Game Design – Unity Workshop

Activity 1

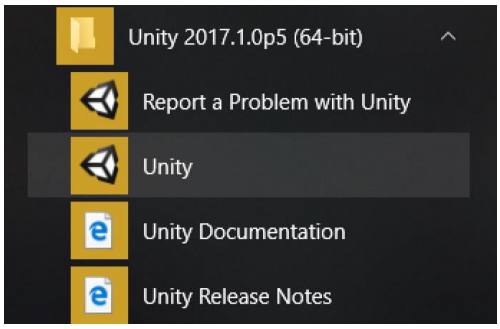
Unity Overview

Unity is a game engine with the ability to create 3d and 2d environments. Unity's prime focus is to allow for the quick creation of a game from freelance to professional. It contains some modelling capability.

Goals:

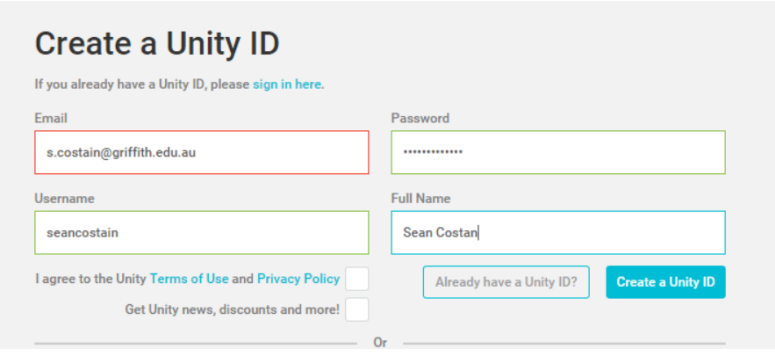
- Introduction to Interface
- Simple model creation
- Player control of model object

Load up unity

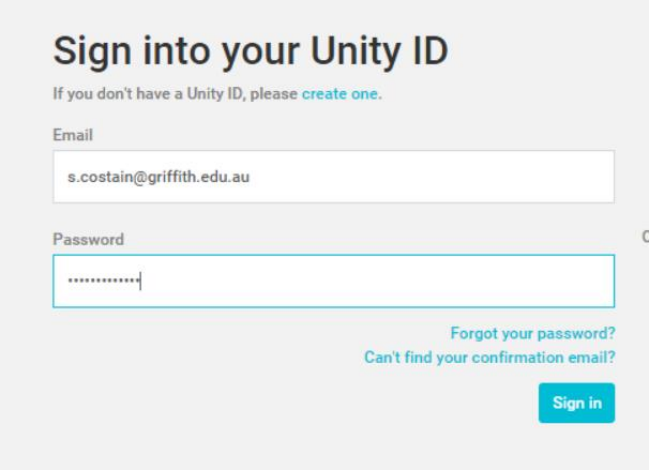


If you don't have an account, you will have to create one. Otherwise sign in with your registered account.

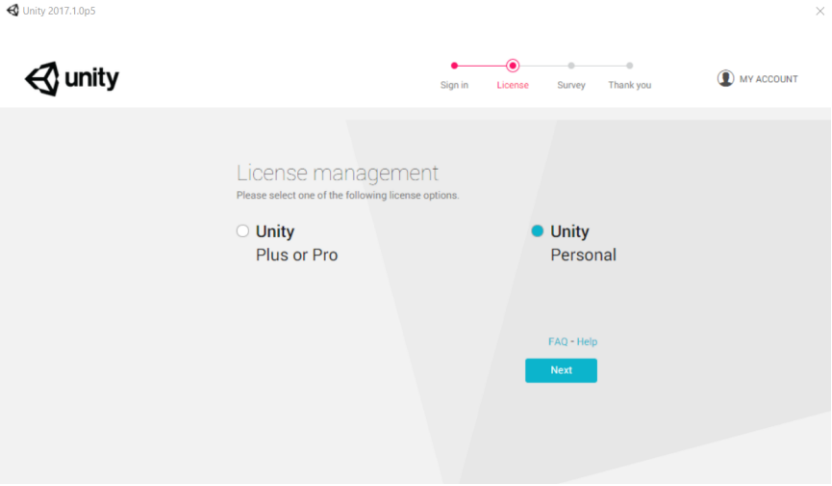
Create:



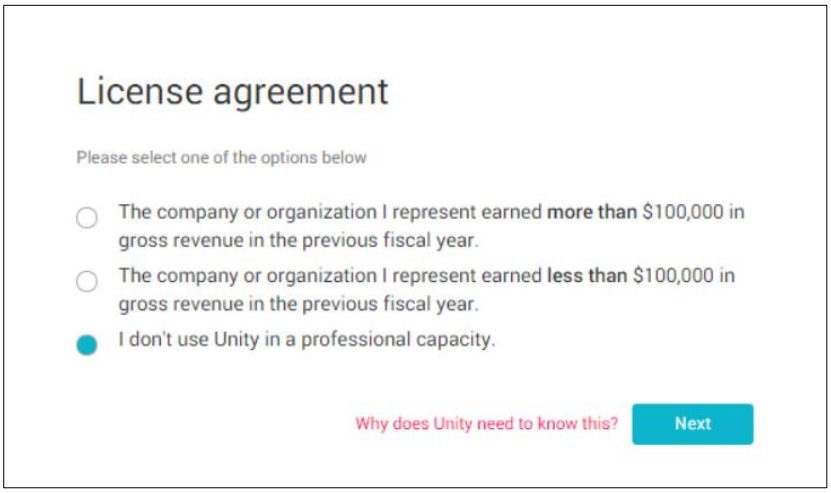
Sign in:



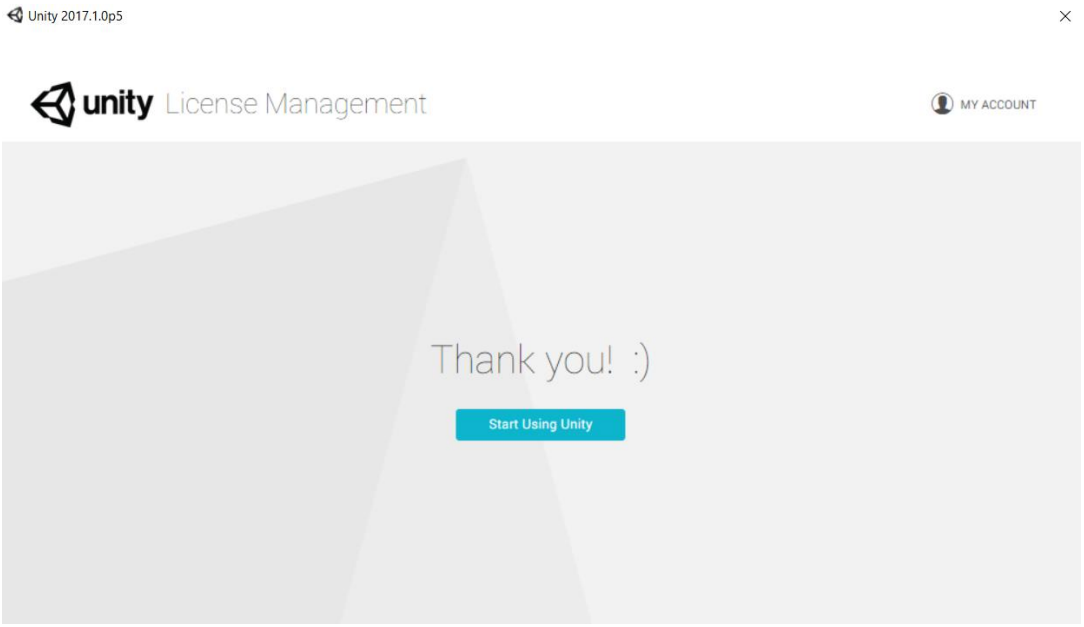
If it pops up, select Unity Personal



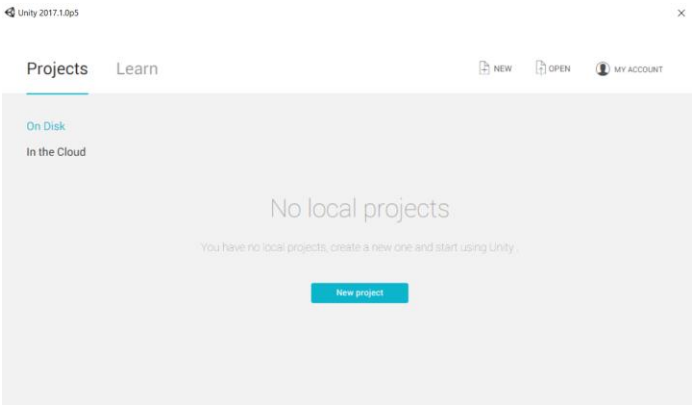
Agree with the license agreement



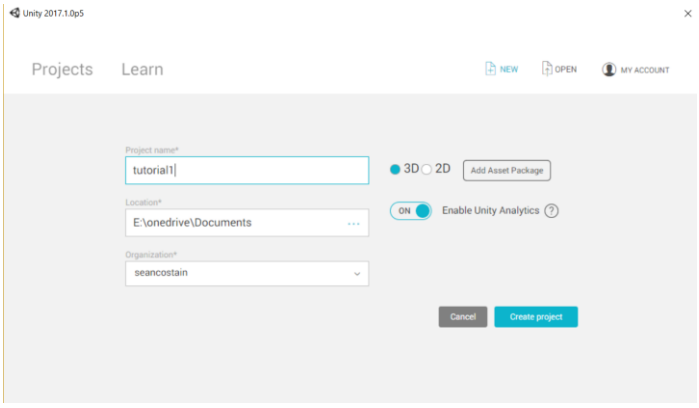
After activation



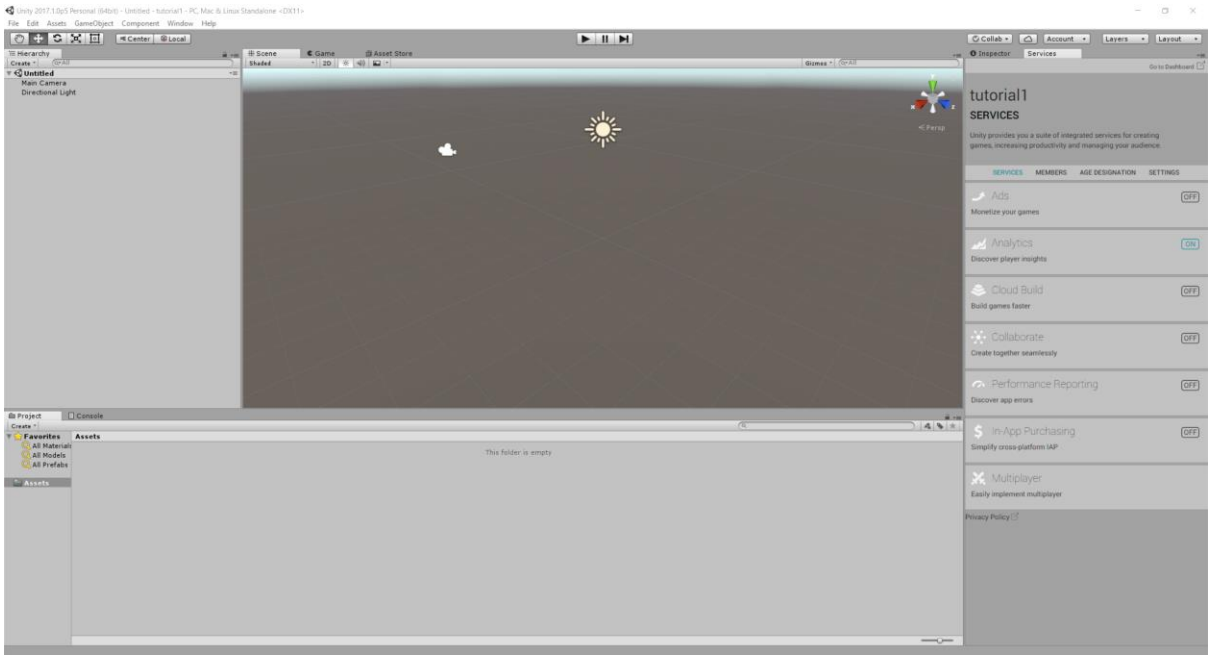
From the start menu, select New Project, On Disk



Fill out the form with a name, in this case, tutorial 1

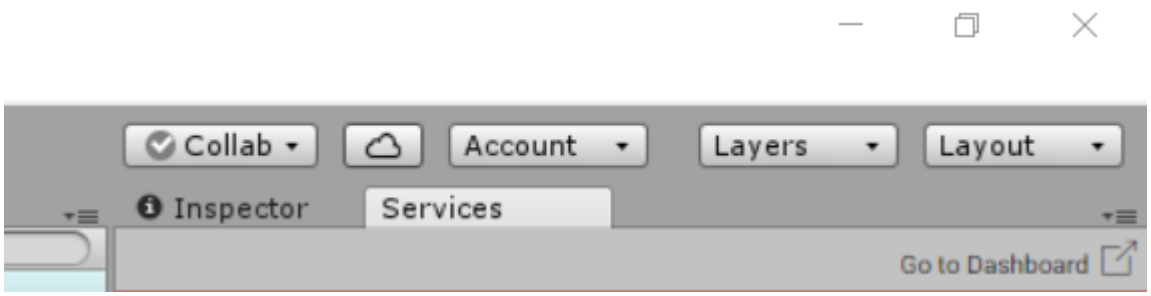


Click on Create Project, this will open Unity into its default layout



Like Maya, there is a bit of learning with the user interface, so let's first change from services to Inspector and have a look.

In the top right, click on the inspector tab



This area will populate when you have objects in the game world or you select an object.

Let's look at the User Interface, there are 5 main areas:

- Toolbar
- Hierarchy Window
- Project Window
- Inspector Window
- Scene View

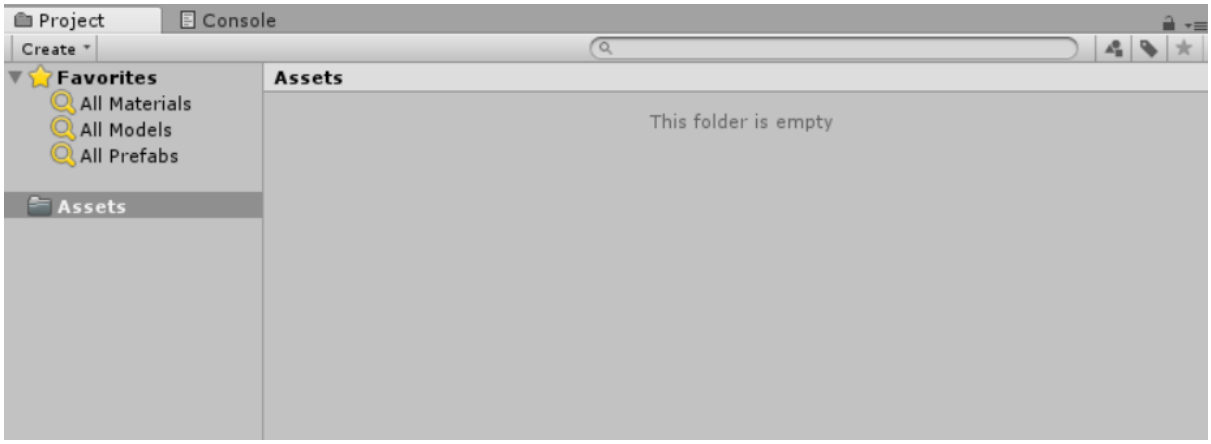
Toolbar

This is the only part of the UI that cannot be re-arranged, it contains the most used buttons for interacting with the interface.

Hierarchy Window

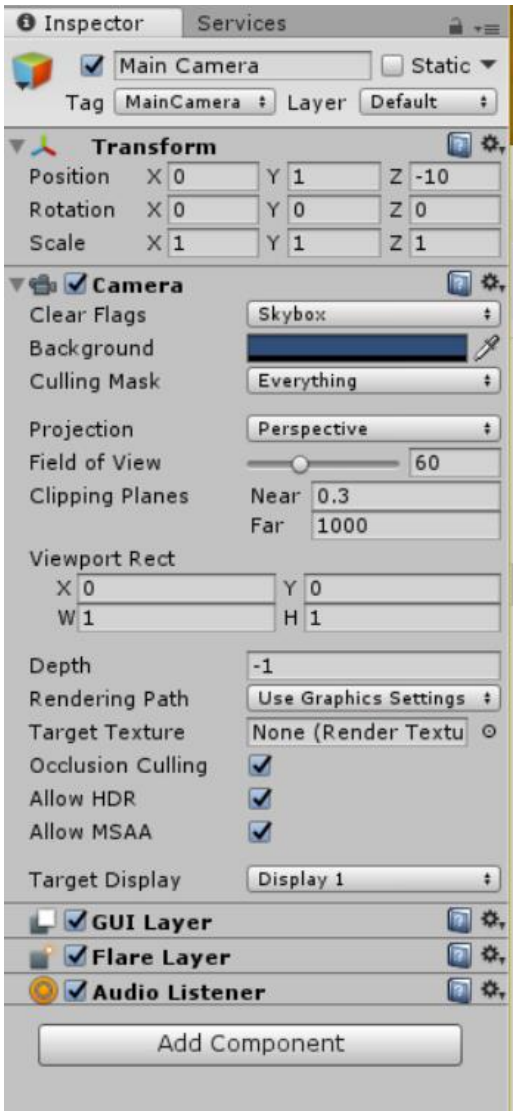
This is a listing of all elements that are within the current project. This allows for the ability to have a parent item with children, i.e. groups of similar objects

The Project Window



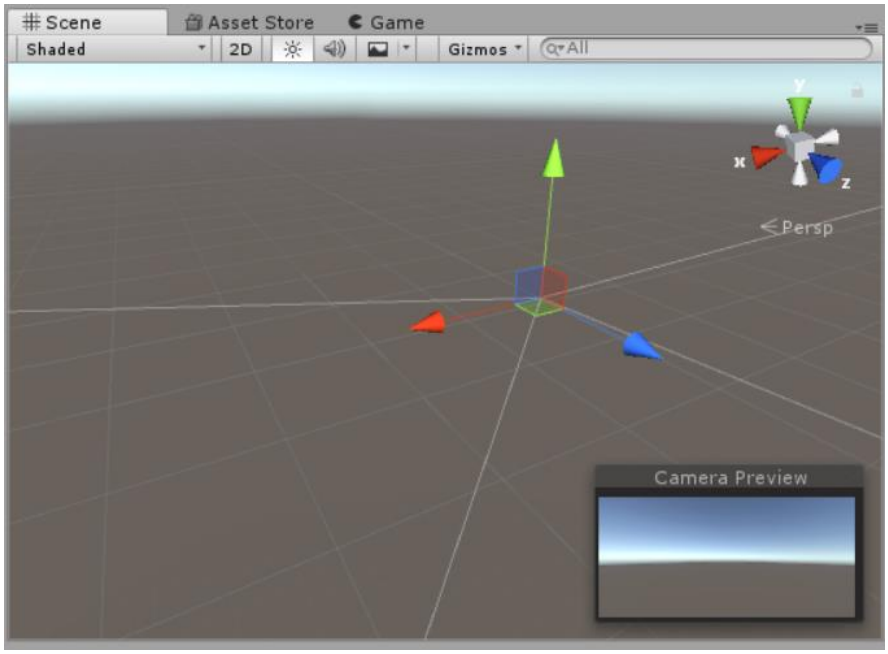
The project window displays all the assets that are in the project. These can be stored in folders to allow for ease of management. This is where the imported assets go.

The Inspector Window



The Inspector window is a graphical view of an asset. Be it a 3D object, camera or light source. Any element in the scene will have a matching field in the inspector window. The add component button allows for additional elements to be added to an object, these items can be items such as rigid body, gravity, scripting for control and so forth.

Scene View



The scene view is where elements of the game are placed, allows for visual navigation and editing of elements. This can be done in either 2D or 3D perspective based upon the project.

Unity Key Shortcuts

Tools	
Keystroke	*Command*
Q	Pan
W	Move
E	Rotate
R	Scale
T	Rect Tool
Z	Pivot Mode toggle
X	Pivot Rotation Toggle

V	Vertex Snap
CTRL/CMD+LMB	Snap

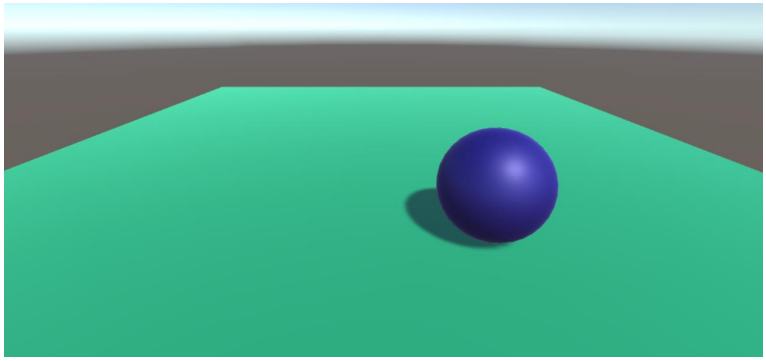
GameObject	
Ctrl/Cmd+Shift+N	New empty game object
Alt+Shift+N	New empty child to selected game object
Ctrl/Cmd+Alt+F	Move to view
Ctrl/Cmd+Shift+F	Align with view
Shift+F or double-F	Locks the scene view camera to the selected GameObject

More Keys here: https://docs.unity3d.com/Manual/UnityHotkeys.html

Build Object: Drop/Move Ball

Now that we have looked at the layout of the interface, it's time to start creating an element in unity. Creation can happen in many different places in unity, it is possible to use the menu to access the create section and right click inside panels such as the hierarchy and assets panels. The more you play with unity the easier it becomes, in this case, we will look at right-click in panels.

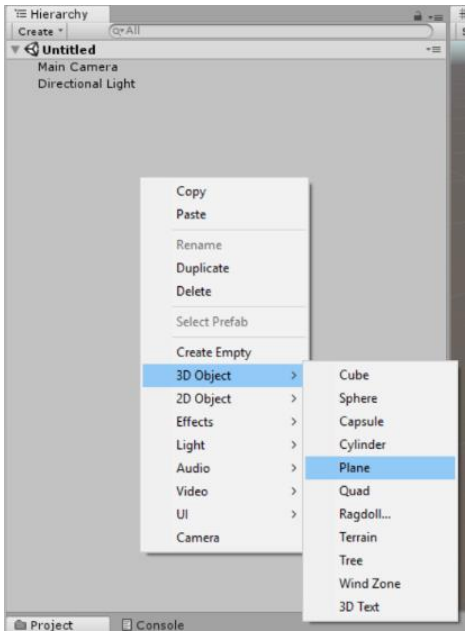
To start with, with need to know what we are building, in this case it is a simple scene that looks like this:



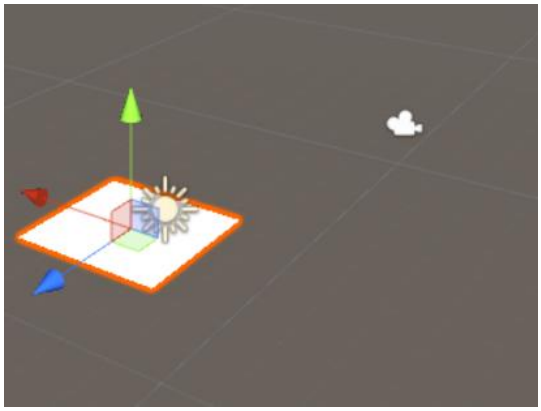
It looks very simple, but it is the basis of understanding elements of the unity system, as such, we can build quickly and learn the basic requirements needed to use unity.

To start with let's create a plane.

Right click in the Hierarchy and select 3D object->Plane

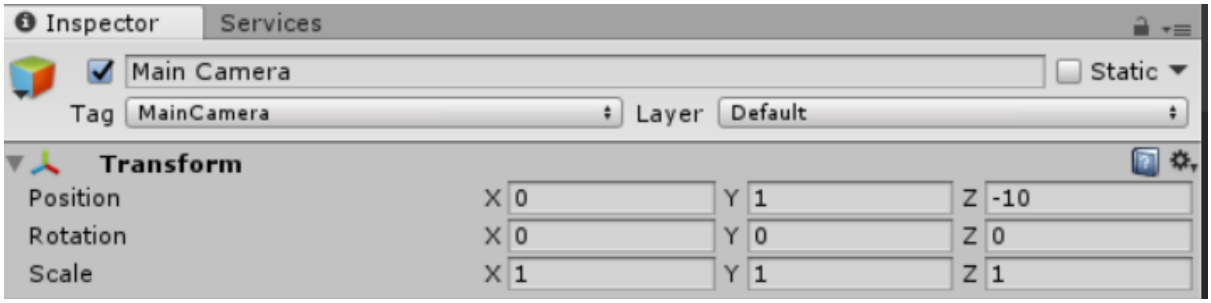


This will produce the following in the scene view



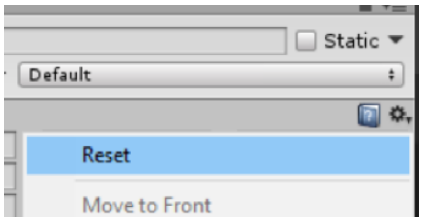
If you examine the scene view, and hierarchy, you will notice that the scene has 3 elements; Main Camera, Directional Light and Plane. The first thing we want to do is centre all these objects, to do this we will click on each element in the hierarchy and then look at it's settings in the inspector.

We will start with the Main Camera, click on it and then look at the Inspector.

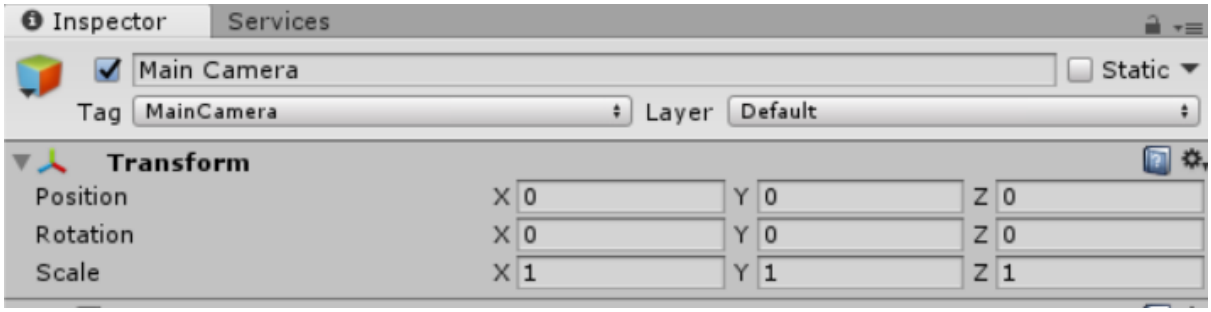


When you click on the main camera, you will see that it has the above position elements, in the transform box, there is a cog, click on this and select reset. We will do this for all elements when we

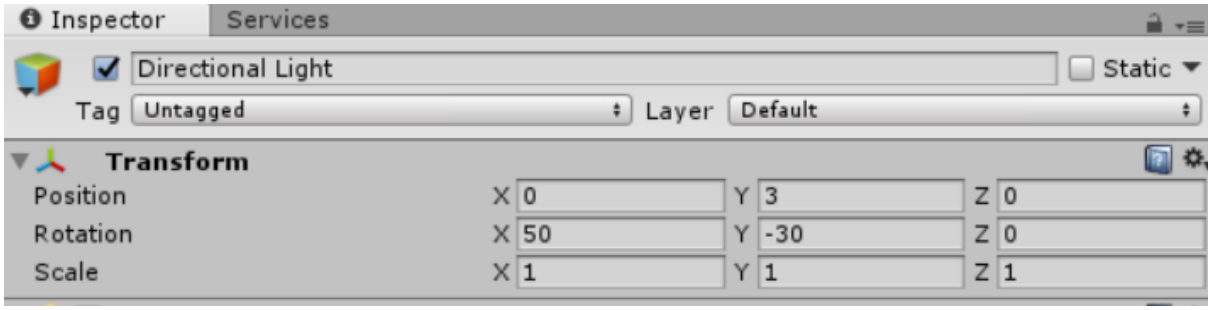
put them into the scene. This places all items at the 0/0/0 point on the scene. From there, we know where and how to move items.



This then provides the new position:

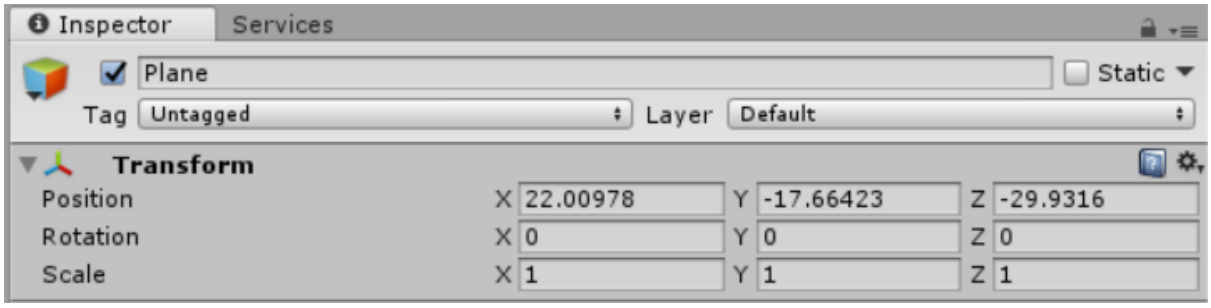


Now, click on the directional light

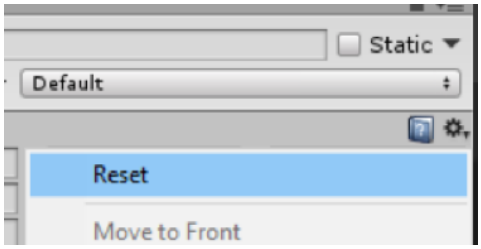


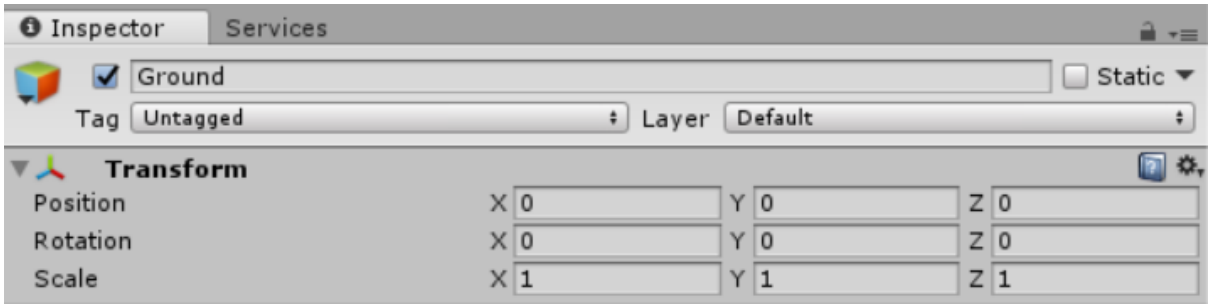
As you can see, the primary difference is the Y position, so the light is above the 0 point and there is a rotation applied. For the moment, leave directional light alone.

Now click on the plane

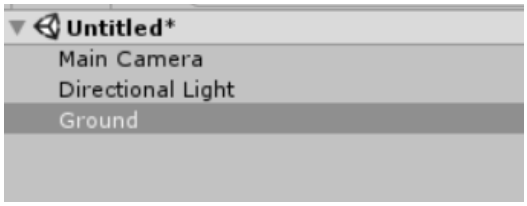


As you can see, the position numbers are all off centre, so we will need to do a reset on this element, we will also change the name to ground.

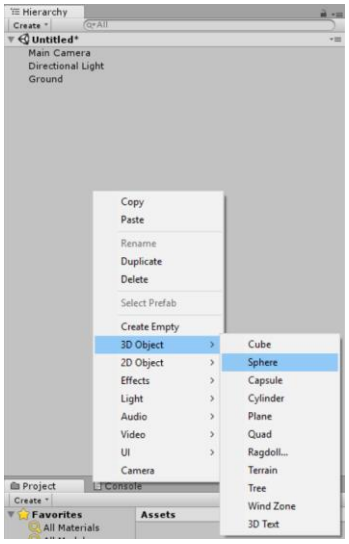




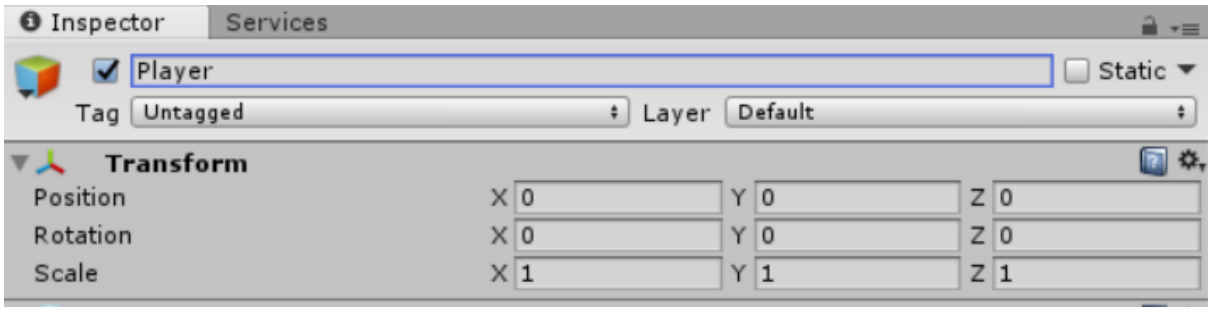
When you look at the hierarchy panel, you will see that the Plane has been renamed as well.



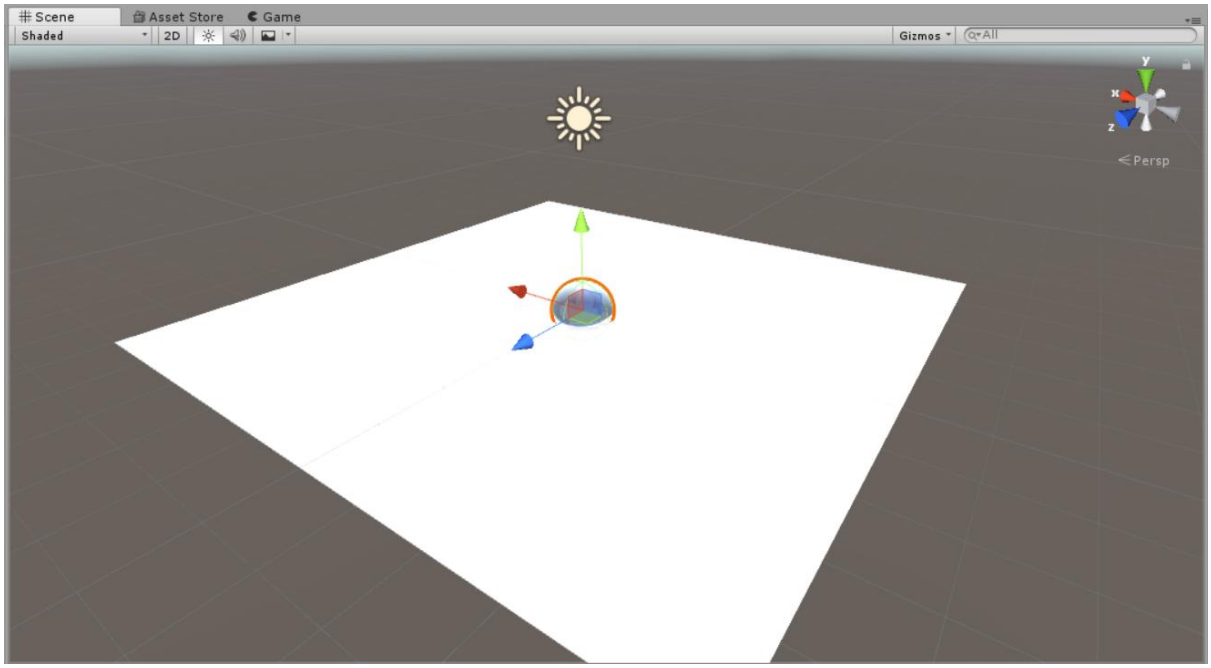
Next, we will create the sphere. Right click in the hierarchy panel and select 3D Object->Sphere



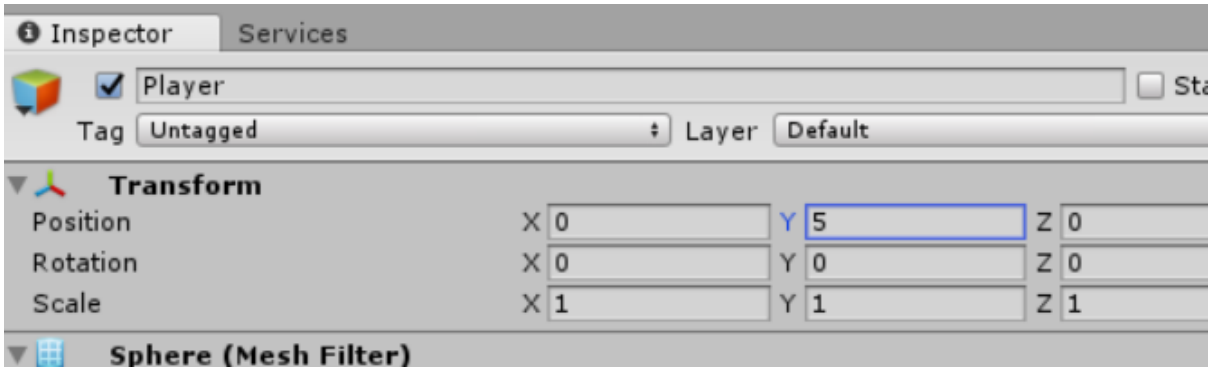
Click in the Inspector Panel, Transform -> reset and then name the Sphere as Player.



Using the scroll wheel, zoom in on the scene, you should see the following.

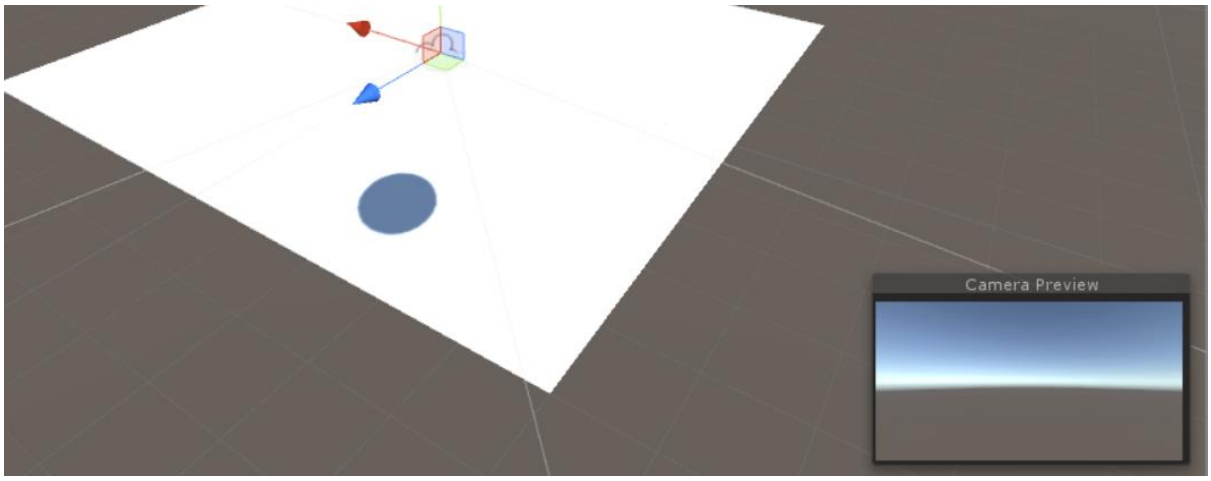


This is normal behaviour for elements we have reset. Now, we want the ball to drop from a height and then be moved by the end user. To do this, select the Player object and then change the Y position in transform to 5.



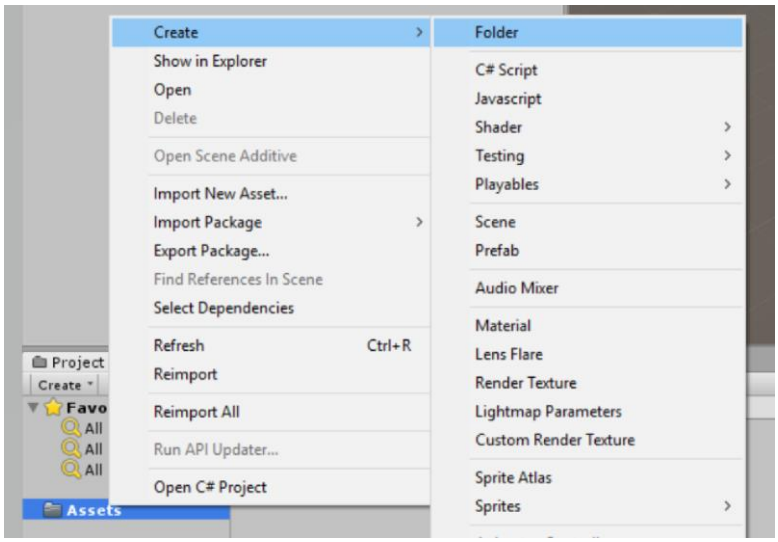
Now we need to check the elements via the Main Camera.

Clicking on the main camera will open a preview window in the scene.

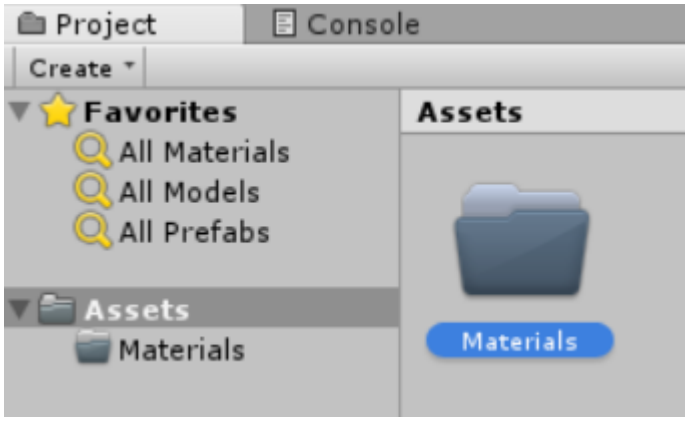


To make this more useful, as we currently have a white sphere on a white plane. So, before we start moving the camera around to view everything, let's create a couple of materials so we can give the ground and ball differing colours. To do this we will create a new folder in the assets pane.

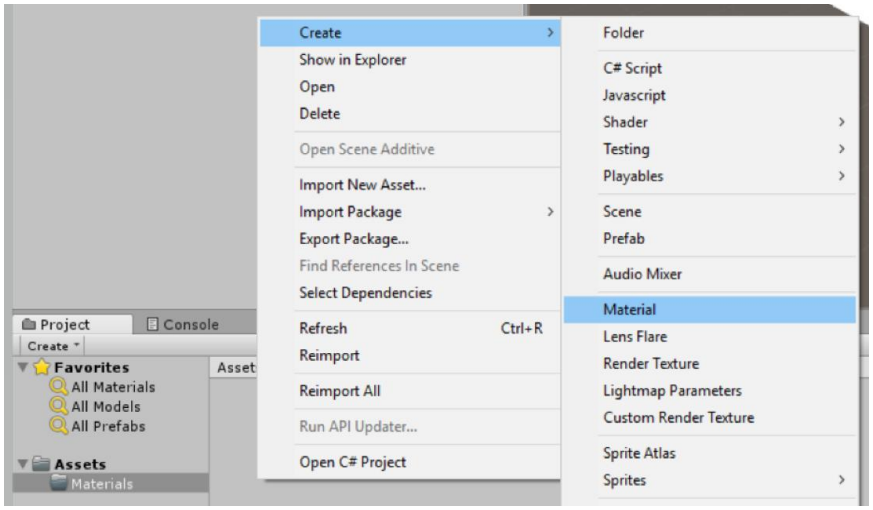
Right click in Assets and select New Folder, call this Materials.



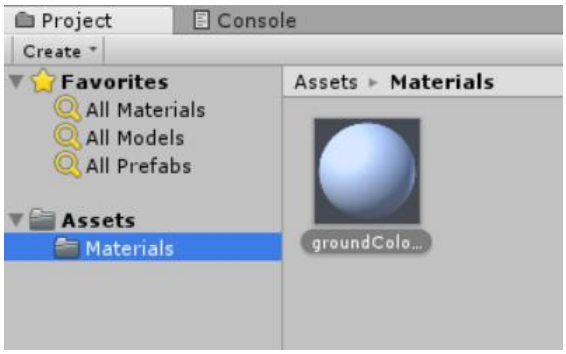
This will produce the following



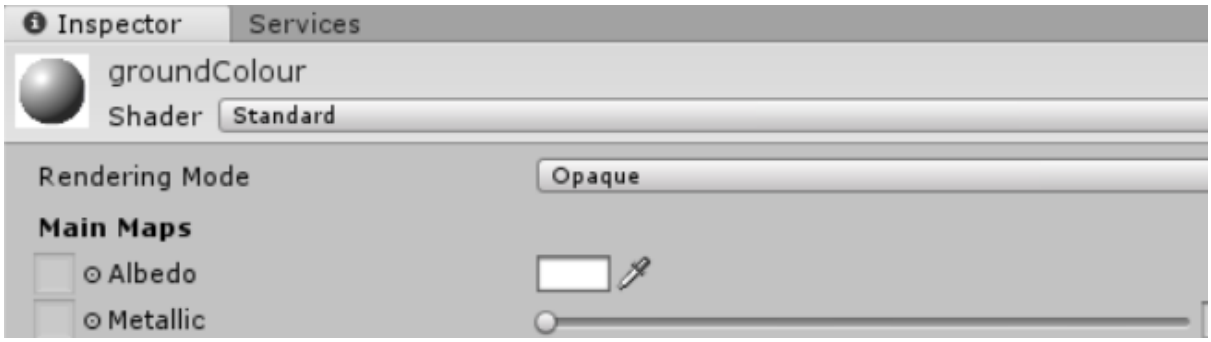
Now, double click into the Materials folder, then right click->Create->Material

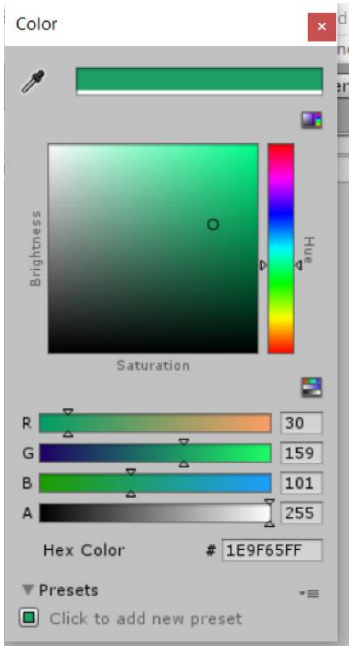


Name this new Material groundColour.



With the material groundColour selected, look at the Inspector Panel, next to the Albedo is a colour box, if you click on that, it will open a colour palette, from here select any colour for your ground.





In this case, I have selected a green:

R: 30

G: 159

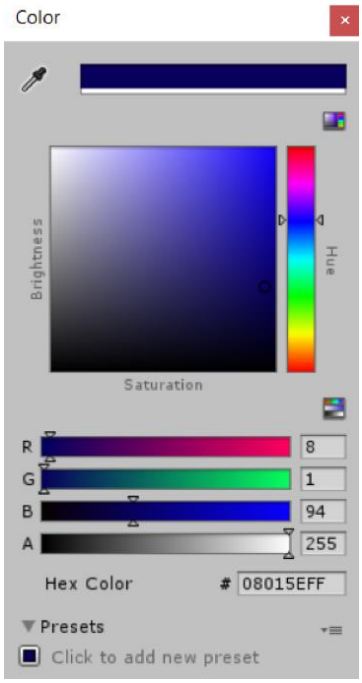
B: 101

A: 255

As you can see, the colours can be done through the colour picker, a known set of RGBA values or even a Hex colour.

Once done you can then click and drag the groundColour from the Assets->Materials folder onto the Ground object in the scene. This will then colour in the ground object.

Repeat this process, making a new material called playerColour, I'll be making a darkish blue material, then drag it onto the player object.



The blue selected is:

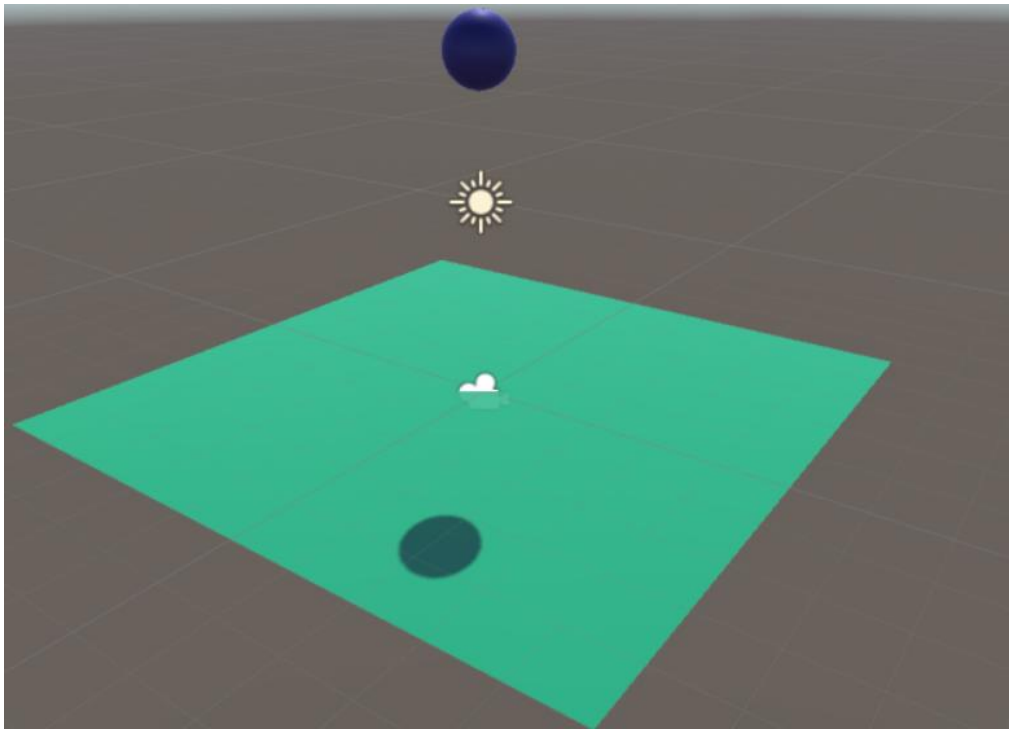
R: 8

G: 1

B: 94

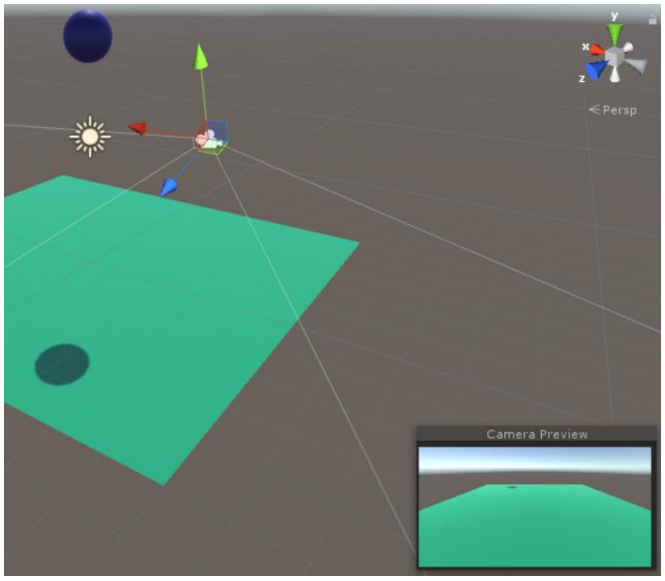
A: 255

This should give the following view in the scene view panel:

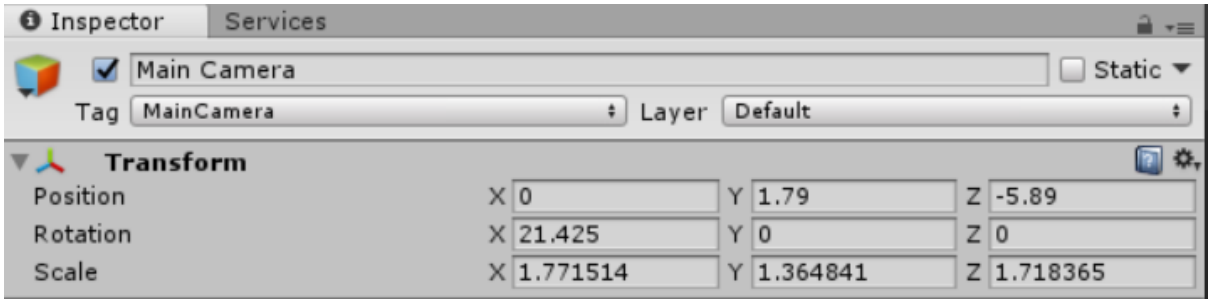


Now that our objects are easily distinguishable, we can return to the main camera and position it to be able to view the ground cleanly. To do this, we can either use the transform co-ordinates or drag the camera around using the move tool (W). Position the camera so that we can see the ground cleanly in the preview window.

This is the view

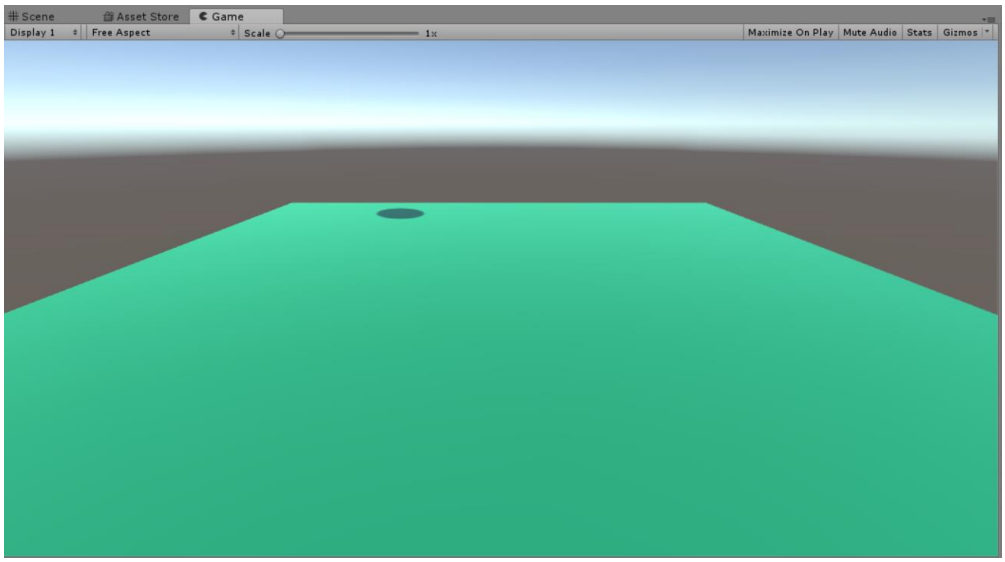


This was achieved via dragging(W) and rotating(E) the camera, the final transform co-ordinates are:



To see what this would look like when we run the game, in the toolbar, click on the play icon

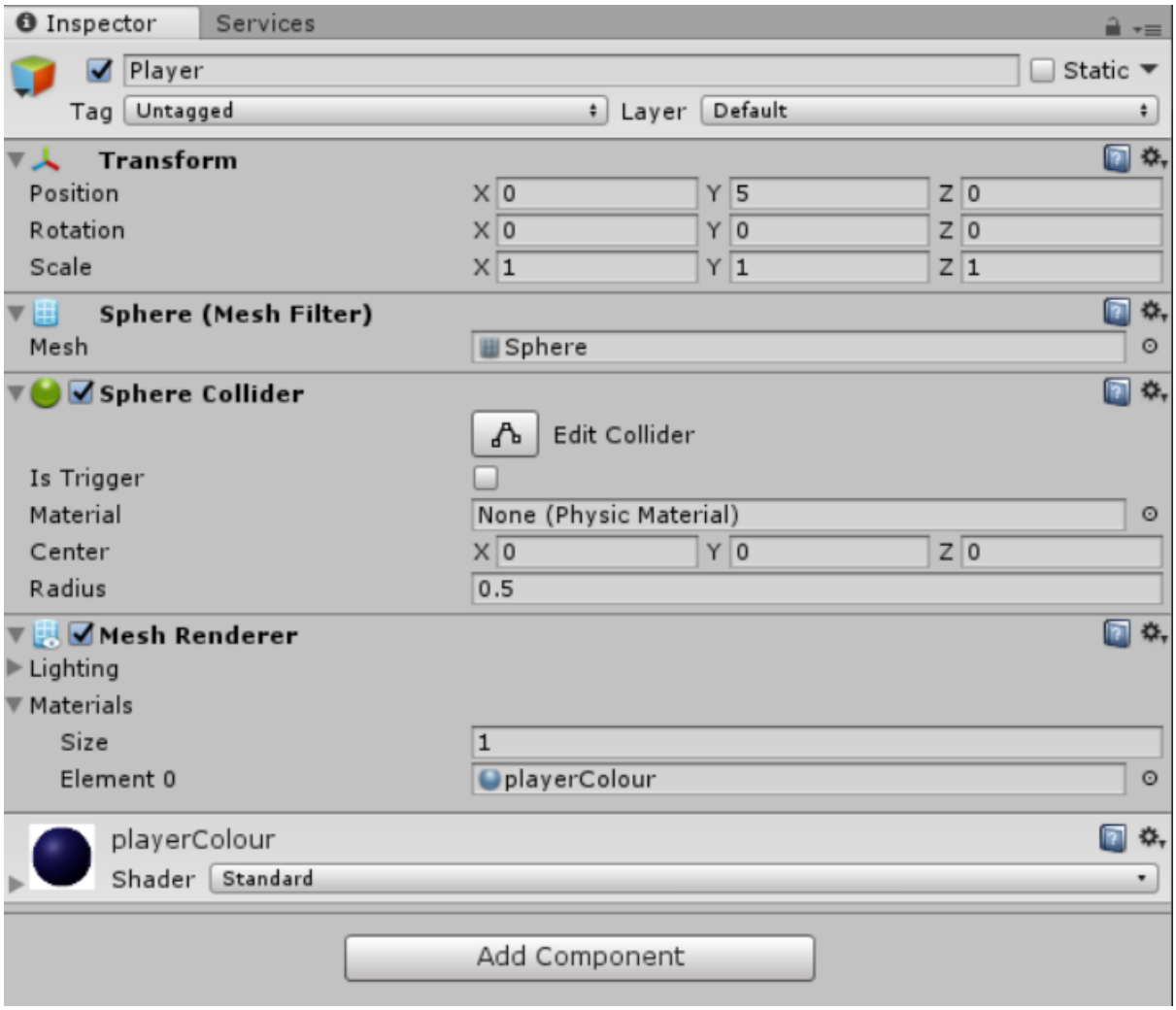
This will swap the scene view for the game view. You should see the following



To return to scene view click on the play button again.

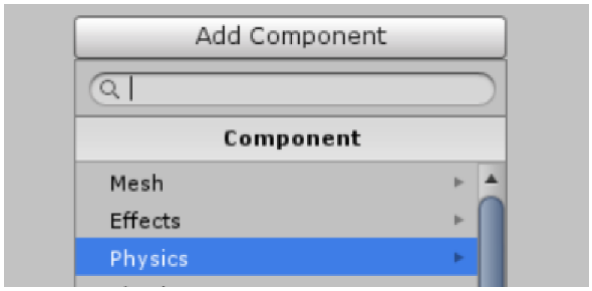
Now that we have all the elements we need, it's time to add some interactivity to the 'game'. To do this, we need to modify the player object with a Rigidbody, this allows it to interact with other objects in the game world, and a script to control it.

Select the player object from the hierarchy and then look at the inspector panel.

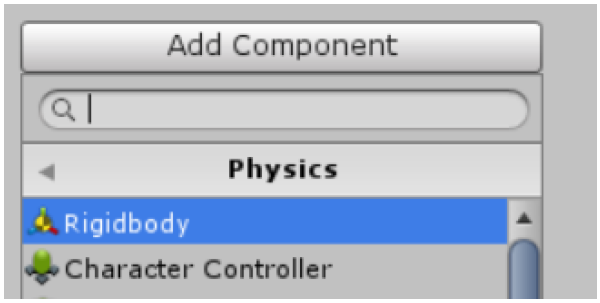


Notice that the player object has a new component called playerColour, this is the material we created earlier and attached via the drag and drop. Now, we need to add a new component called Rigid body, click on the Add Component button

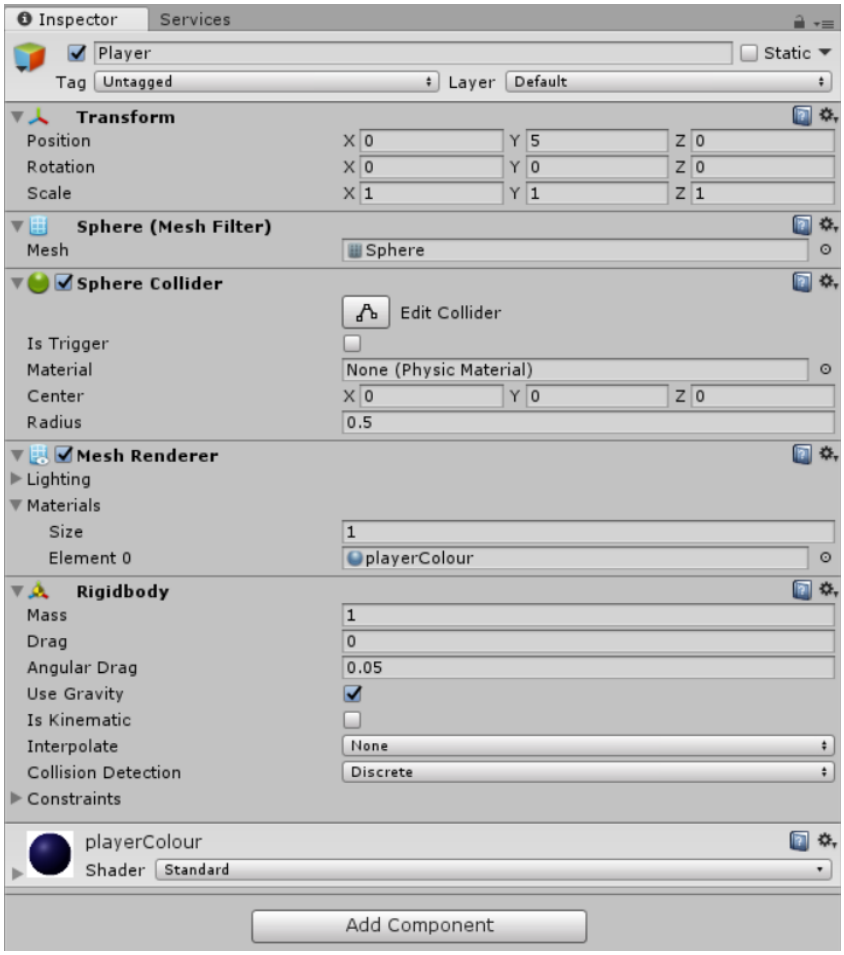
Select Physics



And the RigidBody

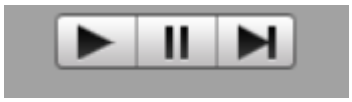


This then attaches the RigidBody body component to the player, it should look like this:

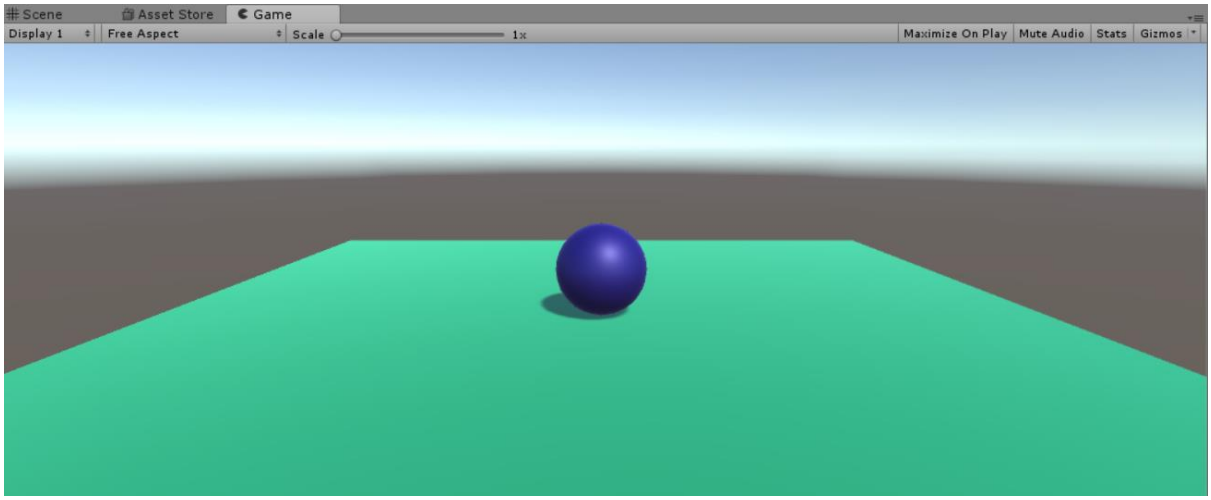


If you hit play on the scene now, the player object will fall from the sky onto the ground object, as you can see, there is a checkbox saying Use Gravity in the RigidBody component.

Push play



You should see the player falling and ending on the ground

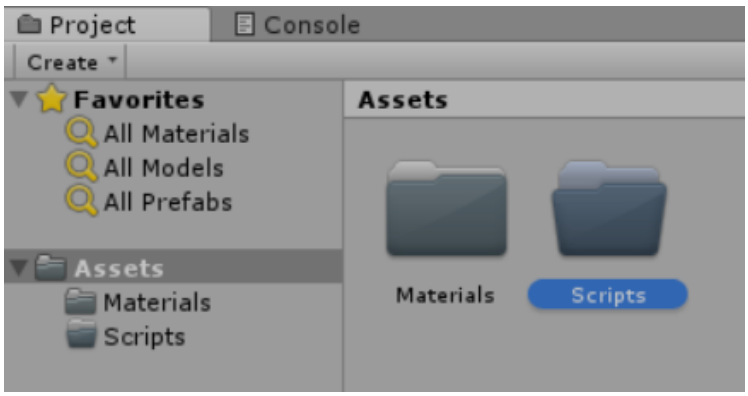


Stop play and now we add a script to control the player.

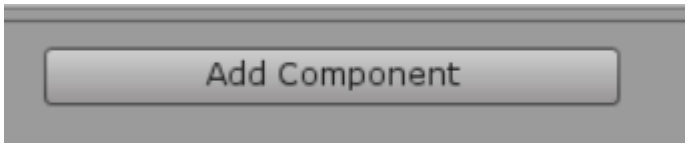
To add the script, we will start by making a folder called scripts in the Assets section, then use the Add Component to create the script. Once this is done, we will need to drag the new script into the Scripts folder.

Add new folder to Assets

Right click->Create->Folder call it Scripts

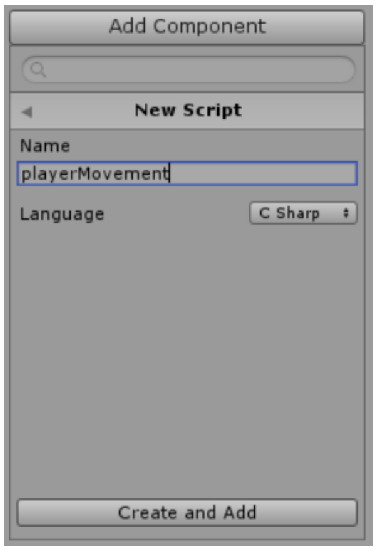


From here click on the Player object in the hierarchy and then go to the Inspector panel and click Add Component.

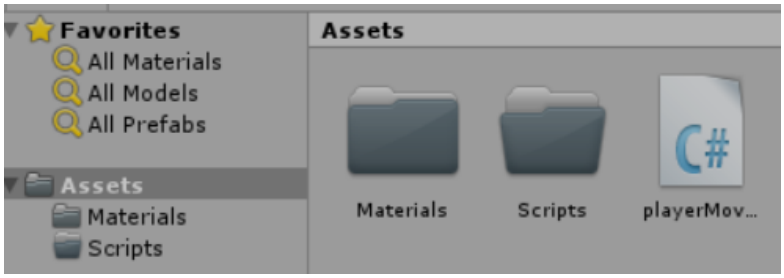


Select New Script

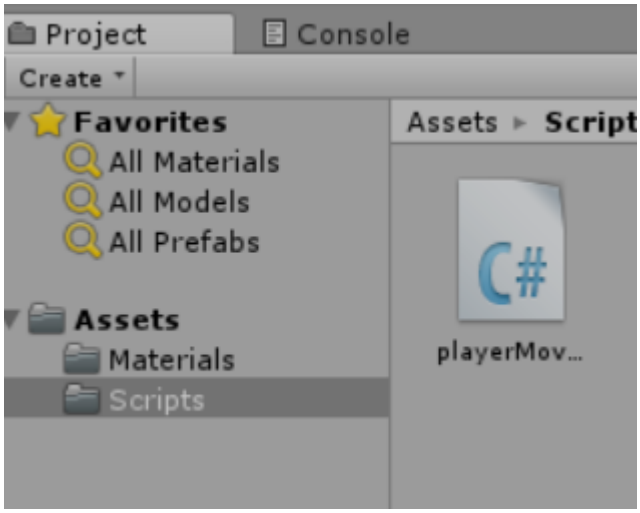
Give the Script a name, in this case it's called playerMovement. We are using the language C Sharp, you'll see it documented as C# sometimes. Then click on Create and Add



Drag the script from the Assets root folder into the scripts folder

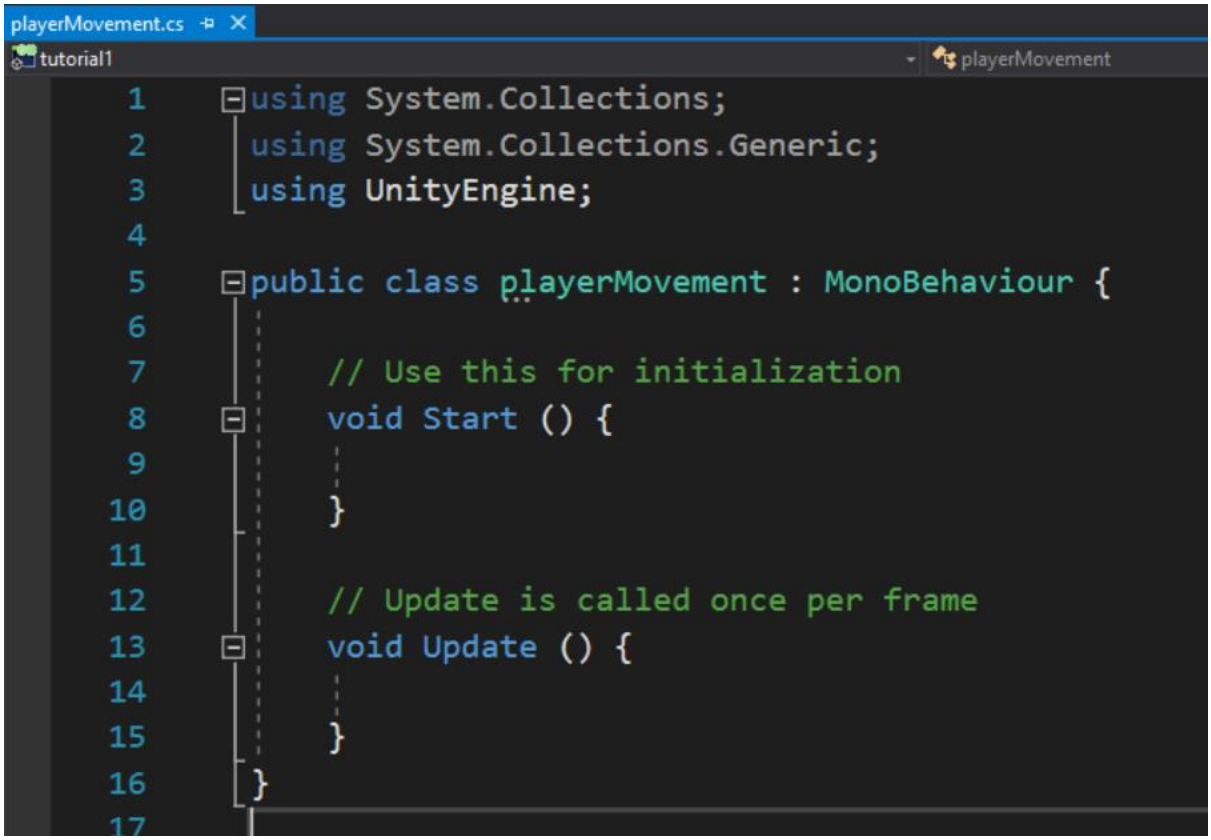


Then go in to the scripts folder



Double Click the newly created script.

It will then open in an editor, this case, the editor used was visual studio 2017. You should see the following:



From here we code what we want to happen, basically what we want to do is apply a force to the rigidbody of the player so it will move. As this is physics based not frame based, we need to rewrite the void Update code to a void FixedUpdate.

```
    // Update is called before physics calculations
    void FixedUpdate () {

    }
}
```

Now, we will put in some code to link to the rigid body. This is done by creating a private variable and then calling it upon start of the program.

```
public class playerMovement : MonoBehaviour {

    private Rigidbody rb;

    // Use this for initialization
    void Start () {
        rb = GetComponent<Rigidbody>();
    }
```

Now we need to think about what type of movement is required, to do this we need to gather the input that occurs. In this case, the cursor keys. In this case, we are working in a 3D environment, this means we have X,Y and Z co-ordinates, we don't require up and down movement, we need to have horizontal and vertical so that means we need to be able to collect X and Z input but can set Y to stay on a singular level.

To do this we use a variable type called float. Float allows for decimal positions, so it is very precise. This code appears inside the fixedUpdate function.

```
// Update is called before physics calculations
void FixedUpdate () {
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");

}
```

Now that we can capture input we still need to apply it, in Unity movement in 3D has 3 options; x,y and z. As such each object we want to have movement requires the ability to hold this information. We've collected 2 parts and we can assign a third, but we still need to make a variable that will hold all this information. This is done in the following line of code.

```
// Update is called before physics calculations
void FixedUpdate () {
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");

    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

}
```

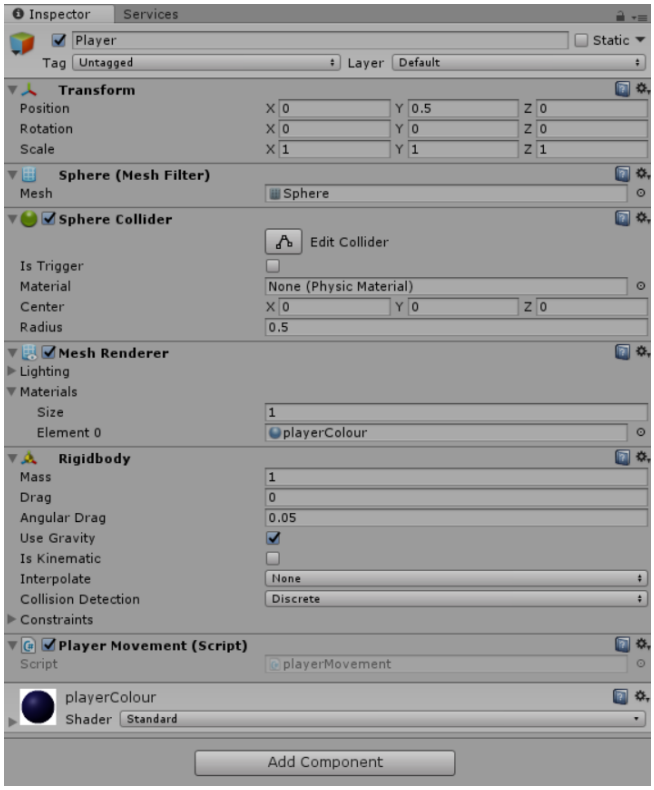
So, what we have done is collect and assign X, Y and Z pieces of information in a Vector 3 variable called movement. But, we haven't done anything with it. This is a common programming mistake where you have gathered all the information and not implemented it. As such, we need to add one more line that will link up the rigidbody of the player object with the input movement collected. This is where the variable rb comes into play.

Add the following code

```
// Update is called before physics calculations
void FixedUpdate () {
    float moveHorizontal = Input.GetAxis("Horizontal");
    float moveVertical = Input.GetAxis("Vertical");

    Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
    rb.AddForce(movement);
}
```

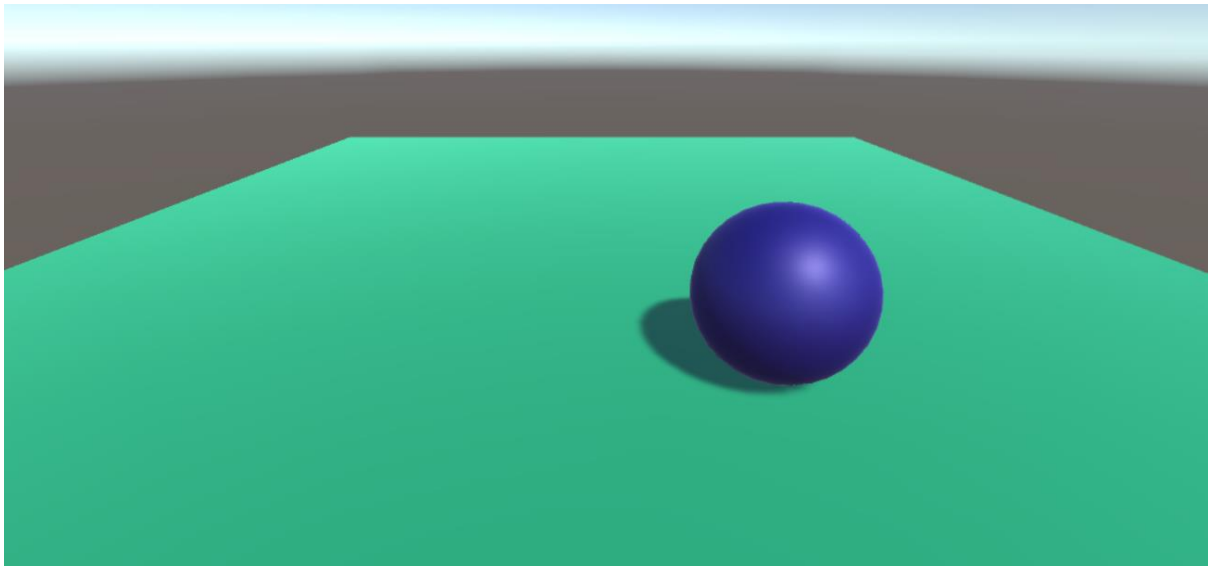
Now that this is done, save the program (Ctrl + S || File->Save) then return to the unity program. If you still have the playerMovement script selected, you will notice that the Inspector has updated with the code you have just written.

Click on Player in the hierarchy and look at the Inspector Panel. This is the base settings for the player, we will be changing this in a few moments, but at the moment this is the default to what we have done. You should see the following:



Run the game by clicking on the play icon

Once the ball has dropped, push the cursor keys and see if anything happens. As you can see there is movement, but we can make it move faster, to do this we need to apply a multiplier to the movement we have set up



. Now, this can be done by setting it in the code or by creating a variable that will allow us to test differing speeds through the inspector. We will set it via the later. But first back to Visual studio to edit the code.

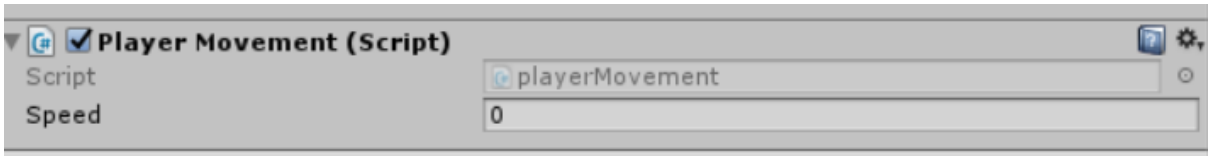
To start with we add a public variable, because this is public, unity will provide this information in the Inspector panel. Then we need to multiply the movement by this new variable.

Your code should look like this:

```
public class playerMovement : MonoBehaviour {

    public float speed;
    private Rigidbody rb;

    // Use this for initialization
    void Start () {
        rb = GetComponent<Rigidbody>();
    }

    // Update is called before physics calculations
    void FixedUpdate () {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        rb.AddForce(movement * speed);
    }
}
```

Save this and go back to unity.

Now, if you select the player object and look at the inspector, you will see that there is an element in the script that is awaiting a value.



If we run the game with speed set at 0, then pushing the cursor keys does nothing. Give it a go, with the game running you will see that we can change this variable. Try some differing numbers to see what a good speed for the player is.

By changing on this element, you can determine values for later.

When the player falls off the plane, stop and start the game to reset.