobile Development ... 329

HOUR 22 Game Revisions ... 341

HOUR 23 Polish and Deploy ... 353

HOUR 24 Wrap Up ... 367

Index ... 373

Table of Contents

Preface xi

HOUR 1: Introduction to Unity 1

Installing Unity 1
Getting to Know the Unity Editor 5
Navigating the Unity Scene View 18
Summary 20
Q&A 20
Workshop 21
Exercise 21

HOUR 2: Game Objects 23

Dimensions and Coordinate Systems 23
Game Objects 27
Transforms 28
Summary 34
Q&A 34
Workshop 35
Exercise 35

HOUR 3: Models, Materials, and Textures 37

The Basics of Models 37
Textures, Shaders, and Materials 42
Summary 48
Q&A 48
Workshop 49
Exercise 49

HOUR 4: Terrain 51

Terrain Generation 51
Terrain Textures 59
Summary 62

Q&A 62
Workshop 63
Exercise 63

HOUR 5: Environments 65
Generating Trees and Grass 65
Environment Effects 72
Character Controllers 77
Summary 79
Q&A 79
Workshop 79
Exercise 80

HOUR 6: Lights and Cameras 81
Lights 81
Cameras 90
Layers 94
Summary 98
Q&A 98
Workshop 98
Exercise 99

HOUR 7: Game 1: *Amazing Racer* 101
Design 101
Creating the Game World 104
Gamification 106
Playtesting 113
Summary 114
Q&A 114
Workshop 114
Exercise 115

HOUR 8: Scripting Part 1 117
Scripts 118
Variables 125
Operators 127
Conditionals 130

Iteration 133
Summary 135
Q&A 135
Workshop 136
Exercise 136

HOUR 9: Scripting Part 2 **137**
Methods 137
Input 142
Accessing Local Components 147
Accessing Other Objects 148
Summary 152
Q&A 152
Workshop 152
Exercise 153

HOUR 10: Collision **155**
Rigidbodies 155
Collision 157
Triggers 161
Raycasting 163
Summary 165
Q&A 165
Workshop 165
Exercise 166

HOUR 11: Game 2: *Chaos Ball* **167**
Design 167
The Arena 169
Game Entities 173
The Control Objects 178
Improving the Game 182
Summary 183
Q&A 183
Workshop 183
Exercise 184

HOUR 12: Prefabs 185
Prefab Basics 185
Working with Prefabs 188
Instantiating Prefabs Through Code 194
Summary 194
Q&A 195
Workshop 195
Exercise 195

HOUR 13: Graphical User Interfaces 197
GUI Basics 197
GUI Controls 199
Customization 205
Summary 211
Q&A 211
Workshop 211
Exercise 212

HOUR 14: Character Controllers 213
The Character Controller 213
Scripting for Character Controllers 216
Building a Controller 219
Summary 225
Q&A 225
Workshop 225
Exercise 226

HOUR 15: Game 3: *Captain Blaster* 227
Design 227
The World 229
Controls 234
Improvements 242
Summary 242
Q&A 242
Workshop 243
Exercise 244

HOUR 16: Particle Systems 245
Particle Systems 245
Particle System Modules 247
The Curve Editor 257
Summary 259
Q&A 259
Workshop 259
Exercise 259

HOUR 17: Animations 261
Animation Basics 262
Preparing a Model for Animation 263
Applying Animations 269
Scripting Animations 272
Summary 273
Q&A 273
Workshop 274
Exercise 274

HOUR 18: Animators 277
Animator Basics 278
Creating an Animator 287
Scripting Animators 294
Summary 296
Q&A 296
Workshop 296
Exercise 296

HOUR 19: Game 4: *Gauntlet Runner* 297
Design 297
The World 298
The Entities 300
The Controls 307
Room for Improvement 314
Summary 315

Q&A 315
Workshop 315
Exercise 316

HOUR 20: Audio 317
Audio Basics 317
Audio Sources 319
Audio Scripting 324
Summary 326
Q&A 326
Workshop 327
Exercise 327

HOUR 21: Mobile Development 329
Preparing for Mobile 329
Accelerometers 333
Summary 338
Q&A 338
Workshop 338
Exercise 339

HOUR 22: Game Revisions 341
Amazing Racer 341
Chaos Ball 345
Captain Blaster 346
Gauntlet Runner 349
Summary 350
Q&A 351
Workshop 351
Exercise 351

HOUR 23: Polish and Deploy 353
Managing Scenes 353
Persisting Data and Objects 356
Unity Player Settings 359
Building Your Game 362

Summary 365
Q&A 365
Workshop 365
Exercise 366

HOUR 24: Wrap Up 367
Accomplishments 367
Where to Go from Here 370
Resources Available to You 371
Summary 371
Q&A 371
Workshop 372
Exercise 372

Index 373

Preface

The Unity game engine is an incredibly powerful and popular choice for professional and amateur game developers alike. This book has been written to get readers up to speed and working in Unity as fast as possible (about 24 hours to be exact) while covering fundamental principles of game development. Unlike other books that only cover specific topics or spend the entire time teaching a single game, this book covers a large array of topics while still managing to contain four games! Talk about a bargain. By the time you are done reading this book, you won't have just theoretical knowledge of the Unity game engine. You will have a portfolio of games to go with it.

Who Should Read This Book

This book is for anyone looking to learn how to use the Unity game engine. Whether you are a student or a development expert, there is something to learn in these pages. It is not assumed that you have any prior game development knowledge or experience, so don't worry if this is your first foray into the art of making games. Take your time and have fun. You will be learning in no time.

How This Book Is Organized and What It Covers

Following the Sam's Teach Yourself approach, this book is organized into 24 chapters that should take approximately 1 hour each to work through. The chapters include the following:

- Hour 1, "Introduction to Unity": This hour gets you up and running with the various components of the Unity game engine.
- Hour 2, "Game Objects": Hour 2 teaches you how to use the fundamental building blocks of the Unity game engine: the game object. You also learn about coordinate systems and transformations.
- Hour 3, "Models, Materials, and Textures": In this hour, you learn to work with Unity's graphical asset pipeline as you apply shaders and textures to materials. You also learn how to apply those materials to a variety of 3D objects.
- Hour 4, "Terrain": In Hour 4, you learn to sculpt game worlds using Unity's terrain system. Don't be afraid to get your hands dirty as you dig around and create unique and stunning landscapes.

- Hour 5, "Environments": In this hour, you learn to apply environmental effects to your sculpted terrain. Time to plant some trees!
- Hour 6, "Lights and Cameras": Hour 6 covers lights and cameras in great detail.
- Hour 7, "Game 1: *Amazing Racer*": Time for your first game. In Hour 7, you create Amazing Racer, which requires you to take all the knowledge you have gained so far and apply it.
- Hour 8, "Scripting Part 1": In Hour 8, you begin your foray into scripting with Unity. If you've never programmed before, don't worry. We go slowly as you learn the basics.
- Hour 9, "Scripting Part 2": In this hour, you expand on what you learned in Hour 8. This time, you focus on more advanced topics.
- Hour 10, "Collision": Hour 10 walks you through the various collision interactions that are common in modern video games. You learn about physical as well as trigger collisions. You also learn to create physical materials to add some variety to your objects.
- Hour 11, "Game 2: *Chaos Ball*": Time for another game! In this hour, you create Chaos Ball. This title certainly lives up to its name as you implement various collisions, physical materials, and goals. Prepare to mix strategy with twitch reaction.
- Hour 12, "Prefabs": Prefabs are a great way to create repeatable game objects. In Hour 12, you learn to create and modify prefabs. You also learn to build them in scripts.
- Hour 13, "Graphical User Interfaces": In Hour 13, you learn to implement graphical user interfaces (GUIs) in Unity. You learn the various components and how to position them on a 2D interface.
- Hour 14, "Character Controllers": In this hour, you learn how to create your own character controllers. You finish up the chapter by building your own custom controller.
- Hour 15, "Game 3: *Captain Blaster*": Game number 3! In this hour, you make Captain Blaster, a retro-style spaceship shooting game.
- Hour 16, "Particle Systems": Time to learn about particle effects. In this chapter, you experiment with Unity's legacy particle system and its new Shuriken particle system. You learn how to create cool effects and apply them to your projects.
- Hour 17, "Animations": In Hour 17, you get to learn about animations and Unity's legacy animation system. You experiment with bringing models to life using assets from the Asset Store.
- Hour 18, "Animators": Hour 18 is all about Unity's new Mecanim animation system. You learn to remap model riggings and apply universal animations to them.
- Hour 19, "Game 4: *Gauntlet Runner*": Lucky game number 4 is called Gauntlet Runner. This game explores a new way to scroll backgrounds and how to implement animator controllers to build complex blended animations.

- Hour 20, “Audio”: Hour 20 has you adding important ambient effects via audio. You learn about 2D and 3D audio and their different properties.
- Hour 21, “Mobile Development”: In this hour, you learn how to build games for mobile devices. You also learn to utilize a mobile device’s built-in accelerometer and multi-touch display.
- Hour 22, “Game Revisions”: It’s time to go back and revisit the four games you have made. This time you modify them to work on a mobile device. You get to see which control schemes translate well to mobile and which don’t.
- Hour 23, “Polish and Deploy”: Time to learn how to add multiple scenes and persist data between scenes. You also learn about the deployment settings and playing your games.
- Hour 24, “Wrap Up”: Here, you look back and summarize the journey you went on to learn Unity. This hour provides useful information about what you have done and where to go next.

Unity Engine Versions

This book was made with the Unity engine version 4.1 and 4.2. The two different versions are nearly identical for your purposes, but do note that some visual elements might have shifted place. For example, in some of the screen images you may note a Terrain menu item in the menu bar at the top of the Unity editor. In version 4.2, that has been moved. Do not worry. All explanations involving the creation and management of terrain have been updated to illustrate the new process. I am just writing this here so that you are not confused if a couple of things look slightly different.

Thank you for reading my preface! I hope you enjoy this book and learn much from it. Good luck on your journey with the Unity game engine!

About the Author

Mike Geig is both an experienced teacher and game developer, with a foot firmly in both camps. He is currently teaches game design and development at Stark State College and the Cleveland Institute of Art. Mike also works as a screencaster for Unity Technologies and is a member of Unity's Learn department. His Pearson video, *Game Development Essentials with Unity 4 LiveLessons*, is a key title on Unity. Mike was once set on fire and has over a million "likes" on Facebook.

Dedication

To Dad: Everything worth learning, I learned from you.

Acknowledgments

A big "thank you" goes out to everyone who helped me write this book.

First and foremost, thank you Kara for keeping me on track. I don't know what we'll be talking about when this book comes out, but whatever it is, you are probably right. Love ya, babe.

Link and Luke: We should take it easy on mommy for a little while. I think she's about to crack.

Thanks to my parents. As I am now a parent myself, I recognize how hard it was for you not to strangle or stab me. Thanks for not strangling or stabbing me.

Thanks to Angelina Jolie. Due to your role in the spectacular movie Hackers (1995), I decided to learn how to use a computer. You underestimate the impact you had on 10-year-olds at the time. You're elite!

To the inventor of beef jerky: History may have forgotten your name, but definitely not your product. I love that stuff. Thanks!

Thank you to my technical editors: Valerie, Jim, and Tim. Your corrections and insights played a vital role in making this a better product.

Thank you, Laura, for convincing me to write this book. Also thank you for buying me lunch at GDC. I feel that lunch, the best of all three meals, specifically enabled me to finish this.

Finally, a "thank you" is in order for Unity Technologies. If you never made the Unity game engine, this book would be very weird and confusing.

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

HOUR 3
Models, Materials, and Textures

What You'll Learn in This Hour:

- ▶ The fundamentals of models
- ▶ How to import custom and premade models
- ▶ How to work with materials and shaders

In this hour, you learn all about models and how they are used in Unity. You start by looking at the fundamental principles of meshes and 3D objects. From there, you learn how to import your own models or use ones acquired from the Asset Store. You finish this hour by examining Unity's material and shader functionality.

The Basics of Models

Video games wouldn't be very *video* without the graphical components. In 2D games, the graphics consist of flat images called *sprites*. All you needed to do was change the x and y positions of these sprites and flip several of them in sequence and the viewer's eye was fooled into believing that it saw true motion and animation. In 3D games, however, things aren't so simple. In worlds with a third axis, objects need to have volume to fool the eye. Because games use a large number of objects, the need to process things quickly was very important. Enter the mesh. A mesh, at its most simple, is a series of interconnected triangles. These triangles build off of each other in strips to form basic to very complex objects. These strips provide the 3D definitions of a model and can be processed very quickly. Don't worry, though; Unity handles all of this for you so that you don't have to manage it yourself. Later in this hour, you'll see just how triangles can make up various shapes in the Unity Scene view.

NOTE

Why Triangles?

You might be asking yourself why 3D objects are made up entirely of triangles. The answer is simple. Computers process graphics as a series of point, otherwise known as *vertices*. The fewer vertices an object has, the faster it can be drawn. Triangles have two properties that make them desirable. The first is that whenever you have a single triangle, you need only one more vertex to make another. To make one triangle, you need three vertices, two triangles take only four, and three triangles require only five. This makes them very efficient. The second is that by using this practice of making strips of triangles, you can model any 3D object. No other shape affords you that level of flexibility and performance.

NOTE

Model or Mesh?

The terms *model* and *mesh* are similar, and you can often use them interchangeably. There is a difference, however. A mesh contains all the vertex information that defines the 3D shape of an object. When you refer to the shape or form of a model, you are really referring to a mesh. A model, therefore, is an object that contains a mesh. A model has a mesh to define its dimensions, but it can also contain animations, textures, materials, shaders, and other meshes. A good general rule is this: If the item in question contains anything other than vertex information, it is a model; otherwise, it is a mesh.

Built-In 3D Objects

Unity comes with a few basic built-in meshes (or primitives) for you work with. These tend to be simple shapes that serve simple utilities or can be combined to make more-complex objects. Figure 3.1 shows the available built-in meshes. (You worked with the cube and sphere in the previous hours.)

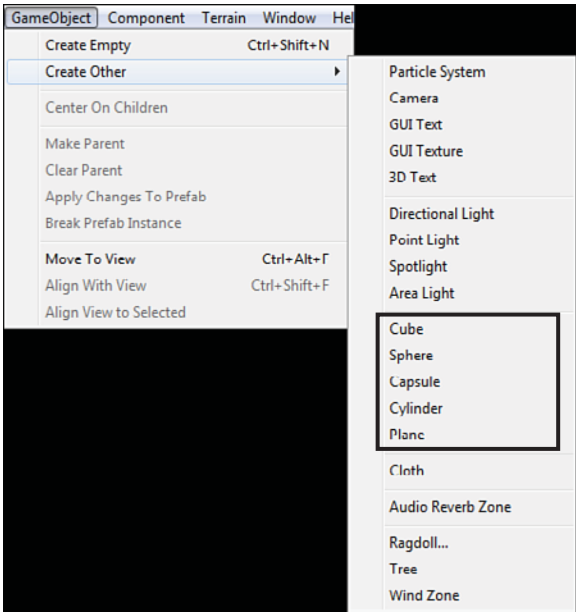


FIGURE 3.1
The built-in meshes in Unity.

TIP

Modeling with Simple Meshes

Do you need a complex object in your game but you can't find the right type of model to use? Nesting objects in Unity enables you to easily make simple models using the built-in meshes. Just place the meshes near each other so that they form the rough look you want. Then nest all the objects under one central object. This way, when you move the parent, all the children move, too. This might not be the prettiest way to make models for your game, but it will do in a pinch!

Importing Models

Having built-in models is nice, but most of the time, your games will require art assets that are a little more complex. Thankfully, Unity makes it rather easy to bring your own 3D models into your projects. Just placing the file containing the 3D model in your Assets folder is enough to bring it into the project. From there, dragging it into the scene or hierarchy builds a game object around it. Natively, Unity supports .fbx, .dae, .3ds, .dxf, and .obj files. This enables you to work with just about any 3D modeling tool.

▼ TRY IT YOURSELF

Importing Your Own 3D Model

Let's walk through the steps required to bring custom 3D models into a Unity project:

1. Create a new Unity project or scene.
2. In the Project view, create a new folder named **Models** under the Assets folder. (Right-click the Assets folder and select **Create > Folder**.)
3. Locate the Torus.fbx file provided for you in the Hour 3 folder of the book files.
4. With both the operating system's file browser and the Unity editor open and side by side, click and drag the Torus.fbx file from the file browser into the Models folder that you created in step 2. In Unity, click the **Models** folder to see the new Torus file. If done correctly, your Project view will resemble Figure 3.2. Notice the Materials folder that was added for you. You will learn more about this later.

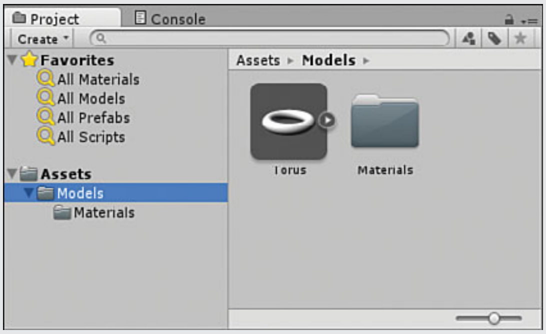


FIGURE 3.2
The Project view after the Torus model was added.

5. Click the **Torus** asset in the Models folder and look at the Inspector view. Change the value of the scale factor from 0.01 to 1 and click **Apply**.
6. Drag the Torus asset from the Models folder onto the Scene view. Notice how a Torus game object was added to the scene containing a mesh filter and mesh rendered. These allow the Torus to be drawn to the screen. If you click the **Torus** object, you see how it is made up of many connected triangles.

CAUTION

Default Scaling of Meshes

Most of the Inspector view options for meshes are advanced and are not covered right now. The property you are interested in is the scale factor. By default, Unity imports meshes scaled down. By changing the value of the scale factor from 0.01 to 1, you are telling Unity to allow the model to enter the scene as the same size as it was created.

Models and the Asset Store

You don't have to be an expert modeler to make games with Unity. The Asset Store provides a simple and effective way to find premade models and import them into your project. Generally speaking, models on the Asset Store are either free or paid and come alone or in a collection of similar models. Some of the models come with their own textures, and some of them are simply the mesh data.

TRY IT YOURSELF ▼

Downloading Models from the Asset Store

Let's learn how to find and download models from Unity's Asset Store. We will be acquiring a model named Robot Kyle and importing it into our scene:

1. Create a new scene (click **File > New Scene**). In the Project view, type **t:Model** into the search bar (see Figure 3.3).
2. In the search filter section, click the **Asset Store: 999+/999+** button (see Figure 3.3). If these words aren't visible, you may need to resize your editor window or Project view window.
3. Locate **Robot Kyle** and select it.

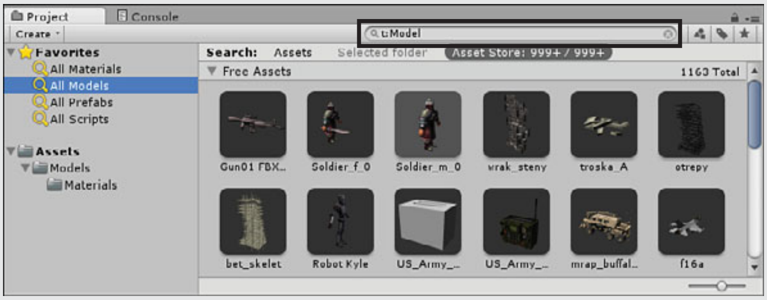


FIGURE 3.3
Steps to locate a model asset.

4. In the Inspector view, click **Import Package**. At this point, you may be prompted to provide your Unity account credentials.
5. When the Importing Package dialog opens, leave everything checked and select **Import**.
6. There will now be a new asset folder called Robot Kyle. Locate the robot model under **Assets > Robot Kyle > Model** and drag it into the Scene view (see Figure 3.4). Note that the model will be fairly small in the Scene view; you might need to move closer to see it.

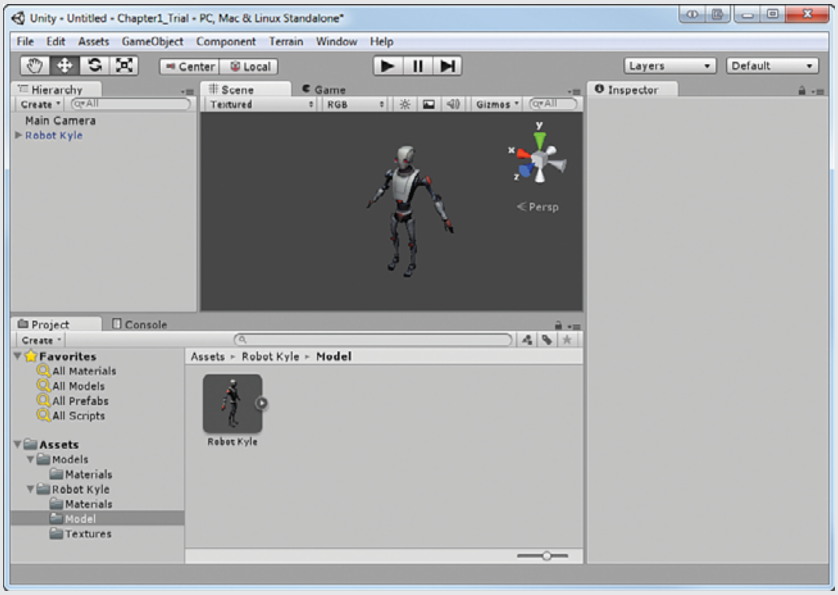


FIGURE 3.4
The Unity project with Robot Kyle Added.

Textures, Shaders, and Materials

Applying graphical assets to 3D models can be daunting if you are not familiar with it. Unity uses a simple and specific workflow that gives you a lot of power when determining exactly how you want things to look. Graphical assets are broken down into textures, shaders, and materials. Each of these is covered individually in its own section, but Figure 3.5 shows you how they fit together. Notice that textures are not applied directly to models. Instead, textures and shaders are applied to materials. Those materials are in turn applied to the models. This way, the look of a model can be swapped or modified quickly and cleanly without a lot of work.

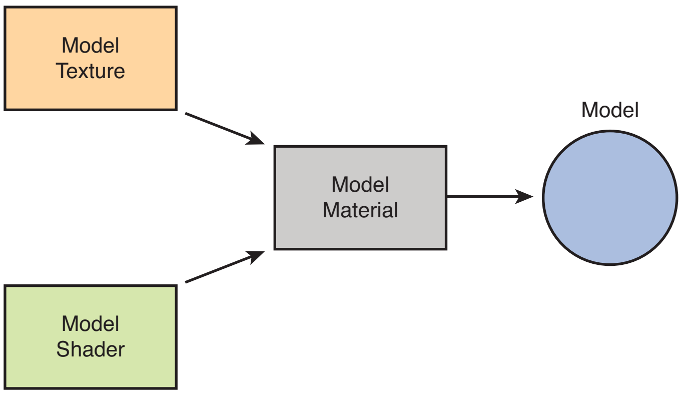


FIGURE 3.5
The model asset workflow.

Textures

Textures are flat images that get applied to 3D objects. They are responsible for models being colorful and interesting instead of blank and boring. It can be strange to think that a 2D image can be applied to a 3D model, but it is a fairly straightforward process once you are familiar with it. Think about a soup can for a moment. If you were to take the label off of the can, you would see that it is a flat piece of paper. That label is like a texture. After the label was printed, it was then wrapped around the 3D can to provide a more pleasing look.

Just like all other assets, adding textures to a Unity project is easy. Start by creating a folder for your textures; a good name would be Textures. Then drag any textures you want in your project into the Textures folder you just created. That's it!

NOTE

That's an Unwrap!

Imagining how textures wrap around cans is fine, but what about more complex objects? When creating an intricate model, it is common to generate something called an *unwrap*. The unwrap is somewhat akin to a map that shows you exactly how a flat texture will wrap back around a model. If you look in the Robot Kyle > Textures folder from earlier this hour, you notice the Robot_Color texture. It looks strange, but that is the unwrapped texture for the model. The generation of unwraps, models, and textures is an art form to itself and is not covered in this text. A preliminary knowledge of how it works should suffice at this level.

CAUTION

Weird Textures

Later in this hour, you will apply some textures to models. You might notice that the textures warp a bit or get flipped in the wrong direction. Just know that this is not a mistake or an error. This problem occurs when you take a basic rectangular 2D texture and apply it to a model. The model has no idea which way is correct, so it applies the texture however it can. If you want to avoid this issue, use textures specifically designed for (unwrapped for) the model that you are using.

Shaders

If the texture of a model determines what is drawn on its surface, the shader is what determines *how* it is drawn. Here's another way to look at this: A material contains properties and textures, and shaders dictate what properties and textures a material can have. This might seem nonsensical right now, but later when we create materials you will begin to understand how they work. Much of the information about shaders is covered later this hour, because you cannot create a shader without a material. In fact, much of the information to be learned about materials is actually about the material's shader.

TIP

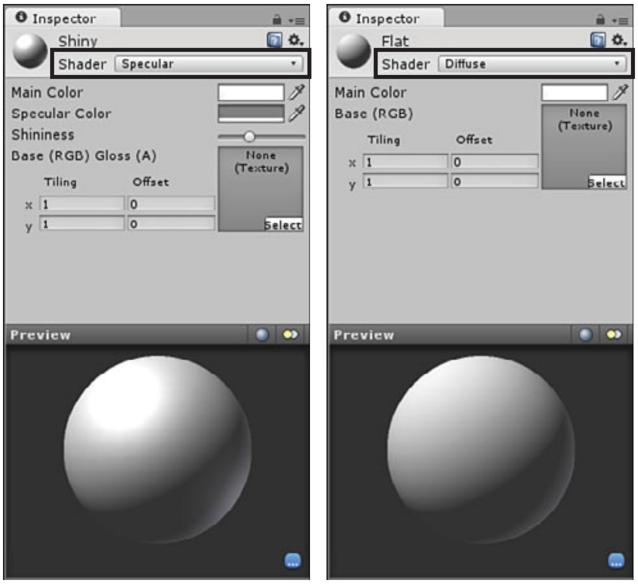
Thought Exercise

If you are having trouble understanding how a shader works, consider this scenario: Imagine you have a piece of wood. The physicality of the wood is its mesh; the color, texture, and visible element are its texture. Now take that piece of wood and pour water on it. The wood still has the same mesh. It still is made of the same substance (wood). It looks different, though. It is slightly darker and shiny. The water in this example is the shader. The shader took something and made it look a little different without actually changing it.

Materials

As mentioned earlier, materials are not much more than containers for shaders and textures that can be applied to models. Most of the customization of materials depends on which shader is chosen for it, although all shaders have some common functionality.

To create a new material, start by making a Materials folder. Then right-click the folder and select **Create** > **Material**. Give your material some descriptive name and you are done. Figure 3.6 shows two materials with different shaders selected. Notice how they each have a base texture, main color, tilling and offsets, and a preview of the material (blank now because there is no texture). The Shiny material, however, uses a specular shader and comes with properties for specular color and shininess. All these properties are covered later in this hour.



Shiny
Shader: Specular
Main Color
Specular Color
Shininess
Base (RGB) Gloss (A)

	Tiling	Offset
x	1	0
y	1	0

Flat
Shader: Diffuse
Main Color
Base (RGB)

	Tiling	Offset
x	1	0
y	1	0

FIGURE 3.6
Two materials with different shaders.

Shaders Revisited

Now that you understand textures, models, and shaders, it is time to look at how it all comes together. Unity has a lot of built-in shaders, but this book is concerned with only a few of the Normal family of shaders. These shaders are the most basic and should be useful for everyone. Table 3.1 lists some of the basic shaders and describes them.

TABLE 3.1 Basic Normal Family of Shaders

Shader	Description
Diffuse	Diffuse is the default shader for materials and is also the most basic. Light is evenly distributed across the diffuse object's surface.
Specular	Specular textures make an object look shiny. If you want to make an object seem to reflect a lot of light, this is the shader to use.
Bumped	Bumped shaders are generally used in conjunction with other shaders (as in bumped-diffuse or bumped-specular). These shaders use a normal map to give the flat texture a 3D, or bumpy, look. These are a great way to give your models a lot of physical detail without requiring complex modeling.

Now that you are familiar with a few of the built-in shaders, it is time to look at some of the common shader properties that you will be working with. Table 3.2 describes the common shader properties.

TABLE 3.2 Common Shader Properties

Property	Description
Main Color	The Main Color property defines what color of ambient light shines on the object. This does not change the color of the object itself; it just makes the object appear different. For example, an object with a blue texture and a yellow main color will not turn yellow but green (because blue with yellow light looks green). If you want your model's color to remain unchanged, select white.
Specular Color	The Specular Color property determines what color the "shiny" parts of a specular model are. Generally speaking, this will be white unless you intend for it to appear as if another color of light is shining on the object.
Shininess	The Shininess property determines how shiny a specular object is.
Texture	The Texture block contains the texture you want to apply to your model.
Normal Map	The Normal Map block contains the normal map that will be applied to your model. A normal map can be used to apply bumpiness to a model. This is useful when calculating lighting to give the model more detail than it would otherwise have.
Tiling	The Tiling property defines how often a texture can repeat on a model. It can repeat in both the x and y axes.
Offset	The Offset property defines whether a gap will exist between edges of the object and the texture.

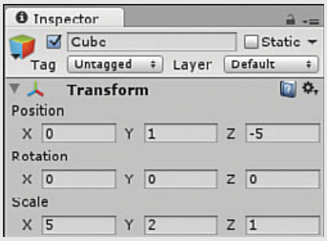
This might seem like a lot of information to take in, but once you become more familiar with the few basics of textures, shaders, and materials, you'll find this a smooth process.

TRY IT YOURSELF ▼

Applying Textures, Shaders, and Materials to Models

Let’s put all of our knowledge of textures, shaders, and materials together and create a decent-looking brick wall:

1. Start a new project or scene. Note that creating a new project will close and reopen the editor.
2. Create a Textures and a Materials folder.
3. Locate the Brick_Texture.png file in the book files and drag it into the Textures folder created in step 2.
4. Add a cube to the scene. Position it at (0, 1, –5). Give it a scale of (5, 2, 1). See Figure 3.7 for the cube properties.



Cube
Tag: Untagged; Layer: Default

Position: X 0, Y 1, Z -5
Rotation: X 0, Y 0, Z 0
Scale: X 5, Y 2, Z 1

FIGURE 3.7
The properties of the cube.

5. Create a new material (right-click the Materials folder and select **Create > Material**) and name it **BrickWall**.
6. Leave the shader as Diffuse, and in the texture block click **Select**. Select **Brick_Texture** from the pop-up window.
7. Click and drag the brick wall material from the Project view onto the cube in the Scene view.
8. Notice how the texture is stretched across the wall a little too much. With the material selected, change the value of the x tiling to be **3**. Now the wall looks much better.
9. Add a directional light to your scene (click **GameObject > Create Other > Directional Light**). Position it at (0, 10, –10) and give it a rotation of (30, 0, 0). We will discuss lighting more in a later hour. This is just here to make your brick wall “pop.”
10. You now have a textured brick wall in your scene. Figure 3.8 contains the final product.

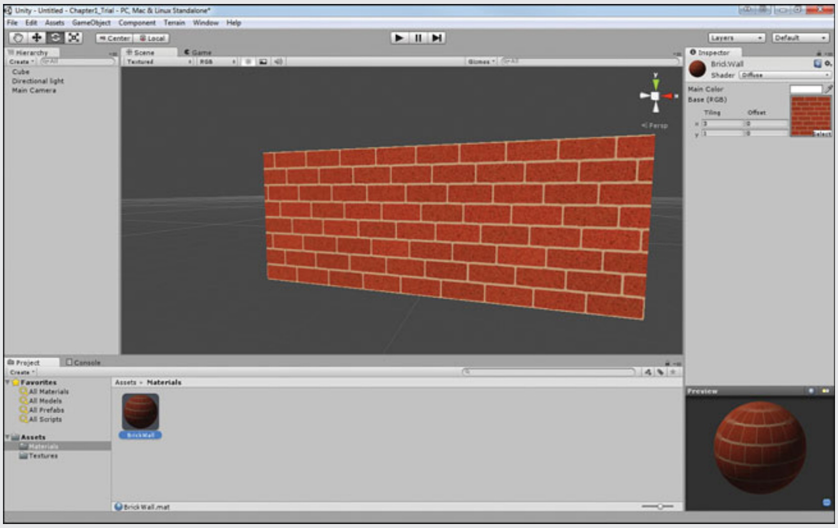


FIGURE 3.8
The final product of this Try It Yourself.

Summary

In this hour, you learned all about models in Unity. You started by learning about how models are built with collections of vertices called meshes. Then, you discovered how to use the built-in models, import your own models, and download models from the Asset Store. You then learned about the model art workflow in Unity. You experimented with textures, shaders, and materials. You finished by creating a textured brick wall.

Q&A

Q. Will I still be able to make games if I'm not an artist?

A. Absolutely. Using free online resources and the Unity Asset Store, you can find various art assets to put in your games.

Q. Will I need to know how to use all the built-in shaders?

A. Not necessarily. Many shaders are very situational. Start with the shaders covered in this chapter and learn more if a game project requires it.

Q. If there are paid art assets in the Unity Asset Store, does that mean I can sell my own art assets?

A. Yes, it does. In fact, it is not limited to only art assets. If you can create high-quality assets, you can certainly sell them in the store.

Workshop

Take some time to work through the questions here to ensure that you have a firm grasp of the material.

Quiz

1. True or False: Because of their simple nature, squares make up meshes in models.
2. What file formats does Unity support for 3D models?
3. True or False: Only paid models can be downloaded from the Unity Asset Store.
4. Explain the relationship between textures, shaders, and materials.

Answers

1. False, meshes are made up of triangles.
2. .fbx, .dae, .3ds, .dxf, and .obj files.
3. False. There are several free models.
4. Materials contain textures and shaders. Shaders dictate the properties that can be set by the material and how the material gets rendered.

Exercise

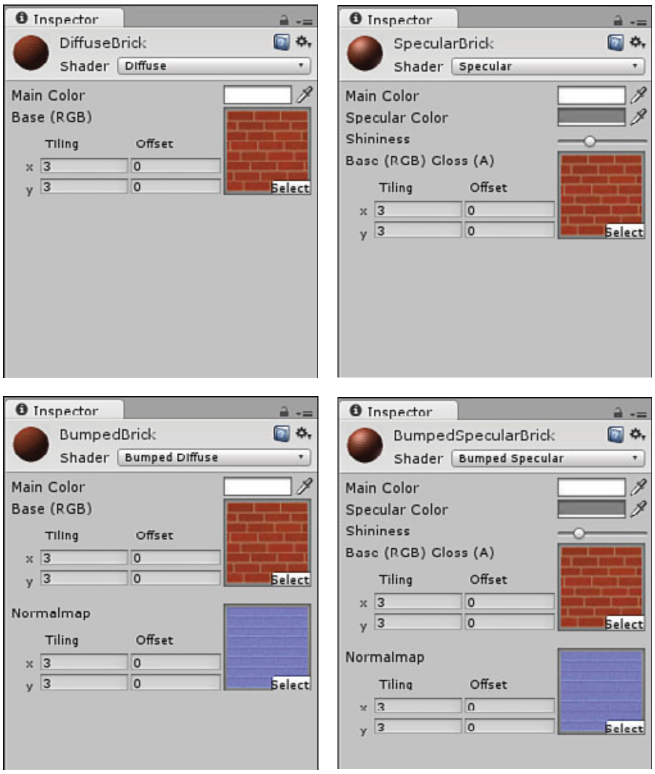
Let's experiment with the effects shaders have on the way models look. You will use the same mesh and texture for each model; only the shaders will be different. The project created in this exercise is named Hour3_Exercise and is available in the Hour 3 book files.

1. Create a new scene or project.
2. Add a Materials and a Textures folder to your project. Locate the files Brick_Normal.png and Brick_Texture.png in the Hour 3 book files and drag them into the Textures folder.
3. In the Project view, select **Brick_Texture**. In the Inspector view, change the aniso level from 1 to 3 to increase the texture quality for curves. Click **Apply**.
4. In the Project view, select **Brick_Normal**. In the Inspector view, change the texture type to **Normal Map**. Click **Apply**.

5. Add a directional light to your project (click **GameObject > Create Other > Directional Light**) and give it a position of (0, 10, –10) with a rotation of (30, 40, 0).

6. Add four spheres to your project. Scale them each to (2, 2, 2). Spread them out by giving them positions of (2, 0, –5), (–2, 0, –5), (–2, 2, –5), and (2, 2, –5).

7. Create four new materials in the Materials folder. Name them **DiffuseBrick**, **SpecularBrick**, **BumpedBrick**, and **BumpedSpecularBrick**. Figure 3.9 contains all the properties of the four materials. Go ahead and set their values.



Inspector
DiffuseBrick
Shader Diffuse
Main Color
Base (RGB)
Tiling Offset
x 3 0
y 3 0

Inspector
SpecularBrick
Shader Specular
Main Color
Specular Color
Shininess
Base (RGB) Gloss (A)
Tiling Offset
x 3 0
y 3 0

Inspector
BumpedBrick
Shader Bumped Diffuse
Main Color
Base (RGB)
Tiling Offset
x 3 0
y 3 0
Normalmap
Tiling Offset
x 3 0
y 3 0

Inspector
BumpedSpecularBrick
Shader Bumped Specular
Main Color
Specular Color
Shininess
Base (RGB) Gloss (A)
Tiling Offset
x 3 0
y 3 0
Normalmap
Tiling Offset
x 3 0
y 3 0

FIGURE 3.9
Material properties.

8. Click and drag each of the materials onto one of the four spheres. Notice how the light and the curvature of the spheres interact with the different shaders. Remember that you can move about the Scene view to see the spheres at different angles.

Index

Numbers

2D audio, 318-319
2D objects, 24
2D Sound Settings property (audio source), 320
3D audio, 318-319, 322
3D objects, 24
- built-in, 38-39

3D Sound Settings property (audio source), 320

A

accelerometers, 332-336
Add Grass Texture dialog, 69
Add Terrain Texture dialog, 60
Add Tree dialog, 66
Alt Negative Button/Alt Positive Button property (axis), 144
Amazing Racer, 101, 106-107, 114, 369
- adding scripts, 109-110
- creating world, 104-106
- design, 101-103
  - concept, 102
  - rules, 102-103
- game control objects, adding, 107-109
- players
  - jumping mechanics, 344
  - movement and horizontal looking, 341-343
- playtesting, 113
- requirements, 103
- revisions, 341-344
- scripts, connecting, 111-113

Angular Drag property (rigidbody), 156
Animate Physics property (Animation component), 269
Animation property (Animation component), 269
Animation property (Texture module), 256
animations, 261-263, 272
- adding, 270
- animating models, 270
- Animation component, 269-271
- applying, 269-271
- assets, 267-268
- idle, 282
- models, preparing for, 263-268
- preparing, 282-287
- rigs, 262-263
- scripting, 272-273
- walk, 284
- walk turn, 285-287
- wrap modes, 271

Animations property (Animation component), 269
Animator view, 289
animators, 277-278, 296
- animation preparation, 282-287

applying idle animation, 289-290
blend trees, 293-292
creating, 287-294
parameters, 290-291
rigging models, 278-281
scripting, 294-295
states, 290-292
transitions, 293-294
apps, Unity Remote, 331-333
area lights, 86
arenas, Chaos Ball, 169
bouncy material, 172-173
texturing, 170-171
arithmetic operators, 127-128
Aspect drop-down menu (Game view), 17
Asset Store, models, 41-42
assets, 9
animations, 267-268
importing, 78
terrain, 66
importing, 59
assignment operators, 128
attaching scripts, 121
audio, 317, 326
2D, 318-319, 323
3D, 318-319, 322
audio listener, 317
changing clips, 326
importing clips, 158-159
priorities, 319
scripting, 324-326
sources, 319-323
starting and stopping, 324-326
testing, 321
Scene view, 321
Audio Clip property, 320
audio listeners, 78, 317
Audition mode (Scene view), 14
axis
properties, 144
rotation, 31
Axis property (axis), 144

B

Background property (cameras), 91
backgrounds, Captain Blaster, 230-231
baking objects, 82
base terrain settings, 71
billboards, 68
blend trees, 291-293
blocks, methods, 139
bool variable, 125
Bounce Combine property (physics material), 161
Bounce property (Collision module), 254
Bounciness property (physics material), 161
bouncy material, Chaos Ball arena, 172-173
box controls, GUIs (graphical user interfaces), 200
breaking prefabs, 194
bugs, halos, 86
Build Settings dialog, 362-363
building games, 362-364
built-in 3D objects, 38-39
built-in methods, 124
built-in objects, 27
bullets, Captain Blaster, 233-234
script, 241-242
bumped shaders, 45
Bursts property (Emission module), 249
buttons
changing scenes via, 356
GUIs (graphical user interfaces), 201-202
Bypass Effects property (audio source), 320
Bypass Listener Effects property (audio source), 320
Bypass Reverb Zone property (audio source), 320

C

C# variable types, 125
calling methods, 141
cameras, 19, 81, 90-94, 98
adding skyboxes to, 72-73
Captain Blaster, 229
falling through the world, 78
layers, 92-97
lens flares, 74-75
multiple, 90
picture in picture, 92-94
properties, 91
split screens, 92-94
Captain Blaster, 227, 242, 369
background, 230-231
bullets, 233-234
script, 241-242
camera, 229
controls, 234-242
design, 227-228
improvements, 242
meteors, 233
script, 235-237
spawn, 237-238
players, 231-233
script, 238-240
revisions, 346-349
triggers, 234
script, 238
world, 229-234
Cascading Style Sheets (CSS), 206
Cast Shadows property (Renderer module), 257
Center property (character controller), 216
Center property (colliders), 158
center scripting variable, 215
centering GUI controls, 200
Chaos Ball, 183, 369
arena, 169
bouncy material, 172-173
texturing, 170-171
chaos balls, 174-176
colored balls, 176-177
control objects, 178-182

design, 167-168
concept, 168
requirements, 168
rules, 168
game controller, 180-182
improving, 182-183
players, 173-174
revisions, 345-346

char variable, 125

character controllers, 77-79, 214, 225
adding, 106, 214-215
building, 219-225
CollisionFlags variable, 218-219
controlling slide, 218
gravity, 222
handling collisions, 219
jumping, 222-223
properties, 215-216
pushing objects, 223-224
rigidbodies, 215
scripting for, 215-219
variables, 215

Clamp Forever property (wrap modes), 271

class declaration section, scripts, 123

classes, contents, 123-124

Clear Flags property (cameras), 91

Clipping Planes property (cameras), 91

code. *See also* scripts
comments, 123
scripting, 117
scripts, 107, 118
adding, 109-110
attaching, 121
basic, 122-124
character controllers, 215-219
class contents, 123-124
class declaration section, 123
conditionals, 130-133
connecting, 111-113
creating, 118-119
Game Control Script, 181
GoalScript.cs, 179
iteration, 133-135
methods, 137-141
operators, 127
player input, 142-146
using section, 123
variables, 125-126
VelocityScript.cs, 176

code listings
Default Script Code, 122
Demonstration of Class and Local Block Level, 126
Game Control Script, 181
GoalScript.cs, 179
VelocityScript.cs, 176

collaborative groups, 370

colliders, 157-159
complex, 159
conflicts, 214
Mesh, 159
mixing and matching, 159
physics materials, 160
properties, 158
trigger, 160-163

Collides With property (World mode), 253

collision, 155, 157, 164
colliders, 157-159
trigger, 160-163
handling, character controllers, 219
particles, 253
raycasting, 162-165
rigidbodies, 155-156

Collision Detection property (rigidbody), 156

Collision module (particle system), 253

Collision Quality property (World mode), 253

CollisionFlags scripting variable, 215, 218-219

Color by Speed module (particle system), 252

Color over Lifetime module (particle system), 251-252

Color property (Color by Speed module), 252

Color property (point lights), 83

comments, 123

concept
Amazing Racer, 102
Captain Blaster, 228
Chaos Ball, 168
Gauntlet Runner, 297

conditionals, 130-133

conflicts, colliders, 214

Console window (editor), 124

Constraints property (rigidbody), 156

control objects, Chaos Ball, 178-182

controllers
character, 77-79, 214, 225
adding, 214-215
building, 219-225
CollisionFlags variable, 218-219
controlling slide, 218
gravity, 222
handling collisions, 219
jumping, 222-223
properties, 215-216
pushing objects, 223-224
rigidbodies, 215
scripting for, 215-219
game, 180-182

controls
Captain Blaster, 234-242
Gauntlet Runner, 307-314
GUIs (graphical user interfaces), 198-205
box, 200
button, 201-202
centering, 200
label, 200
sliders, 205
textarea, 204

textfield, 204
toggle, 202
toolbars, 203
Cookie property (point lights), 83
cookies, 88
coordinate systems, 25
world versus local coordinates, 26-27
Create New Project dialog, 7, 169
cross-platform settings, players, 359
CSS (Cascading Style Sheets), 206
Culling Mask property (cameras), 91, 96
Culling Mask property (point lights), 83
Culling Type property (Animation component), 269
curve editor, 256-258
Cycles property (Texture module), 256

D

Dampen property (Collision module), 254
Dampen property (Limit Velocity over Lifetime module), 251
dark trees, 67
data
persisting, 355-359
saving, 357-359
dataPosition property (touch), 336
Dead property (axis), 144
default particle system, 249
Default property (wrap modes), 271
Default Script Code listing, 122
deltaTime property (touch), 336
Demonstration of Class and Local Block Level listing, 126
Depth property (cameras), 91
Descriptive Name/Descriptive Negative Name property (axis), 144
design
accelerometers, 334
Amazing Racer, 101-103
concept, 102
requirements, 103
rules, 102-103
Captain Blaster, 227-228
Chaos Ball, 167-168
concept, 168
requirements, 168
rules, 168
Gauntlet Runner, 297-298
GUIs (graphical user interfaces), 198
detail object settings, 71
detectCollisions scripting variable, 215
dialogs
Add Grass Texture, 69
Add Terrain Texture, 60
Add Tree, 66
Build Settings, 362-363
Create New Project, 7, 169
Game Settings, 363-364
Importing Packages, 59
Project, 5-7
diffuse shader, 45
dimensions, 23-24
coordinate systems, 25
world versus local coordinates, 26-27
directional lights, 82-86
cookies, 88
disappearing grass, 70
documentation, 371
Doppler Level property (3D audio), 322
double variable, 125
downloading
models, 42
Unity, 2-3
Drag property (rigidbody), 156
Draw Halo property (point lights), 83
Draw mode (Scene view), 14
Duration property (Particle System), 249
Dynamic Friction 2 property (physics material), 161
Dynamic Friction property (physics material), 161

E

editor, 5-17
Console window, 124
Game view, 15-17
Hierarchy view, 11-12
Inspector view, 12-13
Project view, 9-10
Scene view, 13-15
toolbars, 17
effects
environment, 72-76
fog, 74
lens flares, 74-75
skyboxes, 72-73
water, 75-76
particle, 247
picture-in-picture, 94
Emission module (particle system), 249
engine versions, Unity, xiii
environments, 65, 79. *See also* terrain and worlds
adding, 105-106
billboards, 68
character controllers, 77-79
effects, 72-76
fog, 74
lens flares, 74-75
skyboxes, 72-73
water, 75-76
grass
disappearing, 70
painting, 68-70
realistic, 69
mobile development, setting up, 330-331
terrain, settings, 70-71
trees
dark, 67
generating, 65-68
warping, 68
wind settings, 71
equality operators, 129-130
External Forces module (particle system), 253

F

factories, methods, 139
Field of View property (cameras), 91
fingerID property (touch), 336
first project, creating, 6
Flare property (point lights), 83
flares, lens, 74-75
float variable, 125
Flythrough mode (Scene view), 19-20
fog, 74
fonts, GUI controls, 209
for loop, 134-135
Force over Lifetime module (particle system), 251
formats, heightmaps, 54
forums, 371
Friction Combine property (physics material), 161
Friction Direction 2 property (physics material), 161

G

game controller
Captain Blaster, 234-235
Chaos Ball, 180-182
game control objects, adding, 107-109
game control script, Gauntlet Runner, 307-309
Game Control Script listing, 181
game controller, 180-182
game overlay (Scene view), 14
Game Settings dialog, 363-364
Game view, 15-17
games
- adding terrain to, 51-53
- Amazing Racer, 101, 106-107, 114, 369
  - adding game control objects, 107-109
  - adding scripts, 109-110
  - connecting scripts together, 111-113
  - creating world, 104-106
  - design, 101-103
  - playtesting, 113
  - revisions, 341-344
- attaching scripts, 121
- building, 362-364
- Captain Blaster, 227, 242, 369
  - controls, 234-242
  - design, 227-228
  - improvements, 242
  - players, 231-233
  - revisions, 346-349
  - world, 229-234
- Chaos Ball, 167, 183, 369
  - arena, 169-173
  - chaos balls, 174-176
  - colored balls, 176-177
  - control objects, 178-182
  - design, 167-168
  - game controller, 180-182
  - improving, 182-183
  - players, 173-174
  - revisions, 345-346
- creating, 370
  - first, 6
- Gauntlet Runner, 297, 315, 369
  - controls, 307-314
  - design, 297-298
  - entities, 300-307
  - improving, 314
  - revisions, 349-350
  - world, 298-300
- organization, 10
- writing about, 370

Gauntlet Runner, 297, 315, 369
- controls, 307-314
- design, 297-298
- entities, 300-307
- improving, 314
- revisions, 349-350
- world, 298-300

generating terrain, 51-58
Geometric properties (colliders), 158
gizmo (scene), 15
Gizmos button (Game view), 17
GoalScript.cs listing, 179
graphical user interfaces (GUIs). ***See* GUIs (graphical user interfaces)**
grass
- disappearing, 70
- painting, 68-70
- realistic, 69

gravity, character controllers, 222
Gravity Modifier property (Particle System), 249
Gravity property (axis), 144
GUIs (graphical user interfaces), 197-199, 211
- controls, 198-205
  - box, 200
  - button, 201-202
  - centering, 200
  - label, 200
  - sliders, 205
  - textarea, 204
  - textfield, 204
  - toggle, 202
  - toolbars, 203
- creating, 198-199
- skins, 207-209
- styles, 206-207

H

halos, 86-87
Hand tool, 18-19
HDR property (cameras), 91
Height property (character controller), 216
height scripting variable, 215
heightmaps
- formats, 54
- sculpting, 53-54

Hierarchy view, 11-12
- prefab instances, 187

I

idle animations, 282
applying, 289-290
if statement, 131
if/else if statement, 132-133
if/else statement, 131-132
importing
assets, 78
audio clips, 158-159
models, 39-40
terrain assets, 59
Importing Packages dialog, 59
Inherit Velocity property (Particle System), 249
inheritance, prefabs, 186, 192-193
input
key, 143
mouse, 146
multi-touch, mobile devices, 335-336
scripting, 142-146
Input Manager, 143-146
Inspector view, 12-13
script preview, 118
installing Unity, 1-5
instances, prefabs, 186
adding to scenes, 191
Instantiate() method, 194
instantiating prefabs through code, 194
int variable, 125
Intensity property (point lights), 83
interfaces, GUIs (graphical user interfaces), 197-199, 211
controls, 198-205
creating, 198-199
skins, 207-209
styles, 206-207
Interpolate property (rigidbody), 156
Invert property (axis), 144
invisible items, scenes, 96
Is Kinematic property (rigidbody), 156
Is Trigger property (colliders), 158
isGrounded scripting variable, 215
iteration, 133-135
for loop, 134-135
while loop, 134

J-K

Joy Num property (axis), 144
jumping
Amazing Racer, 344
character controllers, 222-223
key codes, 143
key input, 143

L

label controls, GUIs (graphical user interfaces), 200
lakes, creating, 76
layers, 92-97
overloading, 92
Layers drop-down menu, 18, 95
Layout drop-down menu, 18
lens flares, 74-75
license, Unity, activating, 3-4
Lifetime Loss property (Collision module), 254
lighting scenes, 14
Lightmapping property (point lights), 83
lights, 81-88, 98
area, 86
baking objects, 82
cookies, 88
creating out of objects, 86
directional, 82-86
duplicate properties, 81
halos, 86-87
layers, 92-97
point, 82-84
properties, 83
spotlights, 82-85
Limit Velocity over Lifetime module (particle system), 251
listings
Default Script Code, 122
Demonstration of Class and Local Block Level, 126
Game Control Script, 181
GoalScript.cs, 179
VelocityScript.cs, 176
LoadLevel() method, 355-356, 365
local components, accessing, 147-148
local coordinates, versus world coordinates, 26-27
logical operators, 130
Loop property (audio source), 320
Loop property (wrap modes), 271
Looping property (Particle System), 249
loops
for, 134-135
while, 134

M

Main Color property (shader), 46
managing scenes, 353-356
maps, heightmaps
formats, 54
sculpting, 53-54
Mass property (rigidbody), 156
Material property (colliders), 158
Material property (Renderer module), 257
materials, 44, 46
models, applying to, 47
Max Distance property (3D audio), 322
Max Particle Size property (Renderer module), 257
Max Particles property (Particle System), 249
Maximize on Play button (Game view), 17
Mesh Collider, 159
meshes
versus models, 38
simple modeling, 39
meteors, Captain Blaster, 233
script, 235-237
methods, 137-141
blocks, 139
built-in, 124

calling, 141
as factories, 139
identifying parts, 139
Instantiate(), 194
LoadLevel(), 355-356, 365
Move(), 217
names, 138
OnGUI(), 336
parameter list, 138-139
return type, 138
SimpleMove(), 217
writing, 140-141

Min Distance property (3D audio), 322

Min Kill Speed property (Collision module), 254

Min Move Distance property (character controller), 216

minimum requirements, Unity, 4

mobile development, 329, 338
environments, setting up, 330-331
preparing for, 329-333
Unity Remote app, 331-333

mobile devices, 317
accelerometers, 332-336
multi-touch input, 335-336
testing, 333

models, 37-42, 46
animating, 270
applying textures, shaders, and materials, 47
Asset Store, 41-42
built-in 3D objects, 38-39
downloading, 42
importing, 39-40
versus mesh, 38
model asset workflow, 41
preparing for animation, 263-268
rigging, animators, 278-281

modules, particle systems, 247-257
Collision, 253
Color by Speed, 252
Color over Lifetime, 251-252
Emission, 249
External Forces, 253
Force over Lifetime, 251
Limit Velocity over Lifetime, 251
Renderer, 256-257
Rotation by Speed, 253
Rotation over Lifetime, 253
Shape, 250
Size by Speed, 253
Size over Lifetime, 252
Sub Emitter, 256
Texture Sheet, 256
Velocity over Lifetime, 250

MonoDevelop, 119

mouse input, 146

Move() method, 217

multiple cameras, 90

multiple skyboxes, cameras, 73

multi-touch input, mobile devices, 335-336

Mute property (audio source), 320

N

Name property (axis), 144

names, methods, 138

Negative Button/Positive Button property (axis), 144

nested objects, transformations, 33-34

nesting, 11

Normal Direction property (Renderer module), 257

Normal Map property (shader), 46

Normalized View Port Rect property (cameras), 91

O

objects, 23, 27. ***See also*** **specific objects**
baking, 82
built-in, 27
built-in 3D objects, 38-39
character controllers, adding, 214-215
consolidating, 170
control, 178-182
creating lights out of, 86
detail settings, 71
dimensions, 23-24
finding, 148-151
game control, adding, 107-109
keeping, 355-357
layers, 92-97
overloading, 92
modifying components, 151-152
nested, transformations, 33-34
persisting, 355-359
prefabs, 185-186, 194
breaking, 194
creating, 188-191
inheritance, 186, 192-193
instances, 186
instantiating through code, 194
structure, 186-188
updating, 193
pushing, 223-224
rotation, 30-31
scaling, 32
target, transforming, 152
textures, 43-44
transformations, 28-34
hazards, 32-33
nested objects, 33-34
transforming, 148
translation, 29-30

obstacles, Gauntlet Runner, 302
scripts, 311

Offset property (shader), 46

Once property (wrap modes), 271

OnGUI() method, 336

operating systems, supported, 4

operators, 127
arithmetic, 127-128
assignment, 128
equality, 129-130
logical, 130

order, scenes, establishing, 354
organization, projects, 10

P

painting
  grass, 68-70
  textures, terrain, 61
  trees, 65-68
Pan Level property (3D audio), 322
parameter list, methods, 138-139
parameters, animators, 290-291
Particle Radius property (Plane mode), 253
particle systems, 245-246, 259
  creating, 246
  curve editor, 256-258
  modules, 247-257
    Collision, 253
    Color by Speed, 252
    Color over Lifetime, 251-252
    default, 249
    Emission, 249
    External Forces, 253
    Force over Lifetime, 251
    Limit Velocity over Lifetime, 251
    Renderer, 256-257
    Rotation by Speed, 253
    Rotation over Lifetime, 253
    Shape, 250
    Size by Speed, 253
    Size over Lifetime, 252
    Sub Emitter, 256
    Texture Sheet, 256
    Velocity over Lifetime, 250
  particles, 247
    making collide, 253
particles, making collide, 253
Pause button (Game view), 16
per-platform settings, players, 360-361
persisting objects, 357
phase property (touch), 336
physics
  character controllers, gravity, 222
  collision, 155, 157, 164
    colliders, 157-159
    handling, 219
    physics materials, 160
    raycasting, 162-165
    rigidbodies, 155-156
    triggers, 160-163
  jumping, character controllers, 222-223
physics materials, 160
picture in picture, 92-94
Ping Pong property (wrap modes), 271
Pitch property (audio source), 320
Place Tree tool, 66
Planes property (Plane mode), 253
Planes/World property (Collision module), 254
Play Automatically property (Animation component), 269
Play button (Game view), 16
Play On Wake property (audio source), 320
Play On Wake property (Particle System), 249
player input
  key, 143
  mouse, 146
  scripting, 142-146
PlayerPrefs file, saving data to, 358-359
players
  Amazing Racer, movement, 341-343
  Captain Blaster, 231-233
    script, 238-240
  Chaos Ball, 173-174
  Gauntlet Runner, 302-307
    script, 309-310
  settings, 359-361
playtesting Amazing Racer, 113
point lights, 82-84
  properties, 83
  scenes, adding to, 84
position property (touch), 336
power up script, Gauntlet Runner, 301, 311
prefabs, 185-186, 194
  breaking, 194
  creating, 188-191
  inheritance, 186, 192-193
  instances, 186
    adding to scenes, 191
  instantiating through code, 194
  structure, 186-188
  updating, 193
Prewarm property (Particle System), 249
priorities, audio, 319
Priority property (audio source), 320
private variables, 126
Project dialog, 5-7
Project view, 9-10
  prefabs, 185-186
Projection property (cameras), 91
projects. *See also* games
  adding terrain to, 51-53
  attaching scripts, 121
  creating first, 6
  organization, 10
properties
  Animation component, 269
  audio source, 320
  axis, 144
  cameras, 91
  character controllers, 215-216
  colliders, 158
  fog, 74
  lights, 81
  physics materials, 161
  Place Tree tool, 66
  point lights, 83
  rigidbodies, 156
  shaders, 46
public variables, 126
pushing objects, 223-224

R

Radius property (character controller), 216
radius scripting variable, 215
Range property (point lights), 83
Rate property (Emission module), 249
raycasting, 162-165
reading
mouse movement, 146
specific key presses, 143
Receive Shadows property (Renderer module), 257
Render Mode property (point lights), 83
Render Mode property (Renderer module), 257
Render mode (Scene view), 14
Renderer module (particle system), 256-257
Rendering Path property (cameras), 91
repeat buttons, GUIs (graphical user interfaces), 201-202
requirements
Amazing Racer, 103
Captain Blaster, 228
Gauntlet Runner, 298
resources, 371
return type, methods, 138
rigging models, animators, 278-281
rigidbodies, 155-156
character controllers, 215
properties, 156
rigs, animations, 262-263
rotation, objects, 30-31
Rotation by Speed module (particle system), 253
Rotation over Lifetime module (particle system), 253
rules
Amazing Racer, 102-103
Captain Blaster, 228
Chaos Ball, 168
Gauntlet Runner, 298

S

saving data, 357-359
to PlayerPrefs file, 358-359
Scale Plane property (Plane mode), 253
scaling objects, 32
Scene view, 13-15, 18-20
Flythrough mode, 19-20
testing, 321
scenes, 12, 353, 369
adding soldier model to, 268
changing via buttons, 356
character controllers, adding to, 76-78
directional lights, adding to, 82
establishing order, 354
fog, adding to, 74
gizmo, 15
invisible items, 96
lens flares, adding to, 74
lighting, 14
managing, 353-356
point lights, adding to, 84
Scene view, 18-20
setting up, animators, 287-294
skyboxes, adding to, 72-73
spotlights, adding to, 85
switching, 355-356
scope, variables, 126
scripting, 117, 135, 137, 151
accessing local components, 147-148
animations, 272-273
animators, 294-295
audio, 324-326
character controllers, 215-219
variables, 215
conditionals, 130-133
finding objects, 148-151
iteration, 133-135
methods, 137-141
modifying object components, 151-152
operators, 127
arithmetic, 127-128
assignment, 128
equality, 129-130
logical, 130
player input, 142-146
scripts, 107, 118
adding, 109-110
attaching, 121
basic, 122-124
Captain Blaster
bullet, 241-242
meteor, 235-237
meteor spawn, 237-238
player, 238-240
trigger, 238
class contents, 123-124
class declaration section, 123
connecting, 111-113
creating, 118-119
Game Control Script, 181
Gauntlet Runner
game control, 307-309
obstacle, 311
player, 309-311
power up, 311
spawn, 311-312
trigger zone, 307
GoalScript.cs, 179
using section, 123
variables, 125-126
VelocityScript.cs, 176
sculpting
heightmaps, 53-54
terrain, tools, 56-58
worlds, 104-105
Send Collision Messages property (Collision module), 254
Sensitivity property (axis), 144
Separate Axis property (Limit Velocity over Lifetime module), 251
settings, players, 359-361
shaders, 44-47
bumped, 45
diffuse, 45

models, applying to, 47
properties, 46
Shadow Type property (point lights), 83
Shape module (particle system), 250
Shininess property (shader), 46
Shuriken system
modules, 247-257
Collision, 253
Color by Speed, 252
Color over Lifetime, 251-252
default, 249
Emission, 249
External Forces, 253
Force over Lifetime, 251
Limit Velocity over Lifetime, 251
Renderer, 256-257
Rotation by Speed, 253
Rotation over Lifetime, 253
Shape, 250
Size by Speed, 253
Size over Lifetime, 252
Sub Emitter, 256
Texture Sheet, 256
Velocity over Lifetime, 250
value curves, 248
SimpleMove() method, 217
Simulation Space property (Particle System), 249
Size by Speed module (particle system), 253
Size over Lifetime module (particle system), 252
Size property (colliders), 158
Skin Width property (character controller), 216
skins, GUIs (graphical user interfaces), 207-209
skyboxes, 72-73
cameras, adding to, 72-73
scenes, adding to, 72-73
slide, controlling, 218
sliders, GUIs (graphical user interfaces), 205
Slope Limit property (character controller), 216
slopeLimit scripting variable, 215
Snap property (axis), 144
Sort Order property (Renderer module), 257
Sorting Fudge property (Renderer module), 257
sources, audio, 319-323
Space property (Velocity over Lifetime module), 250
spawn script, Gauntlet Runner, 311-312
specific key presses, reading, 143
Specular Color property (shader), 46
specular textures, 45
Speed property (Limit Velocity over Lifetime module), 251
Speed Range property (Color by Speed module), 252
split screens, 92-94
spotlights, 82-85
cookies, 88
Spread property (3D audio), 322
Start Color property (Particle System), 249
Start Delay property (Particle System), 249
Start Lifetime property (Particle System), 249
Start Rotation property (Particle System), 249
Start Speed property (Particle System), 249
starting audio, 324-326
statements
if, 131
if/else, 131-132
if/else if, 132-133
states, animators, 290-292
Static Friction property (physics material), 161
Static Friction 2 property (physics material), 161
Stats button (Game view), 17
Step button (Game view), 16
Step Offset property (character controller), 216
stepOffset scripting variable, 215
stopping audio, 324-326
string variable, 125
structure, prefabs, 186-188
styles, GUIs (graphical user interfaces), 206-207
Sub Emitter module (particle system), 256
supported operating systems, 4
switching scenes, 355-356

T

tabs, adding, 8
Tag Manager, adding new layers to, 95
tapCount property (touch), 336
target objects, transforming, 152
Target Texture property (cameras), 91
terrain, 51, 62. *See also* environments
assets, 66
flattening, 57
generation, 51-58
heightmaps, sculpting, 53-54
importing assets, 59
sculpting, tools, 56-58
settings, 70-71
size, 53
textures, 59-62
creating, 62
painting, 61
Terrain Settings tool, 70-71
testing
Amazing Racer, playtesting, 113
audio, 321
2D, 323
Scene view, 321
mobile devices, 333
textareas, GUIs (graphical user interfaces), 204
textfields, GUIs (graphical user interfaces), 204
Texture property (shader), 46
Texture Sheet module (particle system), 256

textures, 41-44, 46
- baking objects, 82
- cookies, 88
- grass, 69
- models, applying to, 47
- specular, 45
- terrain, 59-62
  - creating, 62
  - painting, 61

texturing Chaos Ball arena, 170-171

Tiles property (Texture module), 256

Tiling property (shader), 46

toggles, GUIs (graphical user interfaces), 202

toolbars
- editor, 17
- GUIs (graphical user interfaces), 203

tools
- Hand, 18-19
- Place Tree, 66
- sculpting, 56-58
- Terrain Settings, 70-71
- transform, 17

touch
- multi-touch input, mobile devices, 335-336
- tracking, 335-336

transform tools, 17
- Hand tool, 18-19

transformations
- hazards, 32-33
- nested objects, 33-34
- objects, 28-34, 148

transitions, animators, 293-294

translation, objects, 29-30

trees
- dark, 67
- painting, 65-68
- settings, 71
- warping, 68

triangles, 38

trigger zone, Gauntlet Runner, 302
- script, 307

triggers
- Captain Blaster, 234
  - script, 238
- colliders, 160-163

Type property (axis), 144

Type property (point lights), 83

U

Unity
- editor, 5-17
- engine versions, xiii
- installing, 1-5
- interface, 7-8
- license, activating, 3-4
- minimum requirements, 4

Unity Remote app, 331-333

updating prefabs, 193

Use Gravity property (rigidbody), 156

user input. *See* player input

using section, scripts, 123

V

value curves, Shuriken system, 248

variables, 125-126
- character controllers, scripting, 215
- CollisionFlags, 218-219
- creating, 125
- private, 126
- public, 126
- scope, 126

Velocity over Lifetime module (particle system), 250

velocity scripting variable, 215

VelocityScript.cs listing, 176

views
- Animator, 289
- duplicating, 8
- Game, 15-17
- Hierarchy, 11-12, 187
- Inspector, 12-13
- Project, 9-10
  - prefabs, 185-186
- Scene, 13-15, 18-20
  - Flythrough mode, 19-20

Visualization property (Plane mode), 253

Volume property (audio source), 320

Volume Rolloff property (3D audio), 322

Voxel Size property (World mode), 253

W-Z

walk animations, 284

walk turn animations, 285-287

warping, trees, 68

water, 75-76

while loop, 134

wind, settings, 71

world coordinates, versus local coordinates, 26-27

worlds
- Amazing Racer, 104-106
- Captain Blaster, 229-234
- Chaos Ball, arena, 169-173
- Gauntlet Runner, 298-300
- sculpting, 104-105

wrap modes, 271

writing methods, 140-141

XYZ property (Velocity over Lifetime module), 250

How can we make this index more useful? Email us at indexes@samspublishing.com