



LEARNING 2D GAME DEVELOPMENT WITH **UNITY**[®]

A Hands-On Guide to Game Creation

MATTHEW JOHNSON
JAMES A. HENLEY

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

Learning 2D Game Development with Unity[®]

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The **Addison-Wesley Learning Series** is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

◆ Addison-Wesley

informIT®
the trusted technology learning source

Safari®
Books Online

Learning 2D Game Development with Unity[®]

A Hands-On Guide
to Game Creation

Matthew Johnson

James A. Henley

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Johnson, Matthew (Computer programmer)

Learning 2D game development with Unity : a hands-on guide to game creation / Matthew Johnson, James A. Henley.

pages cm

Includes index.

ISBN 978-0-321-95772-6 (pbk. : alk. paper)—ISBN 0-321-95772-5 (pbk. : alk. paper)

1. Computer games—Programming. 2. Unity (electronic resource) I. Henley, James A. II. Title.

QA76.76.C672J64 2015

794.8'1526—dc23

2014037406

Copyright © 2015 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

Unity, Unity Free, Unity Pro, and the Unity Web Player are registered trademarks of Unity Technologies.

Adobe®, Flash®, and Photoshop® are registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. THIS PRODUCT IS NOT ENDORSED OR SPONSORED BY ADOBE SYSTEMS INCORPORATED, PUBLISHER OF Adobe® Flash® and Photoshop®.

The Kenney logo and the Made with Kenney Logo belong to Kenney.nl, Netherlands, used with permission.

Mac® is a trademark of Apple Inc., registered in the U.S. and other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

DirectX, Direct3D, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

ISBN-13: 978-0-321-95772-6

ISBN-10: 0-321-95772-5

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing, December 2014

Editor-in-Chief

Mark L. Taub

Executive Editor

Laura Lewin

Development Editor

Songlin Qiu

Managing Editor

John Fuller

Senior Production Editor

Kesel Wilson

Copy Editor

Barbara Wood

Indexer

Jack Lewis

Proofreader

Melissa Panagos

Technical Reviewers

Reshat Hasankolli, II

Sheetanshu Sinha

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Shepherd, Inc.



First and foremost, a big thanks to my friends and family for supporting me throughout this process. You pushed me to start this book and then begged me to finish it!

To my late father, who taught me to inspire and entertain others with my talent and wild imagination: I am forever your biggest fan.

Last, to the love of my life and best friend, Jessica: without you this book would have been just more dust in the wind of my ideas. Thank you for all of your guidance and wisdom, and for believing that I could accomplish this. I love you always and forever.

—Matthew Johnson

For my wonderful wife, who brought me tea whenever I locked myself in my office to write and tolerated being temporarily widowed by this book.

—James A. Henley



This page intentionally left blank

Contents at a Glance

Preface	xix
Acknowledgments	xxv
About the Authors	xxvii
Introduction	1
1 Setting Up the Unity Development Environment	9
2 Understanding Asset Creation	27
3 Creating 2D Sprites	41
4 Building the Game World	53
5 The Basics of Movement and Player Control	71
6 Adding Animations to Our Scene	95
7 Setting Up Player Physics and Colliders	119
8 Creating and Applying Gameplay Systems	135
9 Creating Hazards and Crafting Difficulty	159
10 Creating the Menus and Interface Elements	193
11 Applying Effects to the GameObjects	215
12 Organization and Optimization	247
13 Bringing It All Together	269
14 UGUI	291
Appendix: JavaScript Code Samples	301
Index	323

This page intentionally left blank

Contents

Preface	xix
Acknowledgments	xxv
About the Authors	xxvii
Introduction	1
Introduction to Unity	1
Downloading and Installing Unity	1
Project Wizard	4
Open Project Tab	4
Create New Project Tab	5
Packages	5
Setting Our Project	6
Project Structure	6
Folder Organization	7
File Naming Conventions	7
1 Setting Up the Unity Development Environment	9
Welcome Screen	9
The Unity Interface	10
Menus	10
Toolbar	19
Hierarchy	20
Inspector	20
Project Browser	21
Scene View	23
Game View	24
Summary	25
Exercises	25
2 Understanding Asset Creation	27
File Formats	27
3D Formats	27
2D Formats	28

Importing Our Assets	28
Importing from Inside Unity	28
Importing Premade Assets from the File Browser	29
Creating New Assets	29
Importing Packages	31
Unity Packages	31
Custom Packages	31
GameObjects	33
Our First GameObject	33
Creating a GameObject	34
Components	36
Creating a Component	37
Assign a Component	37
Prefabs	38
Summary	39
Exercises	39
3 Creating 2D Sprites	41
Working in 2D	41
2D Behaviors	41
2D Workspace	42
Building Our Sprites	43
Import Settings	43
Pixels To Units	45
Sprite Editor	45
Sprite Packing	48
Packing Tag	48
Sprite Packer	48
Additional Sprite Packing Resources	49
Summary	50
Exercises	50
4 Building the Game World	53
Level Design 101	53
Setting the Scene	53
Creating a Roadmap	54
Adding Details	55

Getting around Our Scene	56
Scene Gizmo	56
Perspective versus Isometric	57
Camera Controls	57
Manipulating Objects in Unity	59
Transform Tools	59
Z-Depth	61
Settings	62
Our First Level	64
Positioning GameObjects Manually	64
Using the Snap Settings to Position GameObjects	64
Using Grid Snapping to Position GameObjects	64
Efficient Level Design	66
Adding Sorting Elements	67
Continuing On	69
Summary	69
Exercises	70
5 The Basics of Movement and Player Control	71
Coding in Unity3D	71
The Three Languages	71
Choosing the “Right” Language	72
Making the Player Go	72
Different Ways of Handling Movement	72
Creating and Hooking Up Our PlayerController	74
Setting Up a Basic Follow-Cam	83
Introducing the Input Manager	83
Error Handling and Debugging	85
Handling Exceptions	85
Try-Catch-Finally—Gracefully Handling Exceptions	87
Debug.Log() Is Your Friend	89
Using Breakpoints to Halt Code Execution	90
Summary	93
Exercises	94

6 Adding Animations to Our Scene	95
Some Rules for Animation	95
Animation Principles	95
2D versus 3D Animation	96
Transform versus Frame Animation	97
Scripted Animations	98
Imported Animations	98
Creating Animations	99
Animation Component	100
Animation Clip	100
Animation Window	101
Animation Events	107
Animation States	108
Animator Controller	108
Animator Component	109
Animator Window	110
Editing the Player Controller	112
Working with the State Machine	115
Transitions	115
Any State	115
Blend Trees	116
Summary	116
Exercises	117
7 Setting Up Player Physics and Colliders	119
Understanding Physics	119
Mass	119
Gravity	120
Force	120
2D versus 3D	120
6DoF	120
Z-Depth	121
Rotations	121
Physics 2D Settings	122
General Physics Settings	122
Layer Collision Matrix	123
Rigidbody	124

Colliders	125	
Circle Collider	126	
Box Collider	126	
Edge Collider	126	
Polygon Collider	126	
Physics Materials	128	
Constraints	129	
Summary	134	
Exercise	134	
8 Creating and Applying Gameplay Systems	135	
Trigger Volumes in Unity	135	
Trigger2D Functions	135	
Adding Trigger Components to GameObjects	136	
Creating Checkpoints	136	
Scripting the Checkpoint Component	137	
Sizing and Placing Our Checkpoint Trigger	138	
Using Checkpoints with Respawn	140	
Preparing the Pit Trigger Volume	140	
Scripting the Pit Trigger Component	140	
Creating Collectibles	144	
Preparing the Floating Coin Prefabs for Collection	145	
Scripting the CoinPickup Component	145	
Preparing the Popped Coin Prefabs for Collection	147	
Preparing the Coin Box Prefabs	147	
Scripting the Coin Box Component	150	
Scripting the CoinSpawner Component	152	
Hooking It All Together	153	
A Touch of Polish	154	
Tracking the Player's Stats	155	
Summary	157	
Exercises	158	
9 Creating Hazards and Crafting Difficulty	159	
Creating Your First Enemy	159	
Preparing the Slime Enemy GameObject	159	
Inheritance and the EnemyController Component	161	

Scripting the Enemy Slime Component	162
Adding Walls to the Level	164
Handling Collision with Other Slimes	165
Adding Animation to the Slime	166
Dealing Damage	167
Scripting Damage into the PlayerStats Component	167
Creating the Damage Trigger	168
Passing through the Player's Space	170
Adding Damage to the Pits	172
Adding Temporary Immunity Post-Damage	172
Visually Representing Immunity, the Classical Way	175
Handling Player Death	177
Expanding on Platforming	178
Preparing the Moving Platform Prefab	179
Scripting the Flight Points Component	180
Creating Your Second Enemy	182
Preparing the Fly Enemy GameObject	183
Adding Animation to the Fly	184
Scripting the FlyController Component	185
Adjusting the FlightPoints Script	185
Maintaining Your Enemy Arrangements	187
Preparing the Spawn Trigger	188
Scripting the Spawn Trigger Component	189
A Few Words on Challenge	190
Summary	191
Exercises	191
10 Creating the Menus and Interface Elements	193
UI Design	193
Diegetic	194
Non-diegetic	194
Meta	194
Spatial	194
Unity Native GUI	194
GUI Style	195
GUI Skin	195

GUI Controls	195
Compound Controls	197
GUI Class	197
GUI Layouts	197
GUI Text	198
GUI Texture	198
Creating a Splash Screen	198
Title Screen	200
Game Over Screen	201
Game Win Screen	202
HUD	204
Creating the Visuals	204
Creating the Scripts	206
Summary	212
Exercise	213

11 Applying Effects to the GameObjects	215
Introducing the Shuriken Particle System	215
Terms to Know	215
Creating a Particle System	216
Modules and Properties of a Particle System	217
Base Particle System Properties	217
Other Particle System Modules	218
Particle System Curves	219
Adding Particle Effects to the Game	220
Creating a Particle Effect for Coin Boxes	220
Hooking Up the Coin Box Particle Effect	223
Creating a Particle Effect for Damage	223
Calling the Damage Particle System from Code	225
Having a Little Particle Fun	227
Unity's Audio System	227
The Audio Source Component	228
The Audio Listener Component	230
The Audio Reverb Zone Component	230
Adding Sound to the Player	231
Adding Footsteps to the Walk Cycle	231

Adding Sound to the Jump Event	233
Adding Sound to the Damage Event	234
Adding Sound to the Collectible System	236
Applying Sound to the Coin Box	236
Applying Sound to Coin Collection	237
Applying Some Extra Polish	238
Cleaning Up the Camera	238
Cleaning Up Player Death	241
Summary	246
Exercises	246
12 Organization and Optimization	247
Organizing Assets	247
Organizing Our Prefabs	248
Labels	249
Hierarchy	250
Organizing Scripts and Code	253
Organizing the Script Files	253
Organizing the Code	254
Optimizations	261
Prefabs	262
Physics	262
Draw Calls	264
Triangle Count	265
Batching	266
Rendering Statistics Window	266
Summary	267
Exercises	268
13 Bringing It All Together	269
Tying the Levels Together	269
Preparing the Victory Trigger Prefab	269
Creating the Victory Trigger Script	272
Retrieving the Coin Value	274
Hooking Up the Intro Screens	275
Win or Lose: Getting Back into the Action	277
Recovering from Game Over	278
Starting Over from a Win	279

Building and Deploying the Game	281
Web Player's Build Settings	282
PC, Mac, and Linux Standalone Build Settings	282
Cross-Platform Player Settings	283
Web Player's Player Settings	283
The Right Settings for the Job	284
Building the Game for the Web Player	285
Deploying the Game to the Web	285
Post-Deployment	286
Moving Forward	286
Polish Considerations	286
Monetization	287
Final Words	290
14 UGUI	291
UGUI Components	291
Creating Our Example Interface	293
Canvas Component	293
Rect Transform	296
UI Rect Tool	298
Adding the Mask	299
The Event System and Event Triggers	299
Summary	300
Appendix: JavaScript Code Samples	301
Player Scripts	301
Collectible Scripts	307
Enemy Scripts	309
Game System Scripts	310
GUI Scripts	314
Hazard Scripts	320
System Scripts	322
Index	323

This page intentionally left blank

Preface

Why Write This Book?

Since there are hundreds of books on game design and quite a few about using the Unity game engine, you might be asking, “Why even write another book on Unity?” We wanted to write a book about game development with a 2D approach, using an engine that is most widely known for being 3D. There are a bunch of Unity books covering 3D mesh and building game worlds, and fancy game mechanics, but there is really not much in the way of anything about a 2D platformer.

Another goal was to show a simplistic and inexpensive approach to creating your own game. Creating games is tough enough with the amount of time and effort you have to put into it, and rising costs are something no indie developer wants to deal with. Every element of this book uses free software and assets to build the game!

Last, we wanted to write our book around a small game project. Using simple approaches to scripting and asset creation, we wanted to create a game that even someone new to Unity and game development could easily pick up and tackle. Every aspect of the project is covered in the book with clear explanations, examples, and images!

Who Is This Book For?

This book is for those who want to learn more about the process of creating a game and all of the different parts that are involved, from having an initial idea, planning and designing, to the final steps of building and deploying the game to share with others.

This book is also for those who are new to Unity and the new 2D tools that have recently been integrated. We will touch on creating sprites and sprite atlases, applying 2D physics, and adding game scripts, audio, and animations. Almost every aspect of Unity is touched on and explained in detail.

Why Did We Choose to Use Unity?

The core of any game development is the game engine. It needs to handle all of the rules, tasks, and mathematics thrown at it. It also needs to be able to grow and evolve with new technology and the needs of the consumers playing the games.

While it’s possible to develop your own game engine, starting with a well-structured foundation allows you to focus on creating your game content and letting the game engine do the dirty work. There are a dozen great game engines that are capable of this, but Unity excels where others have failed.

Having started out as a great 3D game engine, Unity has blossomed into an end-all development tool for creating games that you can then push to just about every platform available. As time went on, the need for more 2D game tools became obvious, and Unity jumped onboard, creating some of the most intuitive and easy-to-use 2D tools available.

Another reason is how accessible Unity is. While the Pro version has some really great additional features, they are tailored more to teams or people looking to really fine-tune every aspect of their game. We will cover a few of the Pro features, but the free version will work great for us.

What Will You Need?

So what will you need to develop your game? After you have purchased this book, and assuming you have a computer to work on, there is nothing else to buy. All of the assets we will use to create our game are accessible to anyone and readily available on the Internet, the obvious being the Unity engine, which is easily downloadable from their Web site (we cover this in the Introduction).

For the game sprites we were lucky enough to get assets from a great game artist, Kenney Vleugels. We are including these with the ancillaries for this book, but check out all of the amazing resources and game assets on his Web site, www.kenney.nl. He continually adds more and more assets and will even create specific assets at your request.

Last, all of the scripts for our game will be created within the chapters. We will be providing the final scripts along with the project files, but we recommend you follow along and create them for yourself. Having a good grasp of even simple scripting will take you a long way toward creating a game that is truly unique and completely yours.

Register your book at informit.com/title/9780321957726 to access assets, code listings, and video tutorials on the companion website.

How Much Scripting Is Involved?

While we don't dig down into the trenches of writing complex code behaviors, we do cover a lot of the basics and create quite a few scripts throughout this book. Learning a little programming can go a long way in any profession but is highly recommended for game design. Even tests and debugging are helpful and require just a very basic level of scripting knowledge.

How Is the Book Organized?

Our goal for this book was to have those reading it start from the beginning and work their way through it until the very end. Readers can build upon what they learned in previous chapters and continually come back to elements they have already built. With

these building blocks, we hope that at the end, you will have the confidence and skill to either continue building on the example project or start your own game design.

However, we know this is not the case. There will be those who have an understanding of the game development process and are looking for a game authoring engine upon which to build their idea and designs. So we have broken each chapter down into individual lessons. That way those who are looking to learn about a specific mechanic or process can easily jump ahead.

We encourage even those with a general understanding of Unity to read through all of the chapters, as we cover many elements of the Unity engine, both old and new. We have provided notes, tips, and figures throughout to help reinforce or reiterate a specific lesson, so look for these as well.

Here is a summary of each chapter:

- Chapter 1, “Setting Up the Unity Development Environment”
This chapter will familiarize readers with the Unity interface, provide them with an understanding of a Project’s hierarchy, and begin to build the initial Project for the game that will be created throughout this book.
- Chapter 2, “Understanding Asset Creation”
In this chapter readers will start to build the foundation of the game by importing the assets we will use for the game Project. They will get an understanding of how the Unity engine uses GameObjects on which everything in Unity is built. This chapter will break down how Components are the nuts and bolts of a GameObject and how to utilize them to build upon each other for complex behaviors. Last, this chapter will touch upon using third-party assets and packages and how to bring them into our game environment.
- Chapter 3, “Creating 2D Sprites”
In this chapter we will dive into the new tools and features added for building 2D gameplay. We will discuss the sprite editor, as well as some Pro-only features and how we can work around them.
- Chapter 4, “Building the Game World”
In this chapter we will take all of the existing prefabricated GameObjects and start building the world our player will live in. We will learn to use the Transform tools to place our GameObjects, and we will learn about sorting our sprites for layering and depth. Finally, we will go over grouping sprites and the parent-child relationship, and how keeping these organized and named correctly will keep our Scene View and Hierarchy easy to manage.
- Chapter 5, “The Basics of Movement and Player Control”
This chapter will teach a basic understanding of creating scripts and functions to drive input and control the physical behaviors of our GameObjects. We will discuss the basic scripts for controlling user input and building upon these for all the necessary mechanics of our game. We will briefly discuss the Unity native

programming languages and the pros and cons of each. This chapter will also discuss error handling and basic debugging of scripts.

- Chapter 6, “Adding Animations to Our Scene”

This chapter will go into setting up and creating the animations for the GameObjects and sprites. It will discuss creating animations with base transform versus frame animations and the benefits of both methods. We will then discuss creating 2D sprite behaviors with the Animator State Machine. Here we will begin to create the mechanics for the characters for our game.

- Chapter 7, “Setting Up Player Physics and Colliders”

This chapter will discuss adding physics for both 2D and 3D GameObjects. It will discuss setting up GameObject collision and knowing which one is best to maintain game performance. We will also discuss setting up the forces for our GameObjects and creating dynamic physics.

- Chapter 8, “Creating and Applying Gameplay Systems”

This chapter will discuss the creation of key gameplay elements such as picking up collectibles, checkpoints, and respawning. Readers will be taught about Unity’s trigger system and the code methods that it uses. We will also include some design theory related to these systems.

- Chapter 9, “Creating Hazards and Crafting Difficulty”

This chapter will discuss the creation of some basic enemy types and the underlying code that makes them work. We will add damage scripting and teach the player how to hook enemies into spawning logic. This chapter will also touch on some of the design theory related to difficulty and tuning.

- Chapter 10, “Creating the Menus and Interface Elements”

In this chapter we will create the basic menus for getting into and out of our game as well as the game interface elements that will make up the on-screen player information and statistics. We will discuss basic input for menu selections and game screens.

- Chapter 11, “Applying Effects to the GameObjects”

This chapter will guide the reader in adding the final polish to the game assets by adding animations, effects, and audio Components. It will discuss an overview of the Unity particle system, adding audio listeners and effects to our non-character gameplay elements.

- Chapter 12, “Organization and Optimization”

We will go over final tips and recommendations for game optimizations and compressions for deploying to the various platforms. We will also look into some final organization tips for file handling and future revisions.

- Chapter 13, “Bringing It All Together”
In this chapter we will wrap things up. We will package up our game and discuss publishing the game with the Unity Web Player and other platforms. This chapter will briefly detail best practices for monetizing a game and advice for a successful published game. We will then look at publishing the game to the Web.
- Chapter 14, “UGUI”
Chapter 14 is a bonus chapter for the upcoming UGUI system in Unity 4.6. We will take an in-depth look into setting up a new UGUI interface and getting acquainted with the Components and new Rect Transform Component.

Conventions Used in This Book

The book uses a few conventions for explaining best practices and for sharing useful information that is relevant to learning Unity.

- Figures
Figures are used to explain a process or approach that needs a visual example to clarify the information. These are used throughout the chapters as assets and levels are generated to show the game design process taking shape.
- Notes
Notes are used to share additional information with the reader that does not fit exactly into the context but should be explained along with it.
- Tips
Tips are used to share workflows or information not generally known that can help with a specific task or problem.
- Warnings
Warnings are used to signify caveats about established rules or situations in which the game may behave in unexpected ways. These will help you sidestep some of the potential pitfalls of the game development process.
- Code listings
Code listings are used throughout the book and are the bread and butter of the game development process. Code listings contain elements of a script or the entire script that the reader can copy and use or use as a guide when approaching a specific function. Later chapters use code listings to update gameplay to add additional mechanics and features that help polish the gameplay.

- Exercises

At the end of almost every chapter are exercises for readers to complete. These are based on the content of that chapter. After reading the chapter, they should have gained a solid understanding of the information and should be able to use it to complete more gameplay elements. The exercises are built to be easily completed yet require a little bit of discovery and trial and error.

Supplementary Materials

The project development covered within this book will help you build a fully realized game Project, complete with all of the sprites, audio, and script assets that we show. The objective is to help you learn how to create all of these from scratch, but we do understand that you might get stuck or confused about specific processes.

For this reason, we have provided a Web site to grant you access to the entire contents of the Project we will make throughout the book. All of the Scenes, packages, assets, and scripts will be available to you to follow along with, or to reference in case you need a little extra help.

We have provided a series of video learning modules for you to watch and to follow along with as well. These are designed to show the UnityProject coming together, with audio commentary that explains our methods and approach to the game design. The videos recapitulate the book's contents, but sometimes a visual explanation is warranted.

Register your book at informit.com/title/9780321957726 to access assets, code listings, and video tutorials on the companion website.

Acknowledgments

Matthew Johnson

Thank you to Laura Lewin for taking a chance on me, and for not flying to Florida to strangle me. I know the urge was there and was warranted.

Thank you to Olivia Basegio for all of your support and guidance. You truly made this experience enjoyable and as pain-free as you could.

Thank you to Songlin, Reshat, and Sheetanshu, whose wisdom truly made this book that much better. Your suggestions and feedback always seemed to be spot-on.

Thank you to Kenny Vleugels for your great artwork and for making it easy to access it. You are a godsend to game developers everywhere.

Last, to my colleague and friend James Henley: a thousand times over, thank you. Thank you for all of your insight, humor, and hard work. Most importantly, thank you for your understanding and patience in helping me see this book through. Without you stepping in to take the helm, it would never have seen the light of day.

James A. Henley

Foremost, I would like to offer my sincerest thanks to K2, without whom I would not have become the kind of developer that I am today. Years of design theory discussions, feedback, and soundboarding have been integral to my growth as a developer.

I would like to express my great appreciation to the publisher, who provided me with both the opportunity and the means to share some of my knowledge and experience. In particular, I'd like to thank Laura and Olivia for taking the time to answer my questions, no matter how mundane, as well as Songlin, Reshat, and Sheetanshu for their hard work as editors and the many excellent suggestions they made.

I would also like to thank my stream regulars for sticking by me despite the many interrupted or canceled casts this book caused.

Finally, I wish to thank you, the reader, for having the courage to pursue the dream that is game development.

Thank you all.

This page intentionally left blank

About the Authors

Matthew Johnson is a principal 3D artist at Firebrand Games in Merritt Island, Florida. He graduated with a BFA from the International Academy of Design, where he trained in computer animation before going on to study animation at Animation Mentor.

Matthew has been in game development for the past seven years working on more than a dozen AAA racing games, such as NASCAR, Hot Wheels, and the Need for Speed series. He has helped publish titles on almost every platform, including PC, Wii U, iOS, Android, and Steam.

In his spare time Matthew enjoys spending time with his wife and two kids and, when he finds time, pursuing his love for photography.

James A. Henley is an experienced game developer who has worked on several major titles and franchises, including Mass Effect, Dragon Age, Star Wars, and Skylanders, over the past decade. He originally entered the industry via the Neverwinter Nights modding community, where he was able to indulge his desires to craft content, tell stories, and write code all at the same time. He turned that love into a job opportunity at BioWare, where he spent three years with the Edmonton studio and five more with the Austin studio in a variety of design roles before briefly working for Activision.

Currently, James is working as an independent developer on [TITLE REDACTED] and is actively live streaming to share his love of games and game design in an interactive fashion. He may or may not also be a mad scientist. Analysis has proven inconclusive.

This page intentionally left blank

Introduction

Welcome to the exciting world of Unity and game development! We hope you are reading this because you want to learn what we have found to be an exciting and rewarding career with indie game development.

Between the two of us, we have over a decade of game development experience and hope to share our insights with you. Both of us have a strong passion for video games and immersing ourselves in hundreds of different worlds and stories. While game development can take a lot of time and effort, seeing someone else play your creation is well worth it! Imagine an idea you have for a new game that you would love to create. Now imagine creating it and then being able to share it with millions of others. Exactly!

We have created this learning guide to get you up to speed quickly with Unity to create your very own platform game from start to finish. This guide may not cover every little detail of Unity as we are focused on a 2D development platform, but we feel that after reading this you can go on to create your own game and dig deeper into Unity, building on what you learn here.

Introduction to Unity

Let's go over the steps needed to get Unity up and running on your machine. We will then take a look at creating a workflow for organizing your files and recommended steps to avoid problems further along in development. Finally, we will describe the basics of the Unity interface, file menus, and navigation. By the end, we hope you will have a grasp of the Unity user interface and some solid principles for creating your Projects. So let's get started!

Downloading and Installing Unity

Before we dive headfirst into the game development aspects of this book, we need to get our Unity environment up and running. You can download the latest Unity release at their Web site: <http://unity3d.com/unity/download>. While writing this book, we used version 4.5.1. You may end up with a newer version depending on when you bought the book, but as long as it is 4.5.0 or above, you will be able to follow along.

While on the Unity download page, take a look at the System Requirements as well as the License Comparisons pages. The System Requirements page lists the general requirements for Windows or OS X-based machines, along with those requirements needed to publish to the various development environments. If you have a fairly recent Operating System such as Microsoft Windows 7/8 or Apple OS X 10.5 (Leopard) or newer, you will be fine.

The License Comparisons page gives you a full rundown of the features available with the free versus paid versions of Unity, as well as the add-ons to Unity Pro for building your game to their respective platforms. Unity Pro comes with a host of added features that make game creation and debugging a lot easier and even more exciting.

Note

While there are a few Unity Pro features we could benefit from in our game development, you will not need them to follow along in this guide. We will, however, mention a few of these in later chapters just to discuss their advantages for those with access to Unity Pro.

Component Installs

Once you have downloaded and begun the installation, Unity will pause and prompt you with the Choose Components screen (see Figure I.1). The items listed here are additional resources and add-ons that you may wish to install along with the Unity

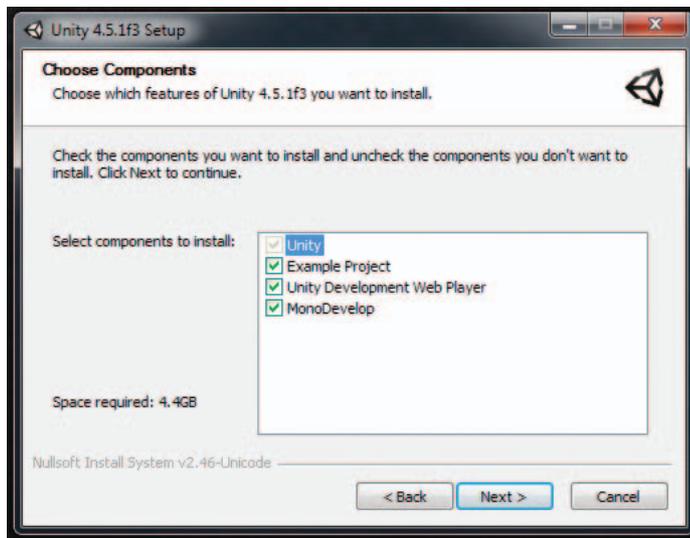


Figure I.1 Unity Choose Components screen

engine. We recommend installing all of them as they will be helpful in your journey developing Unity Projects and games.

Example Project

Angry Bots is a feature-rich Unity Project developed by the minds at Unity Technologies. While it is a fun and immersive game experience, it is of greater significance as a tool for developing your own games. All of the assets are easily viewable in the engine as well as the scripts, Components, and animations. While the features and scope of Angry Bots are beyond what we cover in this guide, we highly recommend looking it over, especially if you are designing any type of 3D game experience.

Note

The Unity Asset Store also carries a bunch of old and new example Projects like this. There are quite a few built by the Unity Technologies team. There is a very good “2D platformer” Project available from them that we highly recommend checking out. We will discuss more about the Unity Asset Store and downloading Projects, packages, and assets in Chapter 2, “Understanding Asset Creation.”

Unity Development Web Player

The Unity Development Web Player will be a vital part of our game development workflow. The Web Player allows you to quickly see what your published game will look and run like on your computer hardware. In later chapters we will see it in action. The Web Player will also be used to publish our game to HTML code for playing through a Web browser such as Firefox or Chrome. This is useful for allowing others to run and test your game from their computers or devices and to get valuable feedback. As we wrap up things in Chapter 13, “Bringing It All Together,” we will go over building for HTML and packaging your game for the Web Player.

MonoDevelop

While we can get so far with our game assets and animations, it’s the core mechanics and gameplay events that create the true experience for the gamer. You can have a hero and a villain character with combat animations, but without gameplay scripts, you won’t be able to move them or have them interact. We can do all of this and more by making a few simple scripts. MonoDevelop is the IDE that allows you to build those runtime events and scripts for Unity. It is by far the most important of the add-ons you can install. If you are fairly new to scripting and Unity, this is a must for creating your game. We will go into MonoDevelop in a lot more depth as we go along, but for now just continue by hitting Next.

Once everything has installed, open Unity by clicking the Unity icon on your desktop. From here you will be given a Unity Activation screen. Click the Register button and finish the registration authorization. Once this is done, you should get a screen for the Project Wizard.

Note

By registering you will be given the option of a 30-day evaluation of Unity Pro and access to the Unity mailing list for upcoming news and updates. Again, we will not have to worry about any of the added Pro features here, but after finishing this guide, if you wish to dive deeper into them, you will be able to update to either the 30-day evaluation or the full version by purchasing Unity Pro.

Project Wizard

The final step (before we see all of the beauty that is the Unity Editor!) is to create our Project. A good way to understand a Project is to think of it as being like building a house. Without having the right pieces such as the walls, doors, and a roof, and if it's not constructed in an organized and methodical manner, things could get unorganized, messy, and come falling down. A Project in Unity can be viewed in the same way: we want to keep it as simple and easy to comprehend as we can for both us and others who are on our team.

Open Project Tab

There are two tabs in the Project Wizard, the first being the Open Project tab (see Figure I.2). Here we can choose an existing Project if we have one. You will use only one Project for each game you are creating. With the house scenario we described, the Project is your house and you really only need one, right? For now we do not have a preexisting Project, so just click over to the Create New Project tab.

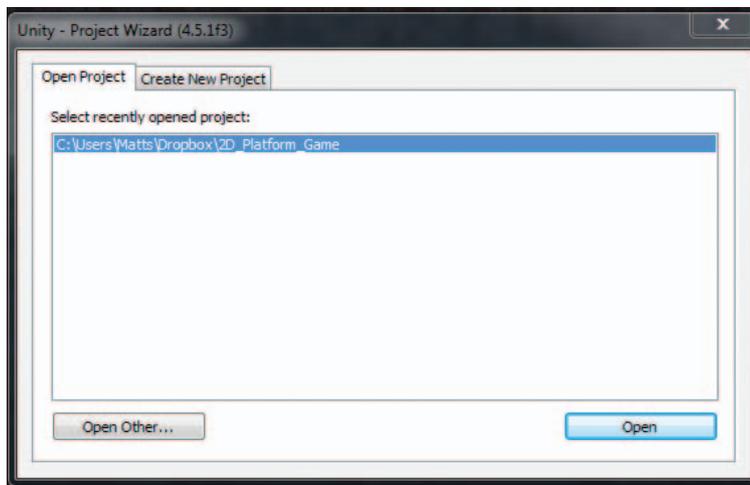


Figure I.2 The Project Wizard—Open Project tab

Tip

We personally like to create a sandbox Project, a second Project that we use for testing game-play scripts. We also use it to test assets so that we won't have to scrap them if we don't need them in our game Project. By importing and then deleting assets in your Project, you can accidentally leave unused assets, or more importantly remove assets or code, that can break your game. Again, keeping our Project clean and clutter-free is vital.

Create New Project Tab

On this tab we will set up the Project that we will use for the duration of this guide and our game. Under Project Location, select the path and name for your Project. By default Unity will name your Project “NewUnityProject” without you entering anything. To follow along we have named our Project “LearningUnity” (see Figure I.3). We have also set up the preference defaults for 2D as we will be building a 2D platform game. You can switch the Unity preferences between 2D and 3D, but Unity will open with a few features defaulted to a 2D setup like this.

Packages

Below the Project Location is a list of packages that come preinstalled with Unity. Packages are collections of assets and GameObjects from a Unity Project that get bundled and exported. You can then take them into other Projects and use them there. This can be very useful for reusing scripts and certain elements rather than having to remake them.

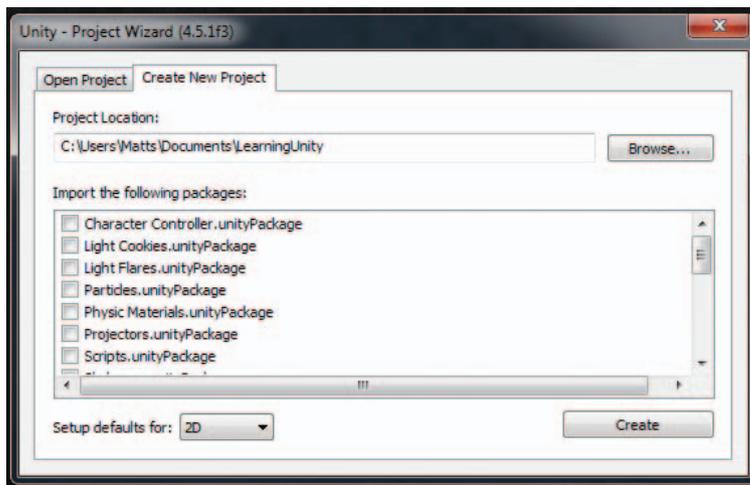


Figure I.3 The Project Wizard—Create New Project tab

Some of these do require a Unity Pro license, but most of them are available in the free version. These packages are very helpful when you're starting out and can get you up and running quickly for testing simple gameplay ideas and mechanics. While we will not be using any of these to start, note that they are here and available to use in your Projects. We will revisit these packages in Chapter 2, "Understanding Asset Creation."

Setting Our Project

One of the most important skills you can have in game development is an understanding of asset management and Project structure. In Chapter 2, "Understanding Asset Creation," we will go more into the asset management side, but here we will cover the Project structure side.

Project Structure

Project structure refers to the hierarchy of files that are used in a Project, how they are named, and their ordered layering. Having a firm grasp of how you keep your Hierarchy organized and named will help you from the beginning of a Project through to the final build so that your game runs at its optimal frame rate. It also allows you to know what is needed in your deliverable game so that there are no unwanted files or folders that could bloat your game. Chapter 12, "Organization and Optimization," will go deeper into optimizing your game for best results, but understand early on that there will be limits on the file size of your game that you cannot exceed. Each device and publisher is different, and you must follow their guidelines when publishing your game.

The best way to understand the Project structure is the house analogy we made earlier. The staple of a house is its name. Some call it "home" or "pad." The point is, the name is the base, just as our Project base should have a name. When we started in the New Project window, we named it "LearningUnity." This is the name we will use throughout, and it best describes what the Project is about.

Your Unity Project is an extension of your Windows Explorer or Finder on Mac. Just as you can create folders, move files in and out, and rearrange things, you can do the same with your Unity Project. Anything you add into your Project will almost instantly be updated and appear in your Explorer Project path and vice versa. This is so important to remember; as we mentioned earlier, adding and deleting files can produce some unintentional consequences.

Note

While it may seem simple to rename or delete a file, remember that Unity has underlying data connections to the files and there may be unintended results.

Folder Organization

When you first build your Unity Project, you will see that it comes with some pre-made folders: Assets, Library, Project Setting, and a Temp folder. The Library, Project Setting, and Temp folders all contain files related to Unity and creating assets for your game. Stay clear of these and most certainly do not delete them.

As we build our Project, we will be creating an abundance of assets, such as models, sprites, scripts, and Materials. Keeping these assets in a neat and organized manner will simply make things easier as your Project continues to grow. At this point we have yet to import anything, but here is an example template for a folder structured for Unity:

```
Assets/  
  Materials/  
  Meshes/  
    Actors/  
      GoodGuy  
      BadGuy_A  
      BadGuy_B  
    Props/  
      GarbageCan_Clean  
      GarbageCan_Dirty  
  Plugins/  
  Prefabs/  
    Actors/  
      GoodGuy  
      BadGuy_A  
      BadGuy_B  
  Scenes/  
    Chapter_1/  
  Scripts/
```

File Naming Conventions

Another valuable tool to have in your workflow toolbox is an understanding of how to use namespaces. Namespaces (not the scripting ones) are a means of keeping your file names simple and short and easy to read and understand at a later point in time. Not only can it become difficult for you and others to find a file if there is no structure, but it can be just as frustrating if the files do not have a clear and concise naming convention.

For example, what if we lazily named two scripts “script_1” and “script_2”? It may be easy to remember just these two, but imagine having dozens, or several hundred, of them. We’re pretty sure that if we went to bed for the night and then reopened Unity

the next day, we would most likely forget what these two scripts contained, let alone a hundred or so of them! Having a concise naming convention in your work can help you stay focused and organized. It works just as well in 2D and 3D packages.

We have compiled a list of some of the best practices to use when dealing with namespaces inside of Unity:

1. Start with the most descriptive word, followed by an underscore. An asset named `alienShip.png` is not bad; however, `char_enemy_alienShip.png` is much clearer.
2. Folders take up very little hard drive space. Use as many as you need, for example, `Assets/_meshes/_characters/_enemy/alienShip.fbx`.
3. Try to use namespaces for linked assets. If you use `alienShip.fbx` for your mesh, try to use `alienShip.cs` for the script and `alienShip_death.anim` for its death animation. Folder management will keep them organized.
4. While we will cover asset labels more in Chapter 2, “Understanding Asset Creation,” they are definitely worth mentioning here. As a sort of internal file system, this simple tool will make locating assets quick and painless.

Note

While these recommended guidelines work for some, they may not feel right to you. Use your best judgment when setting up your Project and when naming your files. The best advice is having a plan in place from the start. As the Project grows and more files are added, understanding your workflow will be key.

From here on we will dive into Unity and game design to start creating our 2D platform game. We have a long road ahead of us, but as they say, “It’s all about the journey and not the destination.” So let’s move on to Chapter 1, “Setting Up the Unity Development Environment,” and get a look at what the power of Unity can do.

Tip

Take a look through the Unity documentation and learning resources provided on their Web site. Unity provides a bunch of their own learning videos, follow-along tutorials, and tips, and the Unity Community and Forum pages are just as valuable.

The Unity Manual is one of your greatest assets when creating games, so remember you always have this available to you as well. You can access the Unity Manual, along with links to their Community, Forum, and Answers pages, from the Help menu inside of Unity. Here are the links to the Internet Web pages for all of this information:

- Unity Documentation: <http://unity3d.com/learn/documentation>
- Unity Tutorials: <http://unity3d.com/learn/tutorials/modules>
- Unity Forums: <http://forum.unity3d.com/>
- Unity Answers: <http://answers.unity3d.com/>

This page intentionally left blank

Building the Game World

In this chapter we will begin building the game world in which our main character will move around. We will need to discuss level design and planning the game design. We will explain how having a solid idea in place will keep you from designing without reason and wasting a lot of time “winging it.”

We will then look at Unity’s Transform tools and the differences between working in 2D and 3D. We will go over the Hierarchy window and how we can use grouping and parent-child relationships to keep things organized and easy to use. Last, we will look at a few other settings to help us more easily build our Scene. By the end, we will have a full level for our character to explore. So with that, let’s get to it!

Level Design 101

Having at least a basic understanding of level design theory, and some knowledge of its rules and principles, will go a long way when you’re creating your game environments. Becoming a great level designer takes a lot of practice and patience, and professional level designers have years of experience. By no means are we even remotely in the same ballpark as them, but we do know a few basic rules that will help us create something that can be fun and entertaining to play.

Setting the Scene

One of the best things to have when putting anything together is a detailed set of instructions. Most times they list the tools you will need to do the job and describe step by step where to begin and how to get all the way through to the final piece. Creating a fun and challenging game level is no different. You will need to know what enemies to encounter, which items to collect, and what puzzles to complete. Having a “road-map” of sorts that lists all of these things will go a long way toward helping you achieve your design.

We have a few things in place to help us; we know that we are creating a 2D-side platform game, and we already have the game sprites we will use to build our levels.

But there are still a few questions we should ask ourselves before we jump into Unity and start throwing down sprites:

- What is the end goal? What are we trying to achieve other than going from point A to point B?
- Will this level be easy or hard to complete? Are the puzzles in it fairly easy or complex to solve?
- Where in the overall game does this level take place? Where are we in the timeline of the story?
- Does this level take place in the daytime or nighttime? What are the weather conditions? Is it bright sunshine or overcast and snowing?
- How has our hero progressed to this point in the game? Do they have new weapons, skills, or upgrades?
- What are some challenges or experiences the user has encountered prior to this point in the game that we may be able to build upon in this new level?

Answering these questions first will help us set some rules and standards, so that we are designing the level we feel will give the user a fun and enjoyable experience.

Creating a Roadmap

Once we have all of the answers to these questions, we can create our level. We suggest putting pen to paper, as they say, to rough out an idea of what it should look like first, before jumping into Unity. This will help us understand things like the distance over a gap, or where to place that hidden gem the player has to find.

We have gone ahead and created that roadmap (Figure 4.1), mapping out the conflicts, puzzles, and behaviors we will create. We will try to explain the theory behind the layout and how we answered those earlier questions. Again, our ways of level design may not be the best or the most effective, but we took what we know and applied it here.

The main objective of our game is getting our hero through the level to enter the door to the castle. Original idea, right? In doing so, the player has completed the puzzle and can move on to the next challenge. But there are some gameplay mechanics we will be adding along the way to make it slightly more challenging.

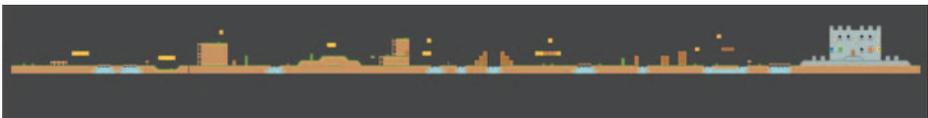


Figure 4.1 Sketched level design

When planning this, our approach was that this would be the first level encountered in the game. That will make it fairly easy for us to map out the details and complexity since this is just a demo for learning the technique. The level will take place in the daytime, it will be fairly simple to solve, and it is designed with the idea that the user has played these types of games before.

A few additions are needed to make this design unique. We tried to use the tools and assets given to us to make something a little different, and we think they work well. We are using some art that has already been created to make this a little easier for us. Now let's take a look at a few rules of level design we might want to take into consideration before fleshing out our level.

Note

Usually all game design, mechanics, and encounters have been solved before asset creation, so you or the art team is making only the art that is needed. Working out all of the design ahead of time will eliminate changes, roadblocks, and issues down the road. Everything should have been planned in advance.

- Players should start out by simply learning the mechanics of moving our hero left and right, along with performing some basic jumping. The first few game screens will be free of enemies to allow players to practice these moves.
- The first obstacle our players will encounter is designed to test their learning curve without any serious repercussions. We want them to succeed, but should they fail, the hero will not lose health or die.
- Our first contact with an enemy is a slow one that weakens our hero only slightly. Its purpose is to set up an encounter so that players learn how they will interact with enemies later on.
- There will be a few simple puzzles for the player to solve. They will use colors to help guide the player toward the solution, for instance, "Find a blue key to open a blue lock."
- Players will know they have succeeded once the castle door opens. To present this, the last puzzle will be placed within viewing distance of the door so that players can see this transition.

Adding Details

Now that we have the basic design and gameplay elements laid out, we can look for ways to liven up the world. It should be noted that this step comes well after the level has been completely designed. All significant changes should be addressed and the level should be signed off. Making things "look pretty" can wait till the end; otherwise you might end up with something that looks great but plays terribly. As the saying goes, it's "icing on the cake." Let's make sure the cake is good first.

Details can have a role in the overall feel and layout of a game and can affect gameplay just like anything else. But didn't we just say that adding details should come after the level design is done? What we mean is that the initial questions have been answered, and we have set the tone and the way we want the level to play out.

A good example of adding details that help the gameplay might be a dirt path to lead the player. Another example might be a waterfall to guide the player downward. Think of ways you can add subtle details to guide the player. We want the levels to be somewhat challenging but also enjoyable, and we want to give the player enough help to actually solve them.

Getting around Our Scene

Up to this point we haven't used the Scene View much for our Project. We did a lot of the legwork: bringing in our assets, building our sprite sheets, and creating our GameObjects. But now we really need to get in and get dirty (so to speak). We should first get comfortable with moving around in the environment. Although we are not dealing much with the concept of 3D depth, we should know how to view our GameObjects and Scene in both 2D and 3D space. Let's get familiar with Scene navigation and object manipulation by creating a test Scene where we can move around:

1. Start by creating a new Scene.
2. In the Scene View control bar, toggle the workspace from 2D to 3D.
3. Create a Cube GameObject by going to the GameObject menu > Create Other and selecting Cube.
4. Reset the Cube's Transform values all to 0. With the Cube selected, in the Inspector, right-click the gear icon to the right of the Transform Component and select Reset.

Tip

If you lose focus of a GameObject, or wish to center your view on a particular GameObject, you can reset the camera by selecting the object and tapping the F key. This will center the camera's focus point to that of the selected object. Note that this needs to be done with the mouse pointer over the Scene View.

Scene Gizmo

To begin, it helps to know how 3D space works. Take a look at the little colored Gizmo in the top right corner of the Scene View. This is the Scene Gizmo (Figure 4.2).

In 3D space, there are three different axes that determine the direction you are facing or the direction in which an object is moving. The red (X-axis), green (Y-axis), and blue (Z-axis) axes on the Scene Gizmo help clarify this. An easier way to understand this concept is that the X-axis runs left to right, the Y-axis runs up and down,



Figure 4.2 The Scene Gizmo

and the Z-axis moves front (near) to back (far). We will get a better understanding of this in just a bit when we move our cube around in 3D space.

Perspective versus Isometric

Our Main Camera works in **perspective view**. This means that we can see our cube object in our Scene with multiple converging angles (perspective viewpoints). Most likely if you haven't moved your view around, you will see two sides of the cube running off into a two-point perspective view.

Isometric view refers to the camera having equal projection of all three axes. This makes your Scene appear as though it has very little depth. We will see more of this isometric relation when we start laying down our objects using a 2D orthographic mode for the camera. Figure 4.3 shows an example of perspective and isometric cameras. Notice how the objects on the left appear to have depth and foreshortening, while the objects on the right appear flat and almost as though they exist on the same plane.

Camera Controls

There are a few different methods for getting around in 3D space inside Unity. If you come from any type of 3D background or have played a third-person-style game, you will easily grasp this concept.

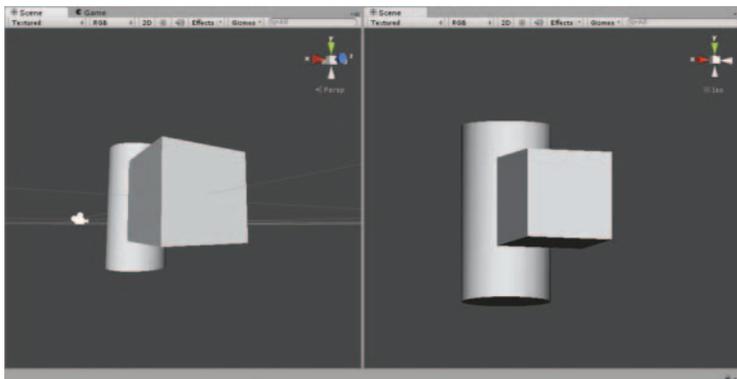


Figure 4.3 A perspective camera (left) and an isometric camera (right)

Arrow Keys

One method of moving around in the Scene View is using the arrow keys on your keyboard. If you are at all familiar with moving around in a third-person game, the arrow keys work the same way: up and down to move you forward and back, and left or right to pan the camera sideways.

WASD Movement

This is the movement that exactly replicates most PC-style games. While the arrow keys method is very similar, this one uses the W, A, S, and D keys for movement while using the mouse to direct the camera for that movement. To enable the WASD keys, you must first hold down the right mouse button.

Mouse Shortcut

The mouse movement is the most efficient method as you can still easily move about in the Scene but also keep the Transform tools for manipulating the GameObjects. While this method is most effective with a three-button mouse, you can use a two-button (no scroll wheel) or even a one-button mouse (most common for Macs or trackpad users). Table 4.1 gives shortcuts to help clarify this.

Hand Tool

Another method is to use the Hand tool (Figure 4.4). You can access it by tapping the Q key on the keyboard. In this mode, you are able to control the camera movements simply by using the mouse.

By holding down the Alt or Ctrl keys, you can orbit or zoom the camera respectively. Also, holding down Shift while using these will increase how fast the camera orbits and zooms.

Table 4.1 Mouse Movement for One, Two, or Three Button Mouse

Action	One Button	Two Button	Three Button
Move	Alt+Ctrl(Cmd) and click-drag	Alt+Ctrl(Cmd) and click-drag	Alt and middle click-drag
Orbit	Alt and click-drag	Alt and click-drag	Alt and click-drag
Zoom		Alt and right click-drag	Alt and right click-drag or use the Scroll wheel



Figure 4.4 Transform tools with the Hand tool selected

Manipulating Objects in Unity

We now have a level that we have fleshed out using paper and pen, and we can start to bring these ideas into Unity. But before we do that, let's check a few settings in Unity and make sure we are prepared to replicate our paper level design in the editor. Making sure we can “copy” things from our design to Unity in an almost one-to-one fashion will save us a lot of time and frustration.

Let's start by opening our 2D_Platform_Game Project. We have gone ahead and updated all of the sprites and made Prefabs for all of the GameObjects we will be using in our game. Go ahead and grab the Project files for Chapter 4 if you want to follow along.

Transform Tools

In order for us to lay down our GameObjects, we need to know how to move them around in Unity. We need to be able to position, rotate, and scale our objects and place them exactly where we need them. We also need to know the key differences between working in 2D and 3D and how we can still attain depth in our game.

Translate

You will mostly make use of the Translate tool when positioning your GameObject. In Figure 4.5 you can see the X-, Y-, and Z-axes just like the Scene Gizmo has. In fact, in 3D mode, you will see that the Translate Gizmo and Scene Gizmo match. This is because in world space we will move along these axes.

The Translate button can be found next to the Hand tool in the Transform toolbar. See Figure 4.4 if you do not remember this. You can also access the Translate tool by tapping the W hotkey on the keyboard.

Simply left-click and drag on one of the colored arrows, and your object will move along that axis. Moving along the direction the arrow points will move the object in a positive direction, and negative in the opposite.

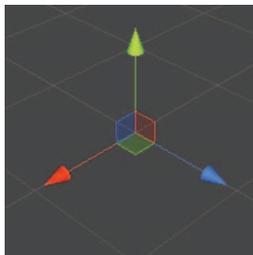


Figure 4.5 Default Translate tool display

Note

The red, green, and blue colors are used to tell us what direction that axis is facing. Red indicates the direction for the X-axis. Green is used for up and down or the Y-axis. Blue is for the Z-axis.

Rotate

Tapping the E key will bring up the Rotate tool. This will let you rotate the object along its pivot point. The colored circles again indicate the axes, but the object will rotate around an axis. This tool is very handy for setting an object at a certain angle. Figure 4.6 shows the Rotate Gizmo.

Scale

The last tool is the Scale tool (Figure 4.7). Scale means to increase or decrease the size of an object in relation to its actual size. Using the axis handles will scale the object only in that one axis. This can be handy if you want to adjust the appearance of an object, such as making a cube into a rectangle or a sphere into an ellipse.

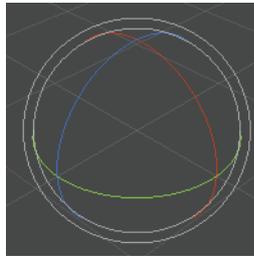


Figure 4.6 Default Rotation tool display

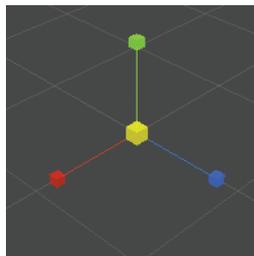


Figure 4.7 Default Scale tool display

Note

Clicking on an individual axis will affect only the object in that axis. Affecting all axes at once is different for each tool. To translate in all three axes, hold Shift and then left-click and drag from the center of the Translate Gizmo. For rotation, select anywhere inside the white circle, but not on a colored circle. With scale you simply left-click and drag within the white cube of the Scale Gizmo.

Z-Depth

Positioning our objects with the Translate tool will help us set up our level and accurately position objects in the Scene. But in a 2D setup this will only help us to place them along the X-axis (horizontal) and Y-axis (vertical). This will work for most everything, but we will want to have some depth to our levels. We want to have some dimension and balance to our world so it doesn't appear flat. Sprites like clouds moving behind our player and allowing the player to walk in front of hills will help add a touch of realism, even though it is 2D.

In a 2D game using Unity, we can still control the Z-depth using the selected sprite's Sorting Layer and Order in Layer attributes. These can be found on the Sprite Renderer of your Sprite GameObject. Figure 4.8 shows an example of this on our Player GameObject.

Sorting Layer

The most effective way we have of sorting our sprites is by using Sorting Layers. Sorting Layers work very similarly to 2D editing package layers. You separate elements into layers, and then place those layers above or behind one another to determine what draws in front of the next. The only difference with Unity is that it draws from bottom to top, with the top being drawn first, and then the next layer down over that. See Figure 4.9 for the Sorting Layers we will set up.

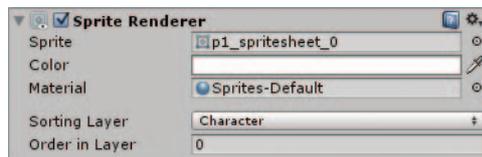


Figure 4.8 Sprite Renderer Component of our Player GameObject

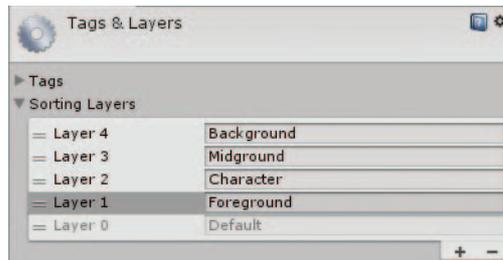


Figure 4.9 Tags & Layers window—Sorting Layers

Note

Sorting Layers are under the Tags & Layers Manager. We discussed where to find these back in Chapter 1, “Setting Up the Unity Development Environment,” but as a quick reminder, they are to the right along the toolbar. Choose the drop-down arrow and select Edit Layers to customize these.

Let’s add a couple of Sorting Layers for the Player GameObject and Scene elements we will be adding:

1. With the Tags & Layers Manager open, click the drop-down to reveal the default Sorting Layer.
2. Click the + icon to add a new layer. Rename this “Foreground.”
3. By default Unity will create new layers below the selected one. Left-click and drag over the Layer 1 text and move this above the Layer 0 slot.
4. Create the remaining three layers for Character, Midground, and Background.

Order in Layer

Another way of layering sprites is with the Order in Layer attribute in the Sprite Renderer. While we could make a bunch of Sorting Layers for each of our sprites, this can be tedious, very hard to work with, and costly for our game. You may have a sprite sheet that consists of elements that you wish to have sort with one another, but you need the entire sheet to sort with other elements. For this we use Order in Layer.

The only difference from a visual standpoint is that Order in Layer sorts based on a value. The higher the value, the later the sprite will draw, with higher values above everything else. You may also find as you create the game that you need to assign a sprite lower than 0. You can use negative values if needed.

Settings

We covered the Pixels To Units measurements for our sprites back in Chapter 3, “Creating 2D Sprites.” This ensures that the size of all of our game elements is comparable

to that of our player, so that all of the GameObjects will be in scale with one another. Once we have our worlds built, enemies placed, and props added, this will make sense, but first we have to build these things.

Another benefit of setting the Pixels to Units size is that it will uniformly align all of the tile sprites, allowing us to easily snap the ends of one tile to the next.

Grid

The grid in our Scene View will help us “snap” pieces of our levels together easily. When we set the Pixels To Units size for the tile sprites (70), it made them equal to their actual dimensions (70 pixels by 70 pixels). This means our tile will fit exactly into a 1×1 Unity grid. Now we can easily “snap” one piece to the next.

Snap Settings

The Snap Settings (Figure 4.10) work in relation to the grid units. Snap Settings allow you to position, rotate, and scale your objects with precise measurements. This tool will become invaluable when we start building our level.

- **Move X (Y and Z):** The number of units the object will move when using snapping
- **Scale:** The percentage an object will scale in size
- **Rotation:** The degree of rotation the object will make

While we do not have to adjust the Snap Settings, it is helpful to know where to locate them. They can be found under the Edit menu (Edit > Snap Settings).

Tip

By holding the Ctrl key and then left-clicking and dragging the tiles, you can easily snap them to each other with precision. This is one method that helps you build your game levels quickly. We also set the pivot point of each sprite to its bottom left, making snapping painless.

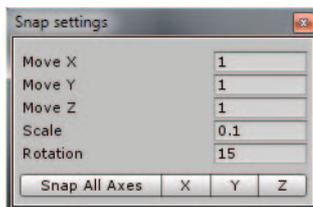


Figure 4.10 Snap Settings with default values

Our First Level

Taking what we have learned from this chapter and our level design concept, let's start building our first level! Find the Chapter 4 project files for the book, and open up the First_Level Scene file. The Project has all of the sprites we will need set up as Prefab objects.

Positioning GameObjects Manually

From the Project window, go to Assets > _prefabs and select the grassMid Prefab. Left-click and drag, then drop this into the Scene View or Hierarchy to add our first piece. Let's adjust the placement by setting the Prefab's Transforms. With the Prefab selected, go to its Transform Component in the Inspector and reset its Position values (the X-, Y-, and Z-values) to 0, 0, and 0 respectively. We have placed our first sprite! Figure 4.11 shows the Prefab with the correct placement.

Using the Snap Settings to Position GameObjects

We can also use the Snap Settings tool described previously in this chapter. This will snap the GameObject with precise values. From the Project Browser, pull in another Prefab asset to make our current ground a little more solid:

1. Find the grassCenter Prefab and drag it into the Scene View, trying to roughly place it below the grassMid Prefab.
2. Open the Snap Settings window by going to Edit > Snap Settings.
3. With the values for Move X, Move Y, and Move Z all set to 1, click the Snap All Axes button.
4. The grassCenter Prefab should now be snapped below the grassMid Prefab and sitting in world space at 0, -1, 0 in X, Y, and Z respectively.

Using Grid Snapping to Position GameObjects

Last, we can precisely position a GameObject by using the grid snap option. This takes the pivot point of the sprite as the position from which it snaps to place it exactly where you intend. You can do this with multiple selected tile sprites as well.

1. Select both the grassMid and grassCenter GameObjects.
2. Duplicate the GameObjects by tapping Ctrl + D. Notice that you now have two of each object in the Hierarchy window.
3. Select one of the grassMid and one of the grassCenter Prefabs from the Hierarchy window.

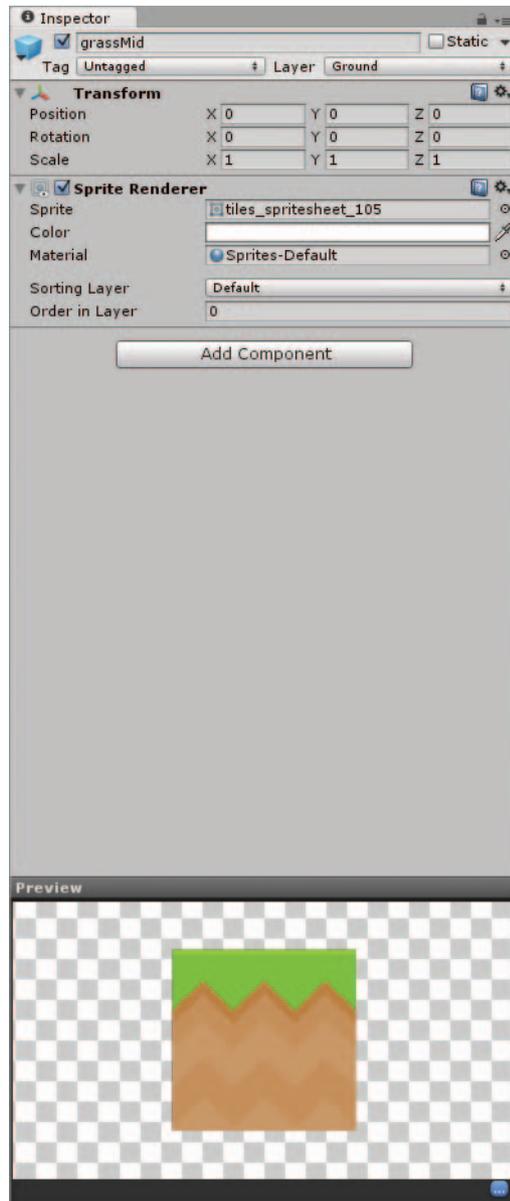


Figure 4.11 Inspector window with grassMid Prefab Component

4. Hold Ctrl and left-click and drag the tiles one snapped unit to the right in the Scene View. You will now have four sprites to make up the first part of our ground-floor tiles.
5. Continue duplicating and positioning these from left to right, until you have a complete ground surface.
6. You should have something that resembles Figure 4.12.

Note

Most often when you duplicate an object a second time, it will spawn from the original GameObject instead of the current one. It is best to deselect and then select the new GameObject before duplicating again. You could also duplicate the number of times you need and then select them individually.

Efficient Level Design

Now that we have the initial ground for our first screen, we will want to duplicate it for the next screen. While it did not take a lot of time to create the first ten, having a quicker solution would make things easier on our level designer! We could simply select

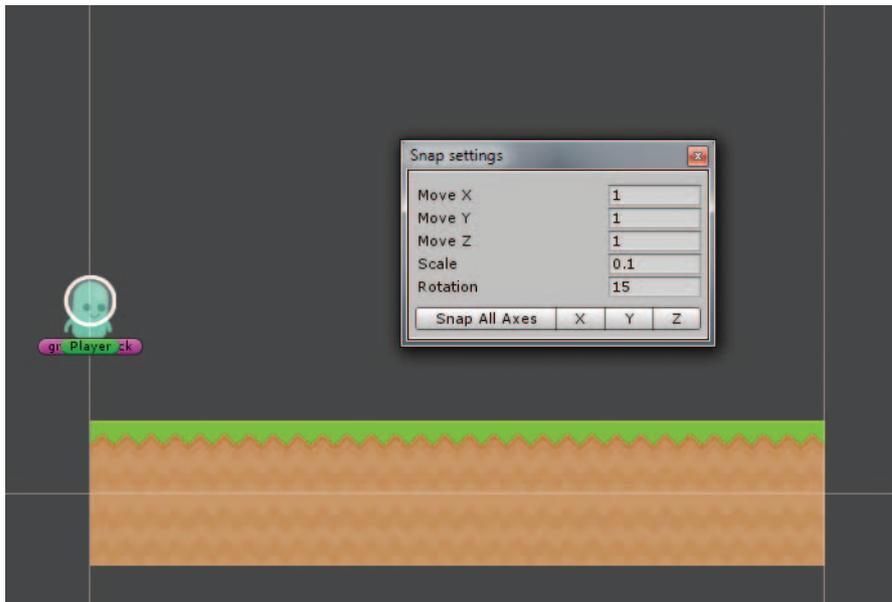


Figure 4.12 Scene View with our initial GameObjects laid out

all 20 grassMid and grassCenter GameObjects, duplicate them, and position them; let's see if we can find a more efficient way:

1. Create an empty GameObject by going to **GameObject > Create Empty**.
2. Double-click on the name or right-click and select **Rename**. Let's rename our GameObject "Screen1."
3. Reset the Screen1 GameObject by resetting its Transform values. Remember the gear icon to the right of the Transform text? Click the drop-down and select **Reset**.
4. Select all of the grass Prefabs up to this point in the Hierarchy.
5. In the Hierarchy window, left-click and drag them to the Screen1 GameObject and release. They will now be parented under Screen1.
6. With the Screen1 GameObject selected, go to **Edit > Duplicate** to create another instance of the tiles we have already placed. Rename this new one "Screen2."
7. Using our grid snapping technique, move the Screen2 group to the right 10 units. The Transform values for Screen2 should be 10, 0, 0 in X, Y, and Z.

Note

Another useful way to make sure your tiles are aligned is with the Snap Settings. Marquee (left-click and drag) a section of tiles or select them in the Hierarchy window. With the tiles selected, open the Snap Settings and click the Snap All Axes button. This will ensure that any tiles are snapped exactly to the absolute grid values. This is a very good technique for cleaning up tiles whose Transforms may have moved slightly.

Adding Sorting Elements

We covered Sorting Layers earlier in the chapter. Now let's set up a couple of instances of them and see how we can add some simple depth and details to our level. Learning how to do this early will help when you have finished the level design and want to add details, and it may give you ideas for adding your own variations.

We will first add a fence GameObject to the foreground so that our Player GameObject will walk behind it. Then we will add water and a ledge. With the ledge sorted to be in front of the water, it will appear as though there is some depth in our Scene. These details will go a long way toward giving the game life.

1. Select the fence Prefab from the `_prefabs` folder. Place this roughly at the end of the first screen, just above the ground.
2. Duplicate this and move it to the right, into the second screen. Make two more copies of this, placing them to the right.
3. Use the Snap All Axes tip from earlier to make sure these are lined up correctly. Your Scene should resemble Figure 4.13.

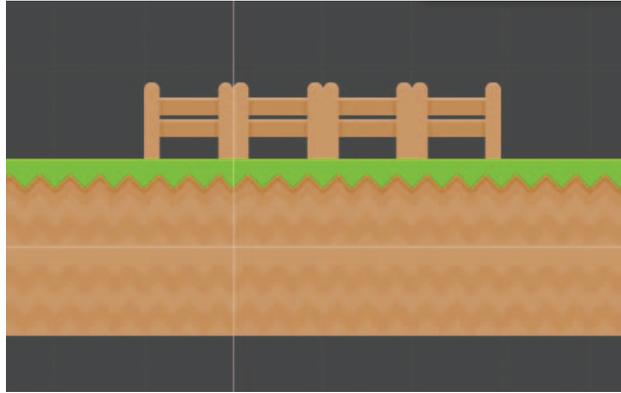


Figure 4.13 Scene View with the newly placed fence GameObjects

4. Make sure to place the first fence Prefab we added to our Scene under the Screen1 parent GameObject. Place the last three under the Screen2 version. This keeps our Hierarchy clean, and we will be able to find elements later on much more easily.

Last, we need to make sure we set the sorting for these, so our Player GameObject will walk behind them. We could do this on a per-object level, but that would take more time as well as make things less consistent. Doing this to the master Prefab in the `_prefabs` folder will instantly update any of these we have added to our Scene.

Select the fence Prefab in the `_prefabs` folder and look at its properties in the Inspector. Select the drop-down for the Sorting Layer and change it from Default to Fore-ground. Now our fence tiles will sort in front of our Player GameObject.

Now let's add the ledge and water. For this we will sort our sprites going back into the Scene. To make our Scene look slightly more realistic, we will change the end of our ground pieces to a ledge and add water:

1. Remove the last `grassMid` and `grassCenter` GameObjects from Screen2 by selecting them in the Scene View or Hierarchy and hitting the Delete key.
2. From the `_prefabs` folder again, select the `grassCliffRight` Prefab and drag it into the Scene View. Place this roughly at the end of the tiles for Screen2.
3. Select the `liquidWater` Prefab and place it directly below the ledge sprite.
4. Last, select the `liquidWaterTop_mid` Prefab and place it directly over the `grassCliffRight` GameObject.
5. Let's set the Sorting Layer for the water so it draws behind the ledge sprite. Again, let's do this in the Prefab so that any instance of the water will automatically sort correctly. Select the water Prefabs and in the Inspector, set the sorting to be Background.

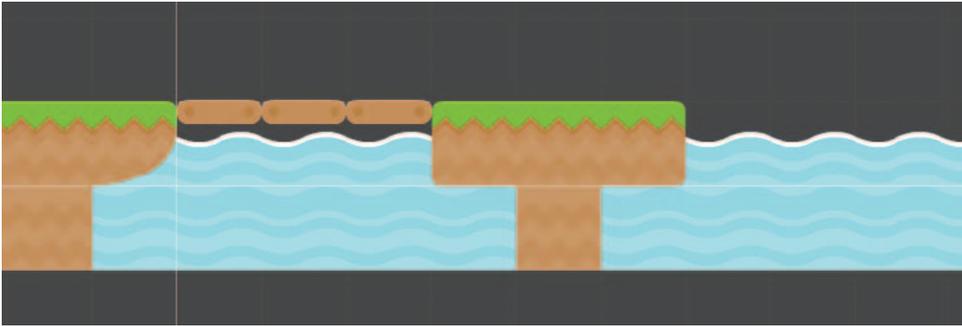


Figure 4.14 Our ledge and water GameObjects placed

6. Finally, we need to make sure the GameObjects are positioned precisely. Select the `grassCliffRight` and the two water GameObjects and hit the Snap All Axes button again.
7. Your Scene should resemble Figure 4.14. We have gone ahead and added a few additional sprites for `Screen3`, including an island and a bridge, as well as finished out the water. Remember to use your `Sorting Layer` and `Order in Layer` attributes to set the priority of the sprites. See the `First_Level_Final` Scene file if you get stuck.

Continuing On

Now that we have a basic understanding of adding Prefabs to our tiles, positioning and snapping them, and sorting our sprites, we can continue. We leave you here to carry on and finish the rest of the level design. You are free to use the design we laid out at the beginning of the chapter or to make your own. Most of what we will be adding in the next few chapters can be done with any variation of the design, but we will show you a few select GameObjects and enemies we have chosen.

Feel free to finish this one and keep this chapter handy should you need to review any information. We have also included the final level completely laid out in the Project files for this chapter. This Unity Scene file is in the `Chapter 4_projectFiles > _scenes` folder and is named `First_Level_Final`. If you run into any roadblocks, check the Inspector and Hierarchy windows.

Summary

In this chapter we learned a great deal of information on designing the first part of our level. We had a brief overview of level design and reviewed a few simple principles to make our level slightly more enjoyable.

After this we dug into Unity and using the Scene View and camera controls to navigate our Scene. Using a variety of methods, you should now be able to easily move around to view and place your GameObjects in world space.

We went into great detail on the Unity Transforms and Transform Components. This is how we will position, rotate, and scale the elements of our game. We went into 2D projections and Z-depth and how we will use sorting and ordering to added depth and priority to our sprites in the game.

Finally, we started building our first level, adding Prefabs, and setting up sorting and some organization for our Scene. We set a few attributes in the master Prefab and saw how this will directly update all instances in our Scene. From here you are left to carry on and finish the design of your level.

In Chapter 6, “Adding Animations to Our Scene,” we will start bringing our characters and Scene elements to life. We will learn how to add a series of complex behaviors and how we can make things move realistically in a short amount of time and with little effort. With that, let’s get moving!

Exercises

1. Finish up the Scene, adding elements such as grass, clouds, and props. Remember to set the position with the Snap Settings tool, and set the priorities for the sprites.
2. Set the Sorting Layers in the master Prefabs themselves in the `_prefabs` folder. This will ensure that all instances we add to our Scene carry them over.
3. Let’s add some simple collision for our ground GameObjects. Select `grassMid` and `grassCliffRight` in the `_prefabs` folder.
4. In the Inspector, click the Add Component button and select `Physics 2D > Box Collider`. This will add a Collider Component for our ground sprites that our Player GameObject will then be able to interact with.
5. The colliders are added, but centered to the pivot of our GameObjects. We need to have the collision cover the bounds of the sprite. Make sure that `grassMid` and `grassCliffRight` are still selected from the `_prefabs` folder.
6. Under the Box Collider Component, find the attribute for `Center` and set its values in `X` and `Y` to 0.5. This will move the collision to center exactly around our sprites.
7. Don’t forget to save your Scene!

Index

Numbers

2D

- 2D animation vs., 96–97
- 2D physics vs., 120–121
- Audio Source component properties, 230
- file formats, 28
- Scene View toggle, 24
- sprites. *See* Sprites, 2D

3D

- 3D animation vs., 96–97
 - 3D physics vs., 120–121
 - Audio Source component properties, 229–230
 - Canvas Component controlling GUI display, 294–296
 - file formats, 27–28
 - rendering 2D spriten, 98
 - Scene Gizmo in, 56–57
 - Scene View toggle, 24
 - transform animation in, 97
 - Unity engine default as, 41
- 3ds Max file format, 27–28
- 6DoF (six degrees of freedom), 120–121
- 16 bits format, 45

A

- A1 routine Components, 162
- About Unity command, Help menu, 18
- Add command, Component menu, 16
- Add Component button, 37–38
- Add Current button, Build Settings, 282
- Add Curve button, Animation window, 102, 105–106
- Add Event button, Animation Events, 102, 107
- Add Keyframe Target, Animation window, 102
- Add-ons, component installs, 2–3

- Adobe Flash, sprite packing, 50
- Adobe Photoshop, sprite packing, 49
- Air control, character movement, 73–74
- Align View to Selected command,
 - GameObject menu, 16
- Align with View command, GameObject menu, 16
- Anchors
 - UGUI Rect Transform, 297–298
 - UI Rect Tool, 299
- Angry Bots example project, 3
- .anim format, 99
- Animation Clips, 100–101
- Animation Component, 97, 99–100
- Animation Events, 107
- Animation Layer, 111
- Animation State Machine, 108, 115–116
- Animation tree, Animator window, 110
- Animation window
 - creating Animation Events, 107
 - creating animations inside Unity, 101
 - creating player walk animation, 105–106
 - editing Animation Clips, 100–101
 - Graph Editor, 102–103
- Animations
 - 2D vs. 3D, 96–97
 - bump, 154–155
 - Fly enemy, 184–185
 - imported, 98–99
 - overview of, 95
 - preparing victory trigger prefab, 270
 - principles of, 95–96
 - scripted, 98
 - Slime enemy, 165–166
 - transform vs. frame, 97–98
 - viewing number in scene, 267

- Animations, creating
 - with Animation Component, 100
 - with Animation State Machine, 108, 115–116
 - with Animation window, 101–104
 - with Animator Component, 109
 - with Animator Controller, 108–109
 - with Animator Window, 110–112
 - assigning Animation Clip, 100–101
 - assigning Animation Events, 107–108
 - with Dope Sheet, 104–105
 - with Graph Editor, 102–104
 - overview of, 99
 - player's idle and walk, 105–107
 - review exercises, 117
 - updating PlayerController to view, 112–115
- Animator Component
 - adding temporary immunity post-damage, 174
 - creating bump animation, 154
 - creating VictoryTrigger script, 274
 - editing PlayerController, 112
 - overview of, 109
- Animator Controller
 - adding animation to Slime enemy, 165–166
 - creating bump animation, 154–155
 - editing in Animator window, 110–112
 - overview of, 108
 - for player character, 109
 - viewing animation in real-time, 105–106
- Animator window, 110–112, 115–116
- Anticipation, in animation, 95
- Any State, animations, 115–116
- API Compatibility Level, Web Player Player Settings, 283
- Appeal, in animation, 95
- `Application.LoadLevel()` function, `SplashScreenDelayed` script, 277
- Apply Changes to Prefab, GameObject, 15
- Apply Root Motion, Animator Component, 109
- Arcs, in animation, 96
- Arrays
 - scripting CoinSpawner Component, 152–154
 - scripting HUD, 208
 - scripting PitTrigger Component, 143
- Arrow keys, camera movement, 58
- Aspect drop-down, Game View, 24
- Assets
 - batching for optimization, 266
 - file naming conventions, 7–8
 - folder organization of, 7, 247–248
 - using as Components in game, 36–39
 - using labels to earmark individual, 249
- Assets, creating
 - 2D file formats, 28
 - 3D file formats, 27–28
 - from Assets menu, 29–30
 - components, 36–39
 - GameObjects, 33–36
 - importing from inside Unity, 28–29
 - importing packages, 31–33
 - importing premade, 29
 - overview of, 27
 - review exercises, 39
- Assets menu, 13–14, 29–30
- Attach to Process window, breakpoints, 91–92
- Audio command, Component menu, 16
- Audio effects
 - adding footsteps to walk cycle, 231–233
 - adding to collectible system, 236–238
 - adding to damage event, 234–236
 - adding to jump event, 233–234
 - recording audio at runtime, 228
 - Unity's audio system, 227–231
- Audio Listener Component
 - defined, 37, 227
 - overview of, 230
- Audio Reverb Zone Component, 227, 230–231
- Audio Source Component
 - 2D sound settings, 230
 - 3D sound settings, 229–230
 - adding footsteps to walk cycle, 231–233

- adding sound to collectible system, 237–238
- adding sound to damage event, 235
- adding sound to jump event, 234
- defined, 228
- properties, 228–229
- Audio toggle, Scene View, 24
- audioSource property, 232
- Authors, about this book's, xxvii
- Automatic Layout, 197
- Automatic Slicing, 46–47
- Avatar, Animator Component, 109
- Awake () function
 - adding footsteps to walk cycle, 231–232
 - CameraFollow script, 239–240
 - editing PlayerController script, 112
- Axes
 - 2D vs. 3D animation, 96–97
 - 2D vs. 3D physics, 120–121
 - CameraFollow script, 241
 - creating bump animation, 154–155
 - navigating with Scene Gizmo, 56–57
 - particle system curves, 219–220
 - positioning GameObject with Transform tools, 59–61
 - sizing Checkpoint trigger, 138–139
 - X. *See* X-axis
 - Y. *See* Y-axis
 - Z. *See* Z-axis, in 3D
- B**
- Background image, UI, 294–295
- Batching
 - optimization with, 266
 - viewing draw cells reduced due to, 267
 - Web Player Player Settings, 283
- Behaviors
 - 2D, 41–42
 - particle system properties, 217–218
 - unexpected, 91
- Best practices, file naming conventions, 8
- Blend Trees, animation, 116
- Block_Bump animation, 155
- Boo, supported by Unity3D, 71–72
- Bounciness, Physics 2D Materials, 128–129

- Box Collider 2D
 - adding walls to level, 164
 - defined, 126
 - preparing Slime enemy GameObject, 160
 - preparing spawn trigger, 188
 - preparing victory trigger prefab, 270
- Break Prefab Instance, GameObject menu, 15
- Breakpoints, in code execution, 90–91
- Bringing it all together
 - building and deploying. *See* Building and deploying game
 - hooking up Intro screens, 275–277
 - levels. *See* Levels, tying together
 - monetization, 287–290
 - polishing considerations, 286–287
 - recovering from Game Over, 278–279
 - starting over from win, 279–281
 - summary, 290
- Build & Run command, File menu, 12
- Build and Run button, Build Settings, 282
- Build button, Build Settings, 282
- Build Settings window
 - buttons in, 281–282
 - cleaning up player death, 244
 - cross-platform players, 283
 - deploying game to Web, 285
 - in File menu, 11
 - hooking up Intro screens, 275–277
 - PC, Mac, and Linux, 282–283
 - post-deployment, 286
 - preparing victory trigger Prefab, 271
 - Web Player Build Settings, 282
 - Web Player Player Settings, 283–285
- Building and deploying game
 - building game for Web player, 285
 - cross-platform Player Settings, 283
 - deploying game to Web, 285
 - overview of, 281–282
 - PC, Mac, and Linux standalone Build Settings, 282–283
 - post-deployment, 286
 - right settings for the job, 284–285
 - Web player's Build Settings, 282
 - Web player's Player Settings, 283–284
- Bump animation, 154–155

- Buttons, GUI
 - defined, 195
 - Game Over screen, 201–202
 - Game Win screen, 202–203
- Buttons, UGUI, 291
- C**
- C# programming language, 71–72
- Camera
 - arrow keys, 58
 - cleaning up, 238–241
 - controls, 57–58
 - creating GameObject, 33–34
 - defined, 37
 - Hand tool, 58
 - managing Scene with multiple, 204
 - mouse shortcut, 58
 - perspective vs. isometric view, 57
 - resetting focus point to selected object, 56
 - setting up basic follow-cam, 83
 - WASD movement, 58
- CameraFollow script
 - cleaning up camera, 238–241
 - JavaScript sample code for, 311–312
- Canvas Component, UGUI
 - defined, 292
 - example interface, 293
 - overview of, 293–294
 - in Rect Transform, 296
 - in Render mode, 294–296
- Cascading Style Sheet (CSS), native GUI, 195
- Catch block, error handling, 87–89, 138
- Center on Children command, GameObject menu, 15
- Center tool, Transform Gizmo Toggles, 19
- Challenge, levels of, 190–191
- Character acceleration, 72
- Check for Updates command, Help menu, 18
- Checkpoints
 - creating, 136–140
 - as critical path material, 140
 - polishing game, 286
 - using with respawn, 140–144
- CheckpointTrigger script
 - JavaScript sample code for, 312
 - scripting Checkpoint Component, 137–138
- CheckXMargin() function, CameraFollow script, 240
- CheckYMargin() function, CameraFollow script, 240
- Choose Components screen, installing Unity, 2–3
- Circle Collider 2D
 - creating damage trigger, 169
 - creating for GameObject, 126–127
 - defined, 126
 - preparing Fly enemy GameObject, 183
 - preparing Slime enemy GameObject, 160
- Classes, GUI, 197
- cleaning up camera, 238–241
- Clear Parent command, GameObject menu, 15, 21–23
- Code
 - calling damage particle system from, 225–227
 - error handling. *See* Error handling and debugging
 - methods of shorthand, 82
 - in Unity3D, 71–72
- Code, organization of
 - optimizing, 254
 - shorthand code, 261
 - using concise functions, 261
 - using constants, 256–261
 - using regions, 254–256
- Code samples, JavaScript
 - collectible scripts, 307–309
 - enemy scripts, 309–310
 - game system scripts, 310–314
 - GUI scripts, 314–320
 - hazard scripts, 320–321
 - overview of, 301
 - player scripts, 301–307
 - system scripts, 322
- Coin box
 - adding sound to, 236–237
 - bump animation for, 154–155
 - defined, 144
 - hooking up particle effect, 223
 - particle effect for, 220–222
 - preparing prefabs for collection, 147–150
- Coin value
 - defined, 144

- for finished coin prefabs, 146
 - retrieving, 274–275
- CoinBox script
 - JavaScript sample code for, 307–308
 - overview of, 150–151
- CoinCounter script
 - creating, 206–207
 - hooking up to Player GameObject, 210–212
 - JavaScript sample code for, 314–315
- CoinPickup script
 - adding sound to coin collection, 237–238
 - JavaScript sample code for, 308
 - overview of, 145–146
 - tracking player stats, 156–157
- CoinPop() function, Coinbox script, 151
- Coins. *See* Collectibles
- coinsCollected variable
 - creating VictoryTrigger script, 273
 - PlayerStats script, 156, 168, 211, 304
 - retrieving coin value, 274–275
- CoinSpawner Component, 152–154
- CoinSpawner script
 - creating, 152–153
 - disabling coin box prefab, 148
 - JavaScript sample code for, 308–309
- CollectCoin() function, 156–157
- Collectibles
 - adding sound to, 236–238
 - adding touch of polish to, 154–155
 - coin box prefabs, 147–150
 - CoinBox script, 150–151
 - CoinPickup script, 145–146
 - CoinSpawner script, 152–153
 - floating coin prefabs, 145
 - hooking it all together, 153–154
 - overview of, 144–145
 - popped coin prefabs, 147
 - script samples for, 307–309
 - tracking player stats, 155–157
- Colliders
 - adding triggers to GameObjects, 136
 - creating checkpoints, 136–140
 - handling collision with other Slimes, 165–166
 - Physics 2D, 125–128
 - reducing triangle count with, 266
 - Trigger2D functions raised by, 135–136
- Collision Detection, optimizing Rigidbody
 - attributes, 263
- Collision module, particle system, 219
- Color by Speed module, particle system, 218
- Color Over Lifetime module, particle system, 218, 224
- Compatibility, Web Player Player
 - Settings, 283
- Component menu, 16
- Components
 - adding to GameObject, 37
 - assigning, 37–39
 - creating GameObject, 33–34
 - overview of, 36–37
 - persisting data between Scenes/game sessions, 157
 - UGUI, 291–293
- Compound Controls, GUI, 197
- Compressed format, 44
- Console
 - filtering contents of, 89
 - using Debug.Log() and spamming, 90
- Constants
 - naming conventions for, 258
 - overview of, 256
 - using shorthand code, 261
- Constants script
 - creating, 256–258
 - JavaScript sample code for, 322
 - preparing victory trigger prefab, 270–272
 - updating PlayerController script, 258–261
- Constraints, Physics 2D, 129–134
- ContactDamage script, 169–170, 320
- Containers, organizing Hierarchy into, 249–252
- Control bars, 23–24
- Controls, GUI, 195–197
- Conventions, used in this book, xxiii–xxiv
- Copy/paste, importing premade assets, 29
- Coroutines, 175, 200
- Create command
 - Assets menu, 13
 - GameObjects, 23
- Create Empty command, GameObject menu, 15, 34–35
- Create Other command, GameObject
 - menu, 15

- Critical paths, 140
- Cross-platform Build Settings, 283
- CSS (Cascading Style Sheet), native GUI, 195
- Culling Mode, Animator Component, 109
- Curves, particle system, 219–221
- D**
- Damage, dealing
 - adding sound to event, 234–236
 - adding temporary immunity post-damage, 172–175
 - calling particle system from code, 225–227
 - creating damage trigger, 168–170
 - creating particle effect for, 223–225
 - handling player death, 177–178
 - overview of, 167
 - passing through player space, 170–171
 - to the pits, 172
 - representing immunity, 175–177
 - updating PlayerStats script, 167–168
- damage property, ContactDamage, 170
- Data, persisting between Scenes/game sessions, 157
- Day 1 DLC, monetization purchase model, 288
- Death, player
 - cleaning up, 241–245
 - handling, 177–178
- deathTimeElapsed variable, cleaning up player death, 243
- Debugging. *See* Error handling and debugging
- Debug.Log() function, 89–90
- Debug.LogError() function, 88, 90–91
- Default Behavior Mode, 2D workspace, 42
- delay property, VictoryTrigger script, 273–274
- Delete command
 - Assets menu, 14
 - Edit menu, 12
 - keyframes, 103–104
- Deployment of game to Web, 285–286
- Details, adding in level design, 55–56
- Development environment setup
 - overview of, 9
 - Unity interface for. *See* Unity interface
 - Welcome screen, 9–10
- Diegetic user interfaces, 194
- Difficulty. *See* Hazards and difficulty
- Direct3D, Web Player Player Settings, 283
- directionAB property, FlightPoints, 180–182
- Discrete tile movement, 72
- Distance Joint 2D constraint, 130
- DLC (downloadable content), 287–288
- Document Editor, MonoDevelop, 75–76
- Document Outline panel, MonoDevelop, 76
- Documentation, Unity, 8
- Dope Sheet, animating, 104–105
- Downloadable content (DLC), 288
- Downloading Unity, 1–2
- Drag and drop
 - adding Component to GameObject, 37
 - adding Rigidbody to give character, 124
 - existing Prefabs, 38–39
- Draw calls
 - optimizing, 264–265
 - viewing number to process before batching, 267
- Draw mode, Scene View, 23
- Duplicate command, Edit menu
 - defined, 12
 - duplicating GameObjects, 64–66
 - efficient level design, 66–67
- Duration property, particle system
 - defined, 217
 - particle effect for coin boxes, 220
 - particle effect for damage, 224
- Dynamic Batching, 283
- Dynamics, creating, 130–134
- E**
- Edge Collider, 126
- Edit menu, 12–13
- Editing
 - Animator Controller, 110–112
 - particles, 216
 - PlayerController, 112–115
 - tangents, 103–104
- Effects
 - adding footsteps to walk cycle, 231–233
 - adding sound to collectible system, 236–238
 - adding sound to damage event, 234–236
 - adding sound to jump event, 233–234

- cleaning up camera, 238–241
- cleaning up player death, 241–245
- overview of, 215
- particle. *See* Shuriken Particle System
- review exercises, 245
- Unity’s audio system and, 227–231
- Effects command, Component menu, 16
- Effects drop-down, Scene View, 24
- Emission module, particle system, 218, 221, 224
- Enemies
 - creating more types of, 286
 - JavaScript sample codes for, 309–310
 - maintaining arrangements, 187–189
- Enemy, creating first. *See also* Damage, dealing
 - adding animation to Slime, 166–167
 - adding wall to level, 164–165
 - EnemyController script and inheritance, 161–162
 - EnemySlime script, 162–164
 - handling collision with other Slimes, 165–166
 - overview of, 159
 - preparing Slime enemy GameObject, 159–161
- Enemy, creating second
 - adding animation to enemyFly, 184–185
 - adjusting FlightPoints script, 185–187
 - FlyController script, 185
 - overview of, 182–183
 - preparing enemyFly GameObject, 183–184
- EnemyController script, 161–162, 309
- enemyFly. *See* Enemy, creating second
- EnemySlime Component, 165–166
- EnemySlime script, 162–164, 309–310
- Error handling and debugging
 - 2D sprite animations, 101
 - halting code execution with breakpoints, 90–93
 - handling exceptions, 85–86
 - overview of, 85
 - using `Debug.Log()`, 89–90
 - using Try-Catch-Finally, 87–89
 - your own code, 138
- Event System, UGUI, 293

- Events
 - adding sound to damage, 234–236
 - adding sound to jump, 233–234
 - player walk animation, 232–233
 - trigger, 135–136
- Exaggeration, in animation, 96
- Examples
 - JavaScript code. *See* JavaScript code samples
 - projects, 3
- Exception handling. *See* Error handling and debugging
- Export Package command, Assets menu, 14
- External Forces module, particle system, 219

F

- F2P (Free to Play), monetization purchase model, 287–289
- Facing, handling player, 79–80
- FBX file format
 - 3D, 27–28
 - importing animations, 99
- File browser, importing premade assets from, 29
- File menu, 11–12
- Files
 - 2D format, 28
 - 3D formats, 27–28
 - audio formats, 227
 - naming conventions, 7–8
 - organizing scripts via filename, 253–254
 - setting project structure, 6
- Filtering Console contents, 89
- Finally block, handling exceptions, 87–89
- Find command, Edit menu, 13
- Find References in Scene command, Assets menu, 14
- First Streamed Level, Web Player Player Settings, 283
- Fixed Layout, 197–198
- Fixed Timestep, Time Manager, 263
- `FixedUpdate()` function
 - adding Try-Catch-Finally to, 87–88
 - for animation, 113–115
 - for basic jumping, 80–82
 - for basic lateral movement, 79
 - CameraFollow script, 240

- FixedUpdate() function (*continued*)
 - defined, 77
 - EnemySlime Component script, 164
 - FlightPoints script, 180–182, 186
 - setting player facing, 79–80
 - Time Manager handling, 262
 - Flare Layer Component, 37
 - flickerDuration property, PlayerStats, 176–177
 - flickerTime property, PlayerStats, 176
 - FlightPoints script
 - adjusting for Fly enemy, 185–186
 - JavaScript sample code for, 312–313
 - moving platforms at set speed, 180–182
 - Flip() function
 - EnemyController script, 161–162
 - EnemySlime script, 165
 - FlightPoints script, 186
 - FlyController script, 185
 - setting player facing, 80
 - Floating coins, 144–145
 - FlyController script, 185, 310
 - Flying enemy. *See* Enemy, creating second
 - Fly_Move animation, 184–185
 - Folders
 - organizing assets into, 247–248
 - organizing for project, 7
 - organizing prefabs into, 248–249
 - organizing script files into, 253–254
 - Follow-cam, setting up, 83
 - Follow-through action, animation, 96
 - Footsteps, sound of player, 231–233
 - footstepSounds property, 232
 - Force
 - physics and, 120
 - Rigidbody behavior as, 124
 - Force Over Lifetime module, particle system, 218
 - Formats
 - 2D file, 28
 - 2D sprite, 44–45
 - 3D file, 27–28
 - audio, 227
 - FPS (frames per second), viewing, 267
 - Frame animation
 - overview of, 97
 - using transform animation with, 97
 - Frame, Animation window, 102
 - Frame Selected command, Edit menu, 12
 - Free to Play (F2P), monetization purchase model, 287–289
 - Friction attribute, Physics 2D Materials, 128–129
 - Full air control system, 73
- ## G
- Game Over screen
 - creating, 201–202
 - recovering from Game Over, 278–279
 - Game system scripts
 - CameraFollow, 311–312
 - CheckpointTrigger, 312
 - FlightPoints, 312–313
 - overview of, 310
 - SpawnTrigger, 313
 - VictoryTrigger, 313–314
 - Game View
 - animation in, 107
 - overview of, 24–25
 - Render mode displaying GUI in, 294–296
 - Render Statistics window in, 266
 - toolbar controls, 20
 - Game Win screen
 - creating, 202–203
 - starting over from win, 279–280
 - Game world, building
 - Grid Settings, 63
 - level design theory, 53–56
 - objects with Pixels to Units, 62–63
 - objects with Transform tools, 59–61
 - objects with Z-depth, 61–62
 - overview of, 53
 - review exercises, 69–70
 - Scene navigation, 56–58
 - Snap Settings, 63–64
 - Game world, building first level
 - continuing on, 69
 - efficient level design, 66–67
 - positioning GameObjects, 64–66
 - sorting elements, 67–69
 - GameObject menu, 15–16, 34–37
 - GameObjects
 - adding trigger Components to, 136
 - Components as “guts” of, 36–39

- creating, 34–37
- creating HUD scripts, 206–211
- creating HUD visuals, 204–205
- creating in Scene View, 23
- creating particle system, 216–217
- Inspector giving detailed description of, 20
- losing focus of/centering view on, 56
- parenting with Hierarchy, 20
- UGUI, 291–293
- understanding, 33–34
- gameObjects property, spawnTrigger, 189
- GameOverScript script
 - GameOver screen, 201–202
 - JavaScript sample code for, 316
 - recovering from Game Over, 278–279
- Gameplay systems
 - adding triggers to GameObjects, 136
 - creating checkpoints, 136–140
 - creating collectibles. *See* Collectibles
 - overview of, 135
 - review exercises, 158
 - tracking player stats, 155–157
 - triggers and, 135–136
 - using checkpoints with respawn, 140–144
- GameWinScript script
 - Game Win screen, 203
 - JavaScript sample code for, 317
 - starting over from win, 280
- GetNearestActiveCheckpoint()
 - function, 143–144
- Gizmos drop-down, Scene View, 56–57
- Gizmos toggle
 - Game View, 25
 - Scene View, 24
- GPU Skinning, Web Player Player Settings, 283
- Graph Editor, 102–104
- Graphics Emulation command, Edit menu, 13
- Gravity
 - adding Rigidbody for interaction with, 124–125
 - physics and, 120
- Gravity Multiplier property, particle system
 - coin boxes, 220
 - damage, 224
 - defined, 218

- Grid Slicing, Sprite Editor, 47–48
- Grid snapping
 - positioning GameObjects, 64–66
 - setting Pixels to Units for tile sprites, 63
- groundCheck property, PlayerController, 77–78, 81
- groundCheckRadius property, PlayerController, 77
- groundLayers property, PlayerController, 77–78
- GUI
 - Layer Component, 37
 - UGUI. *See* UGUI (Unity GUI)
- GUI scripts
 - CoinCounter, 314–315
 - GameOverScript, 316
 - GameWinScript, 317
 - GUIGame, 318–319
 - overview of, 314
 - SplashScreenDelayed, 319
 - TitleScreenScript, 319–320
- GUIGame script, 209–210, 318–319

H

- Hand tool, 19, 58
- Hazards and difficulty
 - creating first enemy. *See* Enemy, creating first
 - creating second enemy, 182–187
 - dealing damage to player. *See* Damage, dealing
 - expanding on platforming, 178–182
 - handling player death, 177–178
 - JavaScript code samples for, 320–321
 - maintaining enemy arrangements, 187–190
 - more types of hazards, 286
 - review exercises, 191
 - understanding challenge, 190–191
- Heads-up display. *See* HUD (heads-up display)
- Health, player
 - carrying forward from level to level, 286
 - creating HUD script to update, 208–211
 - hooking up to Player GameObject, 210–212
 - representing with heart icon, 205

- Heart GUI setup
 - HUD visuals for player health, 204–205
 - updating player health in real time, 208–211
 - Heart icon, for player health, 205
 - Help menu, 17–19
 - Henley, James A., xxv
 - Hierarchy
 - creating GameObject, 33–35
 - efficient level design, 67
 - Hierarchy list, 20
 - organization of, 250–252
 - positioning GameObjects with Snap Settings, 64–66
 - setting project structure, 6
 - Hinge Joint 2D constraint, 130–134
 - Horizontal Scroll Bar. GUI, 196
 - Horizontal Slider, GUI, 196
 - HUD (heads-up display)
 - creating scripts, 206–211
 - creating visuals, 204–205
 - defined, 204
- I**
- Idle state, player's, 115–116
 - `#if UNITY_STANDALONE` tag, recovering from Game Over, 279
 - Image Component, UGUI
 - adding mini-map graphic, 295–296
 - creating background image, 294–295
 - defined, 292
 - sizing with UI Rect Tool, 298–299
 - Immunity
 - adding temporary player, 172–175
 - visually representing, 175–177
 - ImmunityDuration property, PlayerStats, 173–174
 - immunityTime property, PlayerStats, 173–174
 - Import New Asset command, Assets menu, 14, 28–29
 - Import Package command, Assets menu, 14
 - Importing
 - 2D sprites, 43–45
 - animations, 98–99
 - assets from inside Unity, 28–29
 - packages for asset creation, 31–33
 - premade assets from file browser, 29
 - Inherent Velocity property, particle systems, 218
 - Inheritance
 - EnemyController script, 161
 - EnemySlime script, 162
 - Input Manager, 83–85
 - InputField Component, UGUI, 292
 - Inspector
 - building 2D sprites, 43
 - creating GameObject, 33–35
 - hiding public variables in, 77
 - overview of, 20–21
 - viewing animation in, 107
 - Installation
 - component installs, 2–3
 - downloading Unity for, 1–2
 - Interface elements. *See* UGUI (Unity GUI); UI (user interface)
 - Interpolate, optimizing Rigidbody attributes, 263
 - Intro screens, hooking up, 275–276
 - Introduction to Unity
 - component installs, 2–3
 - downloading and installing, 1–2
 - example project, 3
 - file naming conventions, 7–8
 - folder organization, 7
 - MonoDevelop, 3–4
 - overview of, 1
 - project structure, 6
 - Project Wizard, 4–6
 - Unity Development Web Player, 3
 - Is Trigger attribute, colliders, 125–126
 - isDead variable, player death, 243
 - isFacingRight property
 - EnemyController, 162
 - PlayerController, 77
 - PlayHitReaction() function, 226
 - isImmune property, PlayerStats, 173–174
 - Isometric view, camera, 57
 - isTrigger property, floating coin Prefabs, 145
 - isTriggered property
 - CheckpointTrigger script, 137–138, 312
 - PitTrigger script, 142, 321
 - SpawnTrigger script, 189, 313
 - VictoryTrigger script, 272–274, 313–314

J

- JavaScript code samples
 - collectible scripts, 307–309
 - enemy scripts, 309–310
 - game system scripts, 310–314
 - GUI scripts, 314–320
 - hazard scripts, 320–321
 - overview of, 301
 - player scripts, 301–307
 - system scripts, 322
- JavaScript, supported by Unity3D, 71–72
- Johnson, Matthew, xxv
- Jumping
 - adding sound to player, 233–234
 - air control systems for, 72–73
 - implementing basic, 80–82
 - JumpForce property and, 77–78

K

- Keyed frames, Graph Editor, 102–103
- Keyframes
 - adding animation to Fly enemy, 184
 - animation using, 102
 - recorded for current animation, 105

L

- Labels
 - asset naming conventions, 8
 - GUI, 195
 - organization of, 249
- Lateral movement, implementing, 79
- Layer Collision Matrix box
 - overview of, 123–124
 - passing through player's space, 171
 - in Physics 2D Settings, 122
- Layers and Layout drop-downs, toolbar, 20
- Layers, toolbar, 20
- Layout drop-down window, 17, 20
- Layouts, GUI, 197–198
- Level design
 - adding details, 55–56
 - adding timers, 286
 - carrying health forward from level to level, 286
 - creating roadmap, 54–55
 - efficient, 66–67

- overview of, 53

- rules of, 55
- setting scene, 53–54

Levels, tying together

- creating victory trigger script, 272–274
- overview of, 269
- preparing victory trigger prefab, 269–272
- retrieving coin value, 274–275

Licenses

- installing Unity and, 2
- managing in Help, 18
- platforms requiring, 282

Lighting toggle, Scene View, 24

Limit Velocity Over Lifetime module, particle system, 218

Linked assets, namespaces for, 8

Linux, standalone Build Settings, 282–283

Local tool, Transform Gizmo Toggles, 20

Lock View to Selected command, Edit menu, 12

Looping property, particle system

- coin boxes, 220
- damage, 224
- defined, 217

Losing game

- creating Game Over screen, 201–202
- recovering from, 278–279

M

Mac, standalone Build Settings, 282–283

Main Camera

- cleaning up, 238–239
- Components, 36–37
- as first Game Object, 33–34
- follow-cam setup, 83
- perspective vs. isometric view, 57
- properties shown in Inspector, 21–22
- title screen with, 200
- TrackPlayer() function and, 241
- UGUI Render mode settings for, 294

Make Parent command, GameObject, 15

Manage License command, Help menu, 18

Manual Slicing, Sprite Editor, 45–46

Map graphics, 295–296

Mask Component, UGUI, 299

Mask GameObject, UGUI, 296, 298–299

- Mass
 - adding Rigidbody for character, 124–125
 - physics and, 119–120
 - Massively multiplayer online role-playing game (MMORPG) subscriptions, 287
 - Material attribute, Colliders, 125
 - Materials
 - Physics 2D, 128–129
 - reducing draw cells for optimization, 264–265
 - Max Particles property, particle system
 - coin boxes, 221
 - damage, 224
 - defined, 218
 - Maximize on Play, Game View, 25
 - Maximum Allowed Timestep, Time Manager, 263
 - maxSpeed property
 - EnemyController, 162
 - PlayerController, 77–78
 - using breakpoints, 93
 - Maya 3D file format, 27–28
 - Menus
 - Assets menu, 13–14
 - Component menu, 16
 - Edit menu, 12–13
 - File menu, 11–12
 - Game Over screen, 201–202
 - Game Win screen, 202–203
 - GameObject menu, 15–16
 - Help menu, 17–19
 - title screen, 200–201
 - Unity interface, 10–11
 - Window menu, 17
 - Mesh command, Component menu, 16
 - Mesh data, Web Player Settings, 283
 - Meta user interfaces, 194
 - Miscellaneous command, Component menu, 16
 - MMORPG (massively multiplayer online role-playing game) subscriptions, 287
 - Modules
 - music tracker, 228
 - particle system, 218–219
 - Monetization
 - choosing right strategy, 288–290
 - common purchase models, 287
 - overview of, 287
 - MonoDevelop–Unity IDE
 - developing and running scripts with, 3, 75–76
 - initial script setup, 76–77
 - lateral movement, 79
 - properties, 77–79
 - Mouse shortcut, camera movement, 58
 - Move to View command, GameObject menu, 15
 - Move X (Y and Z), Snap Settings, 63–64
 - Movement
 - air control and, 72
 - basic follow-cam, 83
 - character acceleration and, 72
 - coding in Unity3D, 71–72
 - creating/hooks up PlayerController, 74–75
 - discrete vs. smooth tile, 72
 - error handling tools. *See* Error handling and debugging
 - filling in properties, 77–79
 - initial script setup, 76–77
 - Input Manager for, 83–85
 - jumping, 80–82
 - lateral, 79
 - MonoDevelop–Unity IDE, 75–76
 - overview of, 71
 - player’s facing, 79–80
 - Moving platforms
 - Fly enemy GameObject, 183–184
 - Prefab for, 179–180
 - between two points at set speed, 180–182
- ## N
- Namespaces
 - file naming conventions, 7–8
 - searching for file in Project Browser, 21–23
 - Naming conventions
 - constants, 258
 - files, 7–8
 - new projects, 5
 - organizing assets into folders, 247–248
 - organizing script files, 253–254
 - native GUI
 - Controls, 195–197
 - overview of, 194–195

- Skin, 195
- Style, 195
- Navigation command, Component menu, 16
- Navigation, Scene, 56–58
- Nested GameObjects, 179–180
- Network Emulation command, Edit menu, 13
- New Project command, File menu, 11
- New Scene command, File menu, 11
- Nodal tree workflow, Animator window, 110
- Non-diegetic user interfaces, 194

O

- OBJ 3D file format, 27–28
- Object manipulation
 - grid settings, 63
 - Pixels to Units settings, 62–63
 - positioning GameObjects, 64–66
 - Snap Settings, 63
 - Transform tools, 59–61
 - Z-depth, 61–62
- OnGUI() function
 - GameOverScript script, 279
 - GUI layouts, 197–198
- OnGUI system. *See* UGUI (Unity GUI)
- Online references
 - Audio Reverb Zone Component
 - properties, 230–231
 - coroutines, 175
 - documentation and learning resources, 8
 - downloading Unity, 1
 - persisting data between Scenes/game sessions, 157
 - PlayerPrefs class, 274
 - supplementary materials, xxiv
 - Unity Asset Store, 72
- OnTriggerEnter2D() function
 - Coinbox script, 151
 - CoinPickup script, 145–146
 - ContactDamage script, 170
 - defined, 135
 - EnemySlime script, 164, 165
 - PitTrigger script, 143, 245
 - SpawnTrigger script, 190
 - VictoryTrigger script, 273–274
- OnTriggerExit2D() function, 136
- OnTriggerStay2D() function, 136
- Open command, Assets menu, 13

- Open Project command, File menu, 11
- Open Project tab, Project Wizard, 4
- Open Scene command, File menu, 11
- Optimizations
 - batching, 266
 - draw calls, 264–265
 - overview of, 247, 261–262
 - physics, 262–263
 - Prefabs, 262
 - Rendering Statistics window, 266–267
 - summary exercises, 268
 - triangle count, 265–266
- Optimize Mesh Data, Web Player Player Settings, 283
- Order in Layer attribute, 62, 264
- Organization
 - assets, 247–248
 - code, 254–261
 - Hierarchy, 250–252
 - Labels, 249
 - overview of, 247
 - Prefabs, 248–249
 - script files, 253–254
- Overlapping action, animation, 96

P

- Pack setting, Sprite Packer, 48
- Packages
 - defined, 31
 - importing, 31–33
 - overview of, 5
- Packing Policy setting, Sprite Packer, 49
- Packing Tag, 48–49
- Panel Component, UGUI, 291
- Parameters, assigning via animations, 111–112
- Parenting, GameObjects with Hierarchy, 20
- Particle effects. *See also* Shuriken Particle System
 - coin boxes, 220–222
 - damage, 223–227
 - defined, 216
 - using fewest number of, 227
- Particle system
 - creating, 216–217
 - curves, 219–220
 - defined, 215
 - modules and properties of, 217–219

- Particle System Component
 - creating particle effect for damage, 223–225
 - creating particle system, 216–217
 - dragging sprite onto GameObject, 222
- Particles, defined, 215
- Pause command, Edit menu, 13
- Pay once, purchase model, 287–289
- Pay to win, purchase model, 287–289
- PCs, standalone Build Settings, 282–283
- Performance. *See* Optimizations
- Perspective view, camera, 57
- Physics
 - 2D Settings, 122–124
 - 2D vs. 3D, 120–121
 - adding constraints, 129–134
 - Collider types, 125–128
 - creating Materials, 128–129
 - force, 120
 - gravity, 120
 - mass, 119–120
 - optimizing, 262–263
 - Physics 2D Settings, 122–124
 - review exercise, 134
 - Rigidbody, 124–125
 - understanding, 119
- Physics 2D command, 16
- Physics command, 16
- Physics2D check, `FixedUpdate()`, 81
- Pit Prefab, 172
- PitTrigger script
 - cleaning up player death, 245
 - JavaScript sample code for, 320–321
 - using checkpoints with respawn, 142–144
- Pivot tool, Transform Gizmo Toggles, 19
- Pivot values, UGUI Rect Transform, 298
- Pixel Perfect, UGUI Render Mode, 294
- Pixels to Units value, 45, 62–63
- Platform GameObjects, Flight Points script, 180–182
- platformMoving Prefab, 179–180
- Platforms
 - building and deploying game, 281–282
 - PC, Mac, and Linux standalone Build Settings, 282–283
 - Web Player Build Settings, 282
- Play command
 - Animation window, 101
 - Edit menu, 13
- Play on Awake property
 - adding sound to coin box, 236–237
 - particle system, 218, 221, 224
- `PlayClipAtPoint()` function, coin collection, 238
- `PlayDamageAudio()` function, damage event, 235
- Player Idle clip, and walk animation, 105–106
- Player scripts, JavaScript code samples
 - PlayerController script, 301–304
 - PlayerStats script, 304–307
- Player Settings button, Build Settings, 281
- PlayerController script
 - adding basic jumping, 80–82
 - adding breakpoints, 90–93
 - adding `Debug.Log`, 89–90
 - adding footsteps to walk cycle, 231–233
 - adding lateral movement, 79
 - adding sound to damage event, 234–236
 - adding sound to jump event, 233–234
 - calling damage particle system from code, 226–227
 - creating and hooking up, 74–75
 - editing, 112–115
 - handling exceptions, 86
 - handling player's facing, 79–80
 - initial setup, 76–77
 - JavaScript sample code for, 301–304
 - properties, 77–79
- `PlayerIsDead()` function
 - adding sound to damage event, 236
 - cleaning up player death, 242–243
 - handling player death, 177
- `PlayerPrefs` class
 - more information on, 274
 - recovering from Game Over, 279
 - retrieving player's coin value from, 274–275
 - VictoryTrigger script, 271, 273–274
- PlayerStats script
 - calling damage particle system from code, 225–227
 - cleaning up player death, 242–243
 - dealing damage, 167–170

- handling immunity, 172–177
 - handling player death, 177–178
 - hooking up HUD display to, 211–212
 - JavaScript code sample, 304–307
 - retrieving player's coin value, 274–275
 - sound for damage event, 235–236
 - tracking player stats, 155–157
 - `PlayFootstepAudio()` function, 232
 - `PlayHitReaction()` function
 - damage particle system, 226
 - player death, 177
 - `PlayerStats`, 174
 - sound for damage event, 236
 - temporary immunity, 174
 - `playHitReaction` property,
 - `ContactDamage`, 170
 - `PlayJumpAudio()` function, 234
 - Polishing finished games, 286–287
 - Polygon Collider, 126
 - Polygon mesh, 98
 - Polygon triangles, optimization, 264–265
 - Popped coin Prefabs, 147
 - Post-deployment, 286
 - Prefabs
 - adding collision to, 128
 - adding sorting elements to, 67–69
 - attaching physics Materials to, 129
 - coin box, 147–150
 - Coinbox script, 150–151
 - CoinPickup script, 146
 - floating coin, 145
 - Fly enemy, 187
 - level design and, 66–67
 - optimization of, 262
 - organization of, 248–249
 - overview of, 38–39
 - platformMoving, 179–180
 - popped coin, 147
 - positioning GameObjects, 64–66
 - saving checkpoints or pits as, 144
 - victory trigger, 269–272
 - Preferences command, Edit menu, 13
 - Preferences, switching between 2D/3D, 5
 - Premade assets
 - creating new assets, 29–30
 - importing from file browser, 29
 - importing packages, 31–33
 - Presentation, Web Player Player Settings, 283
 - Preset window, anchor, 298
 - Previous/Next Keyframe, Animation
 - window, 101
 - Prewarm property, particle system, 217
 - Project Browser
 - adding Animator Controller from, 108
 - building 2D sprites, 43
 - list of labels in, 249
 - Project Location, 5
 - Project Settings command
 - in Edit menu, 13
 - setting Project workspace to 2D, 41–42
 - Time Manager, 262–263
 - Project Wizard, 4–6
 - Projects
 - creating with Project Wizard, 4–6
 - file naming conventions, 7–8
 - folder organization, 7
 - structure of, 6
 - Properties
 - Animation Component, 100
 - Audio Reverb Zone Component, 230–231
 - Audio Source Component, 228–230
 - CameraFollow script, 240
 - ContactDamage, 170
 - EnemyController, 162
 - FlightPoints script, 180–182
 - Input Manager, 83–84
 - particle system, 217–219
 - PlayerController, 77–79
 - PlayerStats, 174, 176
 - Public variables, 77
 - Purchase models, monetization, 287–290
- ## R
- RawImage Component, UGUI, 292
 - Realism, and character acceleration, 72
 - Receives Events, UGUI Render Mode, 294
 - Record, Animation window, 101
 - Rect Transform, UGUI, 296–298
 - Reduced air control system, 72
 - Reference Manual command, Help menu, 18
 - Refresh command, Assets menu, 14
 - Regions, code, 254–256
 - Registration authorization, 3–4

- Reimport All command, Assets menu, 14
 - Reimport command, Assets menu, 14
 - Release Notes command, Help menu, 19
 - Render Mode
 - Scene View, 23
 - UGUI, 294–296
 - Render Settings command, Edit menu, 13
 - Renderer module, particle systems, 219, 222, 225
 - Rendering command, Component menu, 16
 - Rendering path, Web Player Player Settings, 283
 - Rendering Statistics window, 266–267
 - Repack setting, Sprite Packer, 48
 - Repeat Button, GUI, 195
 - Report a Bug command, Help menu, 19
 - Resolution
 - building 2D sprites, 44–45
 - Web Player Player Settings, 283
 - Respawn
 - for more forgiving replay, 137
 - sizing/placing checkpoint trigger for, 139–140
 - Rigidbody
 - adding constraints, 129–134
 - optimizing physics, 263
 - Physics 2D and, 124–125
 - preparing Slime enemy GameObject, 160
 - Roadmaps, level design, 54–55
 - Rotation
 - 2D vs. 3D physics, 121
 - with Rotate tool, 19, 59–60
 - Snap Settings tools, 63
 - Rotation by Speed module, 219
 - Rotation Over Lifetime module, 219, 225
- S**
- Sample, Animation window, 102
 - Sandbox Projects, testing gameplay scripts, 5
 - Save Project command, File menu, 11
 - Save Scene As command, File menu, 11
 - Save Scene command, File menu, 11
 - Save Search, Project Browser, 23
 - Scale tool
 - positioning GameObject with, 60–61
 - Snap Settings, 63
 - as Transform tool, 19
 - UI Rect Tool as, 298–299
 - Scene Gizmo, 56–57
 - Scene View
 - building 2D sprites, 43
 - building navigation in game world, 56–58
 - creating 2D workspace, 43
 - creating GameObject in, 33–35
 - interface view, 23–24
 - particle system animating in, 216–217
 - Pixels to Units size for tile sprites in, 63
 - viewing animation in, 107
 - Scenes in Build list, Build Settings, 275–277
 - sceneToLoad property, VictoryTrigger script, 273–274
 - Score, carrying forward after winning game, 286
 - Screen Space Camera, UGUI Render Mode, 294
 - Screen Space Overlay, UGUI Render Mode, 294
 - Screen Space, UGUI Rect Transform, 296–297
 - Screen, viewing resolution/memory usage, 267
 - Scripted animation, 98
 - Scripting Define Symbols, Web Player Player Settings, 283
 - Scripting languages, Unity3D, 71–72
 - Scripting Reference command, Help menu, 18
 - Scripts. *See also* GUI scripts; JavaScript code samples
 - building with MonoDevelop, 3, 74–75
 - CheckpointTrigger, 137–138, 312
 - CoinBox, 150–151, 307–308
 - CoinCounter, 206–207, 211–212, 314–315
 - CoinPickup. *See* CoinPickup script
 - CoinSpawner, 148, 152–153, 308–309
 - ContactDamage, 169–170, 320
 - EnemyController, 161–162, 309
 - EnemySlime, 162–164, 309–310
 - FlightPoints, 180–182, 185–186, 312–313
 - FlyController, 185, 310
 - GameOverScript, 201–202, 279
 - GameWinScript, 202–203, 279–280
 - organizing files for, 253–254
 - PitTrigger, 140–144, 245, 320–321
 - PlayerStats. *See* PlayerStats script

- reusing via packages, 5
- SpawnTrigger, 189–190, 313
- SplashScreenDelayed, 199–200, 276–277, 319
 - testing gameplay, 5
 - TitleScreenScript, 200–201, 277, 319–320
 - VictoryTrigger, 272–274, 313–314
- Scroll View, GUI, 197
- Scrollbar Component, UGUI, 292
- Search bar, Project Browser, 21–23
- Secondary action, animation, 96
- Select All command, Edit menu, 13
- Select Dependencies command, Assets menu, 14
- Select Sprite window, 37–38
- Selection command, Edit menu, 13
- Selection Grid, GUI, 196
- Shadow casters, viewing total number of, 267
- Shape Module, particle system, 218, 221, 224
- Shorthand code
 - learning for game development, 261
 - methods of, 82
- shouldChangeFacing property, FlightPoints, 186
- Show in Explorer command, Assets menu, 13
- Shuriken Particle System
 - coin box particle effect, 220–222
 - coin box particle effect hookup, 223
 - creating particle system, 216–217
 - curves, 219–220
 - damage particle effect, 223–225
 - damage particle system from code, 225–227
 - defined, 215
 - modules of, 218–219
 - properties of, 217–218
 - splash screen, 198–200
 - terms to know, 215–216
- Simulation Space property, particle system, 218, 221, 224
- Six degrees of freedom (6DoF), 120–121
- Size by Speed module, 218
- Size Over Lifetime module, particle system, 218, 221, 225
- Sizing
 - Checkpoint trigger, 138–139
 - hiding Sprite Renderer Component, 161
- Sketched level design, of roadmap, 54–55
- Skin, GUI, 195
- Skinned GameObjects, 267
- Skinning, Web Player Player Settings, 283
- Sleeping Mode, optimizing Rigidbody, 263
- Slider Component, UGUI, 292
- Slider Joint 2D constraint, 130
- Slime enemy
 - adding animation to, 166–167
 - adding wall to level, 164–165
 - creating damage. *See* Damage, dealing
 - EnemySlime Component script, 162–164
 - handling collision with other Slimes, 165–166
 - inheritance and EnemyController script, 160–162
 - maintaining enemy arrangements, 187–189
 - preparing GameObject, 159–160
- Slow in and slow out, animation, 96
- Smooth tile movement, 72
- Snap Settings
 - aligning tiles, 67
 - Edit menu, 13
 - manipulating objects, 63
 - positioning GameObjects, 64
- Solid drawing, animation, 96
- Solution Explorer, MonoDevelop, 75–76
- Sorting Layers
 - building first level, 67–69
 - controlling Z-depth, 61–62
 - defined, 42
 - optimizing, 264
 - Order in Layer attribute vs., 62
- Spatial user interfaces, 194
- Spawn
 - adding animation to controller, 155
 - CoinSpawner script, 148, 152–153, 308–309
 - hooking it all together, 153–154
 - preparing spawn trigger, 188
 - spawning vs., 188
 - SpawnTrigger script, 189–190, 313
 - using checkpoints with, 140–144
- SpawnCoin() function, 153
- Speed, moving platforms at set, 180–182
- speed property, FlightPoints, 180–182

- Splash screen
 - creating, 198–200
 - hooking up Intro screens, 275–277
 - SplashScreenDelayed script
 - creating splash screen, 199–200
 - hooking up Intro screens, 276–277
 - JavaScript sample code for, 319
 - Spring Joint 2D constraint, 130
 - Sprite Editor
 - Automatic Slicing, 46–47
 - Grid Slicing, 47–48
 - Manual Slicing, 45–46
 - slicing sprites with, 45
 - Sprite Packing, 48–50
 - Sprite Renderer Component
 - adding walls to level, 164
 - coin box Prefabs, 147–148
 - Fly enemy GameObject, 183–184
 - hiding sizing handles, 161
 - moving platform Prefab, 179
 - of Player GameObject, 61
 - representing immunity, 176–177
 - Slime enemy GameObject, 159
 - victory trigger Prefab, 270
 - SpriteFlicker() function, 175–177
 - spriteRenderer property, PlayerStats, 176–177
 - Sprites, 2D
 - 2D behaviors and, 41–42
 - 2D workspace and, 42–43
 - building, 43
 - debugging animations, 101
 - frame animation with, 97–98
 - Import Settings, 43–45
 - overview of, 41
 - Pixels to Units value, 45
 - reducing draw cells for optimization, 264
 - review exercises, 50
 - Sprite Editor, 45–48
 - Sprite Packing, 48–50
 - Squash and stretch, animation, 96
 - Staging, animation, 96
 - Standalone Input script, UGUI Event
 - System, 299
 - Star sprite, 222
 - Start Color property, particle system, 218, 220
 - Start Delay property, particle system, 218
 - Start() function
 - CoinSpawner script, 152–153
 - editing PlayerController, 112
 - retrieving coin value, 275
 - visually representing immunity, 175–176
 - Start Lifetime property, particle system, 218, 220, 224
 - Start Rotation property, particle system, 218, 224
 - Start Size property, particle system, 218, 220, 224
 - Start Speed property, particle system, 218, 220, 224
 - State Collider, 263
 - State machines, 108
 - Static Batching, 283
 - Stats, Game View, 25
 - Step command, Edit menu, 13
 - Straight-ahead action, animation, 96
 - Structure, project, 6
 - Styles, GUI, 195
 - Sub Emitters module, particle system, 219
 - Subfolders, for script files, 253
 - Subscription, purchase model
 - choosing right monetization strategy, 288–289
 - cross-platform Player Settings, 283
 - defined, 287
 - Supplementary materials, xxiv
 - Switch Platform button, Build Settings, 281
 - Sync MonoDevelop Project command, Assets
 - menu, 14
 - System requirements, installing Unity, 2
 - System scripts, JavaScript code samples, 322
- ## T
- Tags, 35
 - Tags & Layers
 - creating GameObject, 35–36
 - passing through player's space, 170–171
 - Sorting Layers in, 62
 - TakeDamage() function
 - adding temporary immunity, 174
 - cleaning up player death, 243–244
 - handling player death, 177
 - scripting damage into PlayerStats, 167–168
 - Tangent Edit Box, 103–104

- Tangents, Graph Editor, 103–104
 - Template, example folder organization, 7
 - Temporary immunity
 - adding to player, 172–175
 - visually representing, 175–177
 - Testing, Gameplay scripts, 5
 - Text Component, UGUI, 292
 - Text field, GUI, 195
 - Text, GUI, 198
 - Texture Packer, 49
 - Texture Sheet Animation module, 219
 - Texture Type property, coin box, 222
 - Textures
 - GUI, 198
 - importing setting for 2D sprites, 44
 - viewing number of, 267
 - `this` keyword, in scripts, 146
 - Tile sprites, 48
 - Time Manager, 262–263
 - Time Scale, 262–263
 - `Time.DeltaTime`, 173–176
 - `timeElapsed` property, `VictoryTrigger` script, 273
 - Timers, adding to levels, 286
 - Timing, in animation, 96
 - Title screen
 - creating, 200–201
 - hooking up Intro screens, 275–277
 - `TitleScreenScript` script
 - hooking up Intro screens, 277
 - JavaScript sample code for, 319–320
 - overview of, 200–201
 - Toggle Component, UGUI, 292
 - Toggle Control, GUI, 196
 - Toolbar
 - GUI, 196
 - overview of, 19–20
 - Touch Input script, UGUI Event System, 299
 - Tracker modules, music, 228
 - `TrackPlayer()` function, `CameraFollow` script, 241
 - Transform animation, 97
 - Transform Component, 33–34, 37–38
 - Transform Gizmo toggles, 19–20
 - Transform tools
 - controlling camera with Hand tool, 58
 - manipulating objects with, 59–60
 - toolbar, 19
 - with UI Rect Tool, 298–299
 - Transitions, animation, 115–116
 - Translate tool, 19, 59–60
 - Triangle count, optimizing, 265–266
 - Triangles, viewing number of Scene, 267
 - Trigger events, colliders, 125–126
 - `triggerDamage` `GameObject`
 - adding temporary immunity post-damage, 172–175
 - creating damage, 168–170
 - passing through player’s space, 171
 - preparing Fly enemy `GameObject`, 183
 - Triggers
 - adding to `GameObjects`, 136
 - checkpoint, 137–140
 - creating collectibles. *See* Collectibles
 - function of, 135
 - nesting `GameObjects` with, 171
 - spawn, 188–190
 - `Trigger2D` functions, 135–136
 - using checkpoints with respawn, 140–144
 - `triggerSpawn` `GameObject`, 188–190
 - `triggerVictory` `GameObject`
 - creating script, 272–274
 - preparing Prefab, 269–272
 - Truecolor format, 45
 - Try-Catch-Finally statement, 87–89, 138
- ## U
- UGUI (Unity GUI)
 - adding mask, 299
 - Canvas Component, 293
 - components, 291–293
 - creating example interface, 293
 - event system and event triggers, 299–300
 - overview of, 291
 - Rect Transform, 296–298
 - Render Mode, 294–296
 - UI Rect tool, 298–299
 - UI Rect tool, UGUI, 298–299
 - UI (user interface). *See also* UGUI (Unity GUI)
 - design types, 193–194
 - Game Over screen, 201–202
 - Game Win screen, 202–203
 - HUD scripts, 206–211

- UI (user interface) (*continued*)
 - HUD visuals, 204–205
 - splash screen, 198–200
 - title screen, 200–201
 - Unity native GUI, 194–198
 - Unexpected behaviors, 91
 - Unhandled exceptions, 87
 - Unity Activation screen, 3
 - Unity Answers command, Help menu, 18
 - Unity Asset Store, 3, 72
 - Unity Development Web Player, 3
 - Unity Feedback command, Help menu, 18
 - Unity GUI. *See* UGUI (Unity GUI)
 - Unity interface
 - Assets menu, 13–14
 - Component menu, 16
 - Edit menu, 12–13
 - File menu, 11–12
 - Game View, 24–25
 - GameObject menu, 15–16
 - Help menu, 17–19
 - Hierarchy list, 20
 - Inspector, 20–21
 - menus, 10–11
 - overview of, 10
 - Project Browser, 21–23
 - review exercises, 25–26
 - Scene View, 23–24
 - toolbar, 19–20
 - Window menu, 17
 - Unity Manual command, Help menu, 18
 - Unity Manuals, 8, 17
 - Unity Pro
 - 30-day evaluation of, 4
 - defined, 2
 - some packages requiring, 6
 - UnityScript, 72
 - Update() function
 - adding Debug.Log to, 89–90
 - adding sound to jump event, 234
 - adding temporary immunity, 174
 - cleaning up player death, 243
 - defined, 77
 - editing PlayerController, 112–113
 - hooking up title screen, 277
 - implementing basic jumping, 82
 - VictoryTrigger script, 274
 - visually representing immunity, 177
 - Update Mode, Animator Component, 109
 - Updates, checking for Unity, 18
- ## V
- Values, tweaking Rigidbody, 125
 - VBO (vertex buffer object), 267
 - Vector math, 144, 150–151
 - Vector3, Coinbox script, 151
 - Velocity Over Lifetime module, 218
 - Vertical Slider, GUI, 196
 - Vertices, viewing total number of Scene, 267
 - VictoryTrigger
 - adding effects to, 286
 - preparing Prefab, 269–272
 - VictoryTrigger script, 272–274, 313–314
 - View Atlas setting, Sprite Packer, 48
 - Views
 - Game View, 24–25
 - Project Browser, 21–23
 - Scene View, 23–24
 - Vleugels, Kenny, 32
 - Volume, preparing trigger, 140
 - VRAM usage, Render Statistics window, 267
- ## W
- Walk animation, 105–106
 - WASD keys, controlling camera
 - movement, 58
 - waypointA property, 180–184
 - waypointB property, 180–184
 - Web Player
 - Build Settings, 282
 - building game for, 285
 - deploying game to Web, 285
 - Player Settings, 283–284
 - post-deployment, 286
 - using right settings, 284–285
 - Welcome screen
 - accessing in Help menu, 17–18
 - setting up development environment, 9–10
 - Wheel Joint 2D constraint, 130
 - Window, GUI, 197
 - Window menu, 17

Windows Explorer files, Project Browser
 extension of, 21

Winning game
 carrying player's score forward after, 286
 creating Game Win screen, 202–203
 starting over from, 279–280

Workspace, setting to 2D, 42

World Space, UGUI, 294, 297

World tool, Transform Gizmo Toggles, 20

X

X-axis. *See also* Axes
 creating 2D workspace, 42–43
 working in 3D, 41

Y

Y-axis. *See also* Axes
 creating 2D workspace, 42–43
 working in 3D, 41

Z

Z-axis, in 3D, 41. *See also* Axes

Z-depth
 2D vs. 3D physics, 121
 controlling, 61–62
 working in 2D, 41

Zero air control system, 73