PART V

ONLINE APPENDICES

APPENDIX A

STANDARD PROJECT SETUP PROCEDURE

Many times throughout the book, you are asked to create a new project. This is the standard procedure that you should follow each time to create a new project, set up a scene, prepare the Game pane, and so on. We'll also talk about setting up Unity version control, which I highly recommend. Instead of repeating these instructions throughout this book, they are collected here.

The *Set Up* Sidebar for Tutorial Projects

Near the beginning of all chapters in Part III of this book, you'll see a sidebar like this:

SET UP THE PROJECT FOR THIS CHAPTER

Follow the standard project setup procedure outlined in Appendix A to create a new project in Unity. If you need a refresher on the standard project setup procedure, see Appendix A, "Standard Project Setup Procedure."

- **Project Template:** 3D Core
- **Project Name:** Apple Picker
- **Starter UnityPackage:** Find Chapter 29 at http://book.prototools.net
- **Scene Name:** __Scene_0 (rename the default Sample Scene to *__Scene_0*)
- **Game Pane Dimensions:** Full HD (1920x1080)

Don't forget to set up Unity version control for this project. It could save you a ton of time if anything goes wrong. Also make sure to choose the IGDPD Layout for your Unity window.

This appendix outlines what you need to do to follow these directions.

Setting Up a New Project

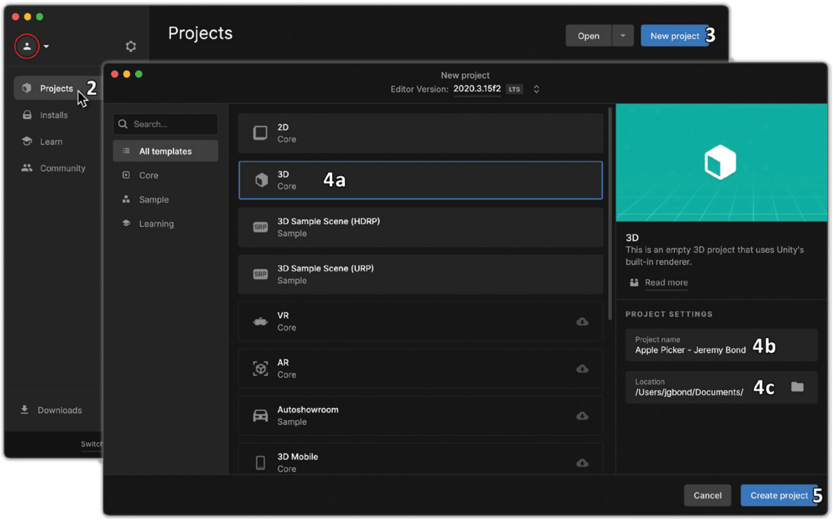
Follow these steps to set up a new project. The procedure is the same for both macOS and Windows:

1. If you haven't yet installed the Unity Hub, please read Chapter 17, "Introducing Unity Hub and the Unity Editor," before attempting the steps here. Chapter 17 leads you through the Unity download process, including:
   - Installing Unity Hub and the latest version of Unity 2020.3 LTS
   - Creating a Unity account
   - Creating your first Unity project
   - Setting up the IGDPD Layout window layout that you see throughout this book

These are all things that you should know before continuing with Appendix A.

Great! Now that you've done all that, let's make a new project (some of this is a repeat of what you read in Chapter 17).

2. Open the Unity Hub. It should open with "Projects" highlighted on the left side of the window, but if it's not highlighted, click *Projects*. (See number 2 in Figure A.1.)



This is an empty 3D project that uses Unity's built-in renderer.

Project name: Apple Picker - Jeremy Bond

Location: /Users/jgbond/Documents/

Figure A.1 Creating your first project in Unity Hub (with numbered steps)

3. Click the *New project* button on the right side of the Unity Hub window.
4. Please configure the *New project* window as it is shown in Figure A.1 (numbers 4a–4c).
   a. **Project Template:** For most chapters of this book, you should choose the ***3D Core*** template unless something different is specified at the beginning of a chapter.

At some point in the future, Unity might get rid of the 3D template and want everyone to move to the Universal Render Pipeline (URP) or High Definition Render Pipeline (HDRP). If that happens, please choose the Universal Render Pipeline. The HDRP is designed for high-end PCs and next-gen consoles, but the URP is generally the better choice for most development.

b. **Project Name:** The Project Name is used for the folder name of this project on your hard drive as well as the name of the project when you set it up for Unity Cloud Services and upload it to Unity version control.

If you're reading this book on your own, the name of the chapter is a fine name for the project. However, if you're in a class with other people, please do your professor a favor and include your name in the project name. This will make it much easier for them to find and grade your work. For example, my project name for the *Apple Picker* project in Chapter 29 would be *Apple Picker - Jeremy Bond*.

c. **Location:** The location that you choose here will be the folder that contains all of your Unity project folders. Click the folder icon on the right side of this field to choose a folder that will work best for you.

You may be tempted to store your Unity projects in a folder that is backed up to the cloud (e.g., a DropBox folder). I *do not recommend* doing this because Unity changes so many files so frequently that in the past, when I put Unity projects in a DropBox folder, it caused issues with having too many files open at the same time. Instead, I highly recommend using Unity version control to back up your projects to the cloud (more info later in this chapter).

4. Click *Create project*.

...and it will probably look like everything crashed. That's not the case; it just takes Unity a little while to launch. You'll see a progress bar for a while as Unity sets up the new project and imports everything to make it work.

When Unity finally gets done setting everything up, you'll see the default Unity Window layout (which I dislike) as shown in the top of Figure A.2. **Please do yourself a favor and switch to the IGDPD Layout** for Unity shown in the bottom half of Figure A.2. You can either download it from this book's website (under Appendix A at http://book.prototools.net) or can read about how to arrange it yourself in Chapter 17, "Introducing the Unity Hub and the Unity Editor."

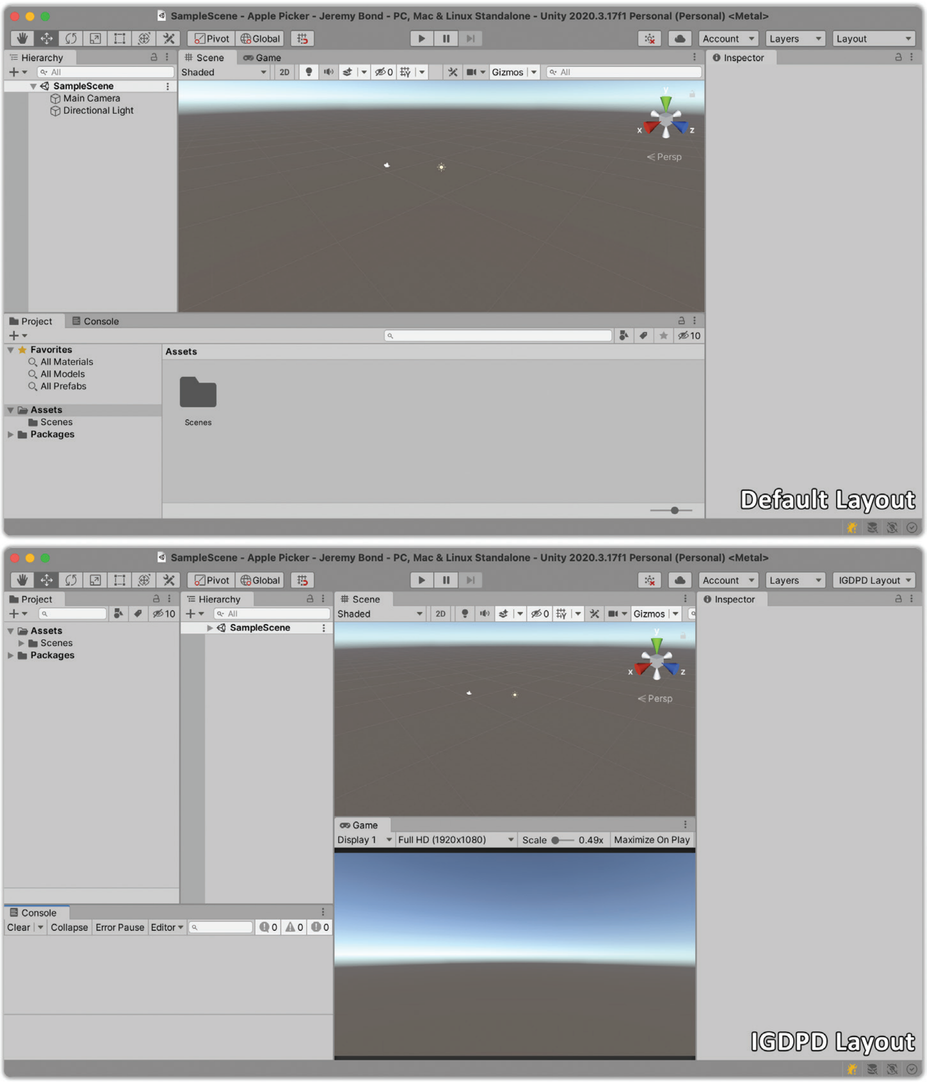


Figure A.2 The default Unity window layout and the IGDPD Layout, the latter being strongly recommended by this book

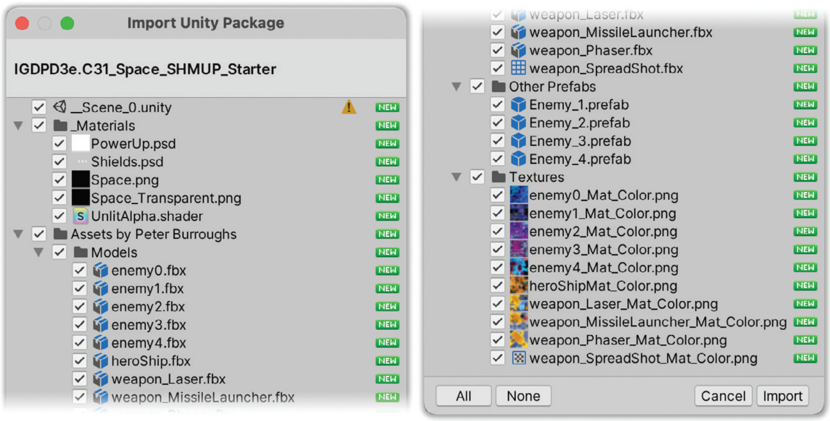
Importing a Starter UnityPackage

The later tutorials in Part II of the book ask you to download a UnityPackage file so that you have some assets to work with. To do so:

1. First follow the URL listed (http://book.prototools.net) and search for the current chapter. On that page, you will find a link to the UnityPackage for the chapter

(e.g., *IGDPD3e.C31_Space_SHMUP_Starter.unitypackage* for Chapter 31, "*Space SHMUP — Part 1*"). If I make improvements to the starter package in the future, you may see a version number at the end of the package name as well (e.g., *…_Starter_v2*).

2. Click the link to download the package to your machine, which will usually place it in your *Downloads* folder.
3. Open your project in Unity and select *Assets > Import Package > Custom Package* from the menu bar. Navigate to the package you downloaded, select it, and click *Open*.
4. An *Import Unity Package* window appears, containing a list of assets. You can see the assets from *IGDPD3e.C31_Space_SHMUP_Starter.unitypackage* in Figure A.3.



```
✓ __Scene_0.unity                         NEW
▼ ✓ _Materials                            NEW
    ✓ PowerUp.psd                         NEW
    ✓ Shields.psd                         NEW
    ✓ Space.png                           NEW
    ✓ Space_Transparent.png               NEW
    ✓ UnlitAlpha.shader                   NEW
▼ ✓ Assets by Peter Burroughs             NEW
  ▼ ✓ Models                              NEW
      ✓ enemy0.fbx                        NEW
      ✓ enemy1.fbx                        NEW
      ✓ enemy2.fbx                        NEW
      ✓ enemy3.fbx                        NEW
      ✓ enemy4.fbx                        NEW
      ✓ heroShip.fbx                      NEW
      ✓ weapon_Laser.fbx                  NEW
      ✓ weapon_MissileLauncher.fbx        NEW

      ✓ weapon_Laser.fbx                  NEW
      ✓ weapon_MissileLauncher.fbx        NEW
      ✓ weapon_Phaser.fbx                 NEW
      ✓ weapon_SpreadShot.fbx             NEW
  ▼ ✓ Other Prefabs                       NEW
      ✓ Enemy_1.prefab                    NEW
      ✓ Enemy_2.prefab                    NEW
      ✓ Enemy_3.prefab                    NEW
      ✓ Enemy_4.prefab                    NEW
  ▼ ✓ Textures                            NEW
      ✓ enemy0_Mat_Color.png              NEW
      ✓ enemy1_Mat_Color.png              NEW
      ✓ enemy2_Mat_Color.png              NEW
      ✓ enemy3_Mat_Color.png              NEW
      ✓ enemy4_Mat_Color.png              NEW
      ✓ heroShipMat_Color.png             NEW
      ✓ weapon_Laser_Mat_Color.png        NEW
      ✓ weapon_MissileLauncher_Mat_Color.png NEW
      ✓ weapon_Phaser_Mat_Color.png       NEW
      ✓ weapon_SpreadShot_Mat_Color.png   NEW
```

Figure A.3 The Import Unity Package window (split to fit on the page better)

You may see image previews as shown for most files in Figure A.3, but it is more likely that models and images will show up with a generic icon (a checkerboard, as shown by the last file in the Textures folders in the figure).

If you see any yellow warning triangle icons (like the one next to __Scene_0.unity in the figure), you can mouse over them, and Unity will tell you what it is worried about. (In this example, __Scene_0.unity has the same GUID (Globally Unique IDentifier) as an existing file (the Sample Scene in this case), so Unity will give __Scene_0 a new GUID on import to avoid any issues).

5. Click the *All* button to check the boxes next to all assets (as shown in Figure A.3), and click *Import*.

This imports all the checked files into the Assets folder of your Project pane. In this example, Unity imports a scene named *__Scene_0*; places four new *textures* and one new *shader* into the *_Materials* folder; and imports many models, textures, and other assets into the *Assets by Peter Burroughs*[1] folder.

Textures are usually just image files. The creation of textures is beyond the scope of this book, but many books and online tutorials cover texture creation. *Adobe Photoshop* is probably the most commonly used image editing tool, but it is very expensive. A very good new commercial competitor is *Affinity Photo* (https://affinity.serif.com/photo), and a common open source alternative is *Gimp* (http://www.gimp.org). But, the easiest image editor to access is *Photopea* (pronounced *Photo-pee*), which is an entirely free, ad-supported online tool that reproduces most of the capabilities of *Photoshop* in a web browser: https://www.photopea.com/.

The creation of shaders like UnlitAlpha.shader is also far beyond the scope of this book. Shaders are programs that tell your computer how to render a texture on a GameObject mesh. They can make a scene look realistic, cartoony, or however else you like, and they are an important part of the graphics of any modern game. Traditionally, Unity used its own unique shader language called ShaderLab,[2] though most recent shader development is done in ShaderGraph,[3] a visual scripting engine for shaders that can produce fantastic results.

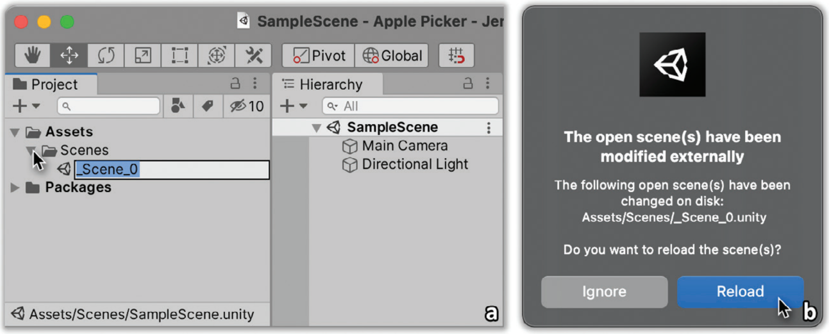
Setting the Scene Name

Most projects will request that you change the name of the SampleScene to __Scene_0 (with two underscores at the beginning of the name). This will help organize things as you eventually make new scenes. By default, a new Unity project will already have a Scenes folder within the Assets folder. To change the name of the scene:

1. Open the *disclosure triangle* next to the *Scenes* folder (under the mouse cursor in Figure A.4a).
2. Select *SampleScene* in the Project pane and then click the name again (or press F2 for Windows).
3. Change the name to *__Scene_0* (or whatever is specified in each chapter) and press Return (or Enter).

1. To learn more about Peter's work, check out his website: https://kairosmith.com/.
2. To learn more about the text-based ShaderLab language, a good place to start is the Unity Shader Reference documentation at https://docs.unity3d.com/2020.3/Documentation/Manual/SL-Reference.html.
3. To learn more about the visual scripting shader language ShaderGraph, check out the ShaderGraph package documentation at https://docs.unity3d.com/2020.3/Documentation/Manual/com.unity.shadergraph.html.

4. Because you changed the name of the scene, a pop-up will appear asking you if you want to reload the scene (Figure A.4b). Click *Reload*.



The open scene(s) have been modified externally

The following open scene(s) have been changed on disk:
Assets/Scenes/_Scene_0.unity

Do you want to reload the scene(s)?

Figure A.4 Renaming the SampleScene

Setting the Game Pane to Full HD (1080p)

The dimensions of the Game pane determine the aspect ratio of any Cameras in the scene as well as the dimensions of the uGUI (Unity graphical user interface) Canvas in the scene. It is very important to lock in your Game Pane dimensions early, because changing them later can cause more work, including requiring you to reposition all uGUI elements.

1. To set the Game pane dimensions, click the *Aspect Ratio* pop-up and choose *Full HD (1920x1080)*, as shown in Figure A.5.

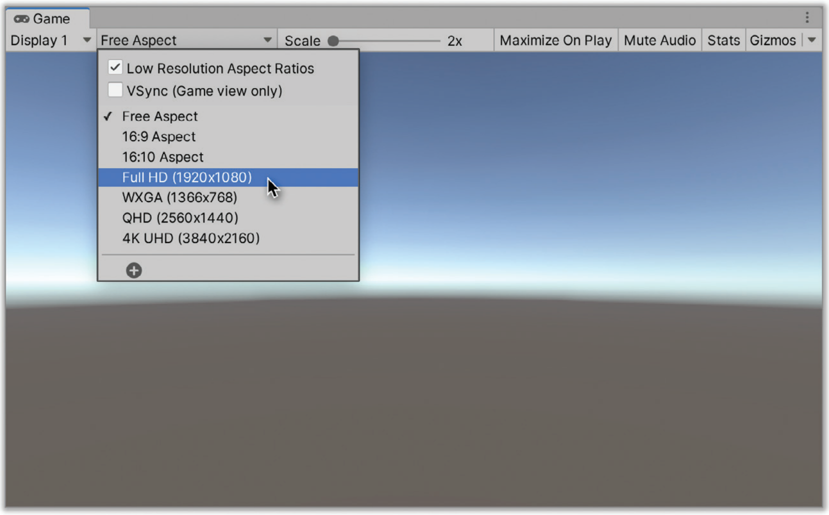


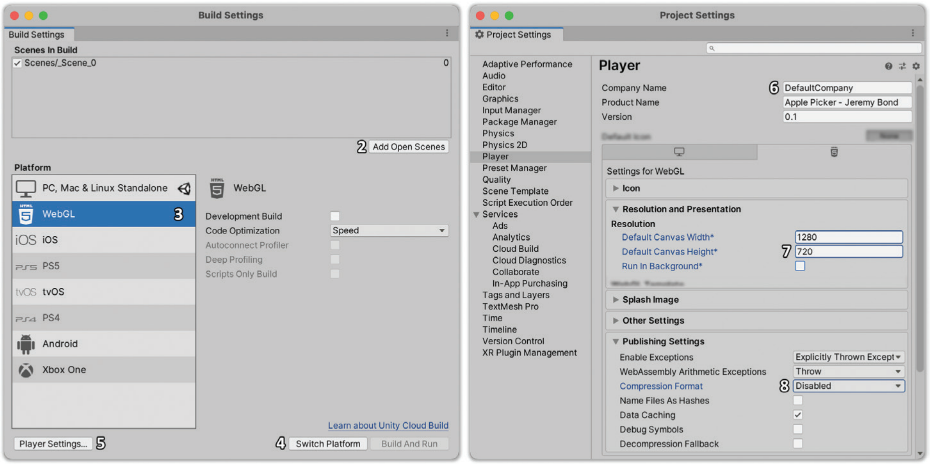
Figure A.5 Setting the dimensions of the Game pane to Full HD (1920x1080)

The *Space SHMUP* tutorial requires different dimensions for the Game pane, but all other projects will work well at Full HD (which is the same resolution as a 1080p television).

Setting Up a WebGL Build

By default, Unity Projects are set up to build an executable for the kind of machine you're using to run Unity (e.g., on Windows, Unity will make a Windows application by default). However, if you want to share your project with other people, the best kind of build to make is WebGL.[4] To switch your Unity to WebGL builds and add __Scene_0 to the build, follow these steps (as shown in Figure A.6):

1. From the main Unity menu, choose *File > Build Settings...*. The Build Settings window appears, showing no scenes in the build. (If there are any existing scenes in the build that you don't want, right-click each scene and choose *Remove Selection*.)
2. Click *Add Open Scenes* to add __Scene_0, which is currently open. It will then appear checked with a number 0 next to it (indicating that it is the initial scene that will be opened) as shown in Figure A.6.



✓ Scenes/_Scene_0 0

WebGL
Development Build ☐
Code Optimization: Speed
Autoconnect Profiler ☐
Deep Profiling ☐
Scripts Only Build ☐

Company Name 6 DefaultCompany
Product Name: Apple Picker - Jeremy Bond
Version: 0.1

Default Canvas Width*: 1280
Default Canvas Height* 7 720
Run In Background* ☐

Enable Exceptions: Explicitly Thrown Except
WebAssembly Arithmetic Exceptions: Throw
Compression Format 8 Disabled
Name Files As Hashes ☐
Data Caching ✓
Debug Symbols ☐
Decompression Fallback ☐

Figure A.6 Build Settings and Player Settings for the projects in this book

3. In the Platform list, click *WebGL*.
4. Click the *Switch Platform* button.

4. WebGL builds are not only cross platform but also significantly smaller than any of the other build types, so they are also the type of build I always ask my students to turn in. For professors and teachers reading this, I usually have my students turn in a WebGL build that I can play. I also require them to add me as a manager to their Plastic SCM organization so that I can see all of their commits to their projects.

Switching platforms will take a little while, but doing so now is *much* better than later. The process has sped up over time, but on a large project that I was developing for iOS and Android in 2014, any time I switched between the two platforms, the switch took more than an hour![5]

5. Click the *Player Settings...* button to open the Player Settings section of the Project Settings window.
6. In the Player Settings pane of Project Settings, set the *Company* Name to whatever you want. Note that the Product Name is automatically set to what you named the project folder earlier in this appendix (as step 4b of "Setting Up a New Project"). This is also where you can set the version number, which becomes particularly important if you want to submit to a store like the Apple App Store (but doesn't matter in the context of this book).
7. Open the disclosure triangle next to *Resolution and Presentation* and set the following:[6]
   a. **Default Canvas Width:** 1280
   b. **Default Canvas Height:** 720
   c. **Run In Background:** false (unchecked)

The Game pane is set to Full HD (1920x1080), but that is too large for a web page on many monitors. 1280x720 is the same aspect ratio as 1920x1080, but it is small enough to fit within a web page on most monitors. 1280x720 is the resolution of 720p televisions.

We set *Run in Background* to false to prevent your WebGL game build from running when its web page is not the focus of your player's computer. If *Run in Background* were checked, the game would continue running, even if the web browser was hidden from view!

All three of these settings have an asterisk (*) next to them in the *Unity Player Settings* window. This asterisk means that the setting is shared between multiple platforms. But not only is it shared between multiple build platforms, **the *Run in Background* setting here also determines whether or not the Unity Editor will pause play mode when the Unity app is in the background**. I always recommend setting *Run in Background* to false. Otherwise, you could accidentally leave Unity playing when you switch away from it and have it using significant computer resources for no reason.

5. The large amount of time to switch platforms is because each platform requires Unity to prepare the assets differently. One of the massive strengths of Unity Cloud Build is that it caches the assets for all build platforms that you ask it to build, avoiding the time to switch platforms that plagues local builds.

6. Note that the Width and Height will be different for *Space SHMUP*, but these are the right settings for any project where the Game pane is set to Full HD (1920x1080).

8. **Open the disclosure triangle next to Publishing Settings and set *Compression Format* to *Disabled*.** *(This is bolded because many of my students miss it.)*

The default Compression Format is Gzip, but web servers must be set up in a certain way for the Gzip compression to be enabled, and many are not. Additionally, if you're trying to run a WebGL build locally on your machine (rather than online from a web server), the Gzip compression won't work on local WebGL builds.

9. Close the *Project Settings* and *Build Settings* windows.

If you are using Plastic SCM version control (which I discuss a little later in this appendix), note that **none of these build settings are saved to Unity version control**. So, unless this changes in the future, you will have to manually set the Build Settings and Player Settings on each person's computer on your team.

Building Your WebGL Build

To actually build the WebGL build, choose *File > Build Settings...* from the main Unity menu bar and then click the *Build* button.

A dialog box appears asking you where to save the build. Unfortunately, on most computers, this defaults to the project folder of your current project, which is *not* where you want to save a build (it will make your project folder much larger for no reason). Instead, choose a place to save it, and name the build something that will make sense later (e.g., *Apple Picker WebGL Build*).

When you do this, you'll actually be naming a folder that contains the several files of a WebGL build:

- **index.html:** The html file that will be loaded by a browser. Though this is necessary, it includes nothing of your actual game.
- **Build:** This folder contains several files that make up both your game and the WebGL version of the Unity engine to run it.
- **TemplateData:** This folder holds all the files and CSS necessary for the index.html page to look nice and work properly when you open it in a web browser.

Unfortunately, opening the index.html file in a web browser is not as easy as it would seem.

Enabling Local WebGL Builds to Run

Due to security concerns, most web modern browsers will refuse to run a WebGL build from a local folder (i.e., any place on your computer). I need to run these builds locally to grade my students' work, so I've found two ways to do it.

Choosing to *Build and Run* a WebGL Build in the Unity Editor

If you're working on a project in Unity and want to play a WebGL build, simply choose *File > Build and Run* from the main Unity menu. If WebGL is your build platform, then Unity will not only make your build but will also start a temporary web server that only exists while you're playing the build that one time. This should work in any browser,[7] but if you close the WebGL window in your browser and then attempt to double-click the index.html file to open it again, it will throw an error and will not play your game in the browser.

Running a Temporary Local Web Server

Creating a temporary web server running on your local machine is the second possible way to view local WebGL builds on your computer. It is surprisingly easy on macOS, but unfortunately, it takes a bit of work on Windows.

Running a Local Web Server on macOS (or Linux)

Because modern macOS is based on Linux, you can do many things from the command line in the Terminal app. One of these is running a web server. Follow these steps to run a web server in the same folder as your WebGL build:

1. Open the Terminal app:
   a. Press *Command-Space* on your keyboard to bring up the macOS Spotlight Search.
   b. Type **`Terminal`** into the Spotlight Search box.
   c. Once you see the Spotlight Search auto-complete to **`Terminal.app`**, you can press *Return* to launch Terminal.

Alternatively, you can find Terminal on most Macs at /Applications/Utilities/Terminal.app.

If you haven't used Terminal before, it might look a bit frightening. There is nothing but text on a blank background. However, you don't need to know much about it to use it for this. To learn more about using the macOS Terminal later, try searching for "introduction to macOS Terminal" online.

2. Back in the Finder, navigate to the folder of your WebGL build.
3. *Right-click* on the folder containing your WebGL build (i.e., not the index.html file but the folder that contains it) and choose *Copy* from the pop-up menu. This will copy the location of the folder to your clipboard.

7. On macOS, you may be asked by the operating system if you want to allow "Node" to accept incoming network connections. Node is the name of Unity's temporary web server (probably because it runs on Node.js), so it's okay to allow these connections.

4. Return to Terminal and navigate to the same folder by following these steps:
   a. In the Terminal, type "**`cd `**" (with a space at the end). In the Terminal, **`cd`** stands for *change directory*.
   b. Press *Command-V* on the keyboard (or *right-click* in the Terminal window and choose *Paste*) to paste the path to your WebGL folder into the Terminal.
   c. Press *Return*.

This will make your WebGL folder the *working directory* (i.e., the active folder in the Terminal). You can type **`pwd`** (i.e., *print working directory*) into the Terminal and press Return to confirm the path to the active folder.

Now that you have the Terminal pointing to the right folder, it's time to start the HTTP server.

5. Type the following into the Terminal: **`python -m SimpleHTTPServer 8000`**
6. Press *Return*.

This uses the built-in programming language Python to run a simple web server for the folder in which you ran it. This is running locally on your computer, so you can access it by:

7. Opening a web browser and typing the following into the address bar at the top:

   `http://localhost:8000`

And that's it! You should now see your WebGL build running in your browser. To quit the server, either switch to the Terminal app and press Control-C, or just quit the Terminal app.

Running a Local Web Server on Windows

Unfortunately, running a temporary web server on Windows is significantly more complex, and it changes based on which version of Windows you are running. The easiest thing to do is to

1. Search the Internet for "windows python web server" and follow the directions there.
2. Once Python is set up and you can run a SimpleHTTPServer, follow the same directions as macOS, except that the command you enter into Windows PowerShell to run the web server differs based on the version of Python that you install:
   - **Python 2:** The command is **`python -m SimpleHTTPServer 8000`**
   - **Python 3:** The command is **`python -m http.server 8000`**

Other than that, the steps should be the same.

Posting a WebGL Build to a Web Server

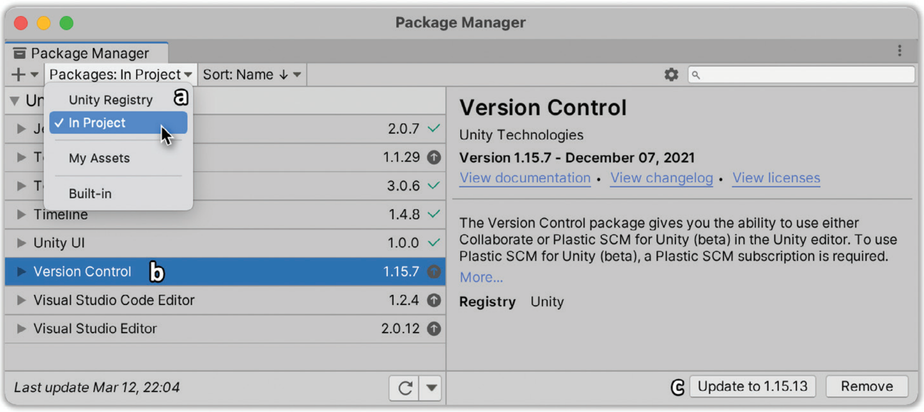
Of course, you can also post your WebGL build to a web server and view it online (like the browsers want you to). Just be sure when you do that you upload the entire build folder (not just the index.html file). If you're going to do this, I recommend that you also remove all the spaces from your WebGL build folder name. Online, spaces become "%20", so if I did not remove the spaces from the example build folder name I mentioned earlier, it would become *Apple%20Picker%20WebGL%20Build*, which is kind of ugly.

Understanding Unity Version Control

In software development, version control (sometimes called source control management) is a way to consistently back up your project to the cloud and work with other developers on the same project without accidentally overwriting each other's work. For several years, Unity had an in-house version control system called Collaborate that integrated nearly seamlessly into Unity. Then, in early 2022, Unity announced that they were automatically switching all customers' projects from Collaborate to Plastic SCM, a more robust source control management solution that Unity had recently purchased.

Unfortunately, at the time of writing, the transition to Plastic SCM is complete but Plastic is not yet integrated into Unity as well as Collaborate was, and there are still several changes happening with every new version, so it is not possible for me to write reliable instructions here for you to set up your projects in Plastic.

What I can tell you is that you will need to make sure you have the latest version of the Version Control package from the Package Manager to take advantage of Plastic. The steps to do so are illustrated in Figure A.7.



Version Control
Unity Technologies
Version 1.15.7 - December 07, 2021
View documentation • View changelog • View licenses

The Version Control package gives you the ability to use either Collaborate or Plastic SCM for Unity (beta) in the Unity editor. To use Plastic SCM for Unity (beta), a Plastic SCM subscription is required.
More...
Registry Unity

Figure A.7 Ensuring that the Version Control package is up to date

To ensure your Version Control package is up to date:

1. Open your project in Unity.
2. From the primary Unity menu, choose *Window > Package Manager* to open the Package Manager window. It will take a moment for the window to load all of the packages.
3. The Packages menu at the top-left of the Package Manager window (Figure A.7a) should show *Packages: In Project*. If it does not, click the menu and choose *In Project*, as shown.
4. You should see that Version Control is already in the list on the left (Figure A.7b), showing that it is installed in your project. However, if you *do not* see it in the list, you need to install the Version Control package from the Unity Registry:

Only follow steps 4a–4c if the Version Control package is not already in your project.

   a. Click the *Packages: In Project* menu (Figure A.7a) and choose *Unity Registry*. This will now show all packages that are created by Unity.
   b. Scroll down in the list and select *Version Control* (Figure A.7b). (The version number you see will not match the one in Figure A.7.)
   c. Click the *Install* button in the bottom-right of the Package Manager window to install the Version Control package. It can take nearly a minute for the package to install and for Unity to recompile all C# code (including the new package). Once this is complete, Version Control is installed.

5. If you see the Version Control package *Update* button (Figure A.7c), click it to update to the latest Version Control package. With Plastic integration still in progress, it can help considerably to have the latest version.

To find the latest information on how to use Plastic SCM in your projects, I recommend going to the Unity web page for Plastic: https://unity.com/products/plastic-scm. There, you can find introductory videos and tutorials that should be up to date with current best practices.

Please don't let any of this information discourage you from using version control in your projects. Even throughout the transition period, Plastic SCM has been drastically better than not having version control and has worked significantly better for my students than more generic solutions like GIT. Plastic will only get better, and by the time you read this, it should be very stable and smoothly integrated into Unity.

I'll end with a quick anecdote about why I use version control. In 2008, I was living in San Francisco, working for Pogo.com at Electronic Arts, and continuing support and development of my grad school team's *Skyrates* game project. At the time, I had been working alone on a complete replacement of the combat in the game (replacing the old ActionScript 2 version with one that ran in ActionScript 3). Disaster struck when my laptop—my only computer—was stolen from my car. It had been at least three months since I had backed up my computer, and all of that work was lost—except for my work on *Skyrates*. I had been faithfully committing my *Skyrates* work to version control at the end of any time I worked on the game. As a result, even though I was working solo, and even though my work had not yet been integrated into the main project, I lost *absolutely none* of that work when my computer was stolen. Once I got my hands on a new computer, I was able to pick up *Skyrates* exactly where I left off. If I hadn't been using version control, I would have lost three months of work on the project and may have abandoned the combat rewrite altogether.

Version Control Best Practices

These are some practices that I **strongly** recommend considering as you start using Plastic or any other version control:

- **Publish commits *frequently*:** Don't work all day and then commit at the end. *Any* time you complete something significant (or even an important step of something significant), please publish a commit. At the very least, you should publish a commit any time you stop working on the project.
- **Make detailed use of the Comment field:** Looking at the Changesets tab of the Plastic SCM pane, you can see that each commit has a number, date, and person who created it, but you must click on each commit to get any information about how many files were changed or which files they were. This means that if you're looking over a dozen or more commits, the **only** way to easily differentiate them from each other is the comment that you wrote. Please do your future self a favor and **write detailed comments** with every commit.
- **Be aware of Plastic SCM storage space limitations:** By default, you should have at least 5GB of Plastic SCM cloud storage with a free Unity account; however, Plastic stores every asset of every commit of your project (rather than storing the difference between commits, as Collaborate did). This means that your space could fill very quickly, and if you exceed that 5GB, Unity will bill you for the overage. However, Unity has committed to sending you a statement at least one week before you are actually billed so that you have time to clear out old commits before you are charged for the usage. You should eventually be able to monitor your Plastic usage on the Unity Dashboard (https://dashboard.unity3d.com).

Summary

If you follow these steps every time you create a new Unity project for this book, you'll be much happier as you work through the book's tutorials. I require my students to go through these steps for all projects they make for me, and it avoids a lot of headaches later.