Lecture 2
A First Look at Unity

98-127: Game Creation for People Who Want to Make Games (S20)

Written by Adrian Biagioli

Instructors:

Adrian Biagioli
(abiagiol@andrew.cmu.edu)

Woody McCoy
(mwmccoy@andrew.cmu.edu)

Carter Williams
(ncwillia@andrew.cmu.edu)

Sebastian Yang
(yukaiy@andrew.cmu.edu)

1 Objectives

By the end of this lesson you will be able to:

- Understand what a Game Engine is, and why we use the Unity game engine
- Utilize common Unity editor functionality and navigate scenes
- Understand Unity's high level architecture: the Scene Graph and GameObjects
- Utilize the inspector to design levels, given art assets and scripts

These lecture notes were written for **Unity 2019.2.3f1**.

2 What is a Game Engine?

Video Games are a very complicated technical challenge. Think about all of the stuff that goes on in your average video game, like Call of Duty. There is a complicated *Graphics Pipeline*, which is the code that communicates very quickly with the user's graphics processor to produce complicated effects (shadows, explosions, hair, muzzle flashes... the list goes on). There is also usually a *Physics Engine* that attempts to simulate how objects would interact with each other (for example, it would figure out how a grenade bounced throughout the scene). There's also some sort of *Audio Mixer* that handles audio effects (for example, a low-pass filter when you are hit with a flashbang). Then there's the *Animation Engine*, which allows characters to move around the scene realistically. Many games include *Artificial Intelligence*, *UI Systems*, and more.

That's a lot of things that *many* games have in common! In fact, it seems like a waste of time to re-do all of these complicated systems for every game that we want to make. This is why Game Engines exist. A **Game Engine** aims to simplify game development by doing all the hard stuff for us (i.e. everything I mentioned in the previous paragraph). This allows game designers like ourselves to focus on the good stuff: how to make our game fun. In addition to this, game engines also tend to simplify *how we code* our games. Many "game-engine-less" games are simply written in C++, which can be hard for beginners. In contrast, Unity is in C#, which is much more approachable.

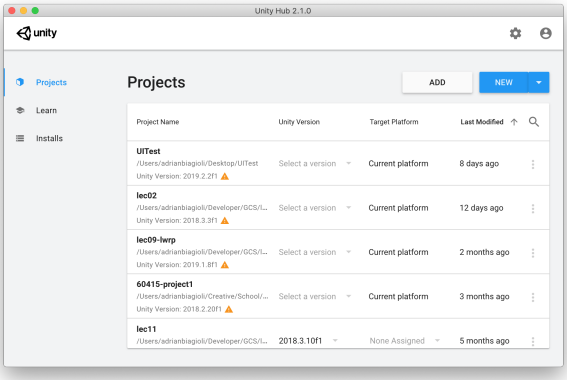
Unity is the most popular Game Engine out there right now for indie game developers, and it is the one that we will use in this course. Right now, Unity's biggest competitor is the **Unreal Engine 4** (UE4), which is developed by Epic Games (creators of *Gears of War* and *Fortnite*). UE4 rivals Unity in many respects and outshines it in some (many will argue that UE4 has better builtin graphics than Unity), but we will not use it in this class because it is generally harder to use for beginners and requires knowledge of C++. Another important difference is how you need to pay royalties if you decide to sell your game: Unity allows you to sell your game without paying any royalties, *until* your game makes $100k per year. After that point, you need to upgrade to Unity "Plus" or "Pro", which has a monthly fee. Unreal requires a 5% royalty of all revenue of your game past $3000, but has no Plus or Pro tier.

3 Getting Started with Unity

If you haven't already, download Unity:

- You can download Unity at https://store.unity.com/download. Accept the license terms and click "Download Unity Hub". The **Unity Hub** is an application that you can use to browse Unity projects and install new Unity versions. To install a new Unity Version, navigate to Installs > Add in the Unity Hub interface.
- If you're a programmer running Windows or Mac, I would recommend downloading Microsoft Visual Studio. Visual Studio and Visual Studio for Mac both have great integration with Unity and are better than the default MonoDevelop: https://www.visualstudio.com.

When you open Unity Hub, you will first see the **project selection screen**. You may need to create a Unity account—do so if prompted. Click "new project" on the top right of the window.

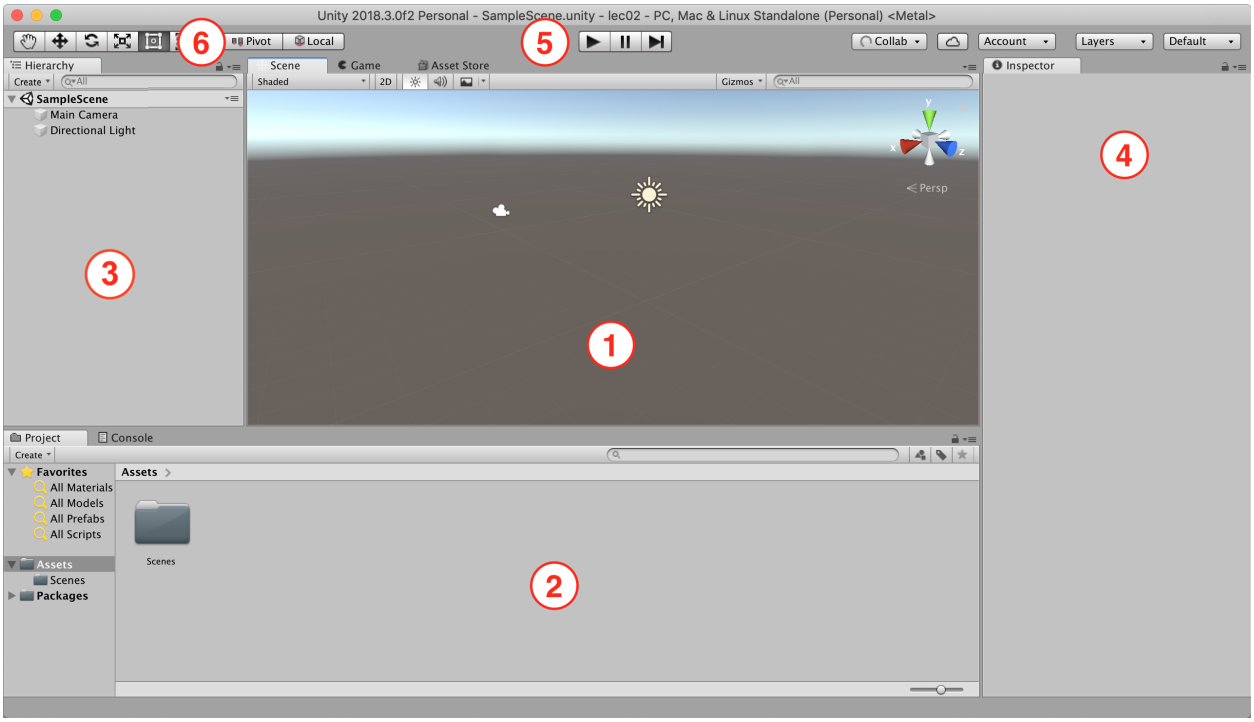


Unity Project Select Window

Enter a name for your new project ("Lecture 2 Project") and specify a location for the project folder. Make sure "3D" is checked. Then click *create project*. **Unity projects are saved in folders on your machine**, so if you browse to the project location you gave then you will find the Unity project files.

The Unity editor should now appear. The editor is divided into **panels**, each providing a different way to interact with your Unity project. If your editor doesn't look like the one on the next page, click Layout > Default on the top right of the editor. Notice that panels can be reorganized and redocked to other panels by clicking and dragging on the tab at the top of each pane. You can also resize panels and drag them out to their own window.

4 Basic Unity Editor Panels



This is the default configuration of the Unity editor, opening a freshly-made project. Each panel that you see in the above screenshot is labeled as follows:

1. The **Scene View** allows you to look into the game world from a perspective other than the in-game camera. You can navigate the scene view as follows:
   - **Right Click + Drag**: Look around
   - **Right Click + WSADQE**: Move camera (FPS controls). Hold shift to move faster.
   - **Left Click**: Select and manipulate objects (see item 6 below)
2. The **Project Window** is a view into the assets of your game. **Assets** are normal files that your game will use like scripts, 3D Models, textures (images), audio, animation data, scene files, and more. The project window behaves much like the Finder on macOS or Explorer on Windows.
   - In fact, these files are located in the `Assets` subfolder of your Unity project! Try right clicking on a file in the project window and select "Reveal…"
3. The **Hierarchy View** lists all of the **GameObjects** that are currently in the scene.
   - In Unity terminology, a **GameObject** is any "thing" in your game. *Everything is a GameObject*: the player, a fence, an enemy, a weapon, a button, an NPC, a scoreboard, ... Right now there are two GameObjects in the scene, a Main Camera and a Directional Light for the Sun.
   - You can select a GameObject by clicking on it: either in the scene view or the hierarchy view.

- If you double click on a GameObject in the hierarchy view you can focus on it in the scene view.

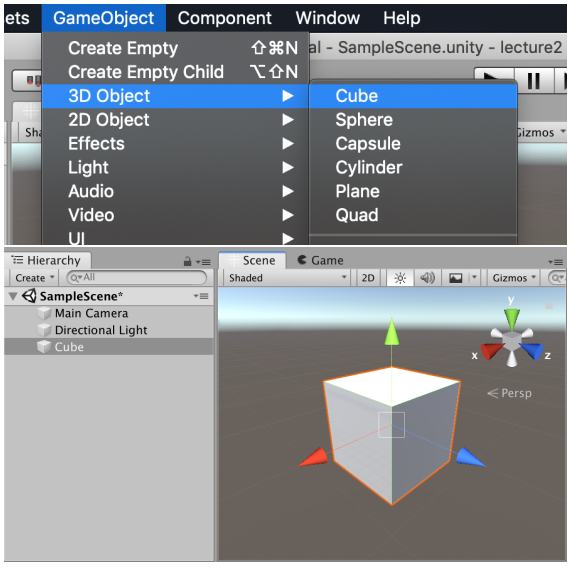
4. The **Inspector** allows you to inspect and modify the properties of whichever GameObject or Asset is selected. For example, you can change the position of a GameObject or the name of a file. More information on how to use the Inspector will follow.

5. The **Play Button** (▶) will shift focus from the Scene View to the **Game View**, allowing you to play your game in real time! When you are in "Play Mode", you can switch back to the Scene/Hierarchy/Inspector Panels and continue to modify the scene. However, as soon as you stop playing (by clicking the play button again) your changes will revert to before you started playing, so be careful! While playing, you can also click the **Pause Button** (⏸) to pause execution of the game (you can still make changes in the Scene View / Inspector while paused). While paused, you can click the **Advance Frame Button** (⏭) to advance by one frame.

6. The **Scene View Tools** change the currently-active manipulator in the scene view. These manipulators allow you to move, rotate, and scale GameObjects–more on this later.

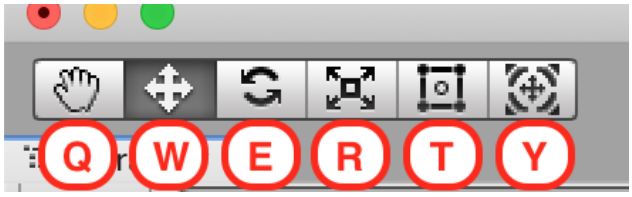
5 The Scene Graph and GameObjects

When you first open Unity, the default scene (Called `SampleScene`) is loaded. In Unity, a **Scene** can be considered a "Level" or a "Screen" in your game. Scenes are saved to a .scene file in your Assets folder. As mentioned above, there are two GameObjects currently in the scene: a Main Camera and an Directional Light. If you don't see any GameObjects in the hierarchy view when you load the project, click on the small triangle next to `SampleScene` in the hierarchy view to expand it. Let's add a third GameObject in the scene; navigate to GameObject > 3D Object > Cube as shown on the right, and a cube should appear in your scene. Notice the new entry "Cube" in the scene hierarchy view.



5.1 Manipulating GameObjects with Scene View tools

The Scene View Tools are located at the top left of the Unity window and are accessible at any time. You can use the Q W E R T Y keys as a shortcut:



Here are each of the manipulators, from left to right:

1. The **Pan Tool**  allows you to click and drag on the Scene View to pan the camera left / right / up / down relative to where you are looking. You do not need to have the Pan Tool selected to right click + drag to move the camera, however.

2. The **Move Tool**  shows a widget when selecting any GameObject that allows you to translate the GameObject throughout the world. Try moving the cube that you made left and right inside the scene. You can hold ⌘Cmd on Mac or ctrl on Windows to snap GameObject translation to regular increments; navigate to Edit ⟩ Snap Settings... to customize snapping settings.

3. The **Rotate Tool**  shows a similar widget that allows you to rotate GameObjects. You can click and drag on the Red/Blue/Green circles to constrain rotation to one axis, or you can click and drag elsewhere on the widget to rotate the GameObject freely. Again, you can hold ⌘Cmd on Mac or ctrl on Windows to snap rotation to regular degree increments. As with translation you can customize snapping via Edit ⟩ Snap Settings... .

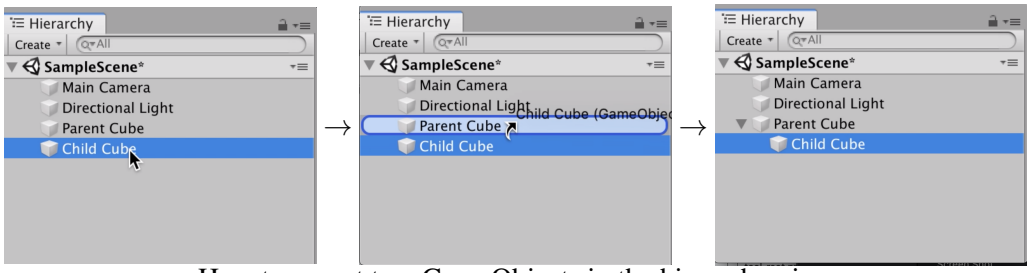
4. The **Scale Tool**  similarly allows you to scale objects in any dimension. You can scale in all dimensions by clicking and dragging on the white box in the center of the widget. Snapping can be used with the scale tool as well.

5. The **Rect Tool**  is generally used in place of the other transformation tools for 2D games and UI elements. It allows you to easily move and scale rectangular objects in the X and Y dimensions.

6. The **Transform Tool**  , added in a recent version of Unity, combines the Move / Rotate / Scale tools into one multi-purpose widget.

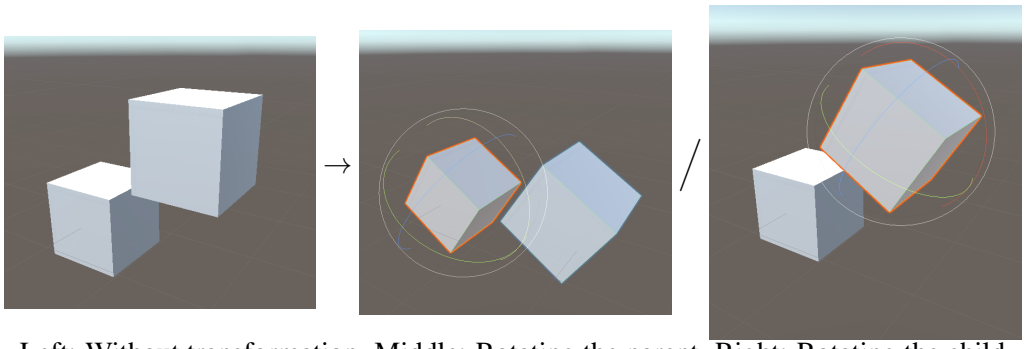
5.2 Parenting GameObjects

Two GameObjects can also be **Parented** to each other. Any transformations that are applied to the parent are also applied to the children. This is useful very often - for example, you might have a sword GameObject that is the child of a character's hand GameObject. You can parent two GameObjects by clicking and dragging one GameObject on top of another in the Hierarchy view.



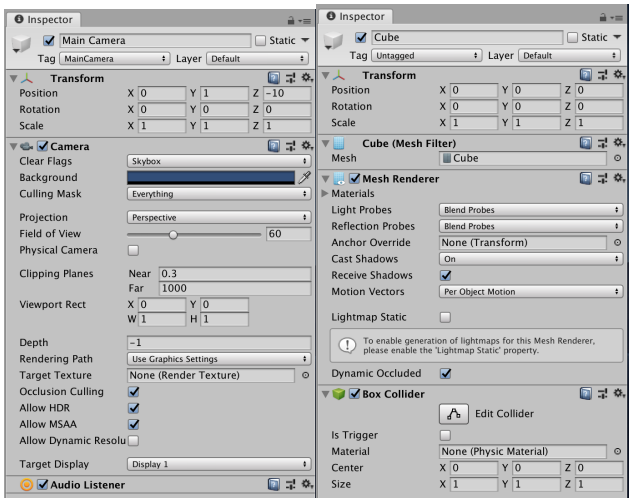
How to parent two GameObjects in the hierarchy view

After two GameObjects are parented, then manipulating the parent will similarly manipulate the child, as shown below:



Left: Without transformation. Middle: Rotating the parent. Right: Rotating the child

6 Composing GameObjects with Components

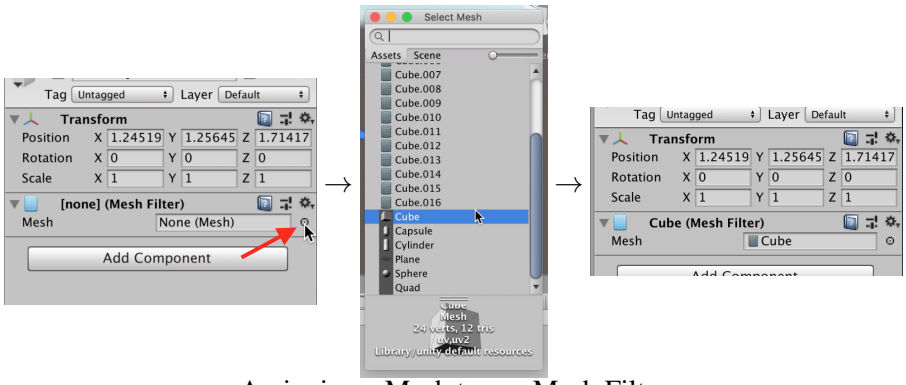


You may have noticed by now that as you select GameObjects in the scene, the Inspector panel (on the right) updates depending on the currently-selected GameObject. Let's take a closer look at the Inspector now. Select the Cube GameObject that we created earlier. At the top of the inspector there is a block of *transform attributes* – these are shared by all GameObjects. When you use any of the transformation manipulators in the Scene View Tools, these values will change accordingly. You can also edit these values (for the position, rotation, and scale of the object) in the inspector. Below the transform attributes there are 3 headings: *Mesh Filter, Mesh Renderer,* and *Box Collider*.

Now click on the *Main Camera* GameObject and take a look at the Inspector. As mentioned, the transform attributes are still there; you can move and rotate the camera as normal (scaling the camera simply does nothing). However the Mesh Filter, Mesh Renderer, and Box Collider have been replaced by a two blocks: *Camera* and *Audio Listener*.

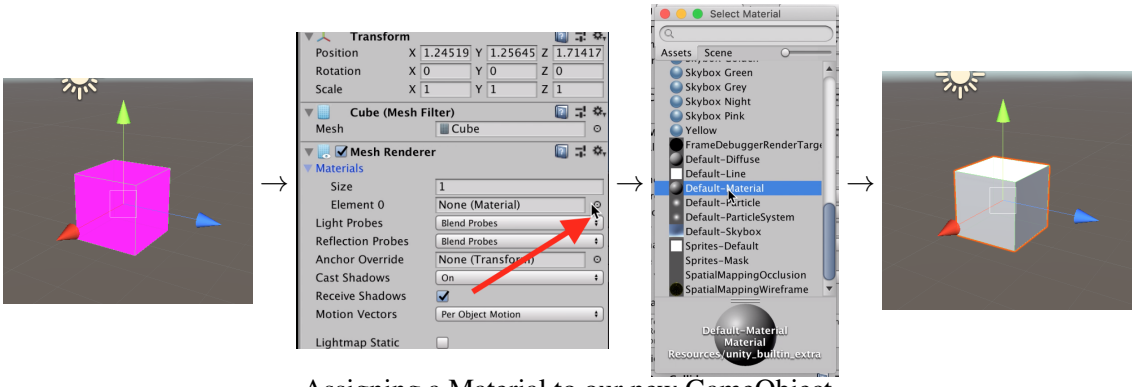
These blocks represent the different Components of each GameObject. A **Component** is the fundamental building block of GameObjects in Unity – *all* of the behavior of a GameObject is performed by its components. For example, the behavior of our cube is encapsulated by its components: the *Mesh Filter* and *Mesh Renderer* define what 3D model should be rendered and how, and the *Box Collider* allows the cube to interact with Unity's physics system. Similarly, the *Camera* allows one to use a GameObject as a viewport and *Audio Listener* enables 3D positional sound. Components are also configurable – try fiddling with the Field of View of the camera for example.

The objects that we have looked at so far were made for us by Unity's interface, however you can build GameObjects "from scratch," by starting with an empty GameObject (an **Empty GameObject** is one without any components). Create an empty GameObject via GameObject > Create Empty and select it in the hierarchy view. Now click the Add Component button, search for *Mesh Filter*, and hit Enter. You should see a *Mesh Filter* added to your GameObject.



Assigning a Mesh to our Mesh Filter

Now try adding the *Mesh Renderer* and *Box Collider* components. You will notice that the Cube now appears in the scene, but it is an ugly purple color. This is because we need to add a Material to the cube. A **Material** is a special type of Asset that defines how a 3D model should render. Let's assign the default material to this mesh – you can do this by expanding the Materials menu and assigning the default material as shown below:



Assigning a Material to our new GameObject

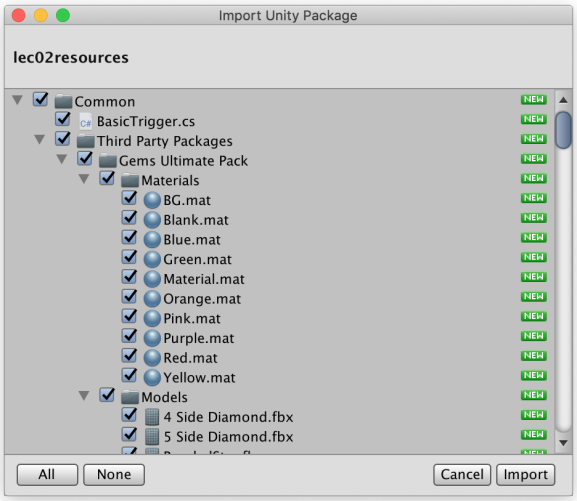
You can create your own material in the Project view. Navigate to Assets > Create > Material, name the material in the Project view, and tweak the material in the inspector. Then you can apply the material as before. Most of the time, we won't need to compose GameObjects manually like this. However, it is very important to understand that all of Unity's builtin functionality is simply exposed through components!

7 Importing Asset Packages

Let's go ahead and import the base assets for this lecture. You can find the base assets at the link below:

http://stage.gamecreation.org/StuCo/F19/packages/lec02resources.unitypackage

This is a .unitypackage file. Unitypackages are a file format specifically made to import project settings and assets into Unity. With your new project open, double click on the downloaded file to open it in Unity. You should see the window to the right. Click "All", then "Import" to import the files.

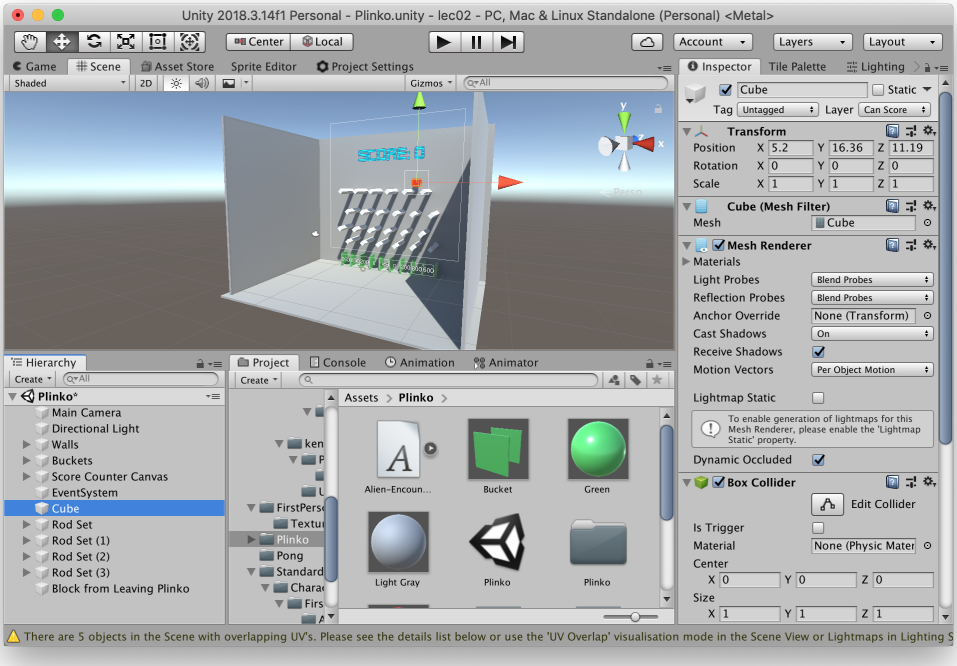


Unitypackage Import Dialog

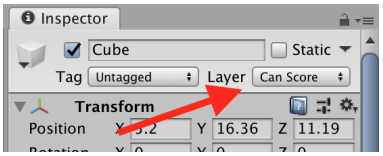
7.1 Using the Unity Asset Store

The *Unity Asset Store* is an online store for .unitypackages. If you don't have access to a larger team, you can use the asset store to get art, scripts, and Editor tools. In fact, the lecture resources (linked above) contain assets from a few asset store packages (the Unity Standard Assets, a 3D Gem Models pack, and some textures for prototyping). You'll need a free Unity account to use the Asset Store.

8 Example Game 1: Plinko



After importing the lecture assets, navigate to `Assets` ▸ `Plinko` ▸ `Plinko.scene` in the project view and double click on the scene file to open it. Press play, and you will see a red cube fall between many pegs into buckets at the bottom of the screen. When the cube falls in a bucket, the score is incremented. Try moving the cube around before playing – notice that it realistically collides with the world and moves in a physically-plausible way, no matter where it starts.



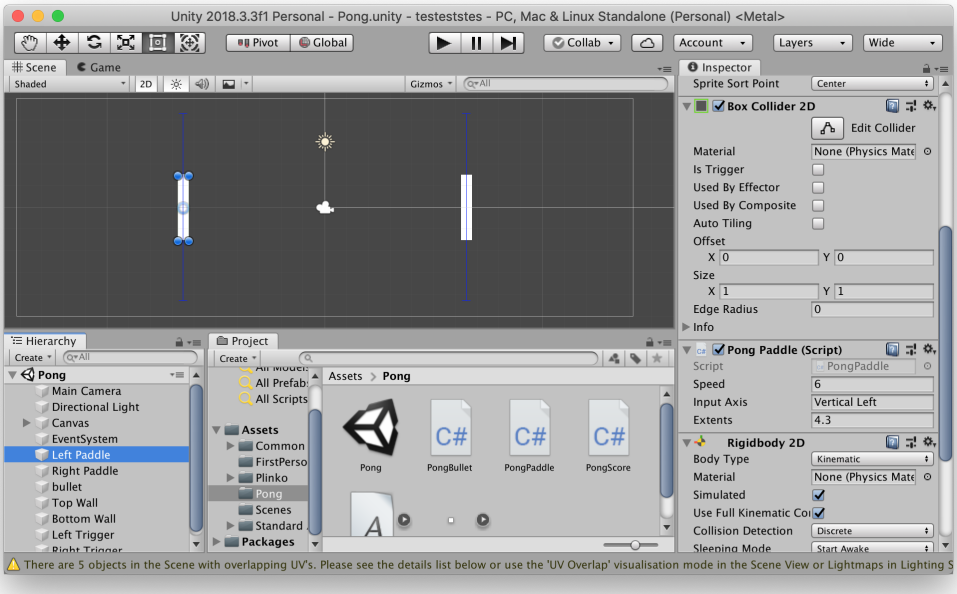
Changing GameObject layers

There are two things that are particularly important about the red cube that enable this simple game. First, the Cube has a Rigidbody component attached. The **Rigidbody** component tells Unity to simulate that GameObject via the physics engine. Unity uses the Collider components attached to the GameObject (or its children) to determine how the object interacts with the world (this cube has a Box Collider attached). Second, the Cube is marked with a layer called *Can Score*. All GameObjects belong to a **Layer** in the scene, and Layers can be used when scripting objects in your game. In this example, all GameObjects in the *Can Score* layer will increase the score when dropped into a bucket. All other GameObjects will not be affected.

In the hierarchy view, search for "Trigger" and click on one of the GameObjects. These are the logical GameObjects that help facilitate scoring. There is a Box Collider here as well, with the "Is Trigger" box checked. "Is Trigger" signals to Unity that this collider shouldn't actually be included in physics calculations (that is, nothing should be blocked by it), but it should still receive collision events for scripting purposes. The other component, called *Basic Trigger*, receives these collision events and passes them on to the scorekeeper script. The "Layer Mask" in *Basic Trigger* is set to *Can Score*, as mentioned. Importantly, the Basic Trigger component is a **Custom Script** created specifically for this game. This is how scripting works in Unity; developers simply create their own custom components!

Now search for the GameObject called "Score Display Canvas". A **Canvas** component allows you to create UI elements such as text or buttons. "Score Display Canvas" contains a Canvas component (as well as some other Canvas-related supporting components) and a script called *Score Counter*. The Score Counter script updates the text at the top of the scene.

9 Example Game 2: Pong

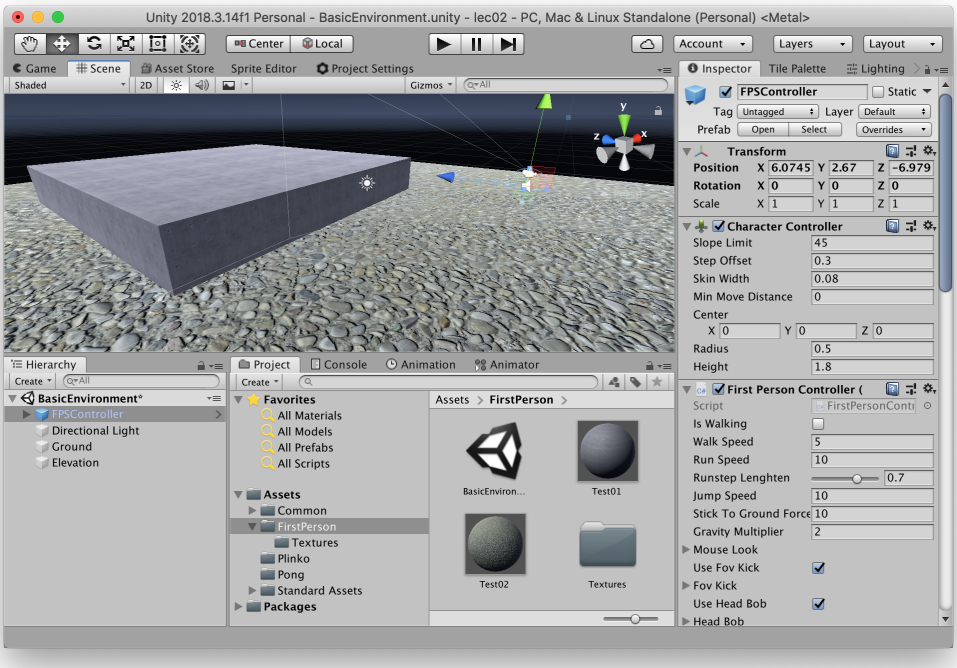


The second example scene is classic Pong! You can try it out by opening the scene at `Assets ▸ Pong ▸ Pong.scene`. You can control the P1 paddle with the W and S keys, and the P2 paddle with ↑ and ↓. This game is an example of a 2D game in Unity; 2D games in Unity are created in largely the same way as 3D games. The main difference is that 2D games tend to use 2D-centric components and the camera ignores the Z dimension. Click the 2D button at the top of the Scene view to toggle between 2D and 3D editing modes. It might also be helpful to switch to the Rect Tool when working in 2D.

Let's compare the GameObjects in this scene with those in the Plinko example. As in Plinko, there is a Canvas GameObject that is used to display the score. A custom component (called *Pong Score*) is attached to the Canvas and handles the score logic. Now inspect the "Left Paddle" GameObject. The components here are different than with the default 3D cube, but largely similar. Instead of the Mesh Renderer that is used to render 3D objects, 2D GameObjects use a **Sprite Renderer** to draw 2D sprites. Unity actually has two different physics subsystems, one for 2D and one for 3D, so all 2D objects use special 2D physics components. Instead of a Rigidbody and Box Collider we use a **Rigidbody 2D** and **Box Collider 2D**.

The *Pong Paddle* script is used on the "Paddle Left" and "Paddle Right" GameObjects to facilitate moving the paddles when pressing the corresponding keys. Try changing the *Speed* attribute to change the movement speed of each paddle. Similarly the *Pong Bullet* script enables the ball/bullet object to move and bounce correctly off of each paddle and the screen boundaries.

10 Example Game 3: First Person Game



The final example isn't a complete game like the previous ones, but it is a good introduction to building expansive environments in Unity.

10.1 The First Person Controller

The **First Person Controller** is part of the Unity Standard Assets and makes it incredibly easy to get a First Person game up and running. We are using the First Person Controller in this scene - check out the GameObject called "FPSController". The **Unity Standard Assets** are a package created by Unity that contains a number of often-useful components and assets (you can download them here). Try changing the parameters of both the *Character Controller* component and the *First Person Controller* component such as Walk/Run speed and height.

To clarify, the *Character Controller* component is built in to Unity and can be used by scripts to move a character GameObject around the scene. The *First Person Controller* component is a script that comes with the Unity Standard Assets that allows the player to look around with the mouse and move with W S A D controls. The "FPSController" GameObject also contains an *Audio Source* component to generate footstep sounds and a *Rigidbody* to receive physics events.

11 Exercise

Design one simple modification to any of the three demos that we went over today, and implement it in Unity. Some ideas include:

- Modifying Plinko by adding more obstacles, so that it is harder to score well
- Modifying Pong so that the ball's behavior is more complex (it is necessary to be familiar with C# scripting first to do this)
- Creating a more complex environment in the First Person scene