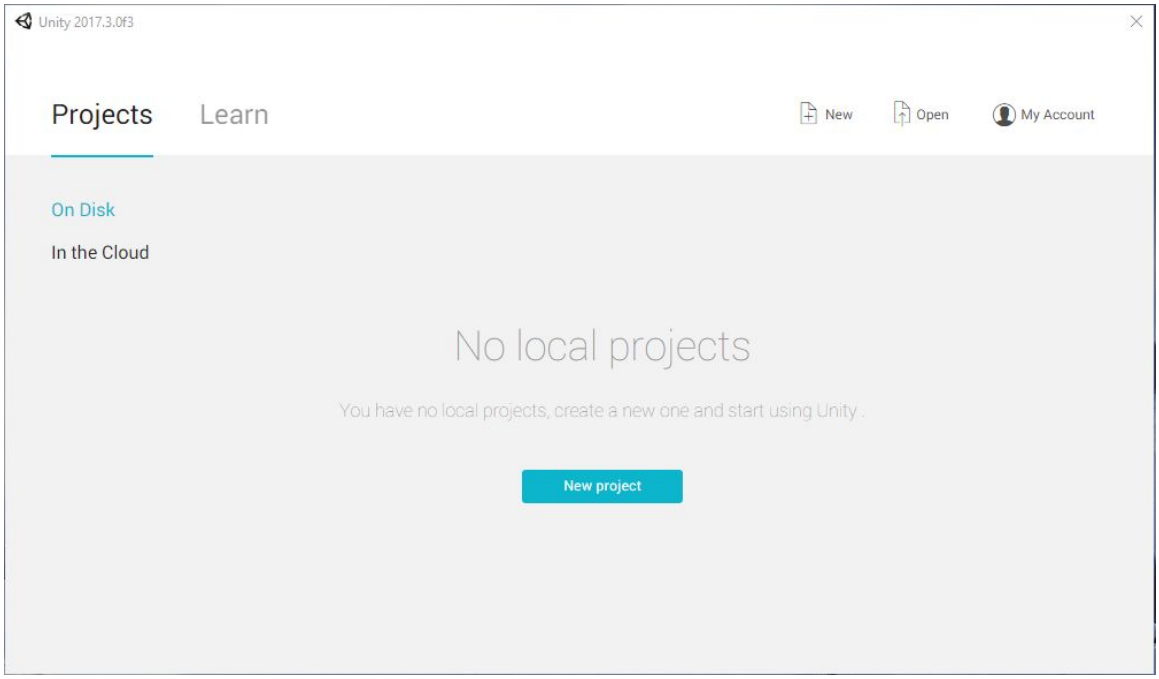
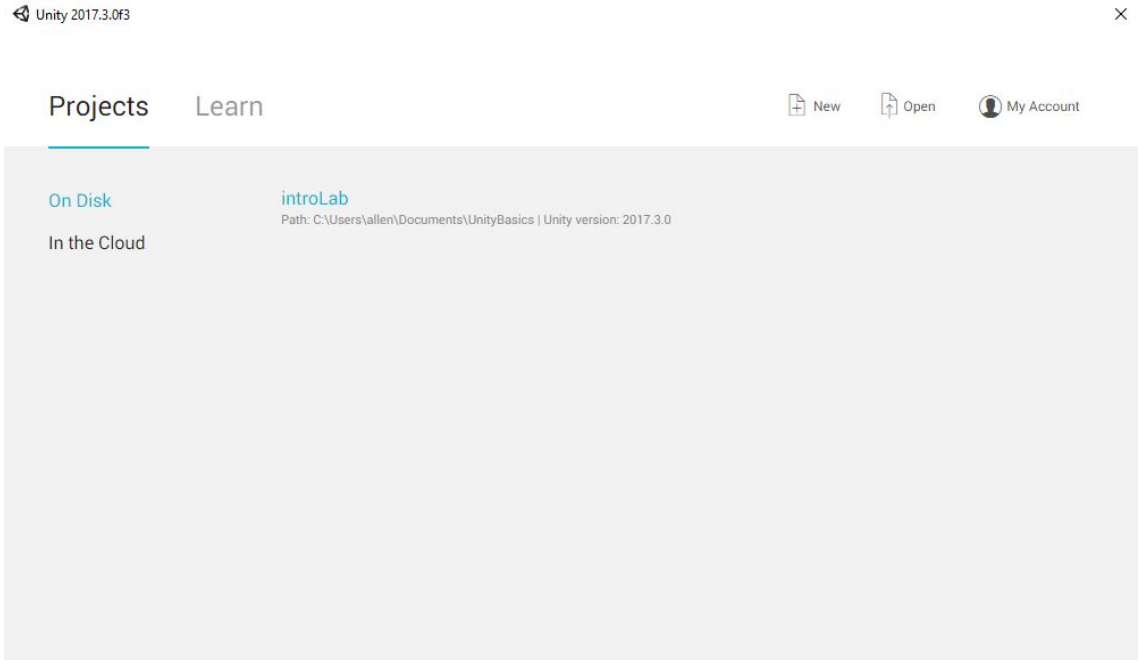
Intro to Unity Basics

Project Setup

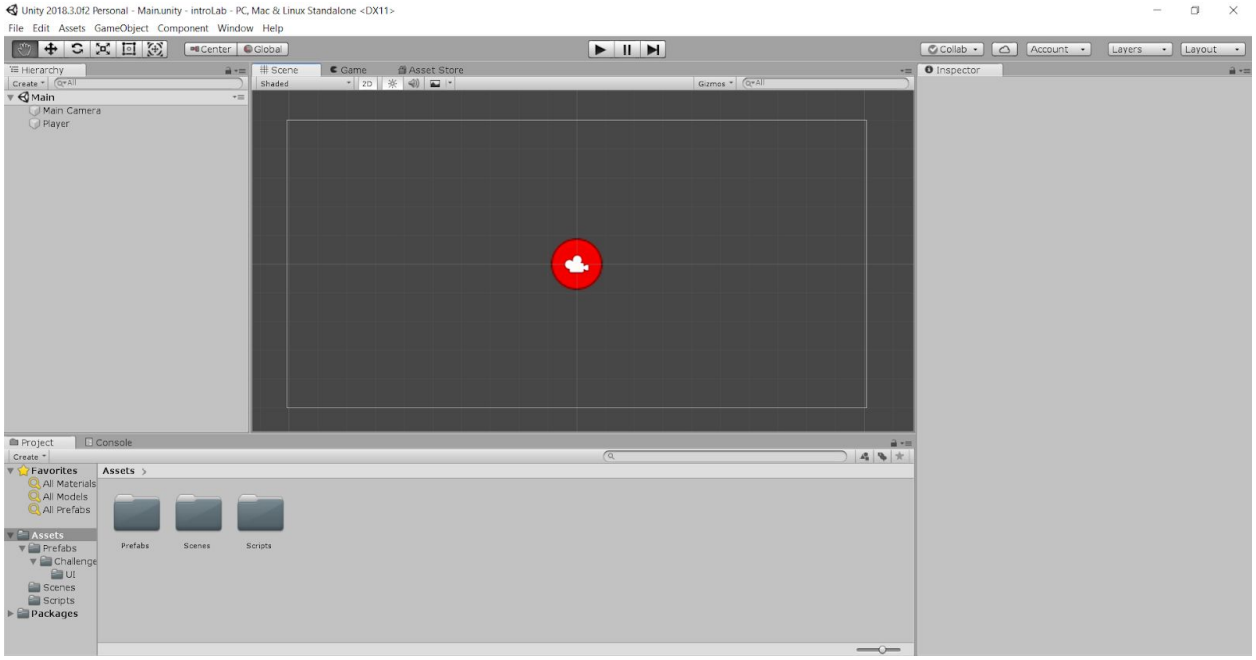
This lab is meant to help walk you through the layout and basics of Unity. When you first open Unity, you should see a screen like this:



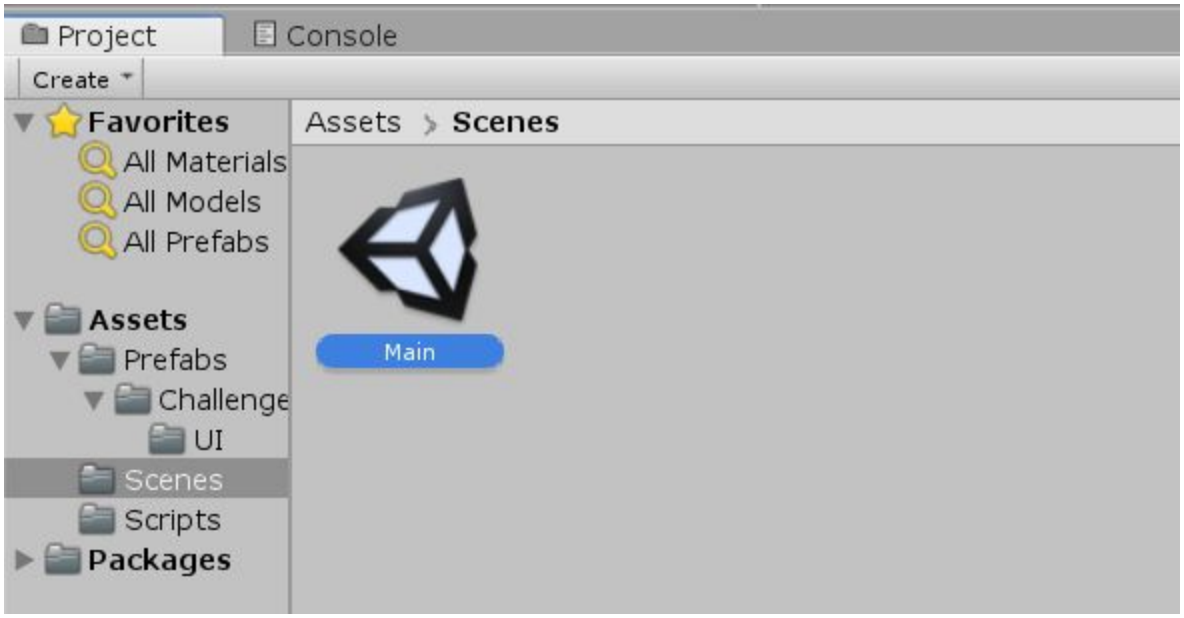
When starting a new project, you select New and select if you want the project to be in 2d or 3d. For this intro lab, we will be opening a pre-existing project. So, if the file doesn't show up like this:



then select Open and find the introLab folder in your computer and select it. Once Unity opens the project, you should see this:



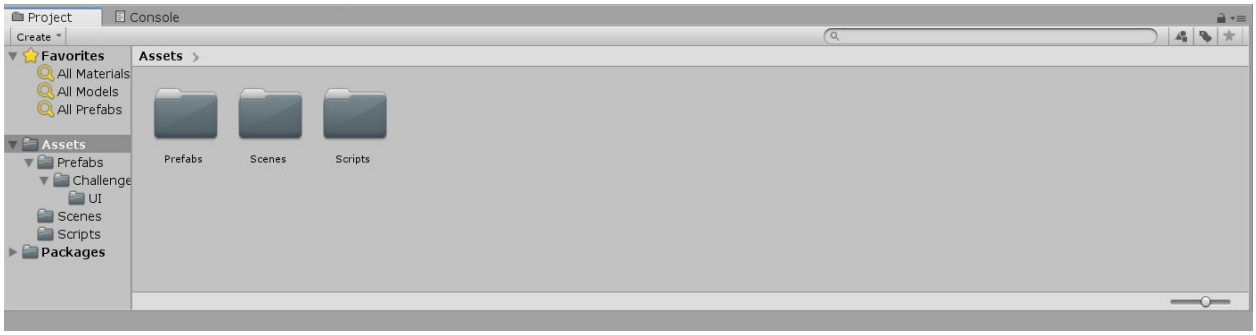
If you don't, then navigate to the Scenes folder in the Project window and double click on the icon that says Main.



Now, let’s begin learning how to navigate Unity. The editor is split into different sections and windows. The windows each do different things and all of them can be opened from the Window tab at the toolbar.

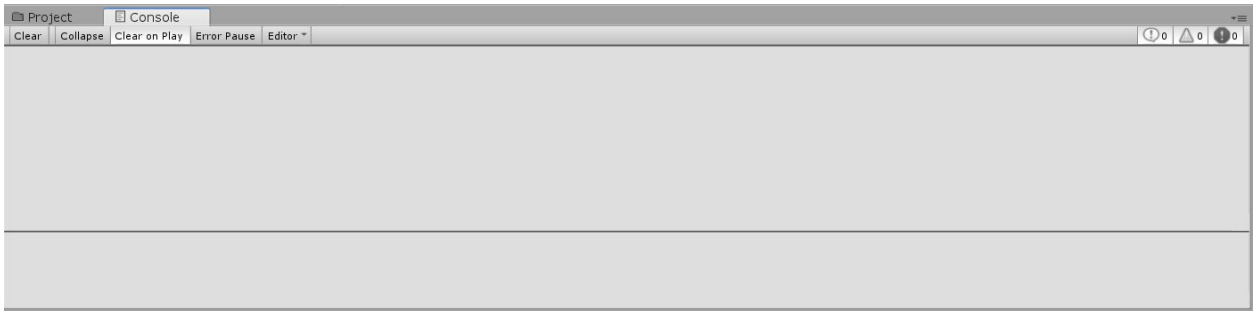
Project and Console

The first window we will look at is the Project window.



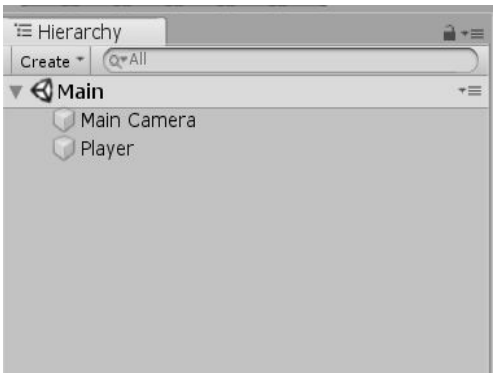
This window shows all of the files and folders that are usable assets for your project. Everything you need, from sprites to scripts will be here. In a new project, the Assets folder will not have anything in it. Here, I have already created a Scenes, Scripts, and Prefab folders. This is to help keep everything organized. If you go to each of the folders, there's only a couple items in each one, but for larger projects, there will be many more items in each folder. Scenes are different files that are made using all of the assets, think of them as different levels. They all draw from the same pre-existing assets, but have a different format and layout.

Next to the Project window is the Console window.

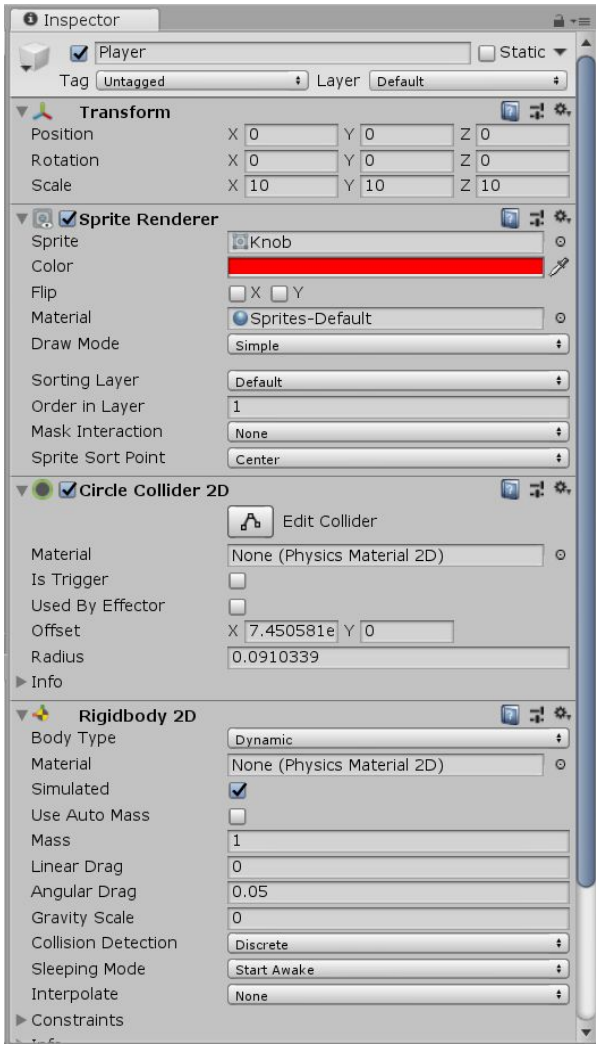


If there are any errors that occur within the project, like compiling errors in code, missing components or invalid code, the errors will show up here. Also, if you print anything in code (using Debug.Log() or print()), it will also show up here. This window can be extremely useful in debugging code.

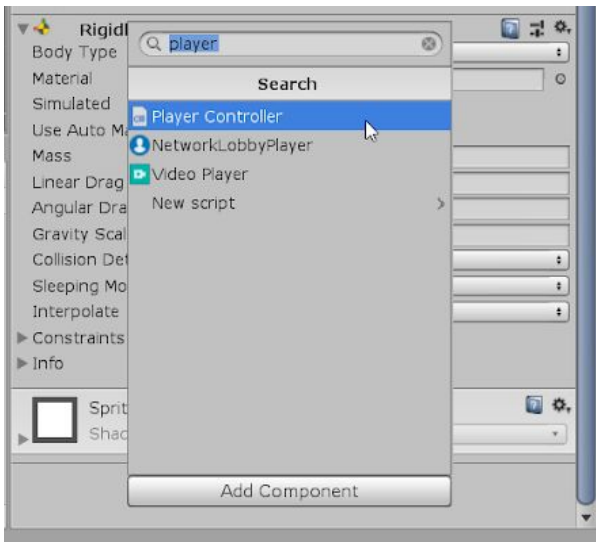
Hierarchy and Inspector



The Hierarchy window contains all of the Game Objects that are currently being used in a scene. Game Objects are the basic building components in Unity. Every object you see is basically some sort of Game Object, and each Game Object can have extra components and scripts attached to it, giving each of them their own properties and behaviors, like colliders and scripts.

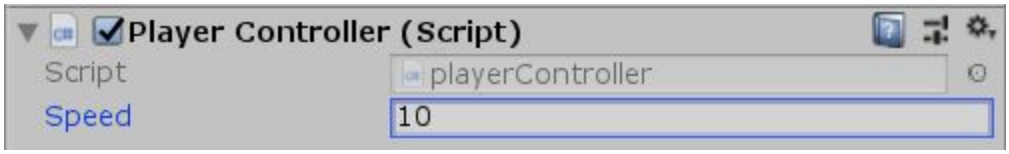


The Inspector window (see left image) allows you to view and edit the properties and components of Game Objects. If you click on the Player game object in the Hierarchy window, you can see all of the components currently attached to the Player. Right now, there should just be Transform, which controls the object's positioning, Sprite Renderer, which allows its sprite to show up on the screen, the Circle Collider 2D, which creates a circle collider (think hitbox) for it to collide with other things, and Rigidbody2D, which controls its physics details.



To add new components or scripts, press Add Component, which is located at the bottom of the Inspector. We are going to add a script to help us control movement. You can either add a pre-existing script or create a new one, but for this lab we've already created one for you. Just type "player controller" in the search bar and it should show up, like in the image on the right. (See top right image) Click on it to add it and it should show up under everything else. Another way to add a script is to simply locate it in the project window and drag it into the inspector.

Now, the Player object will run the player controller script, which will allow us to control movement. We will go into details about scripting on a different day. But for now, **change the number for the player speed inside the player controller script**.



Change this number to any number that is positive.

Scene and Game

Here, you can organize the layout of this particular scene. The Game window shows what the game will look like through the camera when it is running, and the Scene window next to it is where you will actually move and manipulate each item in the scene. There is a toolbar in the top left corner that will help you move around in the scene.

1. Moves the scene view
2. Moves objects vertically or horizontally
3. Rotates objects
4. Changes scale of objects
5. Moves objects freely
6. A combination of 2-5

Above the Scene and Game windows are the play, pause, and step buttons.

1. The play button will start running the game inside the editor so that you can test the game. Hitting the play button while the game is running will stop the game.
2. The pause button will pause the game on a frame when you are playing it, which is useful when you want to be able to tweak something or debug a problem
3. The step button allows you to skip forward a frame. It will pause if the game isn't already paused. This is also useful if you want to debug since it allows you to go frame by frame

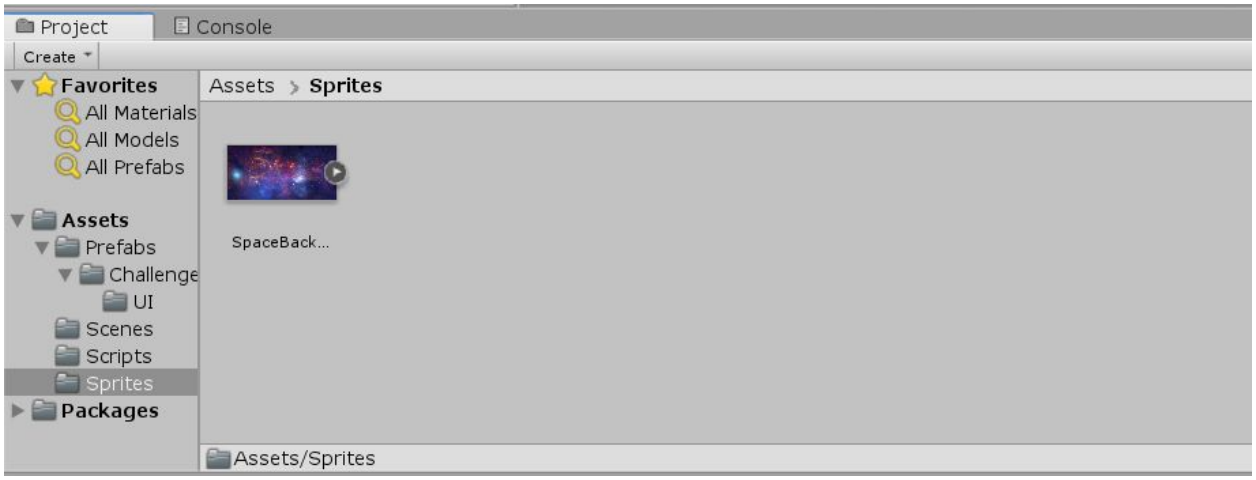
In addition, you can change variables in the inspector while the game is running, but when you stop playing the game, your changes will be reverted back to what they were before you started playing. This is important to keep in mind while testing your game.

Importing Art Assets

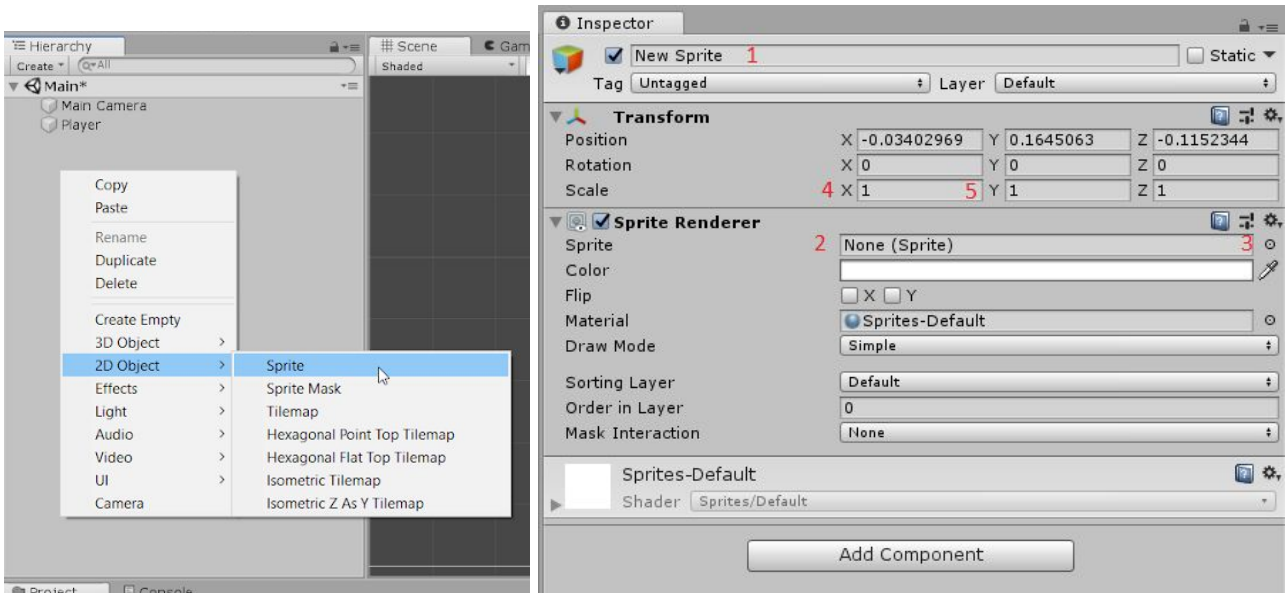
So that's the general layout of Unity. Unity comes with some default art assets, but for the background of this lab, we are going to import an image from outside to use in Unity. In the UnityBasics folder you downloaded should be an image named SpaceBackground.png. We are going to import this into Unity.

First, let us create a folder in our Assets named Sprites, where we will store this background image. **Right click in the empty space next to all of the other folders and make a new folder and name it “Sprites”**.

Now, there are a few ways to import the image. First, you can go into your files and manually move the file from outside of the project folder to inside Assets/Sprites. Or, you can drag the image directly into the Project window of Unity. If you are currently looking into the Sprites folder, this will automatically add it to the Sprites folder.



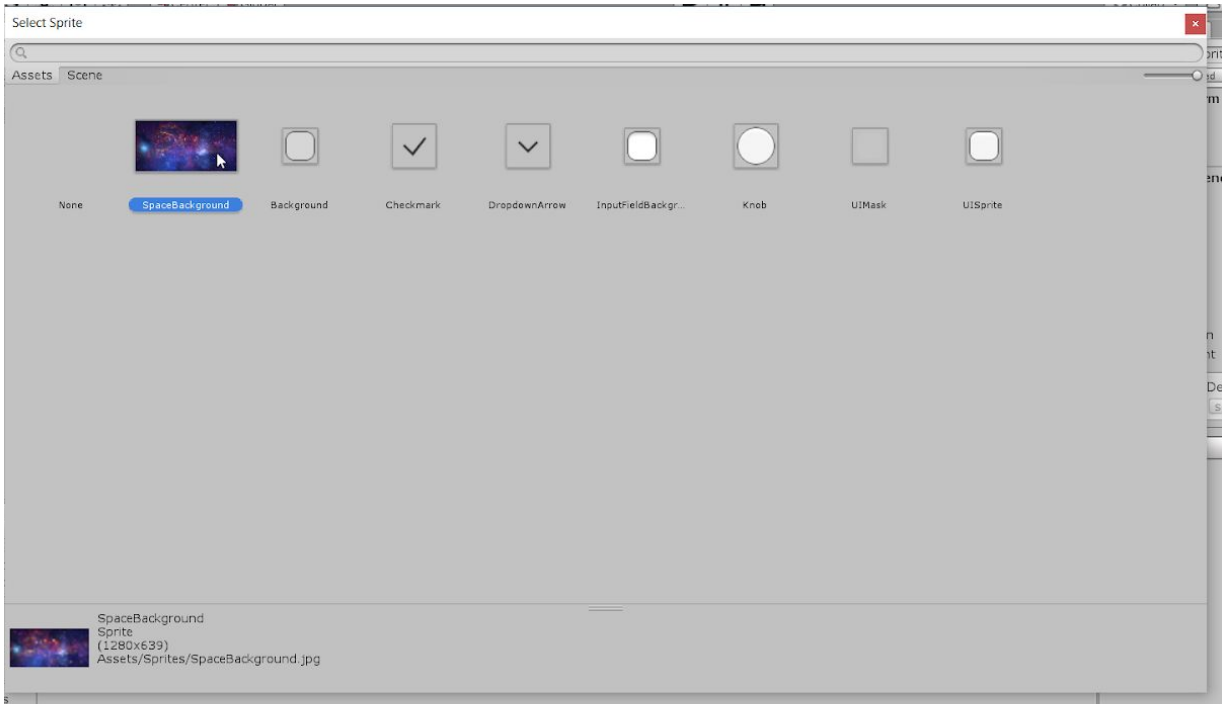
Now, we can add this space image into the background. To do this, we need to create a new Game Object to hold the image. Right click inside the Hierarchy window and select Sprite from inside the 2D Object tab. In the hierarchy, make sure to reset its Transform's position to 0 0 0 so that it is visible.



Left: Making the background Game Object; Right: Inspector with its details

If you click on this now, you should see all of its details in the Inspector. We can change its name to Background (1) and add the space image to its sprite renderer. We can add this with one of two ways. The first is to just drag the image into the box next to the word Sprite (2). The

second is to click the circle (3) and select SpaceBackground.png from the list that shows up.

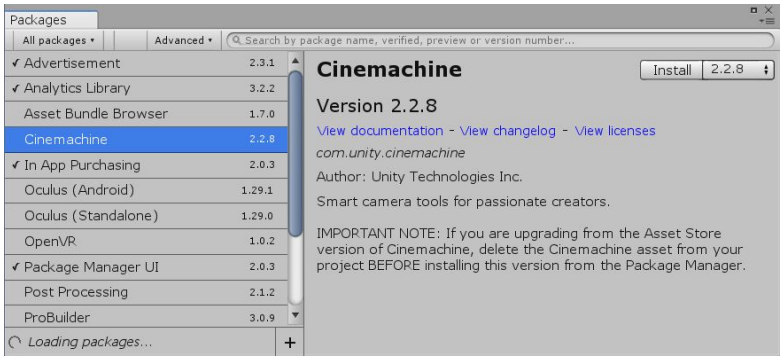


In the inspector, we want to increase the scale of the picture to X = 5 and Y = 5 (numbers 4 and 5 in the picture) so that it is large enough to cover the background. We'll go into the Transform component a bit more later.

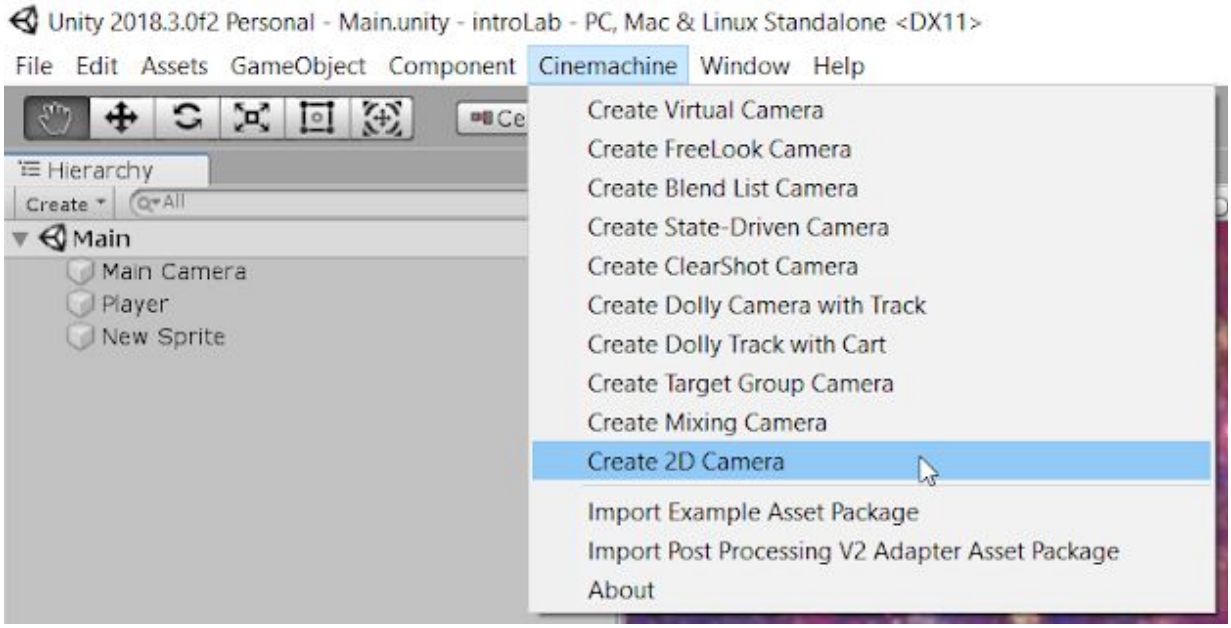
Now, if you hit play, you can see that you can move the red ball around on the background but if you move it off the screen, the camera won't follow the player. Let's fix that.

Packages

A great part of Unity is that there is alot of community support for it, from the forums to community made assets. For the camera, we will be using one of the Unity packages for a smooth follow camera. To access packages, go to the Window tab and find the "Package Manager". It will open the package manager window. Within the All section, if you scroll down, you will find the Cinemachine package

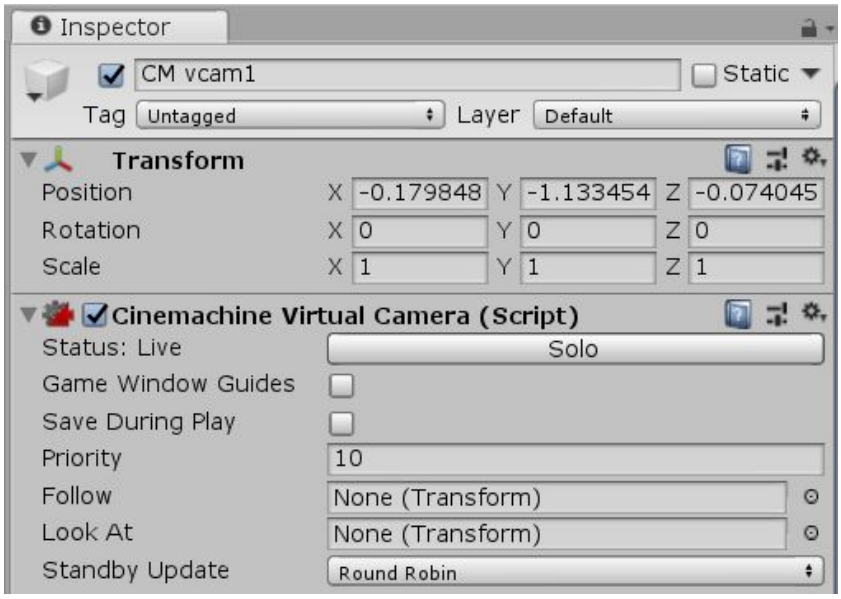


Install the package. Now in the toolbar there should be a tab specifically for Cinemachine. From the tab, create a 2D camera.

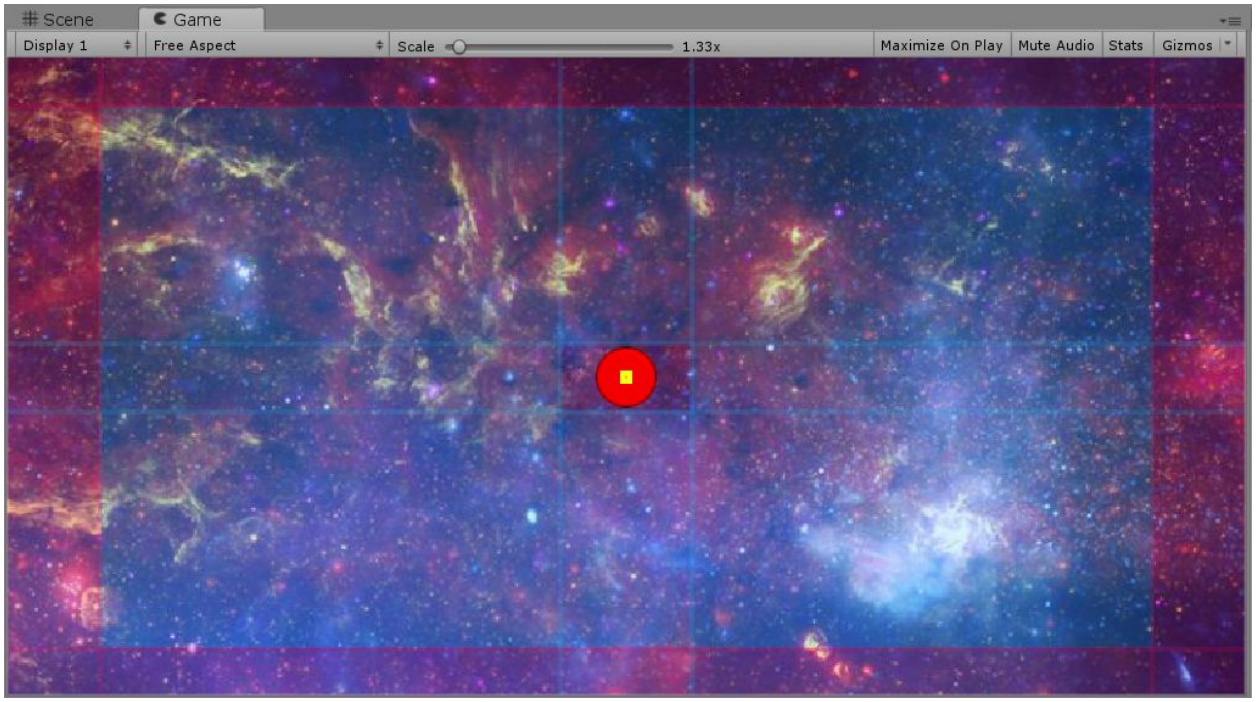


This will attach a Cinemachine script onto the Main Camera that was already in the scene. There will be a new object in the hierarchy as well, the Virtual Machine that controls the Cinemachine. We will only be going through a bit of Cinemachine, but feel free to explore it more!

To get the camera to follow the player, we need to give the camera something to follow. In the virtual machine GameObject, there is a component for the cinemachine script, which has a Follow option.

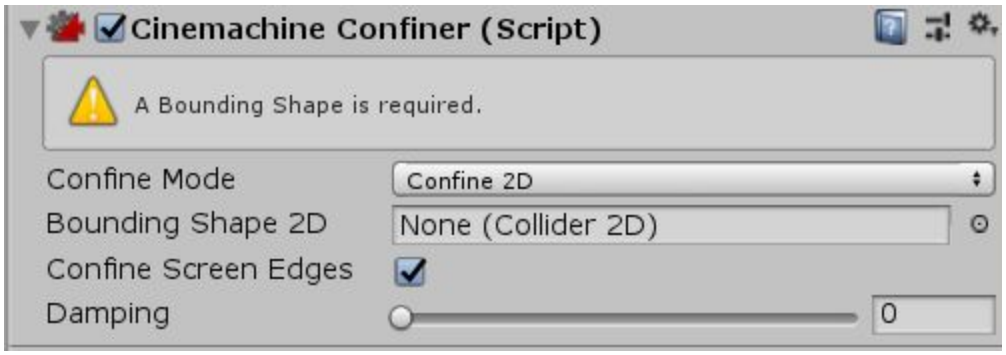


Drag the Player GameObject from the hierarchy into the follow part in the inspector. Now the camera will follow the character. Check the Game Window Guides box to be able to see the camera's boundaries, and in the Body section, change the Dead Zone Width and Height to 0.1 to specify where the camera shouldn't immediately follow. Now if we go into the game view, and while still having the virtual camera selected, we can see this:



The inside box represents the dead zone area that the player can move in and the camera won't follow. While it is inside the blue area, the camera will smooth follow the player. The area outside that, the redish part is where the camera will just follow the player at a one-to-one speed. You can change the size so that the player can move more or less without the camera moving as well. You can play the game and try different sizes with the virtual camera.

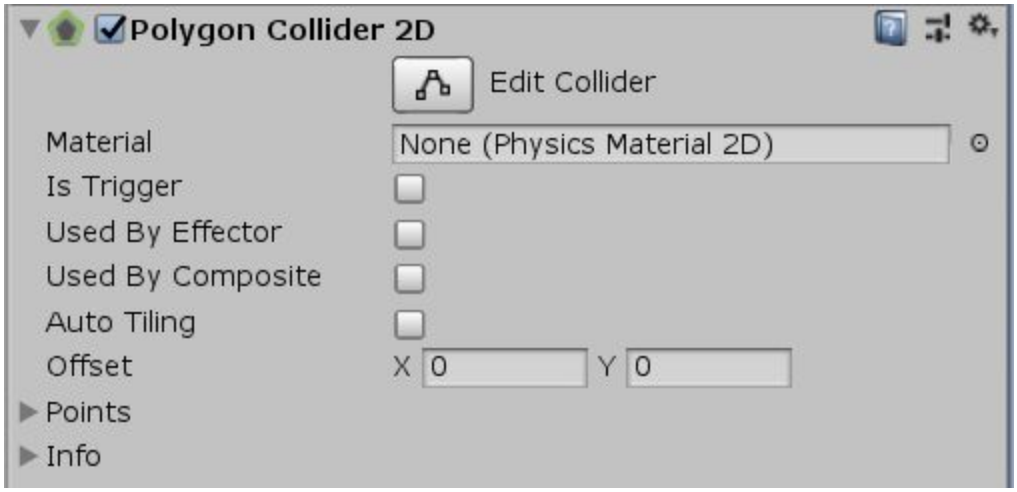
Right now, the camera will follow the player but will not stop at the edge of the background. To do that, Cinemachine has a component that can help us with that. On the virtual camera, we can add an extension by looking at the bottom of the Cinemachine Virtual Camera component. **Add the CinemachineConfiner onto the virtual camera**. It will show up as a component.



To properly confine the camera, we need a Collider component. A Collider component does exactly what it sounds like, it allows objects to collide with each other. For this, we will put the collider onto the space background object that we've made.

For CinemachineConfiners, we need to use a PolygonCollider2D. Since our object is a simple rectangle shape, adding a PolygonCollider should just make it into a rectangle. We add on the PolygonCollider the same way we added the script earlier. **Go to Add Component and add a PolygonCollider2D onto the Background game object**. PolygonCollider2D can be found in the Physics2D section, or you can type in "PolygonCollider2D" to find it. Make sure that it is 2D, otherwise it will add a 3D collider and not work properly.

Colliders act as physical objects and will collide with anything else that also has a collider. Colliders can be changed to be triggers, which do not physically collide. Certain code events will be triggered, but there is no physical collision. In this case, we do not want the background to cause physical collisions, so **check the box that says Is Trigger** in the trigger component.



We want to edit the collider so that we can make sure it covers the entire background. If you are in scene mode and have the background selected, you should see a green outline that shows where the colliders currently are. As long as the green outline surrounds the entire background, it will be ok. If it doesn't, inside the collider component, there is a section to edit the collider. Once you click that you can manipulate the green outlines to fit around space background.

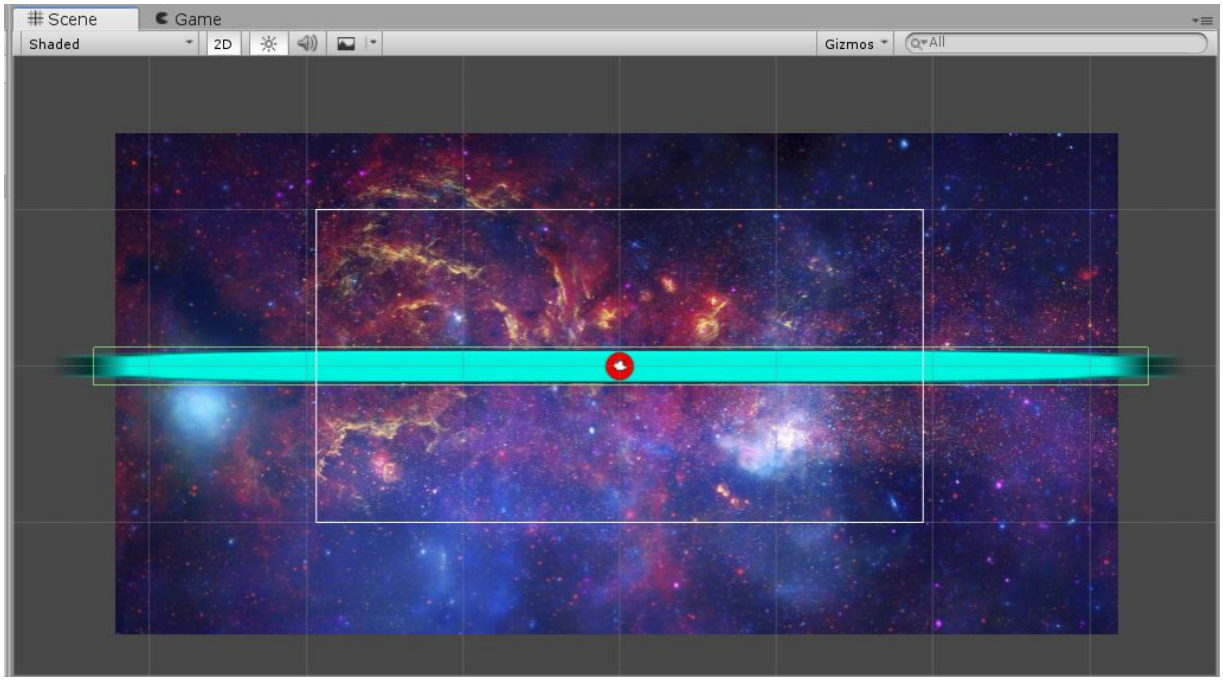
Now that the PolygonCollider is set up, we need to attach it onto the virtual camera. Click on the virtual camera and you can see the spot that the PolygonCollider should go. On the virtual camera, it is the Bounding Shape 2D. **Drag the Background object into that box and it will attach the PolygonCollider to the virtual camera**.

Now, if you hit play, you can see that the camera will follow the player, and it won't show the default blue background no matter how far you go. But then you can go off the screen as far as you want. To limit the player, we should add some walls.

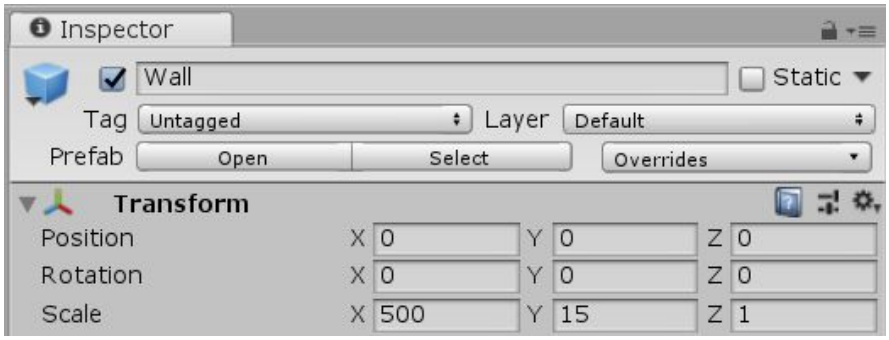
Prefabs

We're going to finish building this simple project by adding some walls into the scene so that the player is physically confined.

First, look in the Prefabs folder. Prefabs are Game Object templates. For example, if you have many enemies, you can create a Prefab with one, prepared with all of your necessary components, and create multiple copies of it in specific locations by simply dragging the prefab into the Scene window, or into the Hierarchy window. We are going to do this with the Wall prefab. Drag the Wall prefab into the Hierarchy window. It should appear like this.

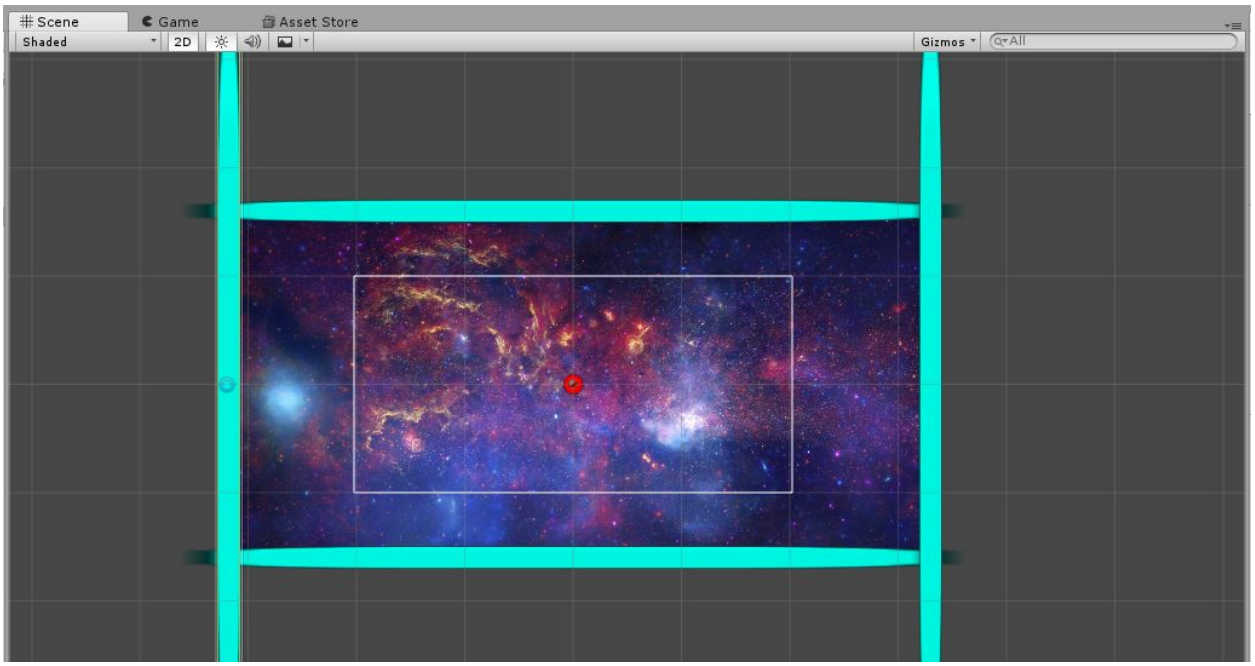


These walls will be at the edge of the screen, so we can move the wall in three different ways. The first way is to manually move the wall in the Scene window. The second is to adjust the values in the Transform component. This method just requires a bit of fiddling to get to the right position.



The final method is to put your mouse on top of the "X" or "Y" in the transform component and drag left or right to move. This is the method that I recommend because it is easy to adjust and you can be pretty precise with this.

Now, repeat this three times so that there is a wall around the ball that just reaches the edge of the background, or like this.



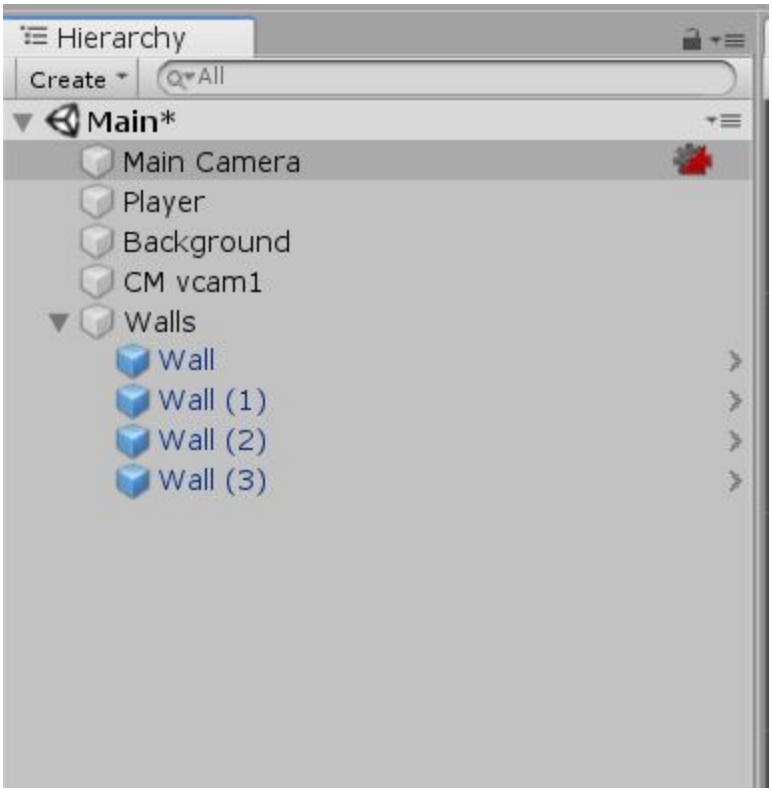
An easy way to modify a wall to become vertical is to change the scale (in the transform component) from being X:500, Y:15 to X:15, Y:500.

Hierarchy

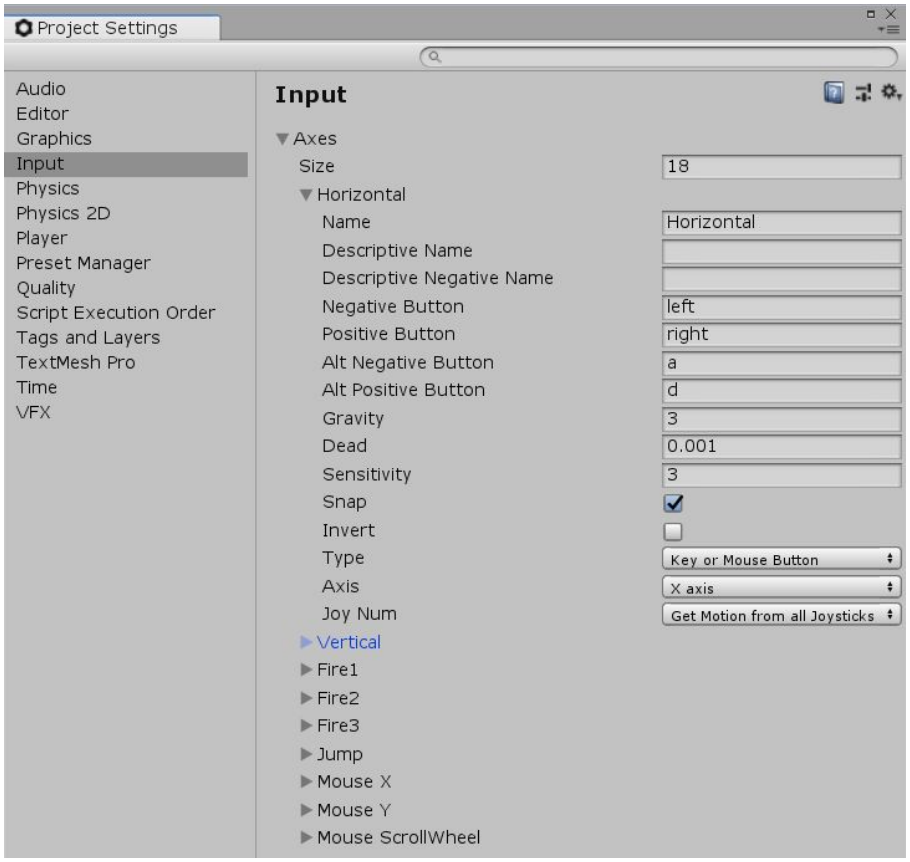
So in the hierarchy, game objects can be organized into parents and children. The parent and child relationship can be used for various things. Children will inherit their parent's positions and transform. RectTransforms (for UI) are dependent on parent transforms, the children of a parent can make up the components of a character, and the parent can be used as an organizational tool to make the hierarchy look neater. In this lab, we will be using a parent Game Object for organization.

We currently have 4 Wall objects in the hierarchy. By making them all children of a parent object, we can make the hierarchy look neater. We can do this by again making a new object but this time, instead of a 2D sprite, **create an empty object** by right-clicking in the Hierarchy and selecting Create Empty and naming it “Walls”.

You might notice that there is no sprite renderer. That is good, we don't necessarily want a sprite to show up. It does have a transform though, because all Unity objects need to have transforms.
Make sure the transform is at (0, 0, 0). Now we can **drag each of the wall objects onto the parent wall** inside the hierarchy. Do this for all 4 and now it looks a lot neater.



Everything is set up now, and the game is playable, but the movement of the player is still a bit off. It might feel a bit slow and sluggish right now. We can change that through the project settings, which can be found in the toolbar through Edit > Project Settings. Then, under project settings, there is an option to edit inputs. Clicking on it will bring up the input settings. Unity provides alot more default inputs than than we are using for this lab. The code only uses 2 of these inputs: Horizontal and Vertical. If you expand either input, you can see a variety of settings.



Some of the basic settings are name (which is referenced by code), negative and positive buttons (which are which buttons control it), type (using mouse, keyboard or controller), and axis (used only for controllers, which axis is being used). The ones that we will be changing to help the movement feel better is gravity and sensitivity. We can **set both of those to 1000** for this lab. The default of 3 is too low for most projects so it is good practice to change them to be higher. If you want to know more about what each of the settings do, you can look at the Unity documentation for each of them: https://docs.unity3d.com/Manual/class-InputManager.html

Now, by pressing the play button, we have a circle that can move around within the confines of the screen on top of a nice space background!

Inside the Prefabs folder, there's another prefab that we haven't used yet. It is meant to be a collectible item. The code has already been provided in scoreController.cs. This script should go on the Player. **Your job is to add it into the game so that the player can collect the collectibles when it runs into them**. An optional challenge that you can do is to add some UI to show how many collectibles have been collected. More details are below in the Challenges section.

Check-off

Run the game and show a facilitator that the player can move move around and collect collectibles, and that the camera smooth follows the player. Also go to the edge of the walls and show that the camera is confined to the walls.

Challenges

These are extra challenges if you finish the lab early and want to work on some more Unity. Some challenges are harder than others, but we are here to help if you need it! If you can't get them now, don't worry! We will teach you how to do all of this in the coming weeks.

- Change the input mapping to anything you want
- Add UI that will show the score of how many collectibles the player has collected. The code is also already provided and can also be found in the Challenges folder. The Canvas and EventsSystem must both be in the hierarchy for UI to work.
- Change the colors of the collectibles. If you are programmer, see if you can try to have them randomly select a color each time the game is started.
- For programmers: write a script that will spawn collectibles randomly

Hints:

- For UI: There is a script specifically made to increase the score as the player gets more collectibles. It should go on the Player Game Object. There are some objects that need to be attached to the script.
- For color changing: The Sprite Renderer controls the color of the sprite.