



Game Programming with Unity and C#

A Complete Beginner's Guide

Casey Hardman

Apress®

Game Programming with Unity and C#

A Complete Beginner's Guide

Casey Hardman

Apress®

Game Programming with Unity and C#

Casey Hardman
West Palm Beach, FL, USA

ISBN-13 (pbk): 978-1-4842-5655-8
<https://doi.org/10.1007/978-1-4842-5656-5>

ISBN-13 (electronic): 978-1-4842-5656-5

Copyright © 2020 by Casey Hardman

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: James Markham
Coordinating Editor: Divya Modi

Cover designed by eStudioCalamar

Cover image designed by Pixabay

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5655-8. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Table of Contents

About the Authorxv

About the Technical Reviewerxvii

Introductionxix

Chapter 1: Installation and Setup 1

 Installing Unity 1

 Installing Our Code Editor 4

 Creating a Project 7

 Summary..... 9

Chapter 2: Unity Basics 11

 Windows 11

 Project Window 12

 Scene Window 13

 Hierarchy Window 14

 Inspector Window 14

 Components 14

 Adding GameObjects..... 17

 Summary..... 20

Chapter 3: Manipulating the Scene 21

 Transform Tools..... 21

 Positions and Axes 23

 Making a Floor 24

 Scale and Unit Measurements 24

 Summary..... 26

TABLE OF CONTENTS

Chapter 4: Parents and Their Children..... 29

 Child GameObjects..... 29

 World vs. Local Coordinates..... 32

 A Simple Building..... 33

 Pivot Points 36

 Summary..... 39

Chapter 5: Prefabs..... 41

 Making and Placing Prefabs 41

 Editing Prefabs..... 42

 Overriding Values 44

 Nested Prefabs..... 48

 Prefab Variants..... 49

 Summary..... 50

Chapter 6: Programming Primer 53

 Programming Languages and Syntax 53

 What Code Does..... 54

 Strong vs. Weak Typing 55

 File-Type Extensions 57

 Scripts..... 57

 Summary..... 59

Chapter 7: Code Blocks and Methods..... 61

 Statements and Semicolons 61

 Code Blocks 61

 Comments..... 63

 Methods 64

 Calling Methods 67

 Basic Data Types..... 69

 Returning Values with Methods 69

 Declaring Methods..... 70

Operators	73
Summary.....	75
Chapter 8: Conditions	77
The “if” Block.....	77
Overloads	79
Enums	80
The “else” Block	81
The “else if” Block	82
Operators for Conditions	83
Equality Operators	83
Greater Than and Less Than	85
Or.....	85
And	86
Summary.....	86
Chapter 9: Working with Objects.....	87
Classes.....	87
Variables.....	89
Accessing Class Members.....	90
Instance Methods	93
Declaring Constructors	95
Using the Constructor	98
Static Members	99
Summary.....	101
Chapter 10: Working with Scripts	103
Usings and Namespaces.....	104
Script Class	105
Rotating a Transform.....	107
Frames and Seconds	109
Attributes	110
Summary.....	112

TABLE OF CONTENTS

Chapter 11: Inheritance 113

 Inheritance in Action: RPG Items 113

 Declaring Our Classes 115

 Constructor Chaining 118

 Subtypes and Casting 122

 Number Value Types 124

 Type Checking 126

 Virtual Methods 127

 Summary 128

Chapter 12: Debugging 129

 Setting Up the Debugger 129

 Breakpoints 131

 Using Unity’s Documentation 135

 Summary 137

Part I: Obstacle Course 139

Chapter 13: Obstacle Course Design and Outline 141

 Gameplay Overview 142

 Technical Overview 143

 Player Controls 143

 Death and Respawn 145

 Levels 145

 Level Selection 146

 Obstacles 146

 Project Setup 146

 Summary 148

Chapter 14: Player Movement 149

 Player Setup 150

 Materials and Colors 152

 Declaring Our Variables 155

Properties.....	158
Tracking the Velocity	159
Applying the Movement	166
Summary.....	170
Chapter 15: Death and Respawning	173
Enabling and Disabling	174
Death Method.....	177
Respawn Method	178
Summary.....	180
Chapter 16: Basic Hazards	181
Collision Detection	181
Hazard Script	188
Projectile Script.....	190
Shooting Script	194
Summary.....	197
Chapter 17: Walls and Goals.....	199
Walls	199
Goals	201
Build Settings for Scenes.....	204
Summary.....	207
Chapter 18: Patrolling Hazards.....	209
Resembling a Patrol Point.....	209
Arrays.....	210
Setting Up Patrol Points	212
Detecting Patrol Points	216
The “for” Loop	218
Sorting Patrol Points	221
Moving the Patroller.....	226
Summary.....	229

TABLE OF CONTENTS

Chapter 19: Wandering Hazards 233

Wander Regions 233

A Basic Editor Extension 236

Editor Scripts 236

Custom Inspectors..... 237

Accessing the Inspector Target 237

Drawing to the Scene 238

Wanderer Setup 240

Wanderer Script 241

Handling the State 243

Reacting to the State 245

Summary..... 247

Chapter 20: Dashing 249

Dashing Variables 249

Dashing Method 251

Final Touches 254

Dash Cooldown 255

Summary..... 257

Chapter 21: Designing Levels 259

Prefabs and Variants 259

Making Levels 261

Adding Walls 263

Level View Camera..... 264

Summary..... 264

Chapter 22: Menus and UI 267

Scene Flow 267

Level Selection Script 269

Summary..... 276

Chapter 23: In-Game Pause Menu	279
Freezing Time.....	279
Summary.....	284
Chapter 24: Spike Traps.....	285
Designing the Trap	285
Raising and Lowering	289
Writing the Script.....	290
Adding Collisions.....	295
Summary.....	297
Chapter 25: Obstacle Course Conclusion.....	299
Building the Project.....	299
Player Settings	301
Recap	303
Additional Features	304
Summary.....	306
Part II: Tower Defense.....	307
Chapter 26: Tower Defense Design and Outline.....	309
Gameplay Overview	309
Technical Overview	311
Project Setup	313
Summary.....	313
Chapter 27: Camera Movement	315
Setting Up	315
Arrow Key Movement.....	318
Applying Movement	320
Mouse Dragging.....	322
Zooming	324
Summary.....	325

TABLE OF CONTENTS

Chapter 28: Enemies, Towers, and Projectiles.....	327
Layers and Physics	327
Basic Enemies.....	329
Projectiles	333
Targeters	339
Towers.....	348
Arrow Towers	351
Summary.....	359
Chapter 29: Build Mode	361
UI Basics	362
The RectTransform.....	366
Building Our UI	368
Events	373
Setting Up	374
Build Mode Logic	379
The Dictionary	385
OnClick Event Methods	388
Summary.....	395
Chapter 30: Play Mode	397
Spawn and Leak Points.....	397
Locking the Play Button	399
Pathfinding Setup	401
Finding a Path	404
Play Mode Setup	409
Spawning Enemies	414
Enemy Movement	417
Summary.....	423

Chapter 31: More Tower Types	425
Arcing Projectiles	425
Cannon Tower	431
Hot Plates	435
Barricades	437
Summary	437
Chapter 32: Tower Defense Conclusion	439
Inheritance	439
UI	441
Raycasting	441
Pathfinding	442
Additional Features	442
Health Bars	442
Types for Armor and Damage	443
More Complex Pathing	443
Range Indicators	444
Upgrading Towers	445
Summary	445
Part III: Physics Playground	447
Chapter 33: Physics Playground Design and Outline.....	449
Feature Outline.....	449
Camera	449
Player Movement.....	450
Pushing and Pulling.....	450
Moving Platforms	451
Swings.....	451
Force Fields and Jump Pads	451
Project Setup	451
Summary.....	452

TABLE OF CONTENTS

Chapter 34: Mouse-Aimed Camera..... 453

 Player Setup..... 453

 How It Works..... 454

 Script Setup 456

 Hotkeys 463

 Mouse Input..... 464

 First-Person Mode 469

 Third-Person Mode 470

 Testing..... 474

 Summary..... 475

Chapter 35: Advanced 3D Movement..... 477

 How It Works..... 477

 Player Script..... 480

 Movement Velocity..... 486

 Applying Movement 490

 Losing Velocity 493

 Gravity and Jumping 494

 Summary..... 496

Chapter 36: Wall Jumping 497

 Variables 497

 Detecting Walls 499

 Performing the Jump 502

 Summary..... 505

Chapter 37: Pulling and Pushing 507

 Script Setup 507

 FixedUpdate 511

 Target Detection..... 513

 Pulling and Pushing 515

Cursor Drawing	517
Summary.....	518
Chapter 38: Moving Platforms.....	519
Scene Setup.....	520
Platform Movement.....	522
Player Platforming	529
Summary.....	532
Chapter 39: Joints and Swings.....	533
Swing Setup.....	533
Connecting the Joints	540
Finishing Touches.....	542
Summary.....	543
Chapter 40: Force Fields and Jump Pads	545
Script Setup	545
Force Field Setup.....	547
Adding Velocity to the Player	548
Applying Forces	549
Summary.....	552
Chapter 41: Conclusion.....	553
Physics Playground Recap.....	553
Further Learning for Unity.....	555
The Asset Store	555
Terrains.....	555
Coroutines	556
Script Execution Order.....	556
Further Learning for C#.....	557
Delegates.....	557
Documentation Comments	558
Exceptions	560

TABLE OF CONTENTS

Advanced C# 562

 Operator Overloading..... 562

 Conversions..... 562

 Generic Types 563

 Structs 563

Summary..... 563

Index..... 565

About the Author



Casey Hardman is a hobbyist game developer, who found inspiration in the capacity for immersion and interactivity provided by games. His area of focus is the Unity game engine. He has nurtured a passion for video games since he was a child. In his early teens, this interest led him on a journey into the world of game design and programming. He is self-taught through a variety of personal projects, some small and some lofty. He has been a regular contributor on various online game development platforms and spends far too much time in front of the computer.

About the Technical Reviewer



Robert Lair has been building software professionally for more than 25 years and has served at just about every position in the SDLC. He has served as President/CEO, VP of Product Development, CTO, Software Architect, Developer, and Scrum Master/Product Manager. He believes that building good software is an art and is passionate about correctly architected software, efficient development processes, documentation, reducing technical debt, organization, and productivity. He specializes in building Unity, web, and mobile applications focused on gamification. You can find out more about Robert at his website, www.robertlair.com.

Introduction

Welcome to the start of your adventure into game programming with Unity. This book is designed to teach you how to program video games from the ground up, while still engaging you with plenty of hands-on experience. It's not focused on completing ambitious projects, and it's not about fancy graphics. We're learning how to program and how to use the Unity engine. Once you have a solid understanding of these integral topics, you can expand your knowledge and make more and more complicated and impressive games.

All of the software we'll be using is cross-platform. This book will mostly stick to Windows-based terminology and examples, but you can still follow through with other major operating systems, like Mac or Linux, with little or no extra trouble.

As for system requirements, any modern computer purchased within the last 6 years or so should have little difficulty running the software we'll be working with. Since we aren't fiddling with high-end graphics or computing long-winded algorithms, the example projects we develop should run fine on most systems. If you have concerns, the official and most up-to-date system requirements for Unity can be found at the official website here:

<https://unity3d.com/unity/system-requirements>

In Chapters 1–12, we'll begin with a primer for the essential concepts of the Unity game engine itself and get all our tools set up and ready for action. Then, we'll get into the nitty-gritty details of programming, and we'll start to actually write code ourselves.

In the remainder of the book, we'll tackle individual game projects one at a time, making playable projects that you can add to later if you please. This is where you'll get much of your hands-on experience. We'll implement actual game mechanics, which is what you're really after as a game programmer, right?

Game Project 1, “Obstacle Course” (Chapters 13–25), will be a top-down obstacle course where the player moves their character with WASD or the arrow keys to avoid touching hazards of various forms: patrolling and wandering hazards, traveling projectiles, and spike traps in the floor. We'll get practice with basic movement and rotation, setting up levels, working with fundamental Unity concepts like prefabs and scripting, and setting up UI.

INTRODUCTION

Game Project 2, “Tower Defense” (Chapters [26–32](#)), will be the basis of a simple “tower defense” game, where the player places defensive structures on the playing field. Enemies will navigate from one side of the field to the other, and the player’s defenses will attempt to fend them off. We’ll explore basic pathfinding (how the enemies navigate around arbitrary obstacles) and further expand on fundamental programming concepts.

Game Project 3, “Physics Playground” (Chapters [33–41](#)), will be a 3D physics playground with first- and third-person camera support for a player character with more intricate mouse-aimed movement, jumping, wall jumping, and gravity systems. We’ll explore the possibilities of Unity physics, from detecting objects with raycasts to setting up joints and Rigidbody.

CHAPTER 1

Installation and Setup

Installing software is somewhat simple – download an installer, run the installer, a menu (sometimes called a “wizard”) pops up, you agree to some terms of use, it asks you where you want to install the program on your computer, maybe it offers some additional options, and then it starts installing. Easy, right? So I won’t go into painstaking detail over the installation process. I’ll just show you what to install.

Installing Unity

Unity is frequently releasing new versions with new features, bug fixes, and little improvements. Because of this, they’ve recently come up with what they call Unity Hub. It’s a lightweight little application that lets you install the actual Unity engine, including older versions of the engine. It also lets you manage older versions of the engine already installed on your computer and view all your Unity projects from one place.

Sometimes it’s useful to keep an old version of Unity around even after you upgrade to the latest version. You may want to work on an older project with the same version you started with, in case some new features or changes aren’t compatible with your old project – things change, and sometimes the new stuff breaks the old stuff. Sometimes the old stuff just gets reworked and isn’t valid in a newer engine. In those cases, you might decide to stick to the old version until you finish a project, to avoid spending unnecessary time changing the way you did something to the “new way.”

So we’re going to install the Unity Hub first, and then we can install the Unity engine itself through the Hub. To download the Hub, navigate to this link in your web browser:

<https://unity3d.com/get-unity/download>

Click the button titled “Download Unity Hub,” as shown in Figure 1-1.

A green rectangular button with the text "Download Unity Hub" in white, centered within the button.

Figure 1-1. *Download Unity Hub button*

The Hub installer will begin to download. Run the installer and follow the prompts.

Once the Hub is installed, run it. You may be prevented from getting very far by Unity asking you to accept a license, which may involve creating an account with Unity. This is a little one-time setup that pretty much stays logged in and accounted for afterward. It won't bother you much once it's done – but still, if you're prompted to make an account, don't forget your password and username!

To understand what the “license” is, know that Unity used to have a free version and a Pro version. They restricted some of the features from the free version, and you would have to pay for the Pro version if you wanted to use these features – the fancy stuff, particularly fancy 3D lighting and effects. Then, they simply opened up nearly all the features to the free version, while the paid versions offered mostly miscellaneous things like heightened support, extra resources, and team collaboration tools. Now, they offer three different “licenses” to use with the engine: Personal, Plus, and Pro.

As long as you or the company you represent made less than \$100,000 in gross revenue in the previous year (that's us), then you can use Unity's Personal license, which is free of charge. If your game development career takes off and you start pulling in some money, you'll eventually have to upgrade to Unity Plus (\$25/month) or Unity Pro (\$125/month) to avoid violating the license. But let's not get ahead of ourselves – we're just hobbyists for now, anyway.

You might also be asked to fill out a little survey. It's just some questions pertaining to how you plan on using the engine, where your interests lie, and other “get-to-know-you” stuff like that.

Once you've got a license and an account, you should see an “Installs” tab on the left side of the Hub (see Figure 1-2).

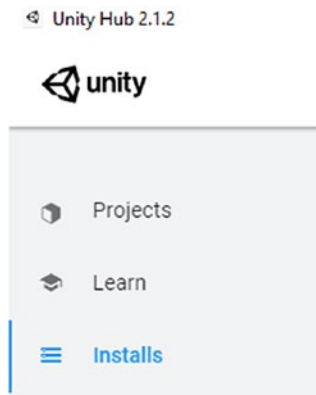


Figure 1-2. Top-left corner of the Unity Hub, with the Installs tab selected

This is where you can see all the versions of the Unity engine you have installed on your computer. You can also install new versions – although installing many versions can quickly use up space on your hard drive, so you may want to uninstall old versions to avoid this. With the “Installs” tab selected, click the “Add” button in the top-right corner. A popup will offer a list of versions you can choose from. The topmost version will be the latest stable release. Click it to select it; then click Next in the bottom-right corner.

You’ll be asked to select what the Unity Hub calls “modules” to install with the engine. These are extra little features to add to the installation, which take up some extra space on your computer if you choose to install them. You can always add modules later, after installation completes, if you find that you want them. Most notable are the Build Support modules, which allow you to build a Unity game project to different operating systems, environments, and hardware.

“Building” a game project is the process of turning it from a Unity project, playable only through the Unity engine, to an actual application used to play the game.

The major platforms that Unity projects can be built to are

- PC, with support for Windows, Mac, and Linux
- Android
- Apple iOS
- WebGL (played in a web browser)
- Xbox One
- PS4

As I said, if you need to build to these platforms down the road (we won't get into them in this book), then you can always install them through the Hub.

You can also select to install the Unity documentation locally as a module. The documentation can be immensely helpful. It's available online as well, and I myself tend to use the online resources, but if you plan on using Unity offline, you may want to install the documentation so it's available without an Internet connection.

For now, you can simply check the Build Support for the operating system you're using, and if you want for the documentation, and uncheck everything else. Then click Done, and the installation will begin. The version you're installing will appear as a box in the main body of the window, and a little bar above it will depict how far along the installation is.

Once it's done installing, the bar will disappear, and you can now create a Unity project with that version of the engine – in a little bit, we'll do that and run the engine for the first time. Another nice feature about the Hub is that it will automatically run the correct editor version of Unity when you open a project (assuming the version is still installed on your computer).

Installing Our Code Editor

You don't write code in the same sort of software that you might write a book or a resume in. Code editors are text editors that are fine-tuned for writing code. They have special highlighting for words and symbols, they know how to format code, and they often come with a slew of features that make it easier and faster for us to write and work with code.

Our code editor of choice is Microsoft Visual Studio Code. It's not to be mixed up with Microsoft Visual Studio. Both are similar (and similarly named) products from the same company, both are free to use, and you could use either one to do the job. They can both edit C# code and integrate with Unity.

Visual Studio Code is designed to be cross-platform and lightweight out of the box, but highly extensible. It has a minimalistic user interface, and most of its features are enabled through installing extensions to add extra functionality, which you do through the software itself.

Visual Studio has support for Windows and macOS, but not Linux. It comes with more features out of the box. It's a very powerful tool and certainly has plenty of uses, including collaboration with teams and other such more advanced features. It is generally "heavier" – more feature-rich but likely to consume more memory and run a little slower.

Code is our choice because I feel it's more suitable for beginners, as it doesn't "get in the way" as much, and most of the advanced features of Visual Studio don't really suit our workflow anyway.

To download Code, head on over to this link in your favorite web browser:

<https://code.visualstudio.com/download>

From there, you can select the correct button to download the software based on your operating system (Windows, Linux, or Mac). The installer should begin to download. Once it completes, run it and follow the instructions it provides.

Once you have Visual Studio Code running for the first time, you'll see a welcome page serving as a hub for various links and resources. Many programmers are picky about the color scheme their code editor uses. I prefer dark schemes myself. Some prefer light – whatever floats your boat. You can easily change it right from the welcome page. Click the "Color theme" button to pop up a list of standard color themes, allowing you to switch to whichever theme you prefer (see Figure 1-3).

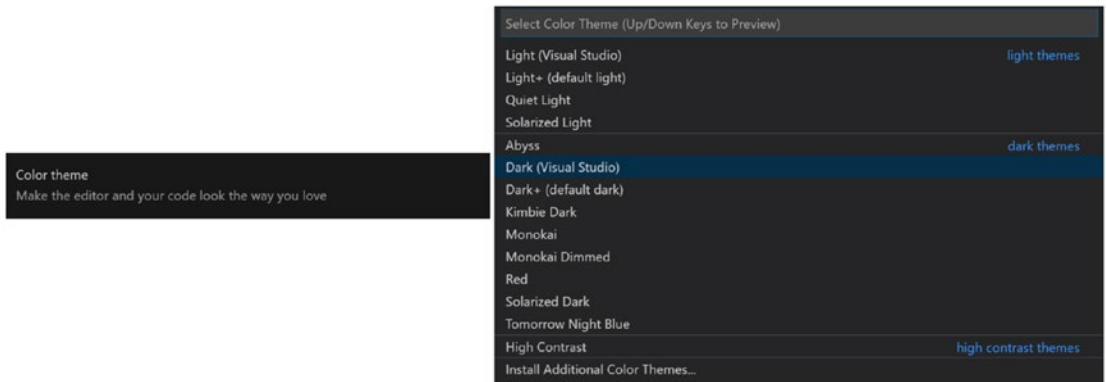


Figure 1-3. Welcome page Color theme button (left) and resulting popup (right)

Once you're satisfied with your colors (you can always change them again later), let's close the welcome page. Any file or page you have open in Code will have a tab at the top left (see Figure 1-4) whether it's an editable code file you're working on or a static page like the welcome page. If you have multiple files open, you can easily switch to view a different one by clicking the tab. Right now, we only have the welcome page open. Let's close it to get a blank slate. You can do this by left-clicking the X button on the tab or by using the Ctrl+W hotkey.

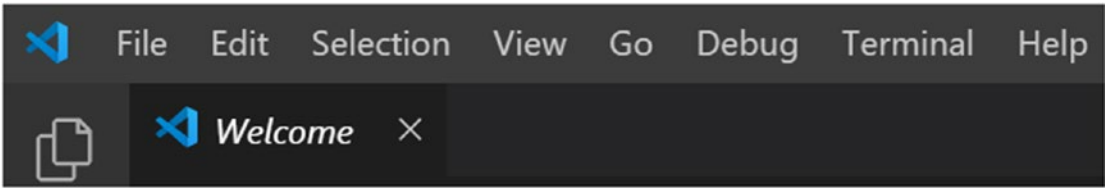


Figure 1-4. The top-left corner of Code, where all page tabs are shown. The welcome page tab is the only one we have open here

Once it's closed, there won't be much going on – just a big blank space, as shown in Figure 1-5.

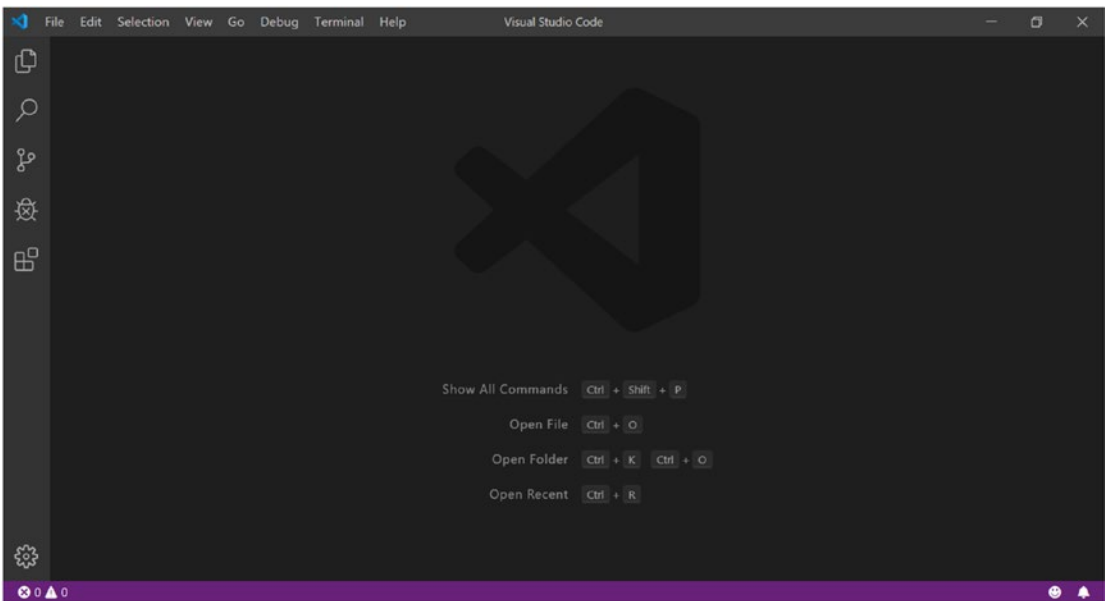


Figure 1-5. View of Visual Studio Code with no files open

You can now install what Code calls “extensions” to add some extra functionality to the editor. Extensions are managed and installed through a button on the left sidebar. The left sidebar has a handful of different forms it can take, all based on which of those buttons you pressed last. Mouse over the buttons to see what they mean and click the Extensions button when you find it (it's the bottom one). You can also press Ctrl+Shift+X to bring it up. This will cause the sidebar to pop up on the left side of the screen. Pressing the button again will fold the sidebar, tucking it away.

Inside the Extensions sidebar, you can search for extensions with the search bar at the top. Click within the search bar and type “C#”. You should see a result simply titled “C#”, with a description “C# for Visual Studio Code.” You’ll also notice that the publisher for the extension is listed beneath the description: “Microsoft,” which so happens to be the company behind Visual Studio Code and C# itself.

Click this extension, and a new tab will pop up, providing details about the extension. Under the main description at the top of the page, you’ll see a button to install the extension. Click that, and the extension will begin installing. If you’re prompted to install any further extensions by popup boxes during this, go ahead and permit them to install.

Next, we’ll install the extension for debugging in Unity. This allows us to “attach” our code editor to Unity, so that we can use the code editor to set up “breakpoints” in the code. A breakpoint is a point in the code that, when reached during the execution of the program, causes the whole program to freeze. While frozen, we can look at pretty much any piece of data from our program that we want and resume whenever we please, among other things. It’s a very handy feature that we’ll use later down the road.

The exact name of the extension is “Debugger for Unity,” published by Unity Technologies. You can find it the same way – by typing the name in the search bar at the top of the Extensions sidebar.

Once you have these extensions installed, we can close Visual Studio Code. We’ll be using it more in Part 2. For now, we’re focused on the Unity editor itself.

Creating a Project

We can now use the Unity Hub to create our first project, so we have an environment to play around in as we learn. In the Unity Hub, click the Projects tab on the left side, and then click the blue “New” button in the top-right corner.

A dialog box will appear (see Figure 1-6), allowing you to select a template to base the project on.

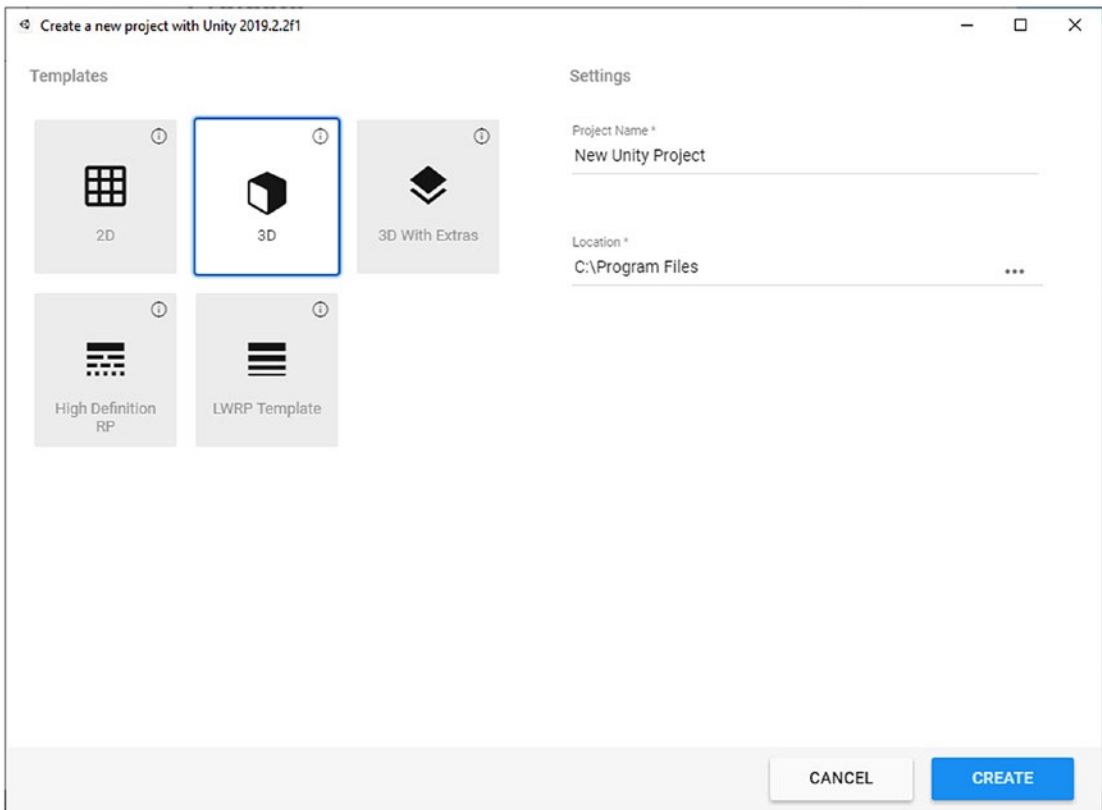


Figure 1-6. *Dialog box for creating a new Unity project*

The template is just a simple starting point for your project. The most bare-bones and simple templates are the first two, titled “2D” and “3D.” These are pretty much blank slates, one set up for 2D and one for 3D.

We’ll start with a blank 3D project, so select the 3D template by clicking it. It will have a blue border around it if it is selected.

To the right side of the template options, there’s a field for the project name (you can change this later) and the directory (file path) on your computer to save the project files to. We’ll name our project “ExampleProject” – note that we aren’t using a space between the two words, because file paths aren’t always fond of them.

You can change the directory to save the project wherever you like. Whatever file path you choose, a folder named after the project will be created inside that folder. That folder is the “root directory” for your project, and all your project’s files and resources will be stored in that folder.

Once you’ve selected the path you want, click the blue “Create” button in the bottom-right corner, and wait for Unity to create the base project files. When Unity finishes, the editor itself will pop up with your brand-new project opened and ready for editing.

Summary

The following is a recap on what we learned in this chapter:

- The Unity Hub program will be used to download new versions of the Unity editor, uninstall old versions you no longer need, create new projects, and open existing projects.
- Opening a project in the Unity Hub will start the Unity editor, which is where we’ll actually use the engine to develop our game.
- A Unity game project is stored on your computer, with all the related files – including stuff we make ourselves like art and code – stored in a “root directory” named after the project name.
- Our code will be written with Visual Studio Code, a text editor designed specifically for writing code. It will offer us useful features that normal text editors don’t have, making it easier to format and navigate our code.

CHAPTER 2

Unity Basics

Now that we have Unity set up and a new project to work with, let's get comfortable with the engine. This is the user interface we will be interacting with quite a lot as we develop our games, after all, so we ought to get to know it early.

Windows

Unity is separated into different windows that serve different purposes. Each window has a little tab at its top-left corner, where the name of the window is written – that is, what type of window it is.

Much like in a lot of computer software nowadays, each of these windows is a separate piece of the program that can be repositioned, resized, or even altogether removed. There are a handful of other window types that aren't being used now, so we don't see them on our screen – but if we ever wanted to use them, we could add them in a jiffy.

You'll notice that if you left-click one of these window tabs, hold the mouse button down, and drag it, the window can be picked up and moved to a different spot in the program. Doing this, you can split one window's space to cover some of it with a different window. If you ever want to do something with a window, chances are you just need to start dragging stuff around and see how Unity reacts.

You can also dock windows beside others to place their tabs side by side (see Figure 2-1). Whichever tab you clicked last will gain focus and fill the space of the window.

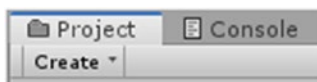


Figure 2-1. Two window tabs docked side by side in the same space. The Project window has focus and will fill the space, but the Console window can be given focus instead by clicking its tab

Unity also lets you save all your current windows and their sizes and positions in a **layout**, which you can assign a name to. This is done with the little dropdown button in the top-right corner of the Unity editor, with the text “Layout” written on it. Click this button to see all of the built-in layouts you can choose from, as shown in Figure 2-2.

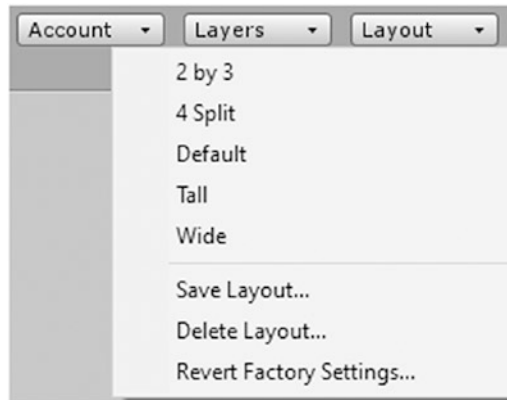


Figure 2-2. The Layout button in the top-right corner of the Unity editor, after it has been clicked to show its options

By default, it will be set to a layout very aptly named “Default.” Try clicking a different layout, and you’ll see Unity automatically restructure all its windows for you. You can also select the “Save Layout...” option in the dropdown list to save your current layout under a new name to keep it around, once you’ve set things up the way you like. That way, if you ever lose a window by accident or accidentally cause some catastrophe to your window setup, you can just reset it to the way you had it by loading your layout with the dropdown.

Layouts can also be useful when dealing with different aspects of game development. Certain activities might be easier to do with a different set of windows in different positions. The ability to save layouts makes it easier for us to hop back and forth between activities – we can do it with just a few clicks.

The default layout has all the most important windows. Let’s go over them to learn about what they do.

Project Window

The Project window shows us all our **assets**. An asset is the game development term for some piece of work we can use in our game – art, sound effects, music, code files, game levels, things like that.