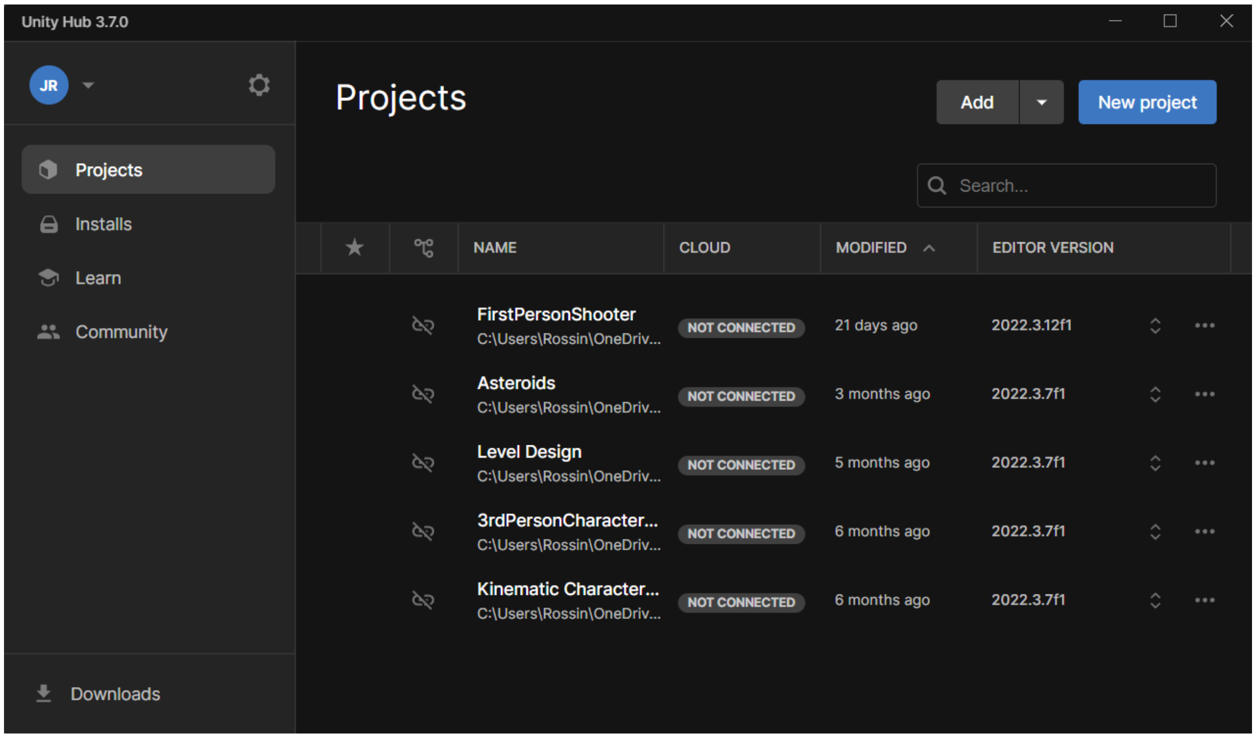
Intro To Unity:

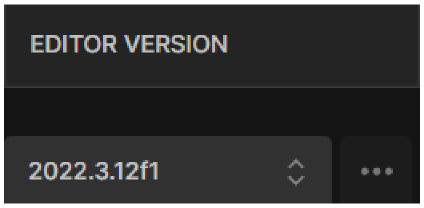
Unity Hub

1. The first thing we should talk about when starting out with Unity is the **Unity Hub**. The Unity Hub is a place where you can manage our projects and download different version of Unity.



NAME	CLOUD	MODIFIED	EDITOR VERSION
FirstPersonShooter C:\Users\Rossin\OneDriv...	NOT CONNECTED	21 days ago	2022.3.12f1
Asteroids C:\Users\Rossin\OneDriv...	NOT CONNECTED	3 months ago	2022.3.7f1
Level Design C:\Users\Rossin\OneDriv...	NOT CONNECTED	5 months ago	2022.3.7f1
3rdPersonCharacter... C:\Users\Rossin\OneDriv...	NOT CONNECTED	6 months ago	2022.3.7f1
Kinematic Character... C:\Users\Rossin\OneDriv...	NOT CONNECTED	6 months ago	2022.3.7f1

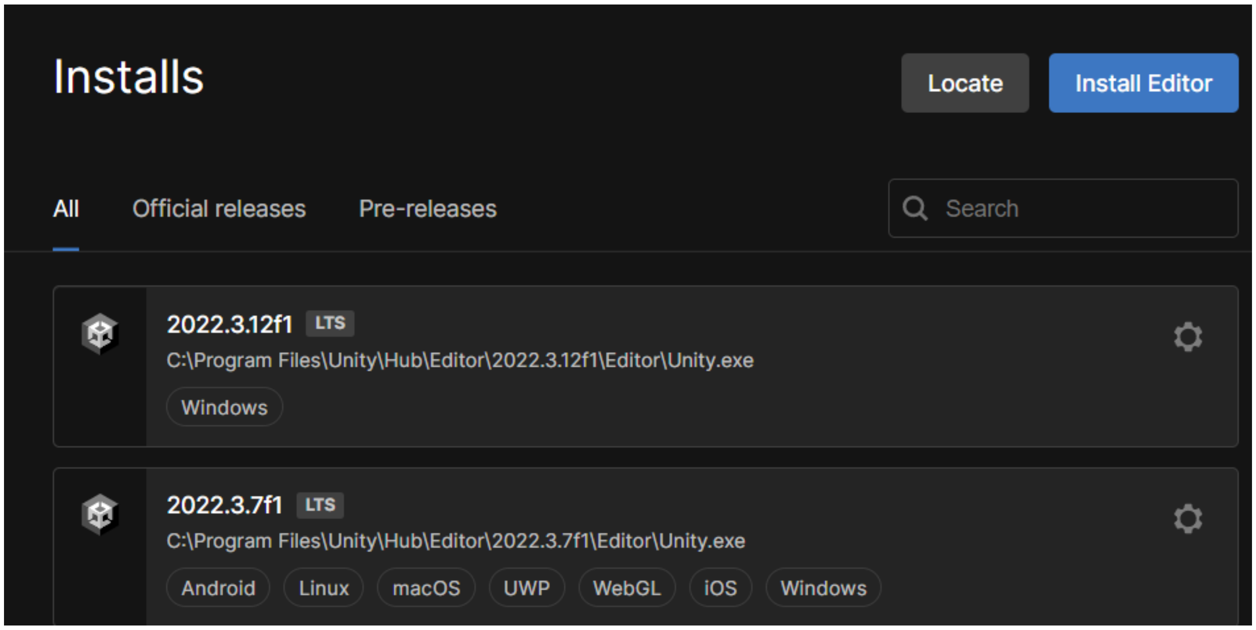
2. When we have a project listed on unity hub it will display what editor version of unity we are using for the project.



This is important because when working on a game dev team we need to make sure **all the members of the team are using the same editor version**. If not, when the team merges the parts of the project together, they will run into errors.

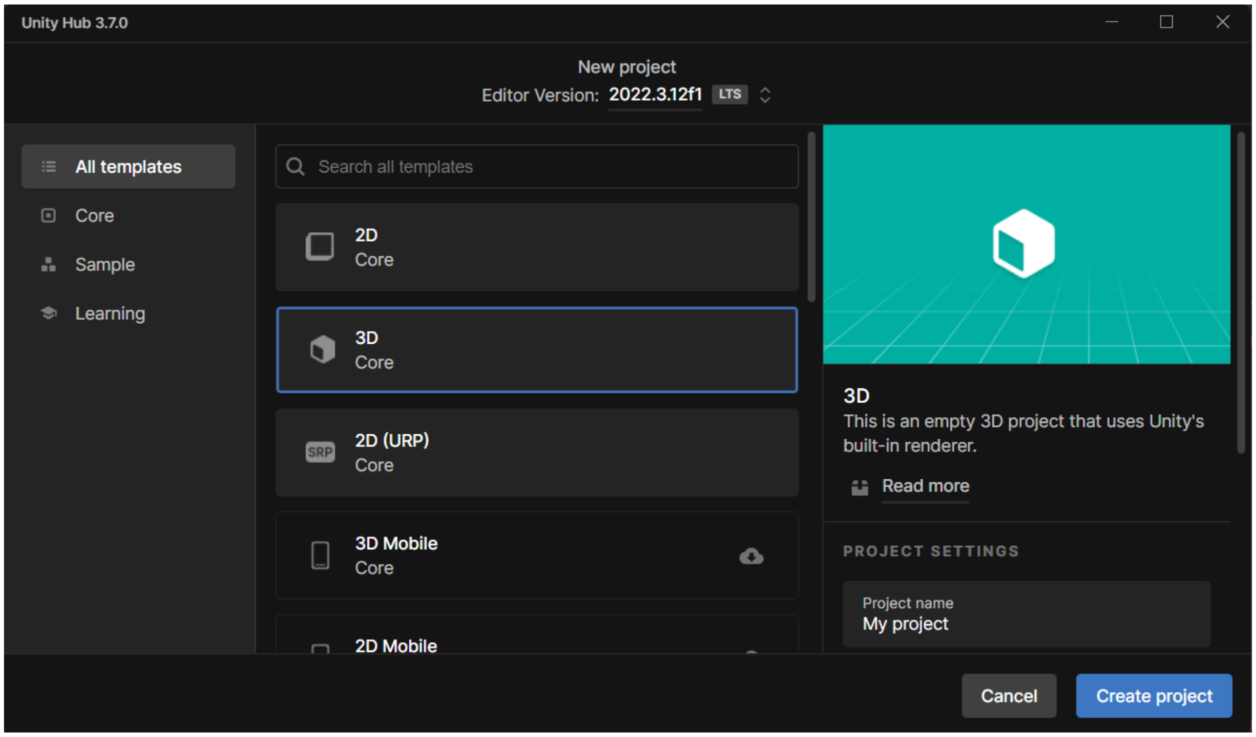
For any projects we do in class we will use the **LTS** version. **LTS** stands for **Long Term Support.** All LTS versions should be able to work with each other, so even if a team members is on different versions of LTS when the projects get merged everything should work just fine.

3. For our class we need to make sure that we have unity version **2022.3.40f1 LTS**. We can check what version of unity we have by going to the **Installs** tab on the unity hub.



Creating a New Project

1. Let's create a new unity project. To do so, on the projects tab click the blue **New Project** button at the top right-hand corner. Once you do you will be brought to the **New Project** window.


Unity Hub 3.7.0

New project

Editor Version: 2022.3.12f1 LTS

3D

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

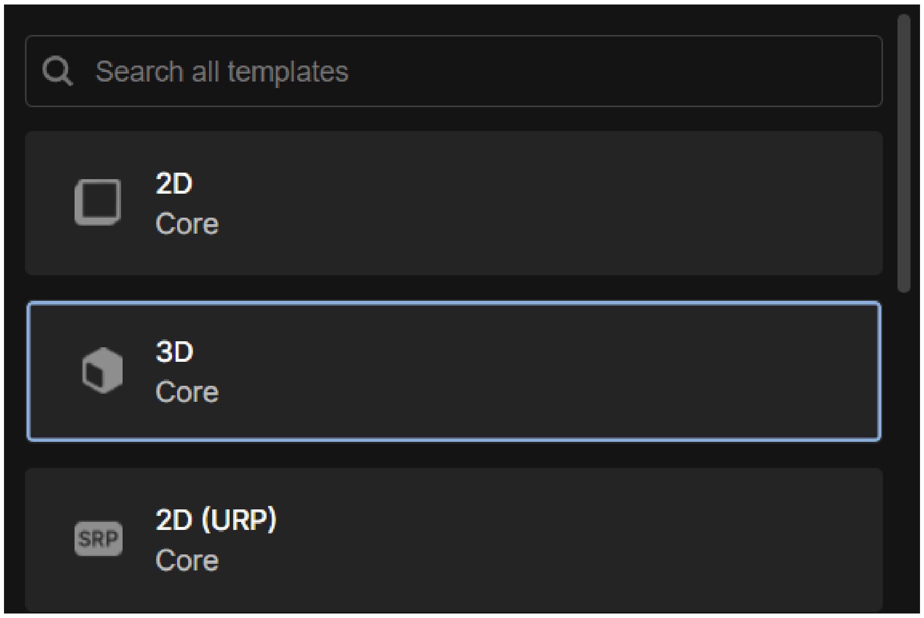
Read more

PROJECT SETTINGS

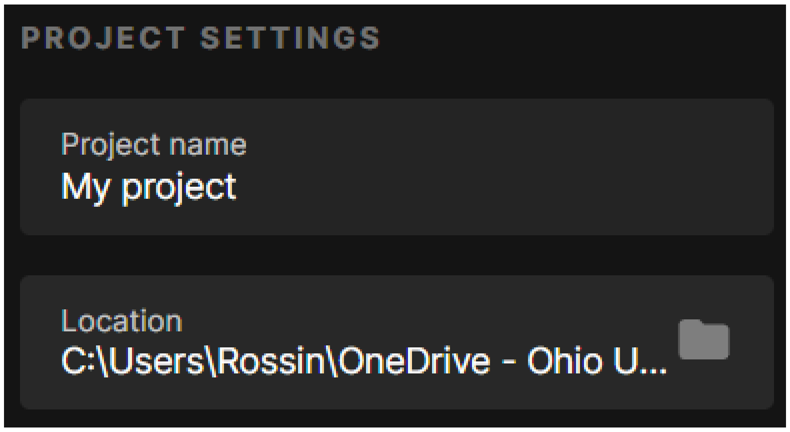
Project name: My project

2. First, we need to take note of what **unity version** we are going to make this project in. At the top, set the version to **2022.3.1.12f1 LTS**.

3. Second, we need to set what template our project will be based off of. In the middle of the screen, you should see that there are a variety of different template options. A template will essentially configure unity in such a way to make developing whatever game we are making easier. For our project we need to select the **3D Core** template. It should be located at the top of the list.



4. Lastly, over on the right-hand side, find the **Project Settings**. Here we will name our project and select the location of the project on our computer.

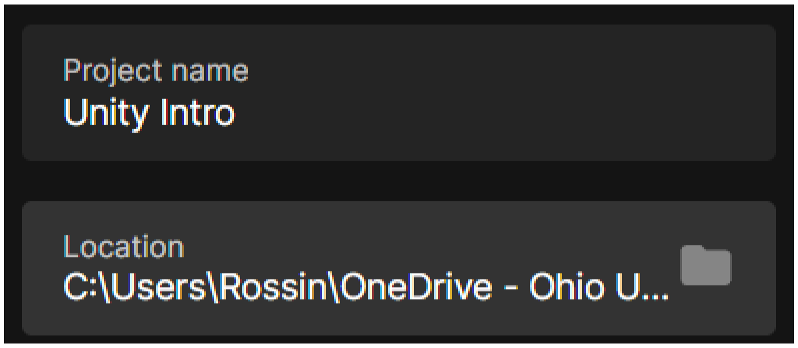


PROJECT SETTINGS

Project name
My project

Location
C:\Users\Rossin\OneDrive - Ohio U...

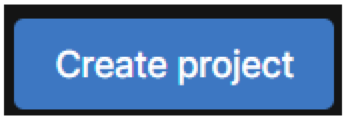
Name the project **Unity** and set the **location** of the project to the **local repository we made in the last lesson.**



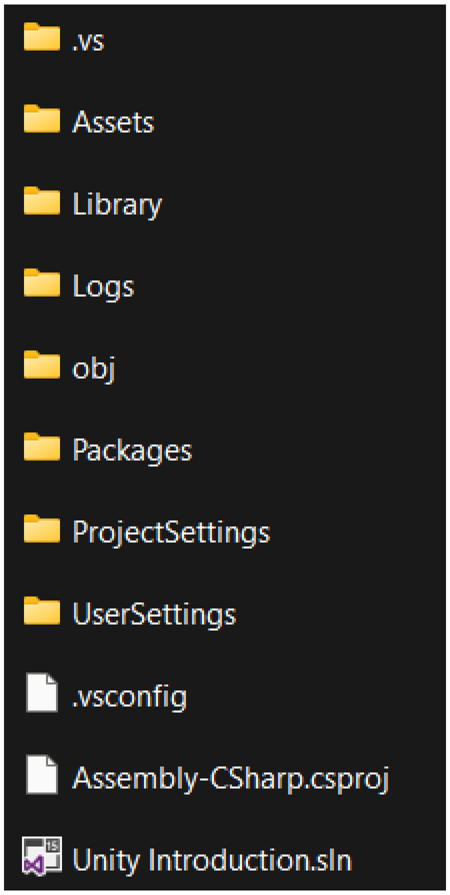
Project name
Unity Intro

Location
C:\Users\Rossin\OneDrive - Ohio U...

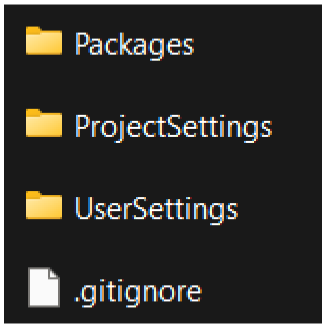
5. Once everything is set up, click on the **Create Project** button at the bottom right-hand corner. Unity will then start to create the project. It will take a few minutes for the project to be created.



6. Once the project is made, we can check the local repository to see the files of the unity project.



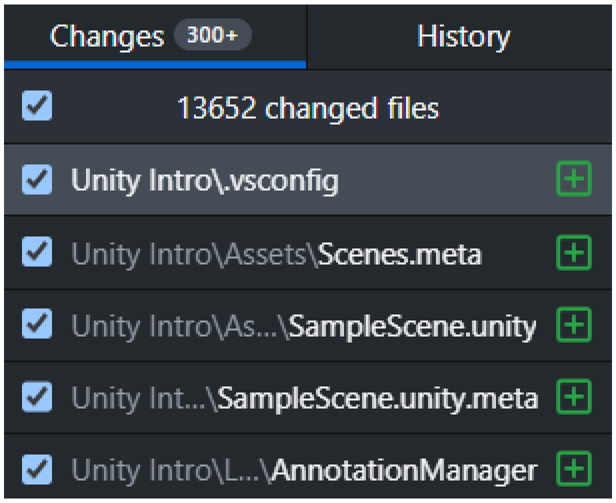
7. The only thing that we need to **make sure to do** is place the **.gitignore** file in with these files.



We need to do this to ensure that we will be able to push the project to GitHub. Without the .gitignore file we will run into a large file error.

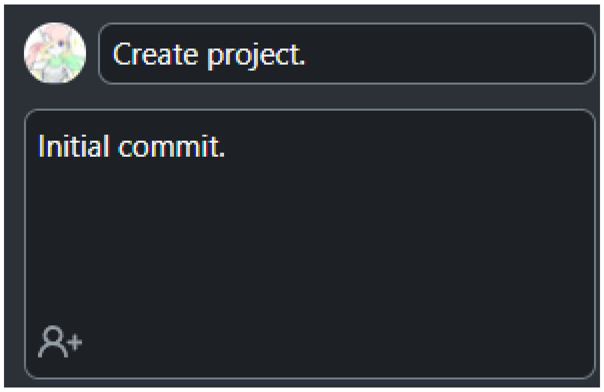
Commit the Project

1. Now that our project is made, lets commit it to GitHub. Open **GitHub Desktop**. We should see all the files that have been added to the repository.



13652 changed files
Unity Intro\.vsconfig
Unity Intro\Assets\Scenes.meta
Unity Intro\As...\SampleScene.unity
Unity Int...\SampleScene.unity.meta
Unity Intro\L...\AnnotationManager

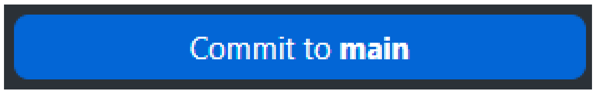
2. Title the commit and give it a ddescription.



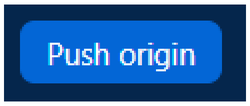
Create project.

Initial commit.

3. Once done, press the commit to main button.

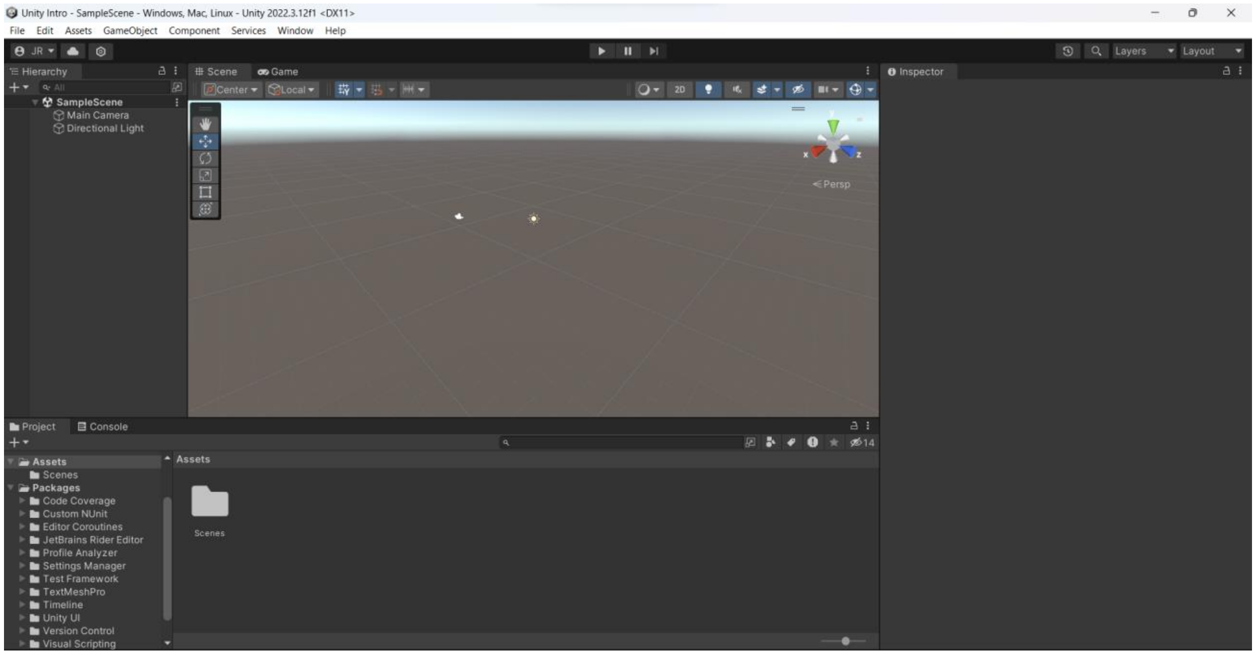


4. Then push to origin.

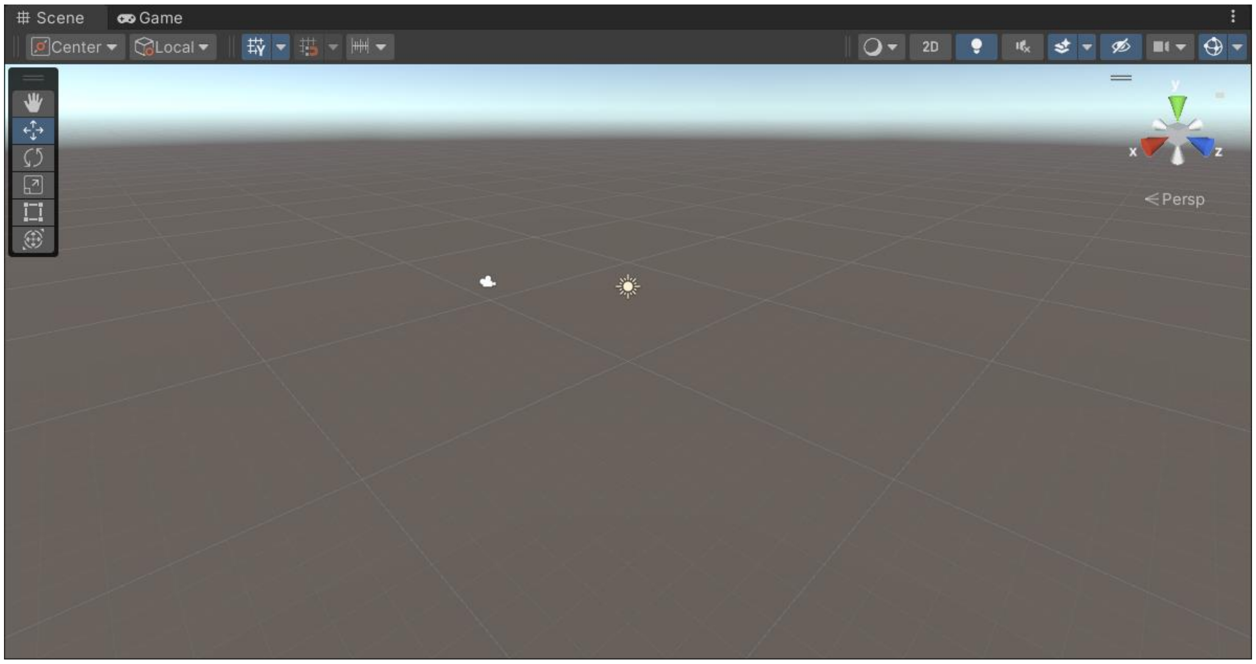


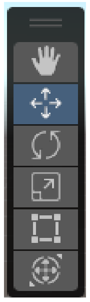
Unity Basics

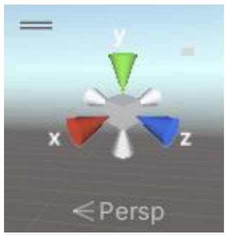
1. If we open the project, we will be greeted with a variety of different windows.



2. The first window we should look at is the **Scene** view, located in the center of the unity window. This window allows us to add game objects to our game. This can be anything from the player, to coins, to level geometry. Currently in our scene there is a directional light and a camera.



3. There is a tools panel to the top-left hand side of the screen. These tools allow us to move, resize or rotate a game object.

The short cuts for these tools are the QWERTY keys on the keyboard.

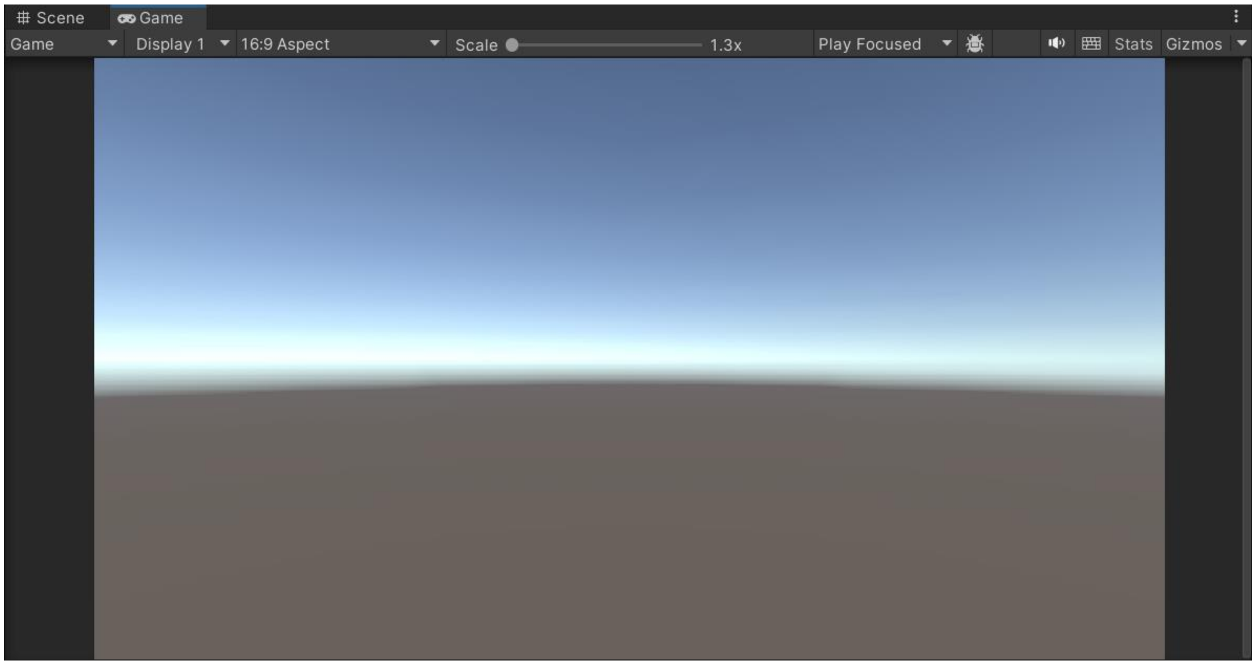
4. There is also a **View Gizmo** that tells us how we are currently viewing our world in 3D space. We can click on the text labeled **Persp** to change from a **perspective** view to an **orthographic** view. You can click on the text one more time to go back to the perspective view.


5. Just above the View Gizmo we can see a few buttons to press. I want to draw attention to the **2D** button.

   Clicking on this button will change the view of our game from a 3D view to a 2D view. We will be using the 2D view for the games that we make in class.

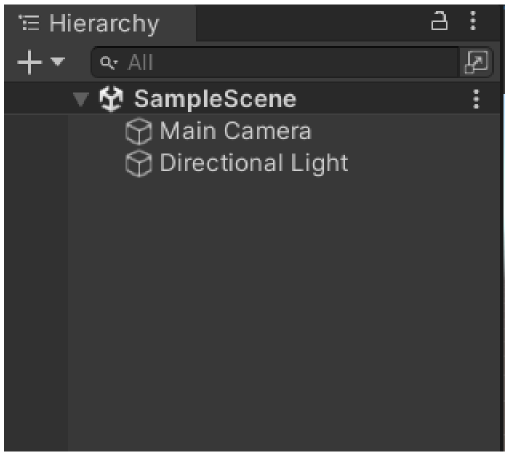
6. At the top-left hand corner of the window you will see two tabs, one labeled Scene and the other labeled **Game**.

   By clicking on the game tab, we can see how our game will look through our currently active camera.



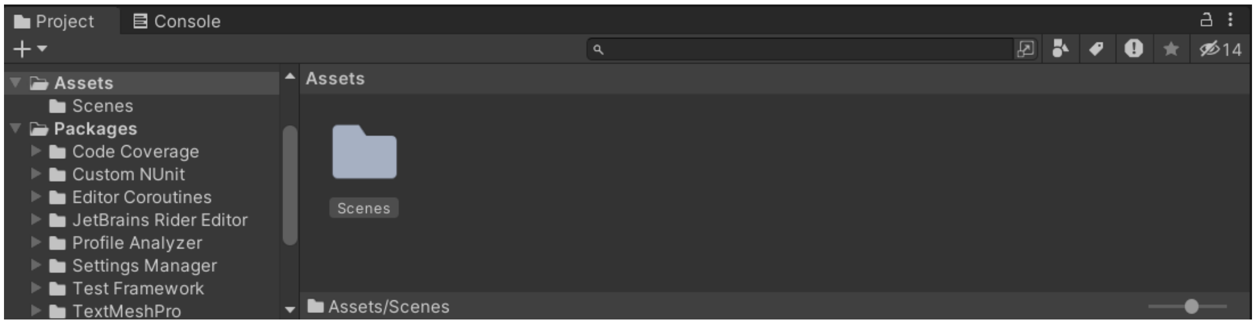
Since there is currently nothing interesting to look at let's switch back to the **Scene** tab.

7. Next, let's look at the **Hierarchy** window over on the left-hand side of Unity.



The **Hierarchy** window will display all the game objects that are currently in the scene. We can also see what the relationships for each game object that is in the scene.

8. Next, let's look at the **Project** window.

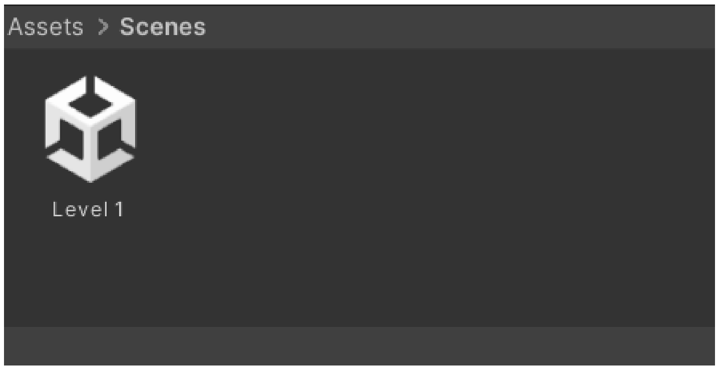


The **Project** window shows us all the files that are currently within our game project. By default, there will be two central folders, **Assets** and **Packages**.

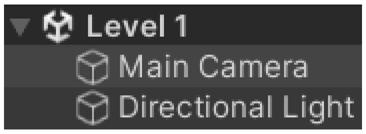
Assets are where we will be adding in our own custom files for the game. This can be things like textures, models, music, anything that would be for developing a game. By default, there is already a folder within this folder called **Scenes.**

Packages can be thought up as addons that we get for our unity project. By default, Unity already has a bunch of addons that are imported in whenever we make a project. Some of these addons are for displaying text to a screen, using visual scripting or for version control.

9. If we open the folder called **Scenes**, we will see that there is a **Scene** object named **SampleScene**. This is the current scene that is open in our scene view. We can think of scenes as "levels" that will be in our game. For this project let's name the scene **Level 1**.

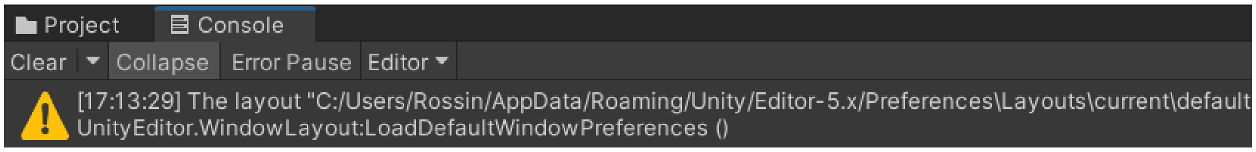


The new name of our scene will be reflected in the hierarchy.



We can look to the hierarchy to tell use what scene we are currently in.

10. It is important to note that there is the **Console** tab right next to the project tab. If we click the console tab, we will be able to see any errors or messages that we get when we playtest our game.



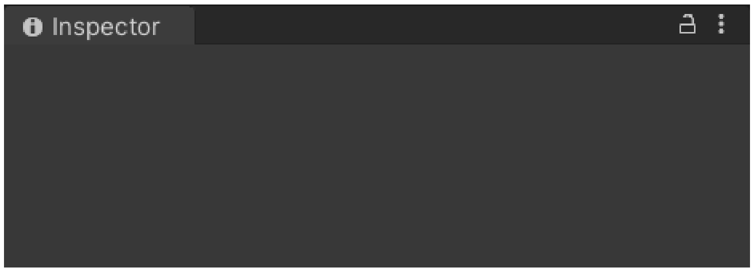
```
[17:13:29] The layout "C:/Users/Rossin/AppData/Roaming/Unity/Editor-5.x/Preferences\Layouts\current\default
UnityEditor.WindowLayout:LoadDefaultWindowPreferences ()
```

Something we need to be aware of are the types of messages we receive. There are **Warnings** and **Errors**.

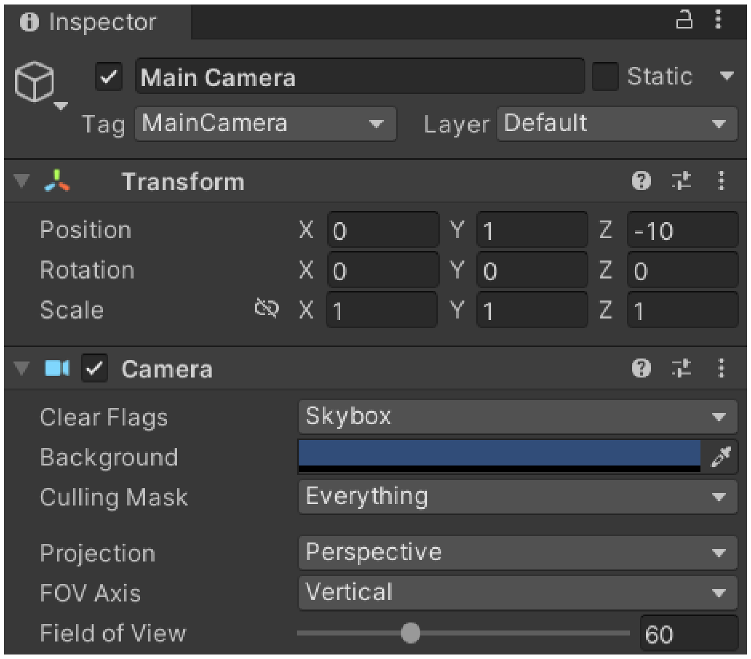
A **Warning** has an ⚠️ image next to it. A warning is Unity telling us that we might want to look into something. Most of the time they are harmless, and they should allow us to playtest our game.

An **Error** has an ❗ image next to it. An error is something we need to fix within our project. If an error occurs, we will not be able to playtest our game.

11. Lastly is the **Inspector** window located at the top-right hand corner of Unity.



By default the inspector will be blank until we click on an object. Select the camera and then look back at the inspector. Once we do, we will see a lot of information on the inspector.


Inspector

✓ Main Camera | Static

Tag MainCamera | Layer Default

Transform

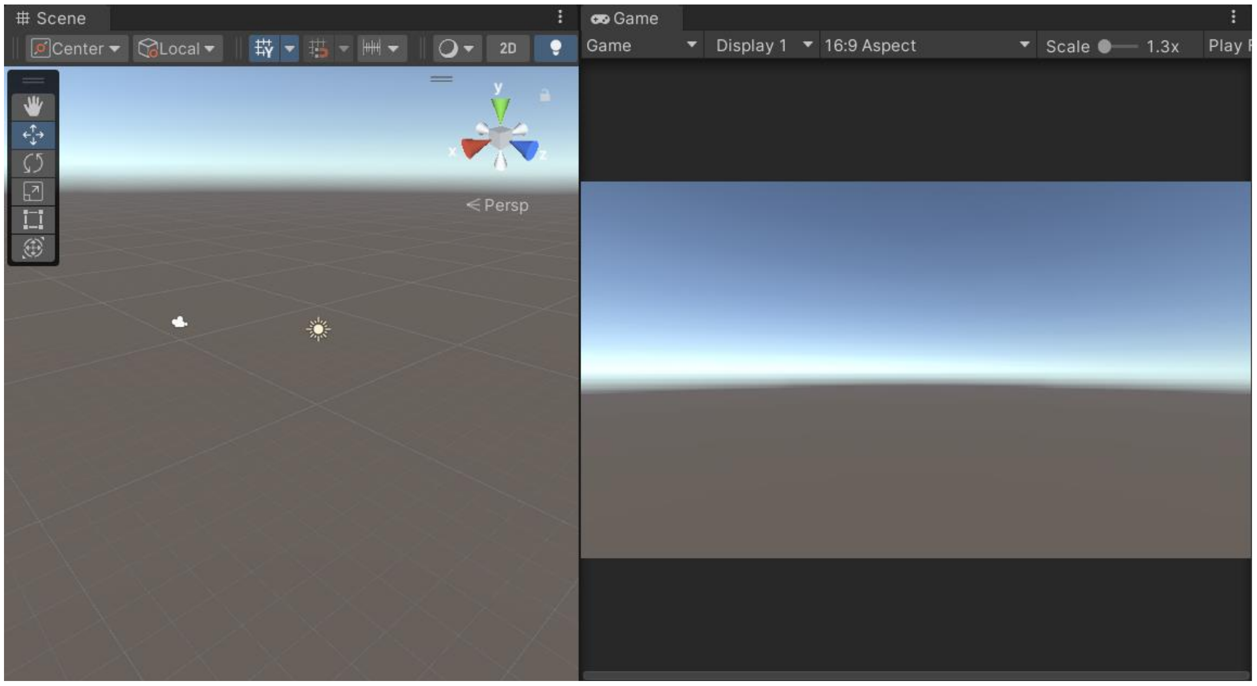
	X	Y	Z
Position	0	1	-10
Rotation	0	0	0
Scale	1	1	1

✓ Camera

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
FOV Axis	Vertical
Field of View	60

The **Inspector** allows us to see **components** on the game object that is selected. We won't go into too much information about components just yet, but for the time being, you can just think of them as, well, components. A part that can give your game object some functionality.

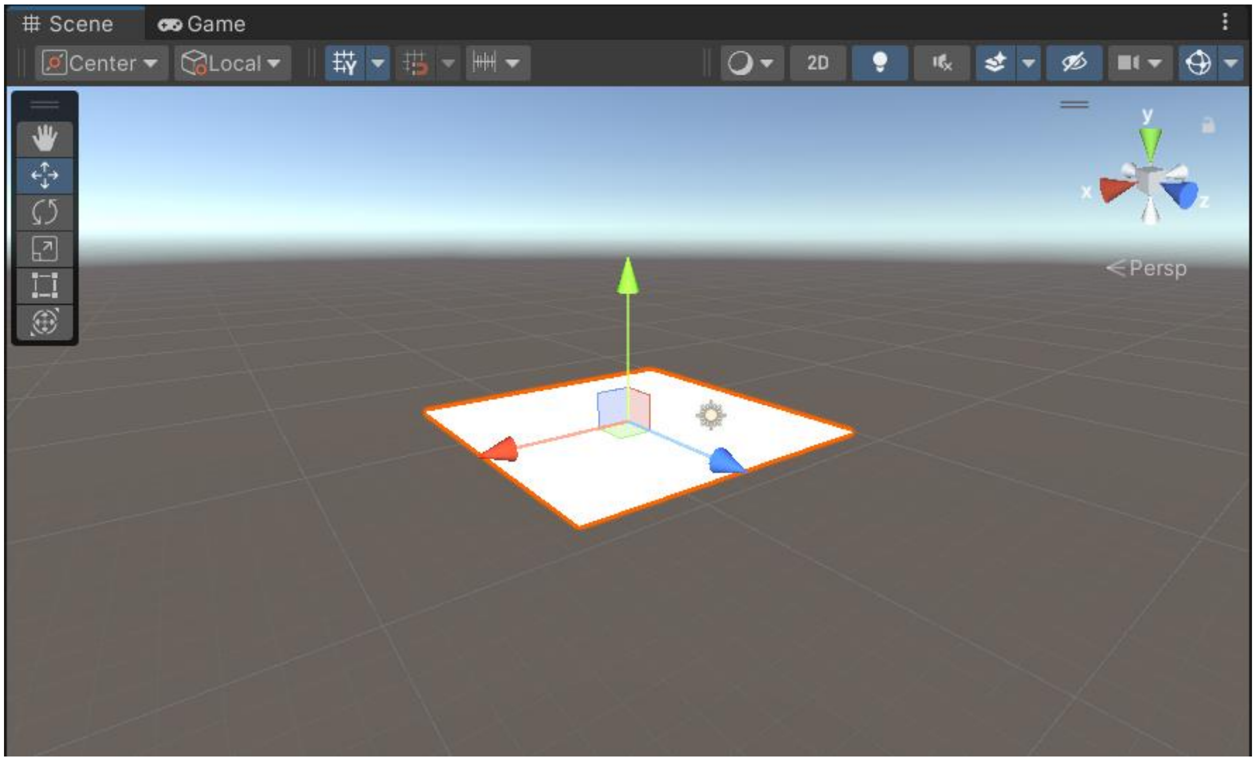
12. This then covers all the default panels in Unity. One last thing of note is that we can drag and drop the window tabs to different locations to customize the Unity layout. For instance, I can have the scene and game tab side by side each other.



If we ever need to reset the layout of our project, we can go to **Window > Layouts > Default**.

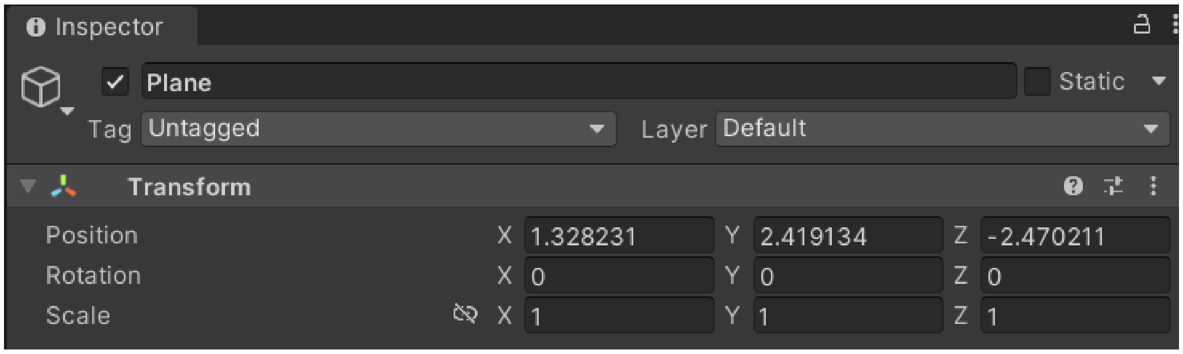
Adding Objects to the Scene

1. Let's begin to create a simple level within our scene. To do so, let's first start off by adding some simple shapes to the scene. Go to **GameObject > 3D Object > Plane**. This will add a plane to the scene.



We can use the **Move, Rotate** and **Scale** tools to affect our game object. However, let's take a look at the inspector.

2. With our plane selected, find the **Transform** component in the **Inspector**.



Inspector

Plane — Static

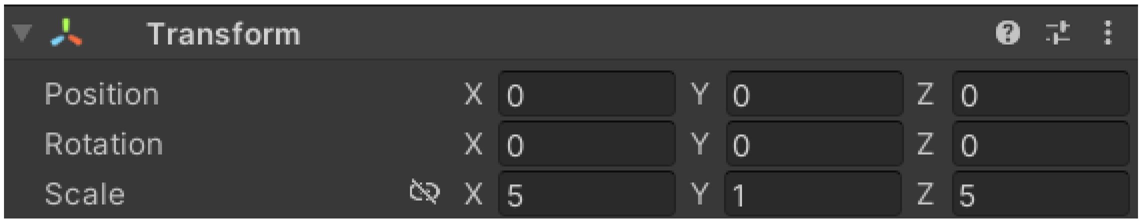
Tag: Untagged — Layer: Default

Transform

Position	X 1.328231	Y 2.419134	Z -2.470211
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

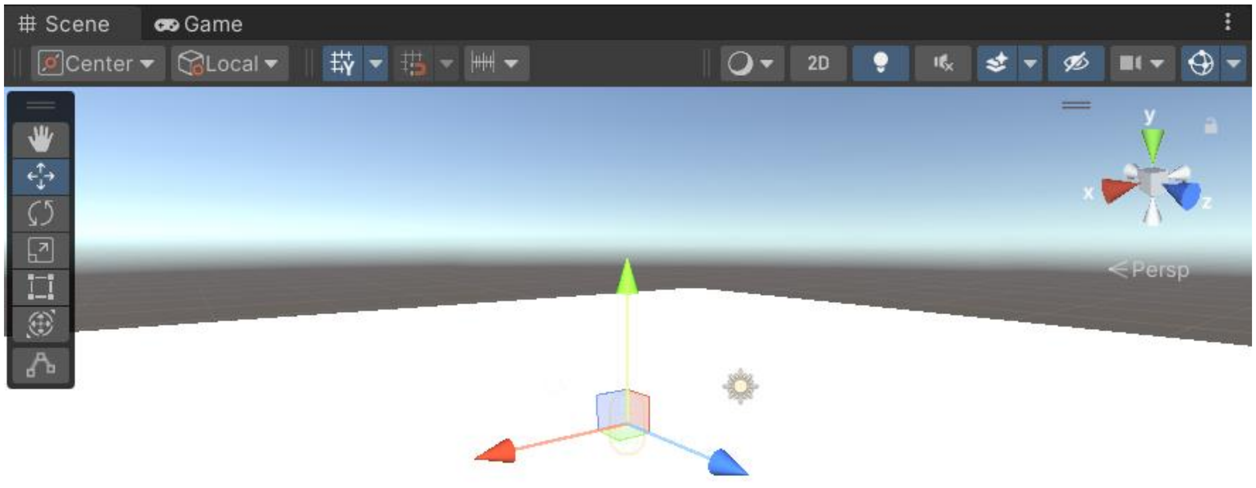
Here we can see the **Position, Rotation** and **Scale** of the game object. Let's set the **Position** to **0** for the **X, Y and Z**. This will move the object to what is known as the **World Origin**, which is just 0 in the XYZ position.

Let's also increase the **X** and **Z scale** of the object to **5**. This will make the plane larger in the X and Z axis.



Transform	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	5	1	5

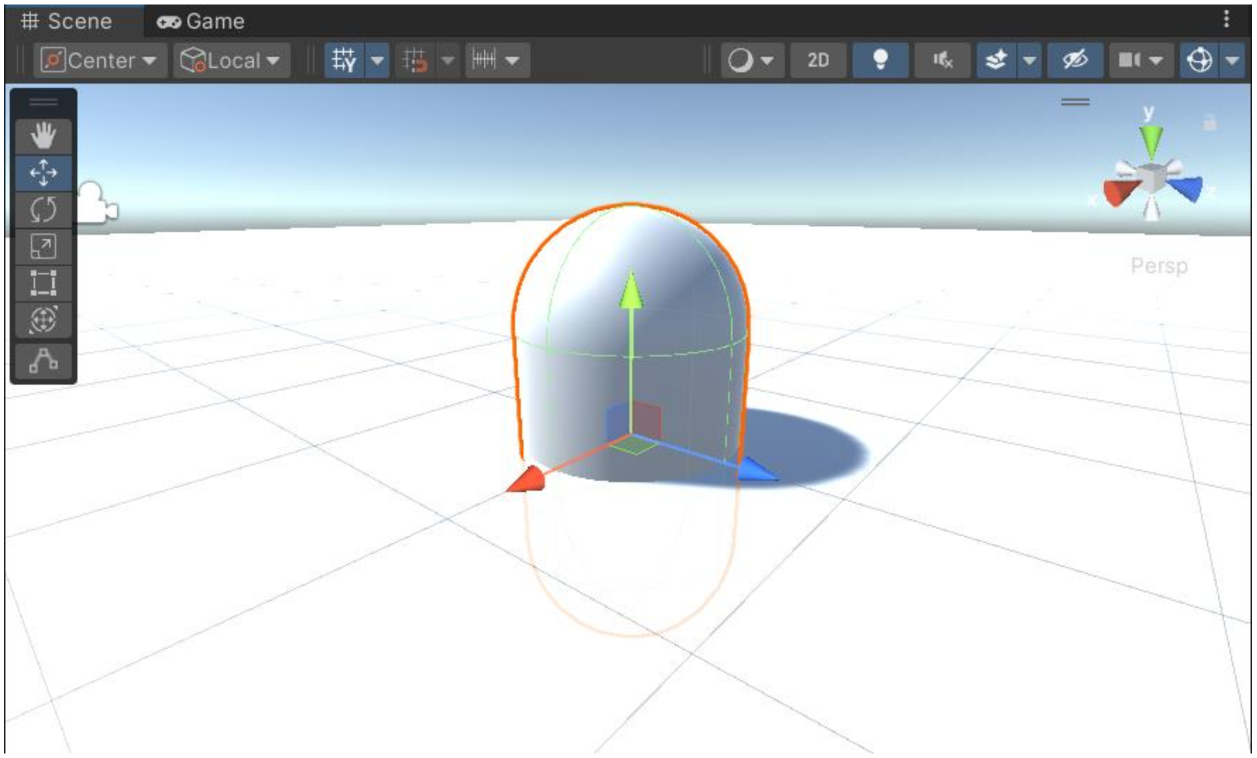
3. Next let's add a new **Capsule** game object. Again, we can go to **GameObject > 3D Object > Capsule** to create this object. Set its position to the world origin.



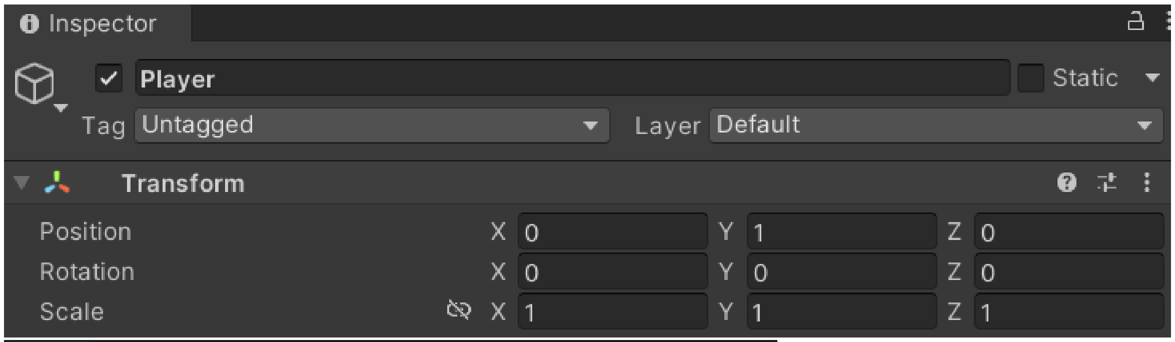
4. Our capsule is a bit far away, so let's take a moment to explore how we traverse the scene view. With our **Mouse** overtop of the scene view window, we need to right click and hold. This will transform the mouse into an **eyeball** icon.

   If we drag the mouse around, we will be able to look around our scene. To move, we can press and hold one of the **WASD** keys. This moves us in the selected direction relative to the camera. You can also hold the **Q** key to move down and the **E** key to move up.

   Lastly, to quickly move to an object. We can select the object in the hierarchy and hit the **F Key**. This allows us to frame the selected object in the center of the view.



5. With the capsule selected, go to the inspector, and set its name to **Player** and set the **Y Value** of the **Position** to **1**. The player should now be above the ground.



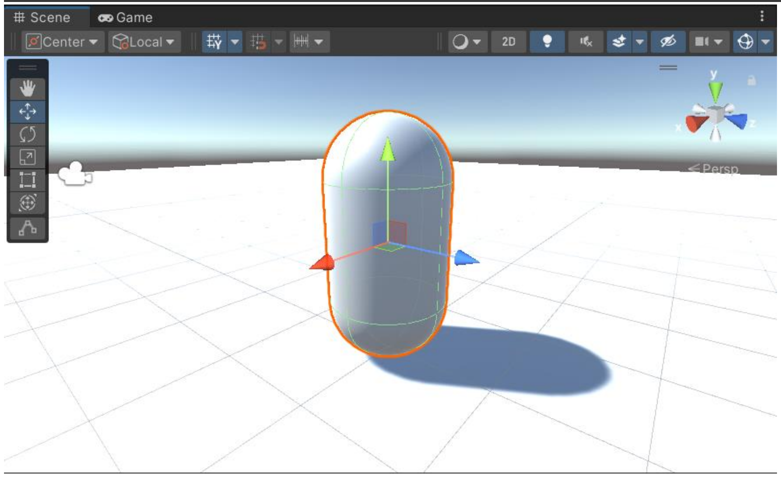
Inspector

Player — Static

Tag Untagged — Layer Default

Transform

	X	Y	Z
Position	0	1	0
Rotation	0	0	0
Scale	1	1	1

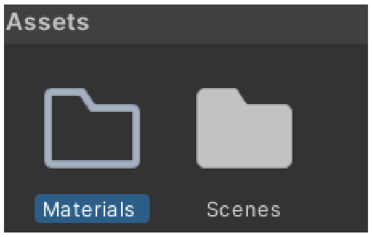


Even through we called the capsulePplayer, they are going to be more of a stand in for a player character. Typically, when we make games, we want to have some object that represents the player in them. That way we can properly scale the objects of the game to fit to the player's size.

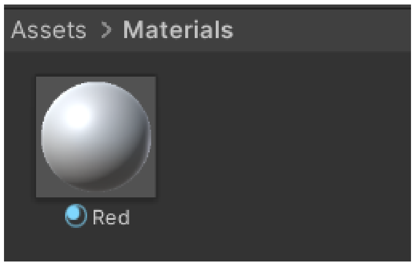
Materials

1. Right now, our player is just a little bit hard to see since they blend in with the color of the plane. To fix this let's create a **Material** for the player.

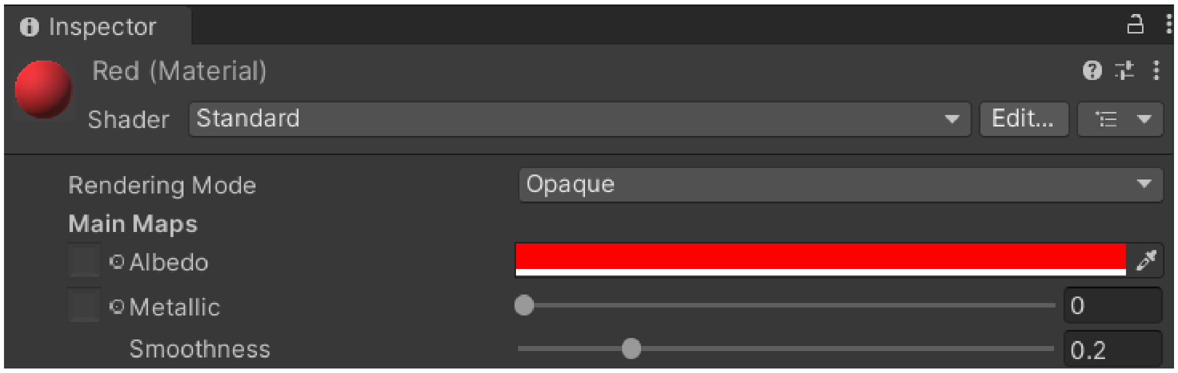
2. In the **Assets** folder right click and create a new folder called **Materials**.



Then inside that folder right click and create a new **Material**. Name the new material, the color you want the player to be.



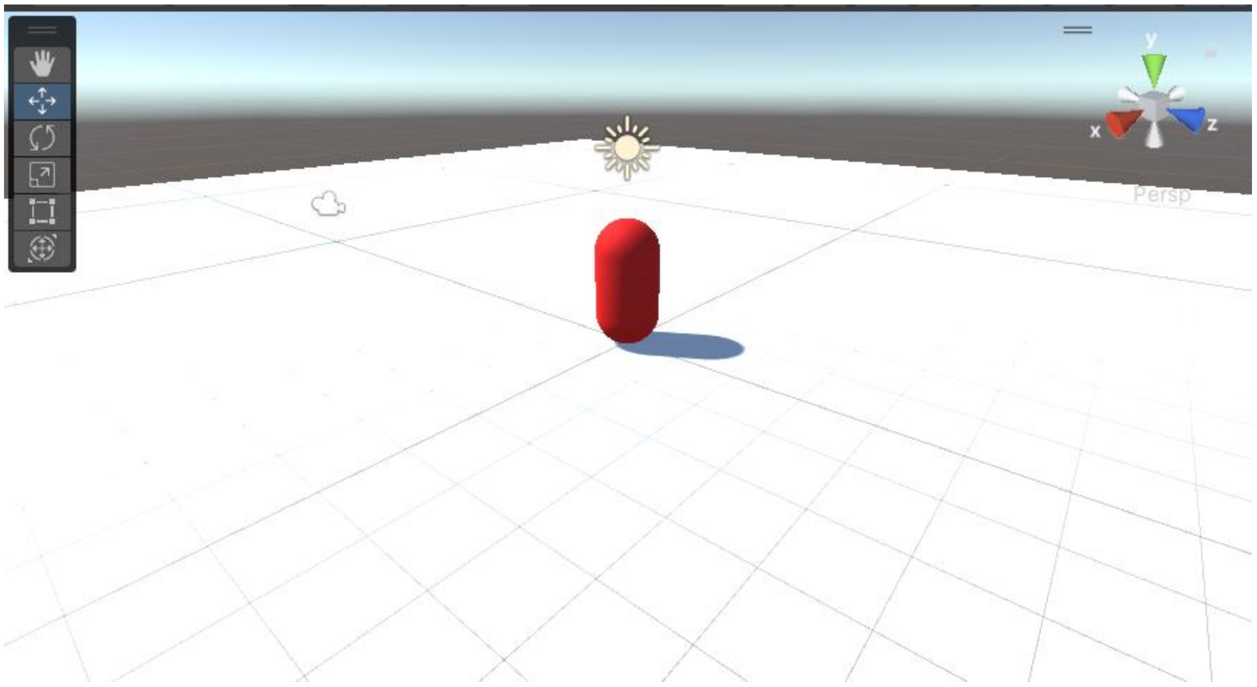
3. With the material selected we can view its properties in the inspector.



Shader: Standard
Rendering Mode: Opaque
Main Maps
Albedo
Metallic: 0
Smoothness: 0.2

Here I change the albedo color to red and lower the smoothness value to 0.2.
Albedo is another name for color.
Smoothness determines how light will reflect off of an object. A lower value will make our player less reflective.

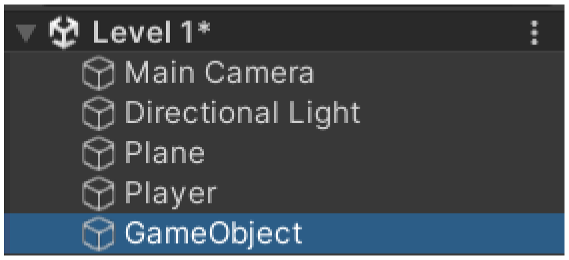
4. We can then just drag the material onto the player to change their color.



Grouping/Parenting Objects

1. One last thing we should cover is how to group or **parent** objects together. Say we are building a structure in Unity out of simple shapes. If we had to move the structure, we would need to individually select each part and then move it. To avoid doing this we can use an **Empty GameObject** as a container for our structure.

2. Go to **GameObject > Create Empty**. This will create an empty game object in the hierarchy.

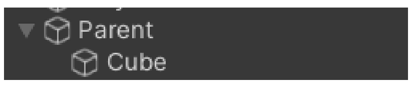


Level 1*
Main Camera
Directional Light
Plane
Player
GameObject

An empty game object is probably the simplest thing within unity, it only has a transform component attached to it. Let's name it **Parent**.

Parent

3. In the hierarchy, right click on the Parent object and select **3D Object > Cube**. If we look at the hierarchy, the cube will be within the parent game object. What we have done is make the cube a **Child** of the parent object.

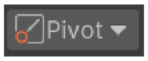


We can have multiple child objects within a parent object.

4. For a demonstration of what this means, set the cube's **X position** to **1**. If we rotate the parent object the cube will also rotate. But if we just rotate the cube the parent will not be affected.

   A child object will inherit the transformations of the parent, but the parent will not inherit the transformations of a child.

   Note: Be sure to set the handle position to **Pivot** rather than center.



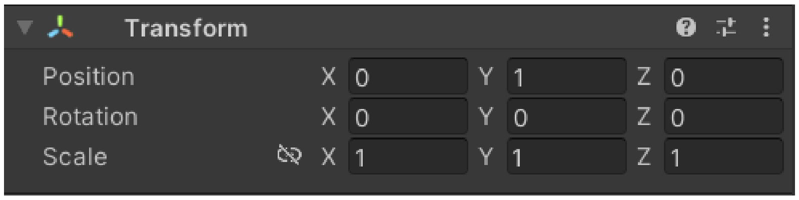
Components

1. Now it's time to talk a little bit about **Components**. Components contain properties which you can edit to define the behavior of a GameObject. Every GameObject will have components attached to it.

2. Since we have a few GameObjects already in our scene objects, let's take a look at a few components that are attached to them.

Transform

1. The **Transform** component keeps track of the **Position, Rotation** and **Scale** of an object along the **X, Y** and **Z** axis. This component will most likely be attached to every object that we add with in game.

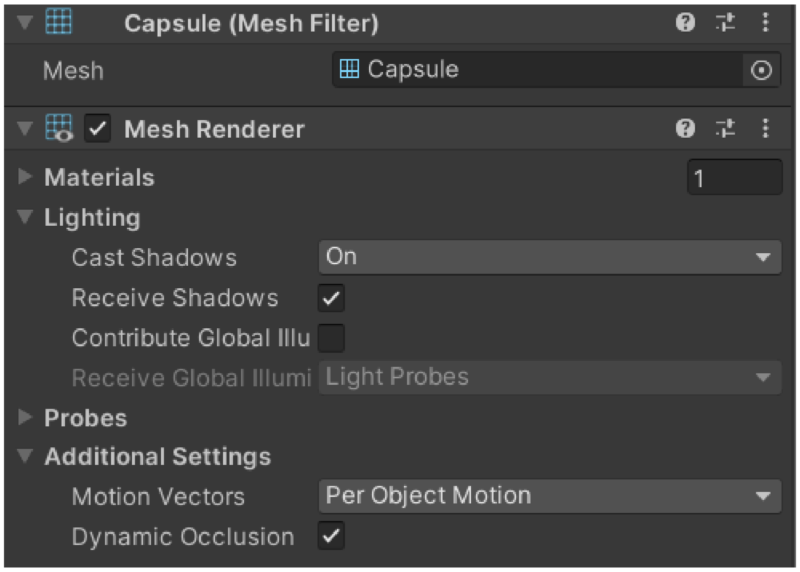


Transform

Position	X	0	Y	1	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Mesh Filter and Mesh Renderer

1. The Mesh Filter and Mesh Renderer components work hand in hand with each other. A **Mesh Filter** component holds a reference to a mesh while the **Mesh Renderer** renders the mesh that the Mesh Filter references.



Capsule (Mesh Filter)

Mesh	Capsule

Mesh Renderer ✓

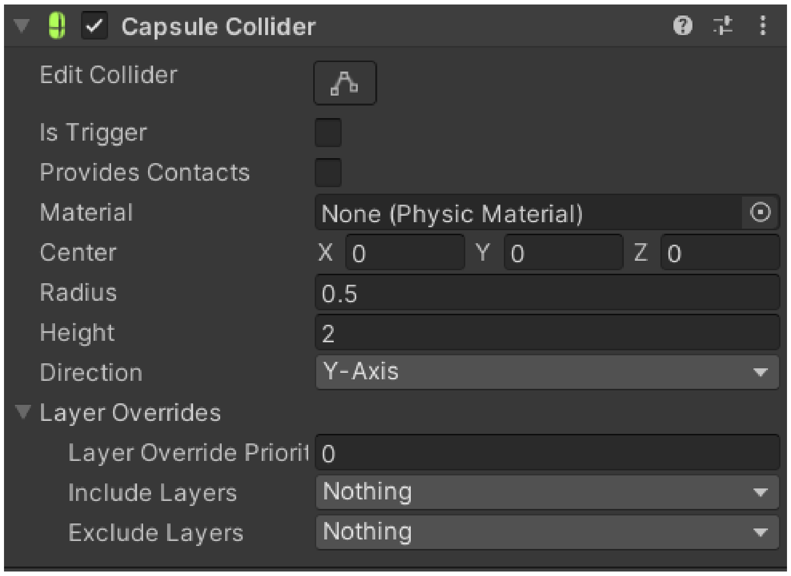
Materials	1
Lighting	
Cast Shadows	On
Receive Shadows	✓
Contribute Global Illu	
Receive Global Illumi	Light Probes
Probes	
Additional Settings	
Motion Vectors	Per Object Motion
Dynamic Occlusion	✓

On the mesh renderer we can control what materials are on the object as well as if the object will cast shadows.

Colliders

1. **Colliders** are what allow our GameObjects to detect collision. There are a variety of different collider component types that we can attach to our objects.

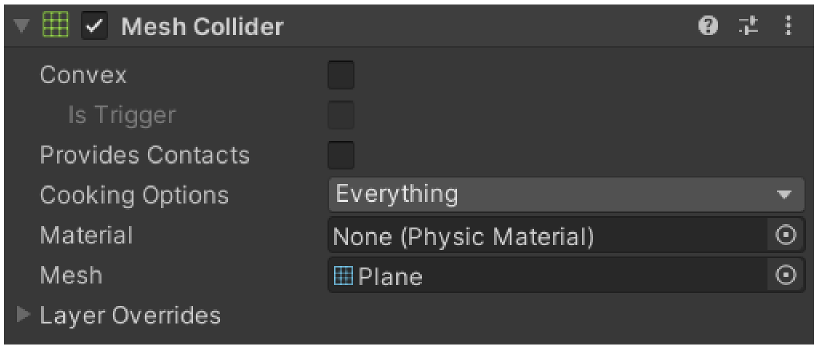
2. There are primitive collider shapes such as **Box, Capsule, Sphere** and **Cube** colliders. They are called "primitive" because they are defined by simple geometric shapes. This is good for us as game developers since primitive colliders are the most efficient type of **collider** in Unity.



Capsule Collider ✓

Edit Collider	
Is Trigger	
Provides Contacts	
Material	None (Physic Material)
Center	X 0 Y 0 Z 0
Radius	0.5
Height	2
Direction	Y-Axis
Layer Overrides	
Layer Override Priorit	0
Include Layers	Nothing
Exclude Layers	Nothing

3. There are also advanced colliders such as **Mesh** or **Terrain** colliders. These will typically be used for complex models such as our level geometry. The precision of these colliders comes with a higher processing overhead than primitive colliders.



Mesh Collider

Convex	☐
Is Trigger	☐
Provides Contacts	☐
Cooking Options	Everything
Material	None (Physic Material)
Mesh	Plane
Layer Overrides	

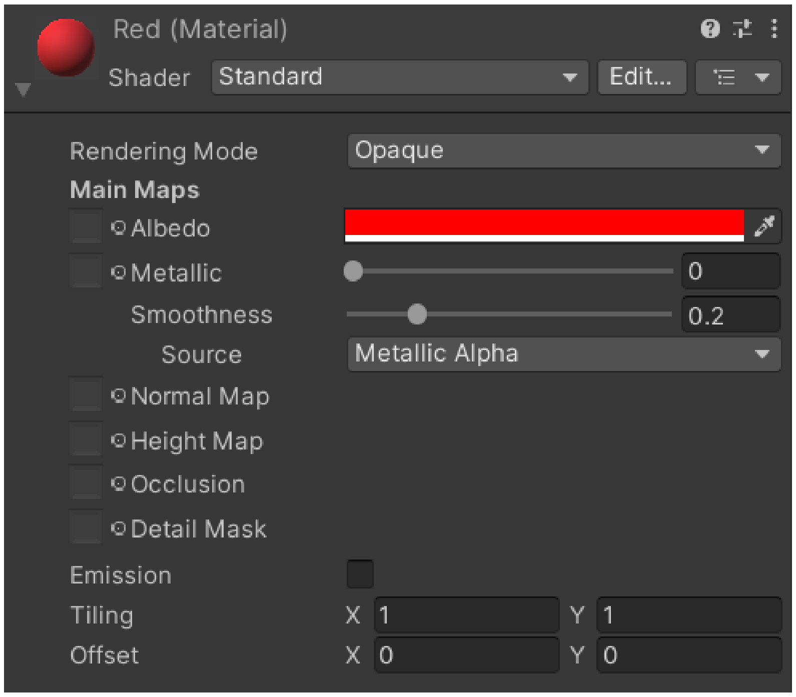
4. One last thing to mention about colliders is that we can turn them into what are known as **Triggers**. Where a normal collider would stop upon touching another collider, a trigger allows for the collider to pass through it, however it still reports the collision to the game engine.

Is Trigger ☐

This is good for us game designers in a case where we want to cover an area of the level with a collider that will trigger a cutscene in a game.

Materials

1. **Material** components allow us to see the materials that are currently on the object. We can use this component to set the properties of the material such as the Color, Smoothness, Normal Map or Tilling values.



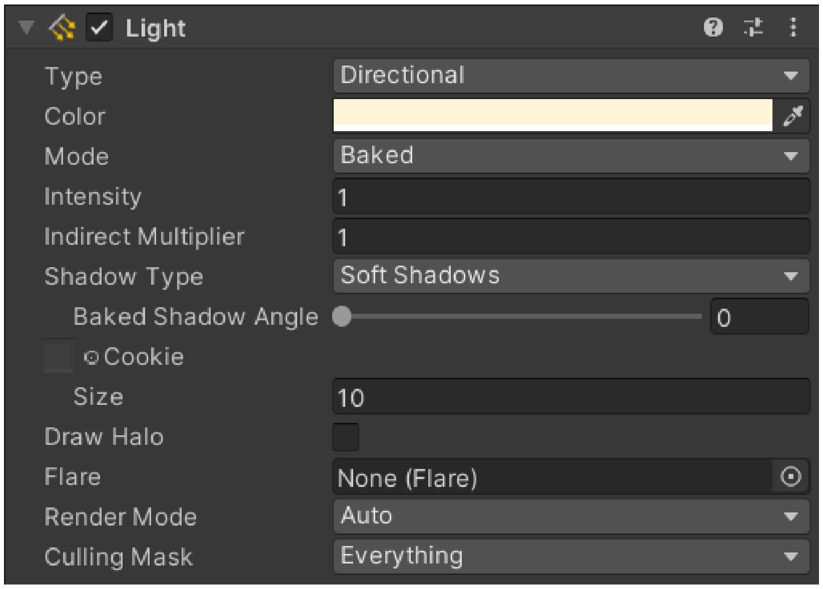
Red (Material)

Shader Standard | Edit...

Rendering Mode	Opaque
Main Maps	
Albedo	🟥
Metallic	0
Smoothness	0.2
Source	Metallic Alpha
Normal Map	
Height Map	
Occlusion	
Detail Mask	
Emission	☐
Tiling	X 1 Y 1
Offset	X 0 Y 0

Light

1. A **Light** component is used to control the light sources within the game. We can set what type of light the light source is, the color or if the light will be able to cast shadows.



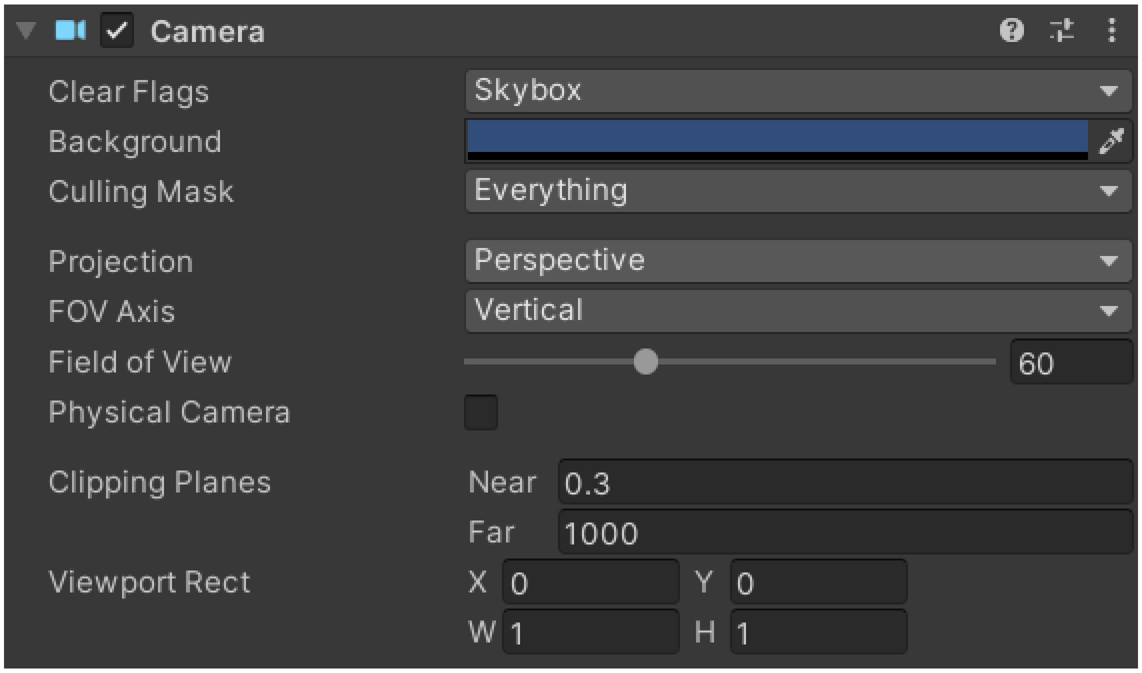
Light

Type	Directional
Color	
Mode	Baked
Intensity	1
Indirect Multiplier	1
Shadow Type	Soft Shadows
Baked Shadow Angle	0
Cookie	
Size	10
Draw Halo	☐
Flare	None (Flare)
Render Mode	Auto
Culling Mask	Everything

2. We also have the option of **Baking** our lights into our scene. Baking a light means calculating the effect of lights, like illumination and shadow, and storing this information in a separate texture called a lightmap. With the information stored, we can delete the light, but the lighting information will still appear in the scene.

Camera

1. We can use the camera component to change what the camera in our game sees. This can range from anything to seeing the skybox in the game, whether the camera is in perspective or orthographic view or when objects will begin to fade in/out of our game.



Camera

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
FOV Axis	Vertical
Field of View	60
Physical Camera	
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1

Unity Manual

1. One last thing to mention since we are talking about different parts of Unity is the **Unity Manual**. Here you can read about any aspect of Unity that you may be having questions about.

2. To find out information about something in Unity all we need to do is click the **Question Mark** icon ?. We can find this on any of our components at the top-right hand corner of the component.

3. Once we click it, we will be brought to the manual page that covers the topic on that part of Unity.

4. **Remember, the manual is your friend.** It will play a key part in helping you understand Unity.

Class Work – Create a Level

1. Now that we know how to add objects, materials, group objects together and a few components, let's create a level.
    a. Students will design a level by adding in game object to use as level geometry.
    b. The level geometry must stay within the bounds of the plane.
    c. Students should add materials to the object to make the level easier to navigate.