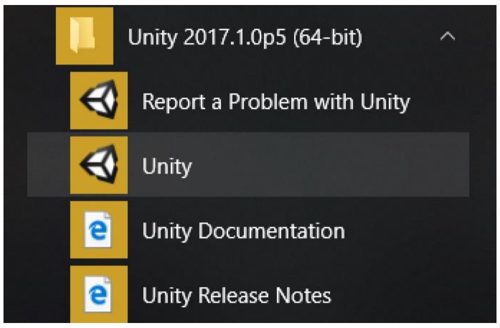
Game Design – Unity Workshop

Activity 2

Goals:

- Creation of small world
- Creation of character
- Scripting of player movement and camera following

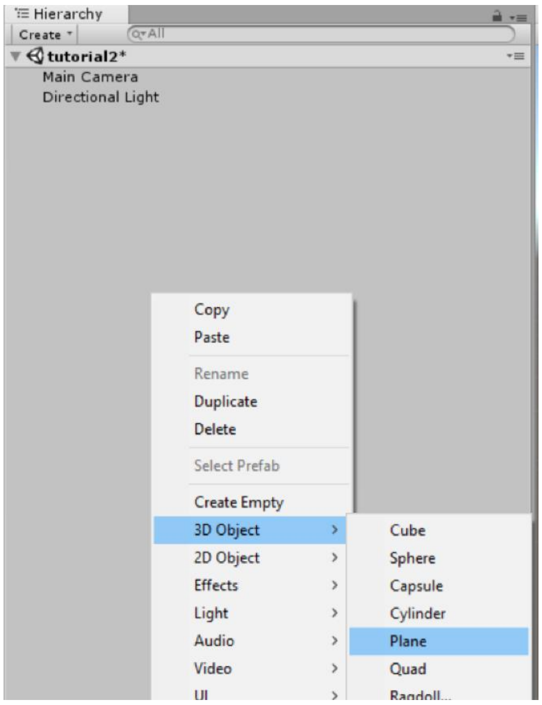
Load up unity



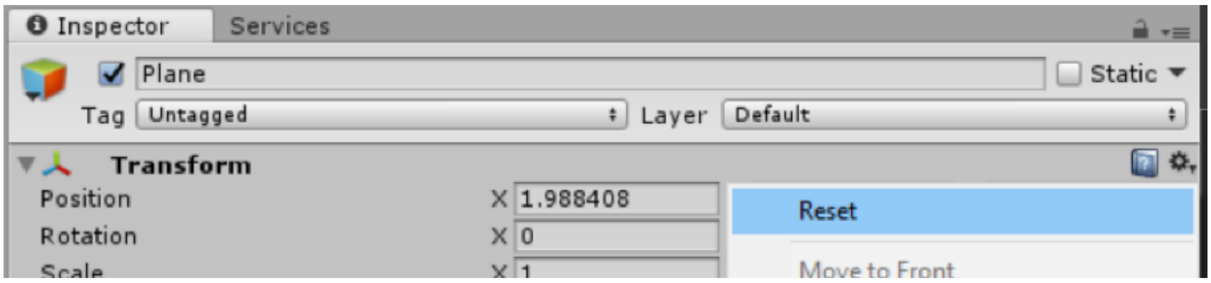
Build Object: Mini World and basic Chase Cam

Aim: multi-level flat spheres to move the players object along.

To start with right-click in the hierarchy panel and create a plane



Next, go to the inspector panel and reset the position from the cog icon.



Plane

Tag: Untagged Layer: Default

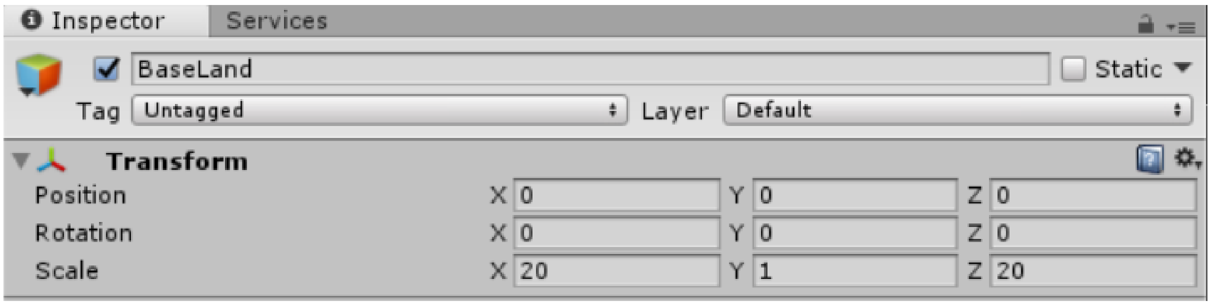
Transform

Position X 1.988408

Rotation X 0

Scale X 1

Once this is done, change the name of the plane to BaseLand and modify the scale of the object to match the following



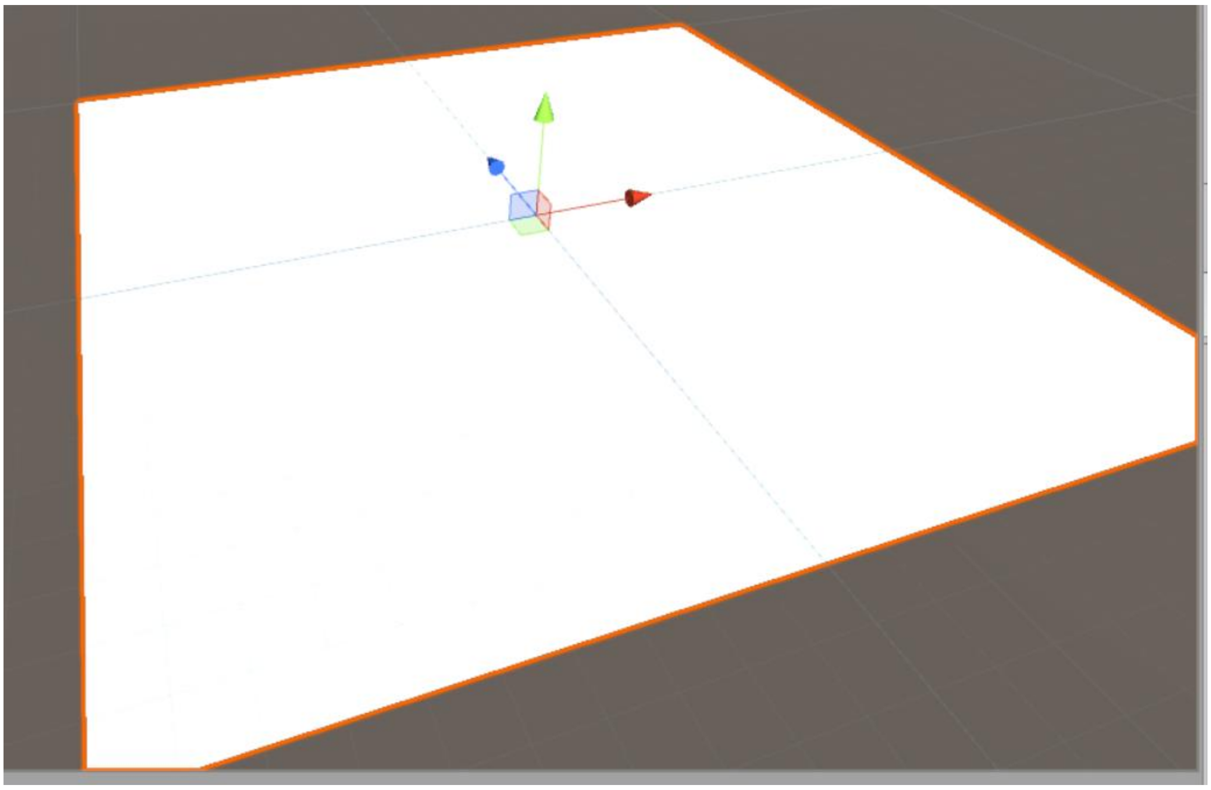
BaseLand

Tag: Untagged Layer: Default

Transform

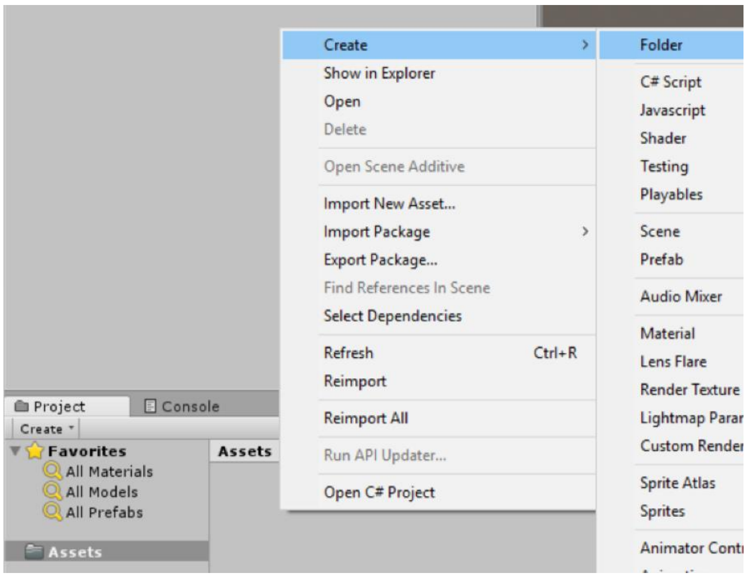
	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	20	1	20

You should see the following in the scene view

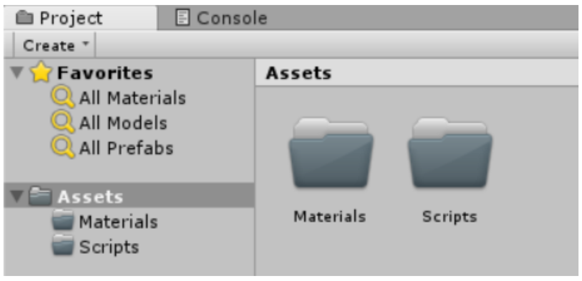


Next, we will create a material and apply it to the BaseLand object.

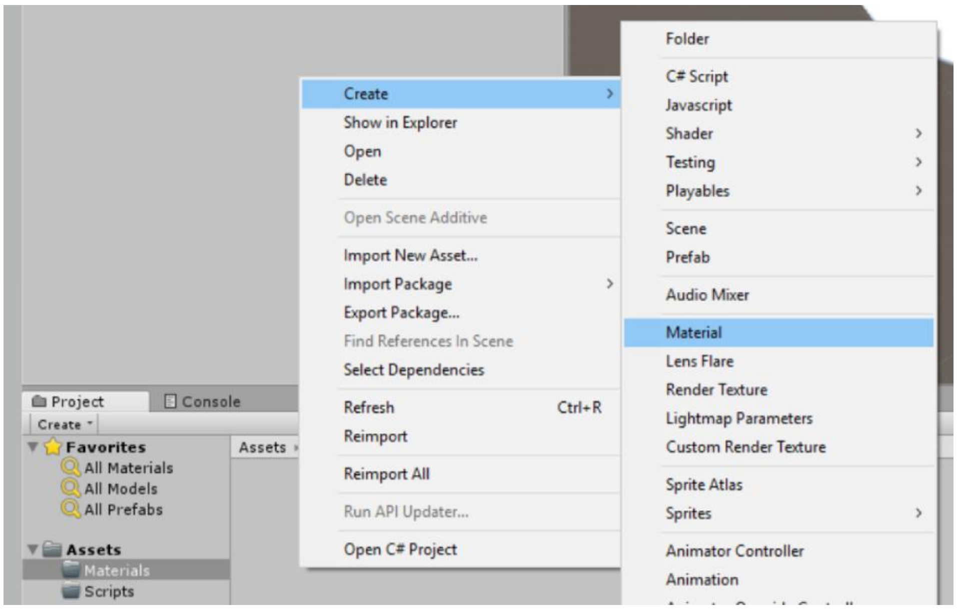
Right click in the Assets panel and create a folder called Materials and one called Scripts



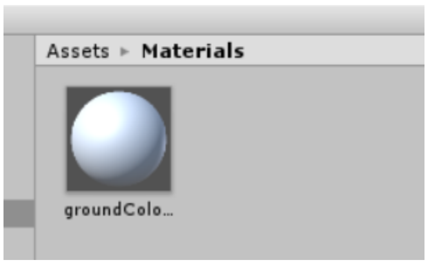
You should end up with the following

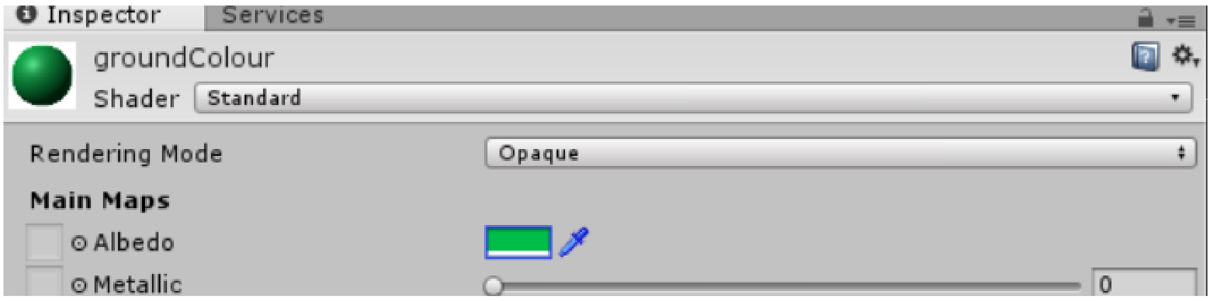


Go into the Materials folder and Create a new Material. Right-click Create->Material

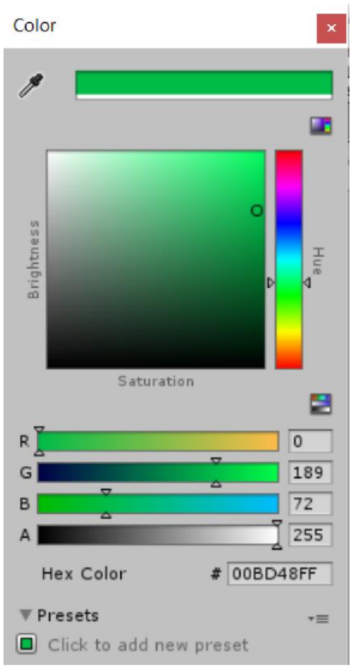


Name it groundColour and then in the inspector panel, click on the colour selector and assign a colour to the material.





groundColour
Shader Standard
Rendering Mode Opaque
Main Maps
Albedo
Metallic 0



R 0
G 189
B 72
A 255
Hex Color # 00BD48FF

In this case, the rgba format is

R: 0

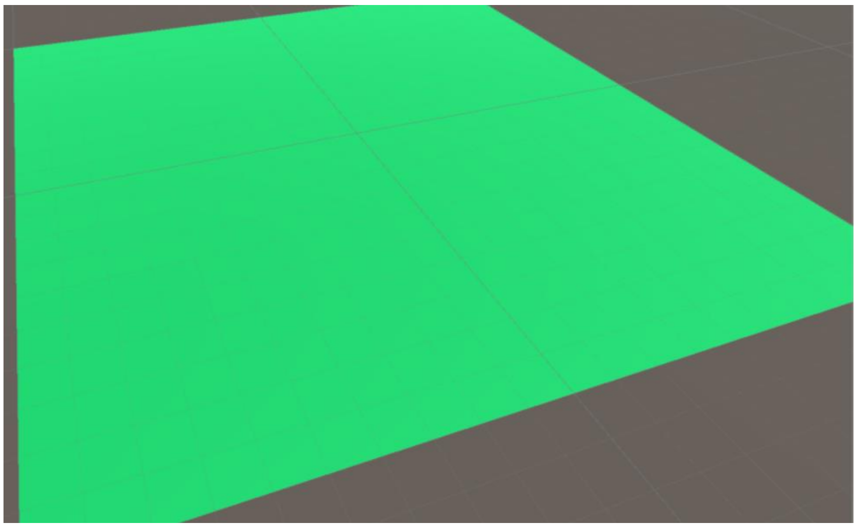
G: 189

B: 72

A: 255

You can use any colour you like for this part of the tutorial.

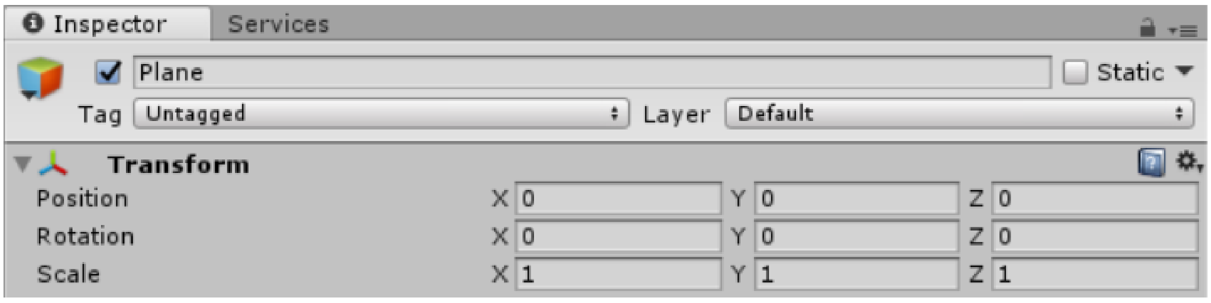
Next drag the colour on BaseLand object and you should see the following in the scene view.



Next we are going to add a few more levels. Right click on BaseLand in the hierarchy pane and create a new plane. Two things will happen, one a new plane with the same attributes as BaseLand is created in the exact same position and two; the new object is now a child of BaseLand.

This is the introduction to grouping, with elements called Parents and Children. The children will inherent the Parents attributes, but not assigned materials.

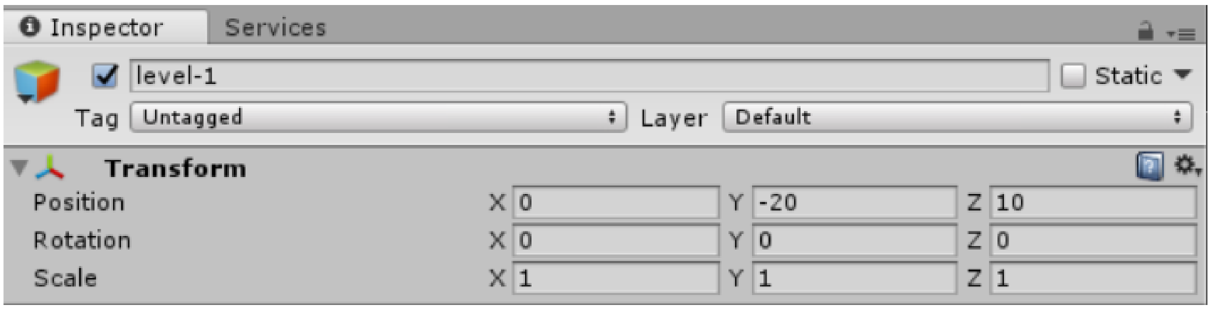
Now we rename and move the new plane, in the Inspector Panel change the name of plane to level-1 and move its position to 0/-20/10



Plane

Tag: Untagged Layer: Default

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

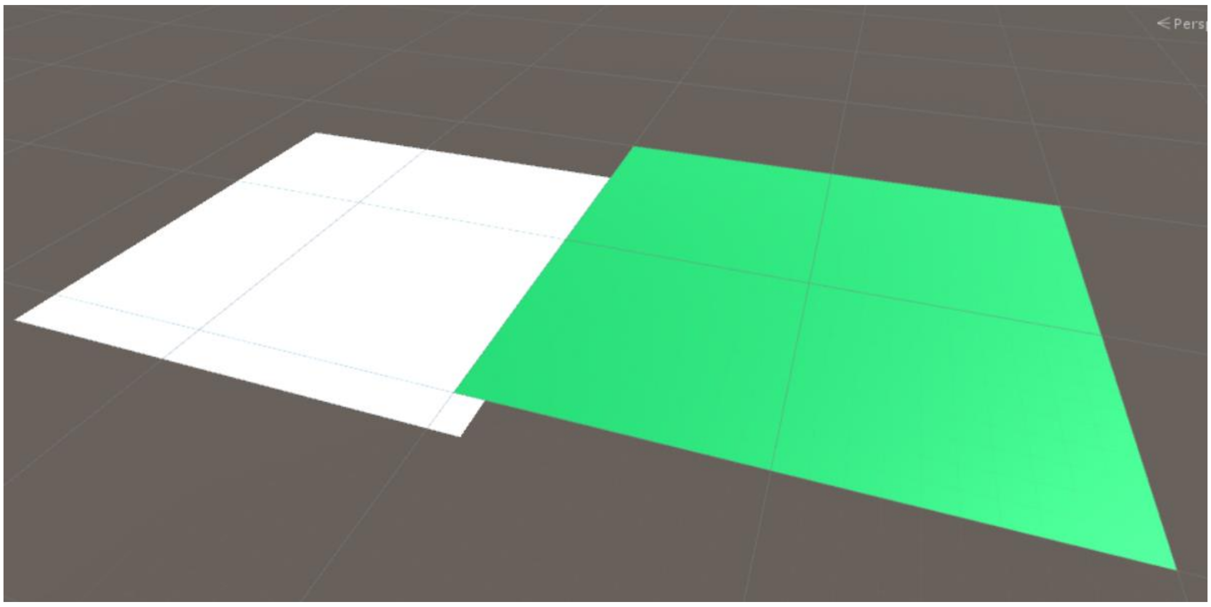


level-1

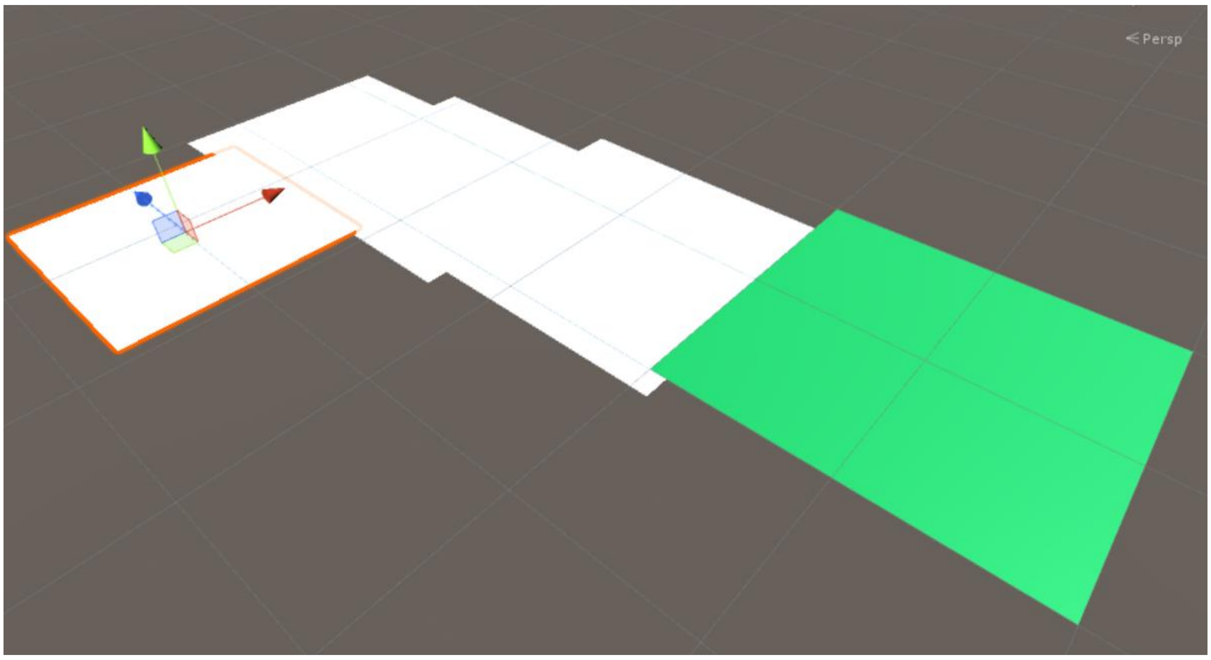
Tag: Untagged Layer: Default

	X	Y	Z
Position	0	-20	10
Rotation	0	0	0
Scale	1	1	1

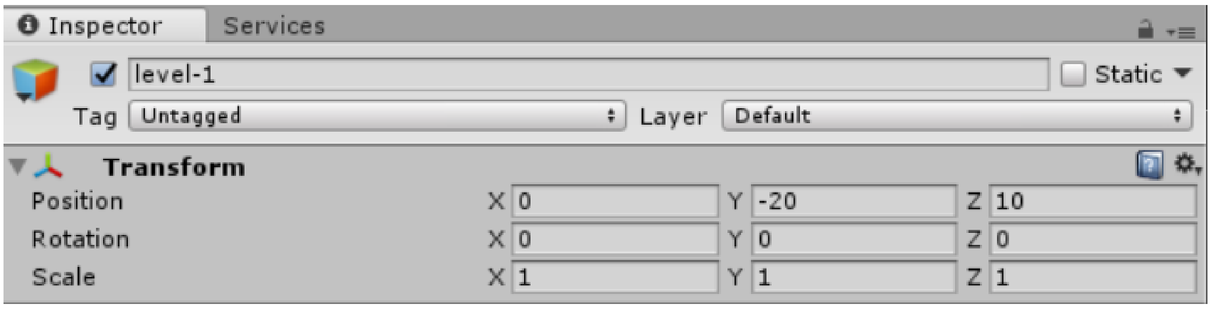
This should give the following in scene view



Create three more, label them level-2, level-3 and level-4, then move then into the below positions, it should look like this:



Here are the Inspector Panel transforms

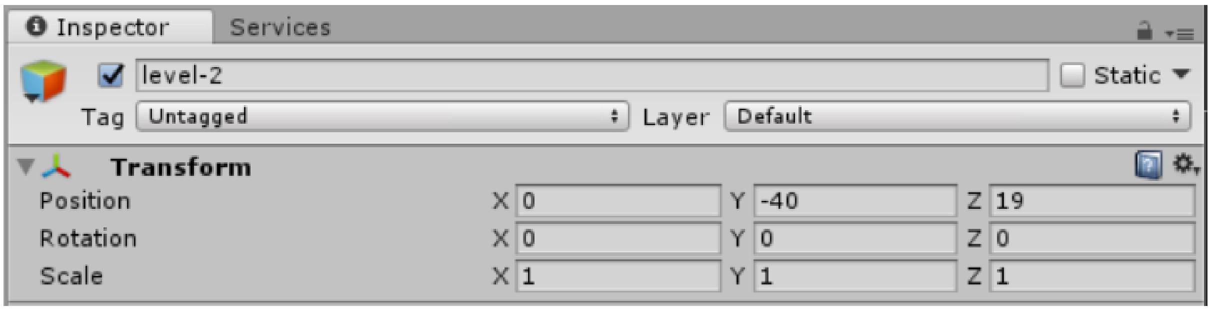


level-1 Static

Tag Untagged Layer Default

Transform

Position	X 0	Y -20	Z 10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

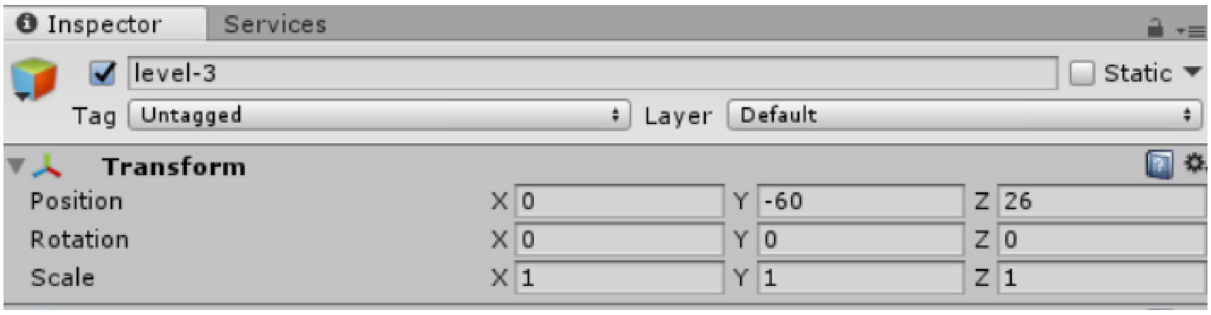

Inspector | Services

level-2 | Static

Tag Untagged | Layer Default

Transform

	X	Y	Z
Position	0	-40	19
Rotation	0	0	0
Scale	1	1	1

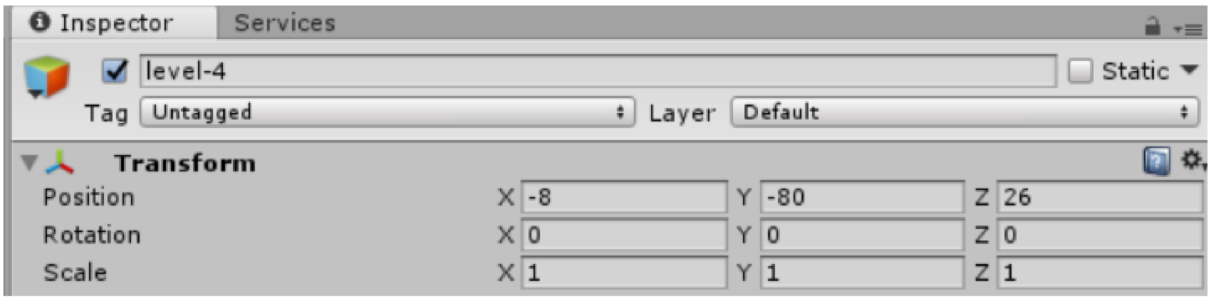

Inspector | Services

level-3 | Static

Tag Untagged | Layer Default

Transform

	X	Y	Z
Position	0	-60	26
Rotation	0	0	0
Scale	1	1	1


Inspector | Services

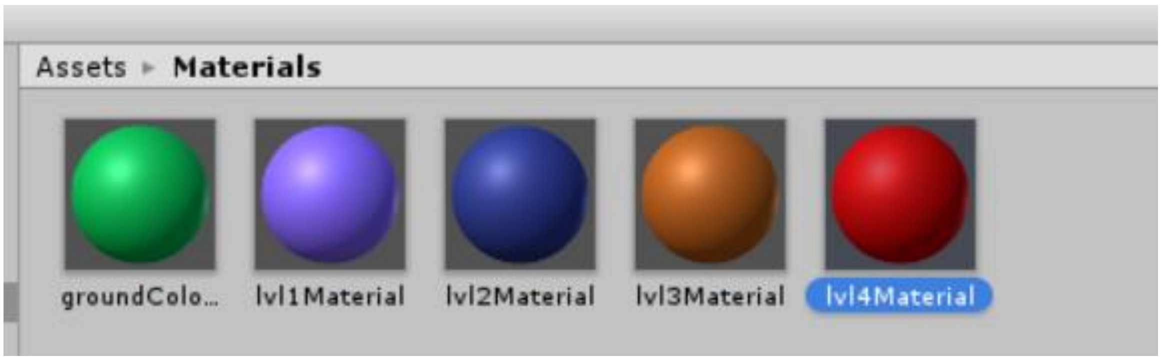
level-4 | Static

Tag Untagged | Layer Default

Transform

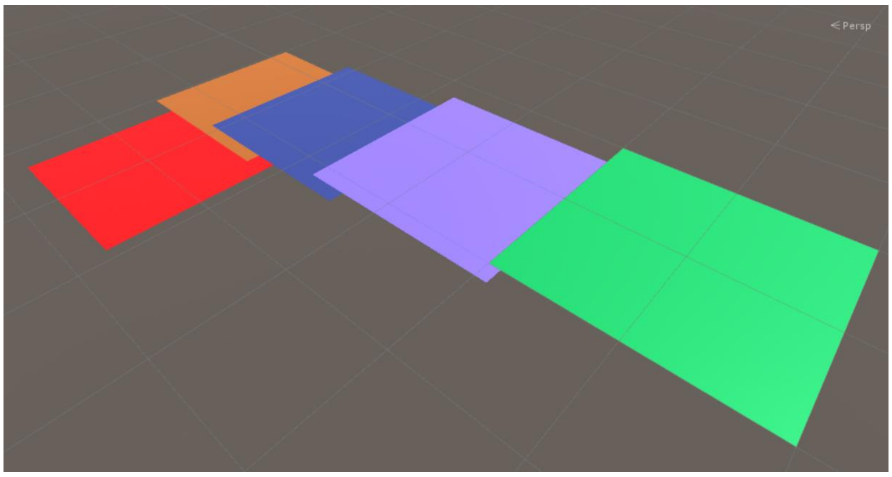
	X	Y	Z
Position	-8	-80	26
Rotation	0	0	0
Scale	1	1	1

Next create 4 new Materials and apply to each level



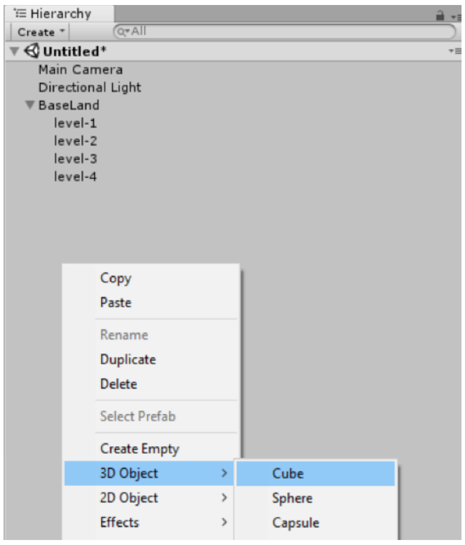
	Lvl1Material	Lvl2Material	Lvl3Material	Lvl4Material
Red	141	42	197	249
Green	101	54	89	0
Blue	242	133	19	0
Alpha	255	255	255	255

After applying the materials to the levels, you should see the following in the scene view

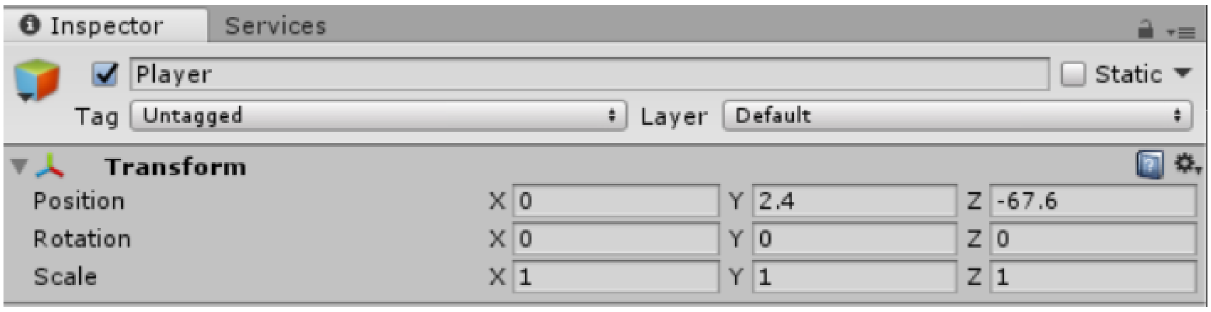


Next we will make the player model, in this case, let's make a simple cube to move around. In the hierarchy panel, right click and create 3D Object->Cube. We want the player to not be part of the BaseLand group, so make sure BaseLand is not selected when you create the Cube, if it is, you can drag the Cube out of the group.

Create cube



When this is done, you will see the cube appear below the centre of the scene, to fix this reset the position of the cube, change its name to Player for ease of reference and position the cube just above and near the back of BaseLand .



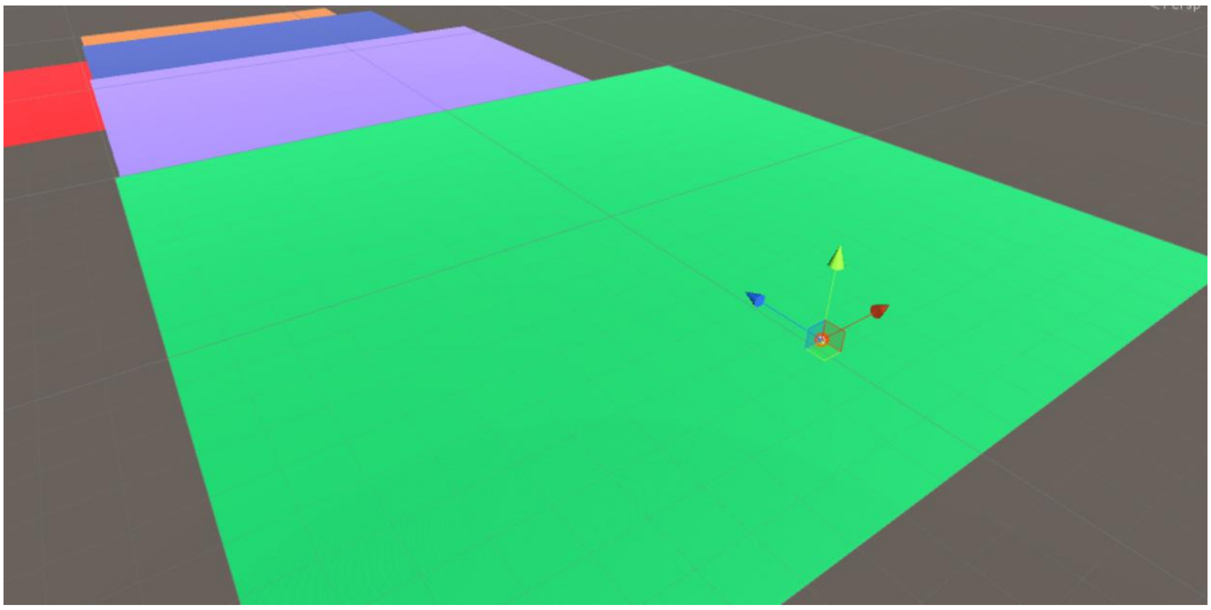
Player

Tag Untagged Layer Default

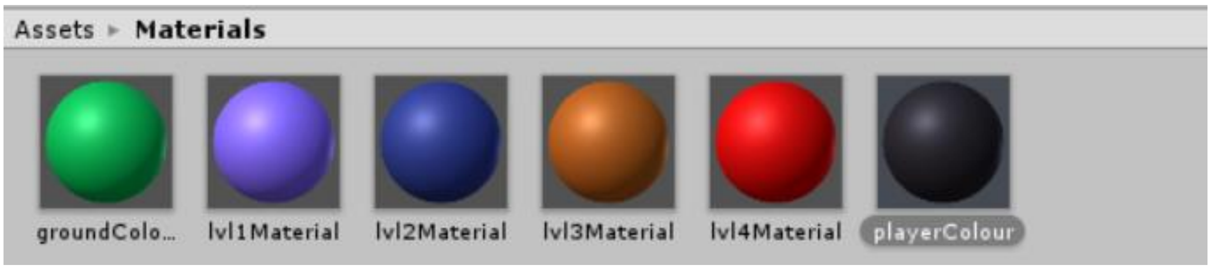
Transform

Position	X	0	Y	2.4	Z	-67.6
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

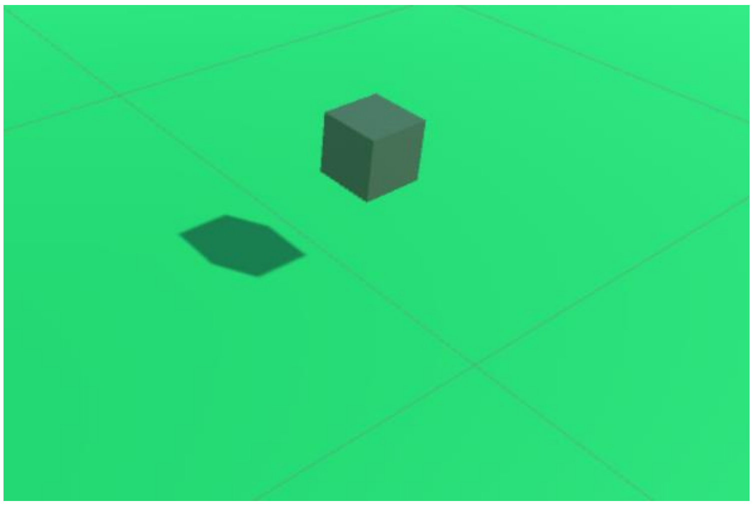
Scene view should look like this, remember to rotate the view using alt and mouse down to scroll/rotate the viewing camera.



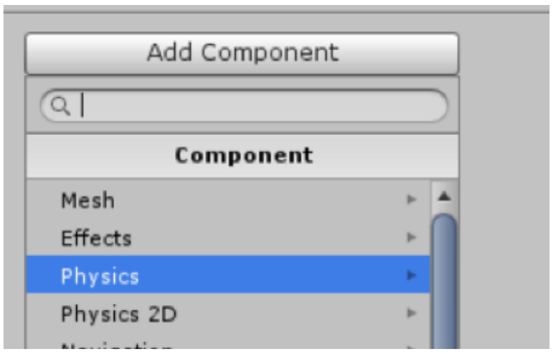
Create a new material for the player and apply it. Just to make things a little different, use the following RGBA settings; R:34, G:34, B:34, A:170 and in the inspector, change the rendering mode to transparent. Then apply it to the player object.

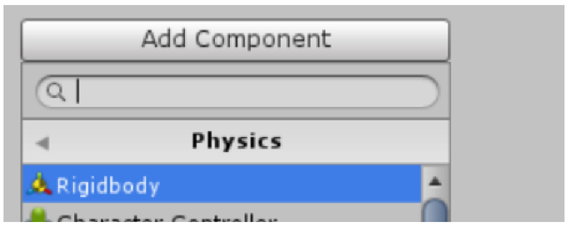


Creates the follow scene view:

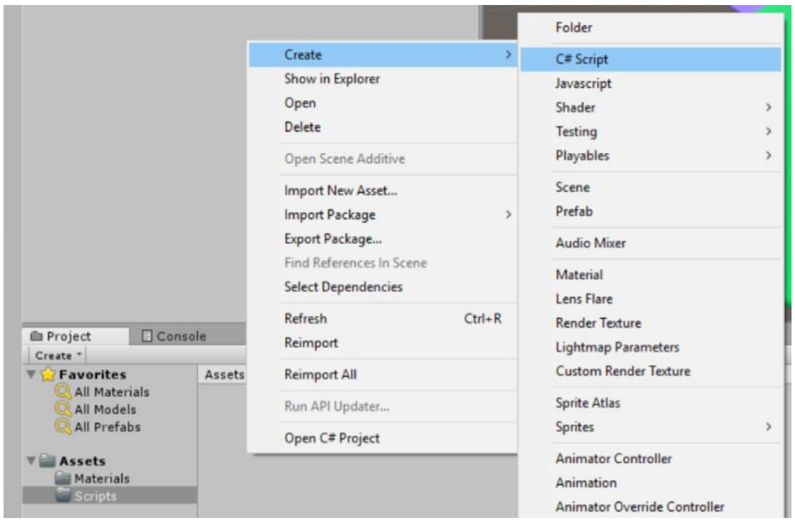


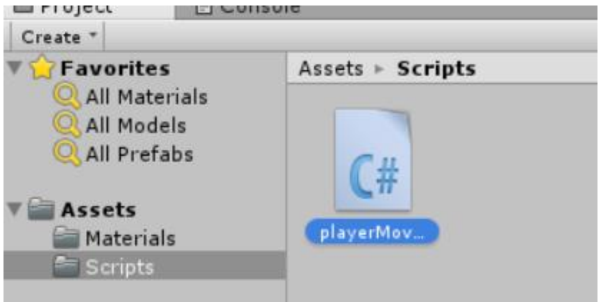
Next, we add movement to the Player object. To start with we add a RigidBody to the Player. This can be done by clicking on Add Component->Physics->RigidBody



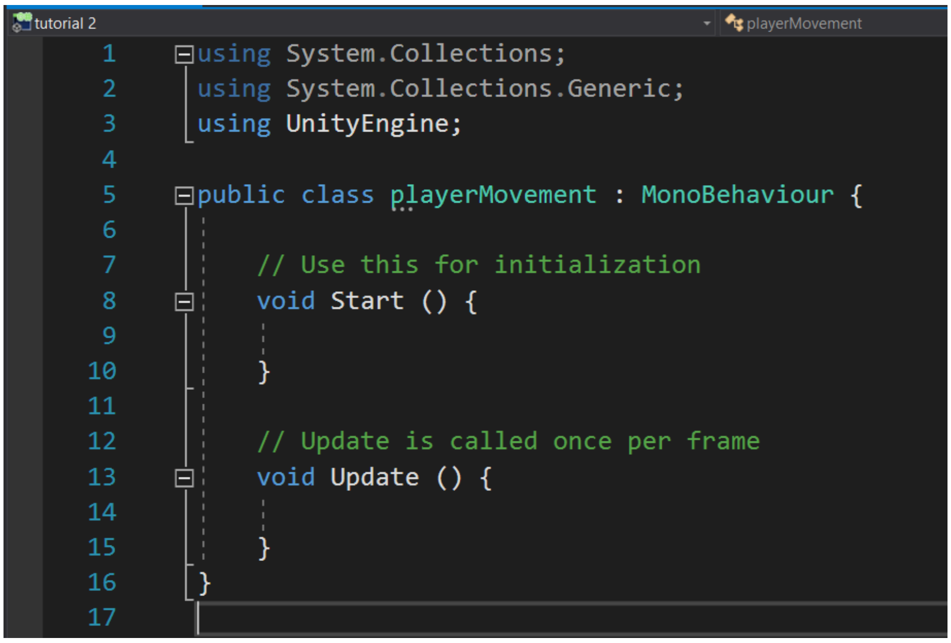


From here we add a new script to the player object, in the Assets folder, click into Scripts, then right click and create new C# Script, call this script playerMovement.





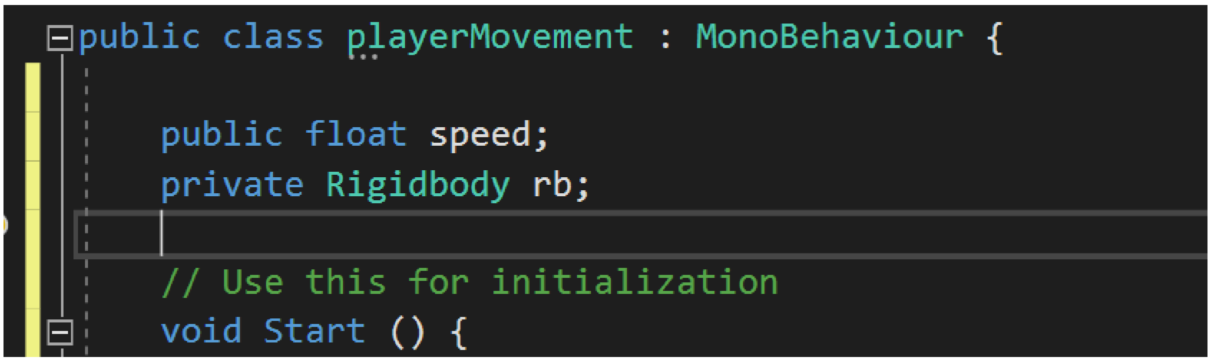
Now, double click the script, it should open in visual studio.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class playerMovement : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

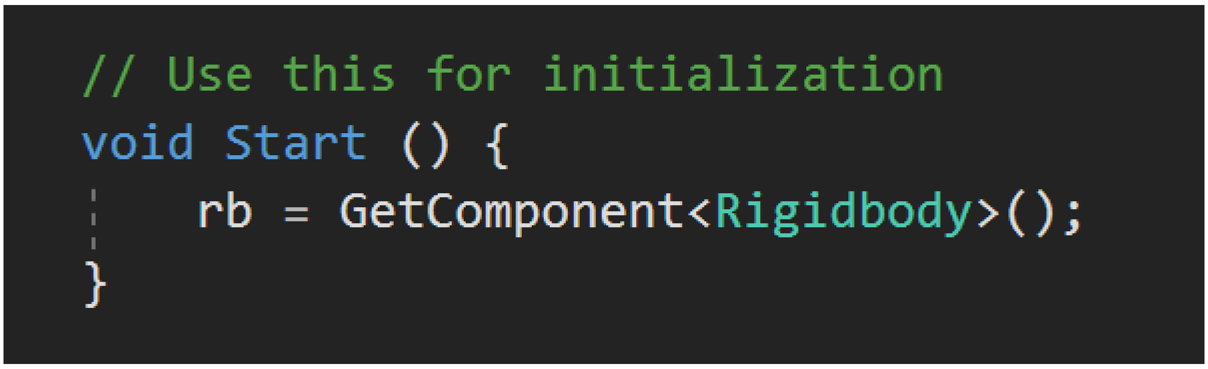
As we did in the last tutorial, we need to be able to programmatically access the player object, to do this we will create a couple of variables above the void start, this will be the speed (used for speed of movement) and rigidbody (Allow the object to be interacted with).

The code looks like this



```
public class playerMovement : MonoBehaviour {

    public float speed;
    private Rigidbody rb;

    // Use this for initialization
    void Start () {
```

Next, we initialise the access to the rigidbody this is done in the start function. We get all of the components of the rigidbody and make then accessible via the rb variable

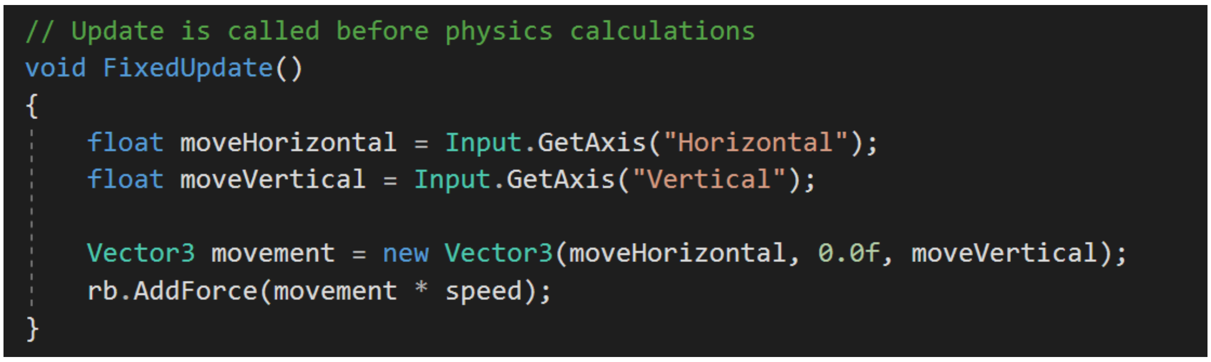


```
// Use this for initialization
void Start () {
    rb = GetComponent<Rigidbody>();
}
```

Next, we add in the ability to capture the movement via input, i.e. a key is pressed (WASD or Cursor keys). This is done by creating a couple of float variables called moveHorizontal and moveVertical.

From here, we implement a Vector3 variable (x, y, z). This variable holds the horizontal, vertical and Y position of the player object.

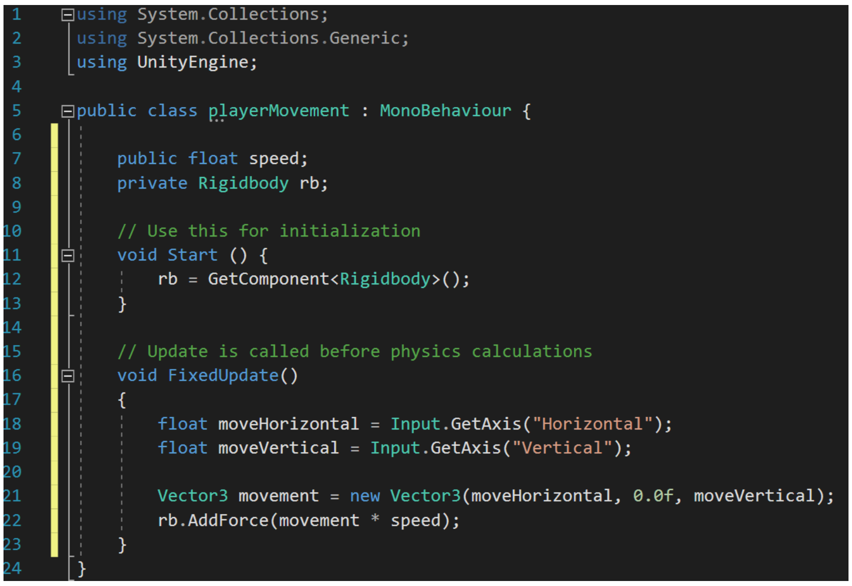
Finally, we apply force to the rigidbody to make it move when the keys are pushed. The force is calculated by movement * speed. The code below is what should be created.



```
// 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);
}
```

Notice that the Update function is replaced with the FixedUpdate function.

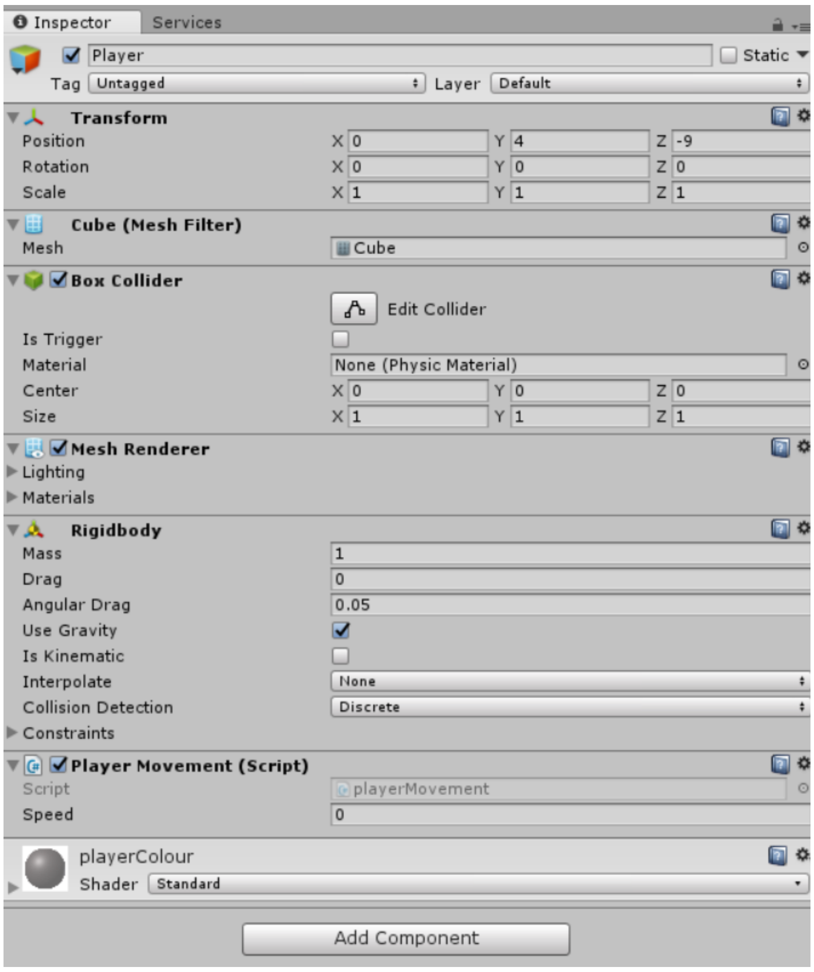
The overall code for player movement is:



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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 (Ctrl+S or File -> Save), then return to unity.

From here, drag the newly created script from the Assets folder on to the player object, so the script is linked to the object.


Inspector Services

Player Static

Tag Untagged Layer Default

Transform

Position	X 0	Y 4	Z -9
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Cube (Mesh Filter)

Mesh: Cube

Box Collider

Edit Collider

Is Trigger	☐		
Material	None (Physic Material)		
Center	X 0	Y 0	Z 0
Size	X 1	Y 1	Z 1

Mesh Renderer

Lighting

Materials

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete

Constraints

Player Movement (Script)

Script	playerMovement
Speed	0

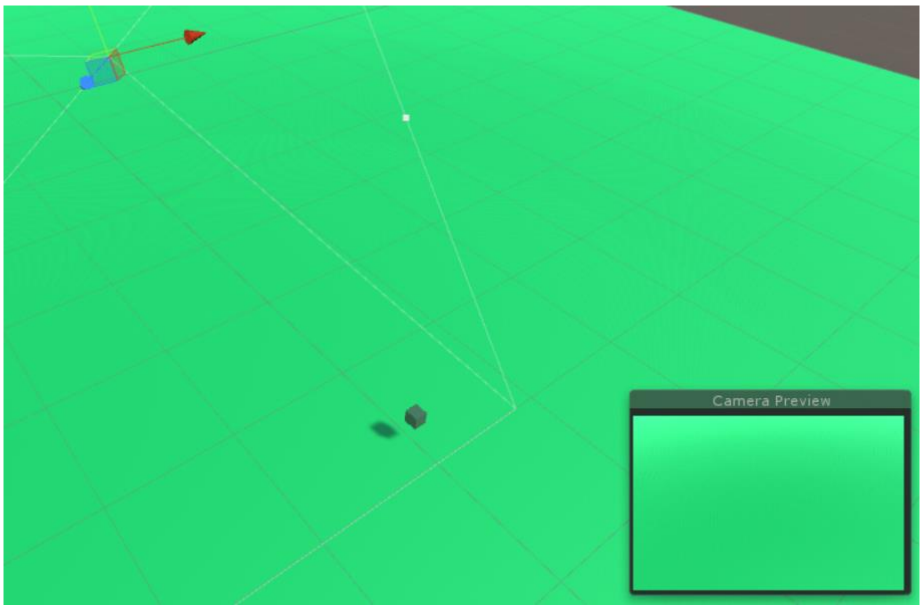
playerColour

Shader Standard

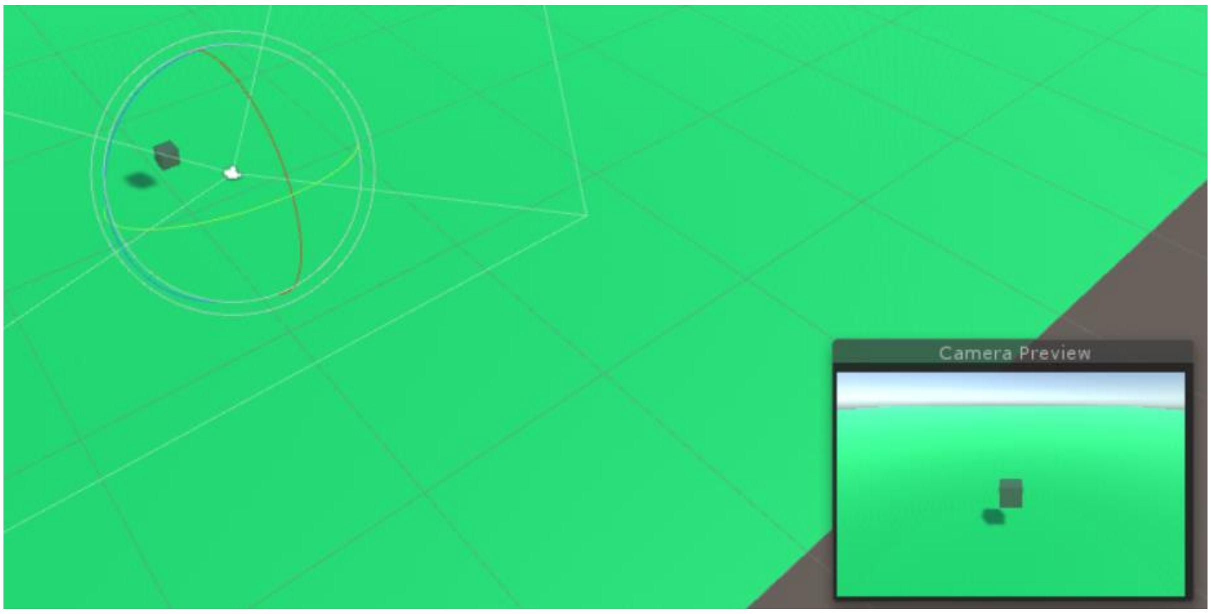
Add Component

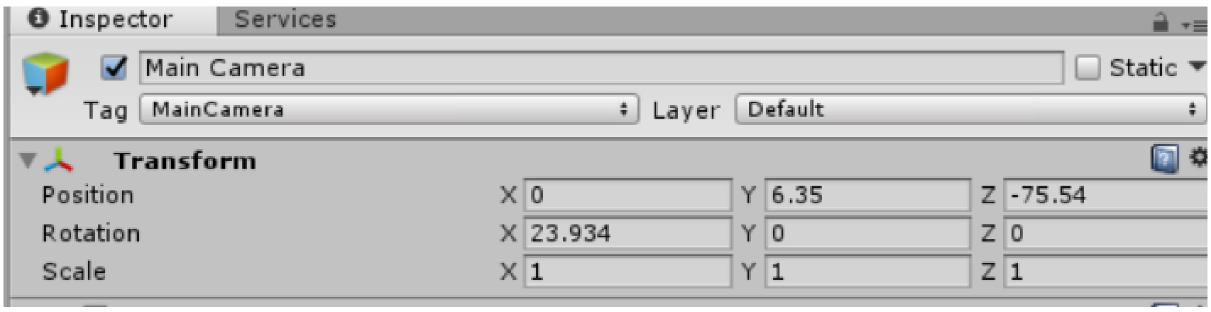
Before we test the code, let's check what we will see by clicking on the Main Camera in the hierarchy view. The camera indicates that we won't see much, as such let's re-position the camera to see the player drop onto the BaseLand object.

Original View



Changed View




Inspector | Services

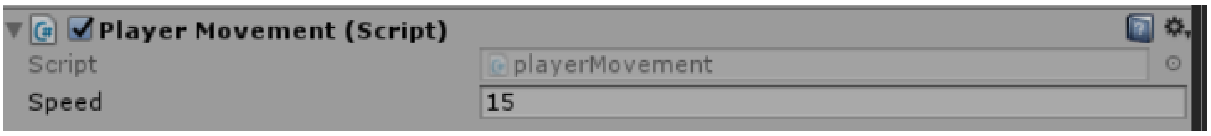
Main Camera ☐ Static

Tag MainCamera Layer Default

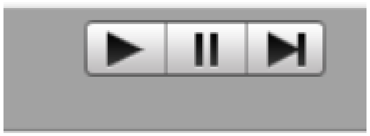
Transform

Position	X 0	Y 6.35	Z -75.54
Rotation	X 23.934	Y 0	Z 0
Scale	X 1	Y 1	Z 1

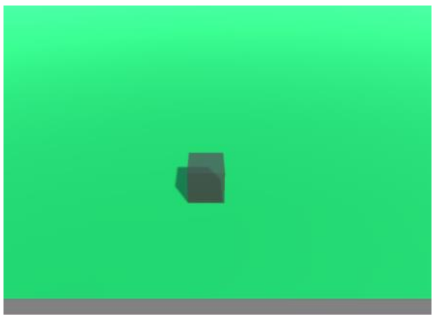
Assign the player a base speed (15) and then test, push the play button


Player Movement (Script)

Script	playerMovement
Speed	15



This is what happens in the scene

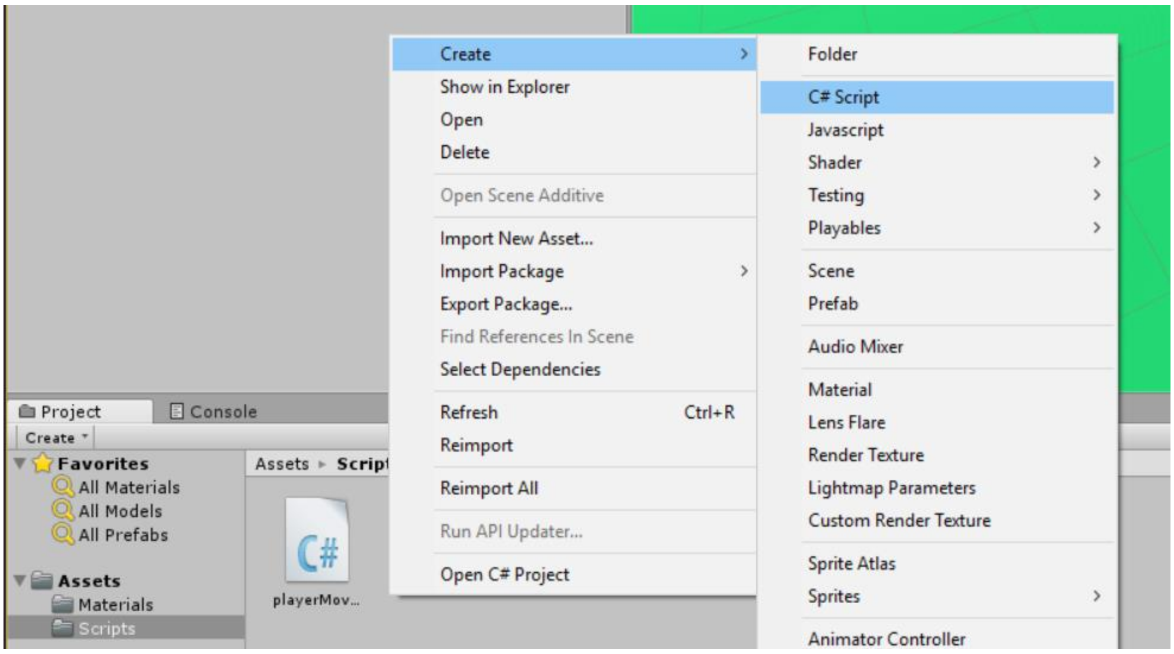


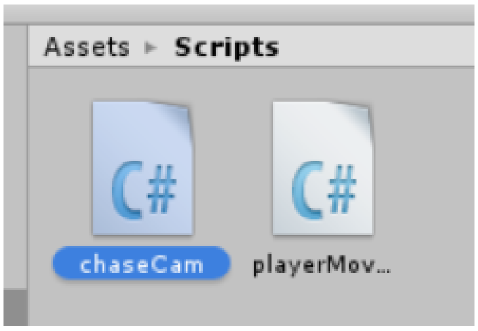
We see the cube fall and by using the keys (WASD or Cursor) we can see the cube bounce and move around the sphere, the main issue we have is that the camera isn't following the player, so we have no way of getting the cube to level 4.

Script 1

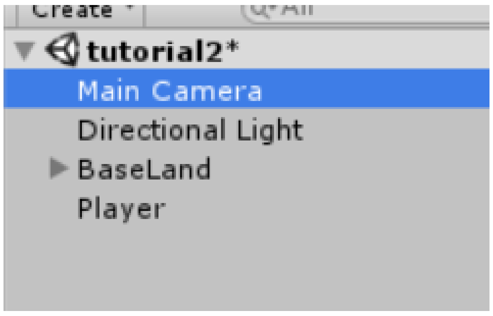
To make this happen, we need to create a new script and apply it to the camera.

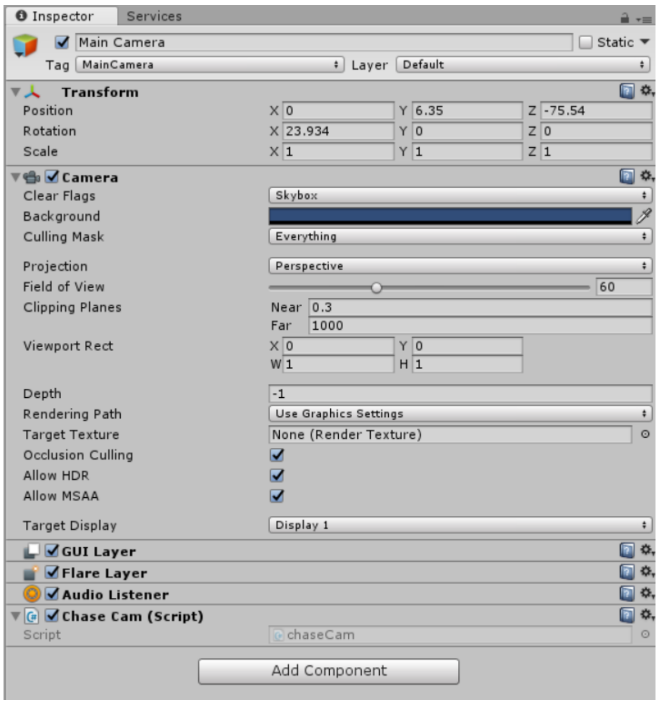
Right click in the Scripts folder and create a new C# script called chaseCam.





Select Main Camera from the hierarchy panel and then drag the chaseCam script into the Inspector Panel.




Inspector | Services

Main Camera — Static
Tag: MainCamera — Layer: Default

Transform

Position	X 0	Y 6.35	Z -75.54
Rotation	X 23.934	Y 0	Z 0
Scale	X 1	Y 1	Z 1

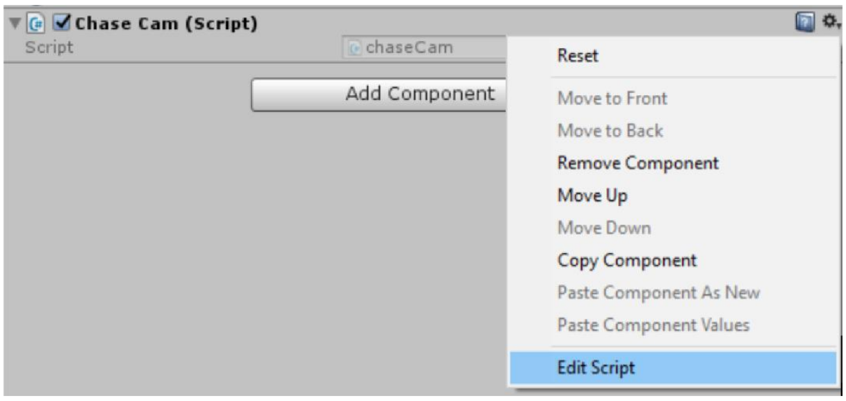
Camera

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
Field of View	60
Clipping Planes	Near 0.3, Far 1000
Viewport Rect	X 0, Y 0, W 1, H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
Allow HDR	✓
Allow MSAA	✓
Target Display	Display 1

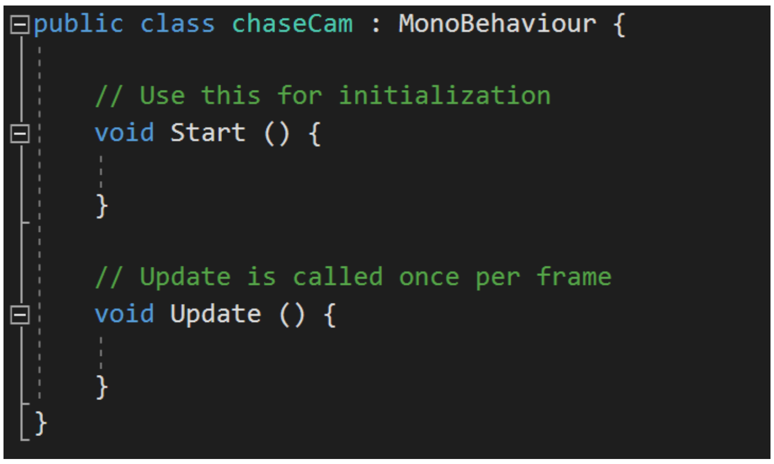
GUI Layer
Flare Layer
Audio Listener
Chase Cam (Script)
Script: chaseCam

Add Component

From the inspector panel, we will use the cog to edit the script, this will open the script in visual studio.

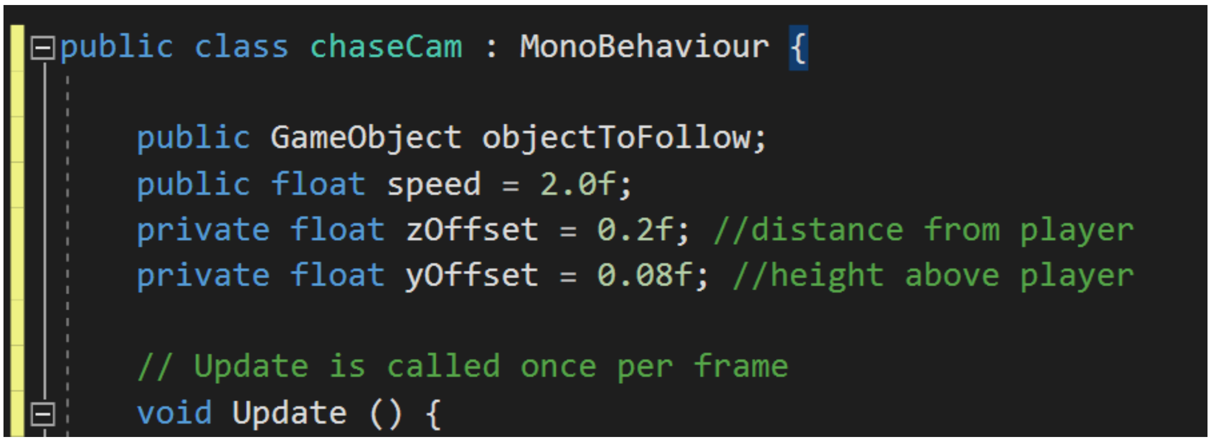


The initial view is an empty script.


```
public class chaseCam : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

To modify this script we need to access some information, we need to tell the camera which game object it needs to follow, what speed it needs to follow at and then we need to apply a smooth transition for animation by using the Maths function Lerp. A maths lerp basically alters between the start and end value of two numbers. When applying this to a position transform it ends up being a smooth transition.

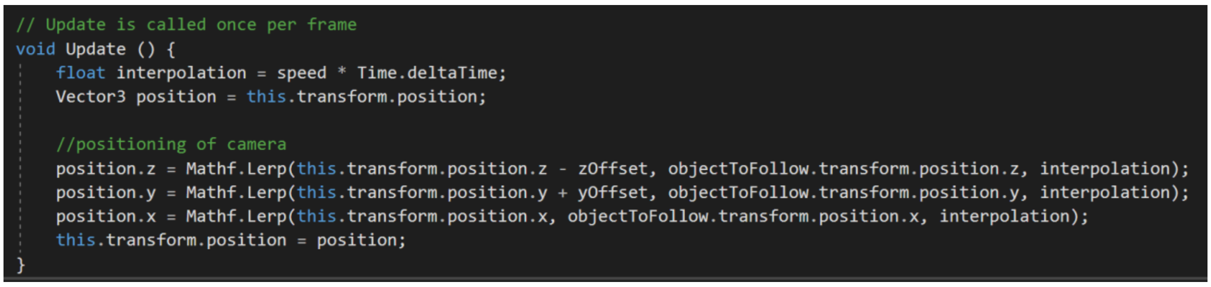
To start with, we will need to add some variables outside the update function.



```
public class chaseCam : MonoBehaviour {

    public GameObject objectToFollow;
    public float speed = 2.0f;
    private float zOffset = 0.2f; //distance from player
    private float yOffset = 0.08f; //height above player

    // Update is called once per frame
    void Update () {
```

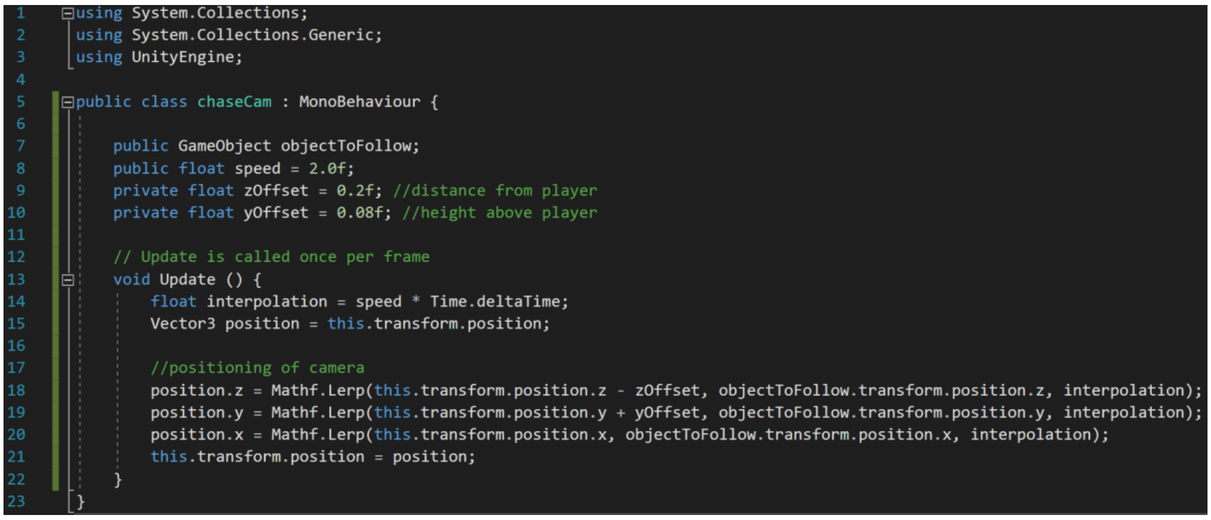
The public GameObject will be selectable inside unity.

From here, we need to modify the update function, like this



```
// Update is called once per frame
void Update () {
    float interpolation = speed * Time.deltaTime;
    Vector3 position = this.transform.position;

    //positioning of camera
    position.z = Mathf.Lerp(this.transform.position.z - zOffset, objectToFollow.transform.position.z, interpolation);
    position.y = Mathf.Lerp(this.transform.position.y + yOffset, objectToFollow.transform.position.y, interpolation);
    position.x = Mathf.Lerp(this.transform.position.x, objectToFollow.transform.position.x, interpolation);
    this.transform.position = position;
}
```

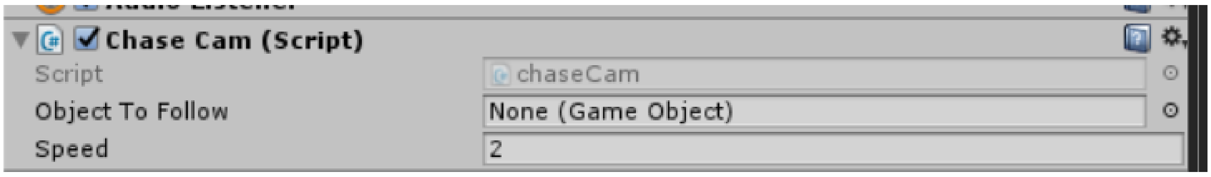
This set of code does all of the calculations required to keep the camera moving behind the player. The total code looks like this:



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class chaseCam : MonoBehaviour {

    public GameObject objectToFollow;
    public float speed = 2.0f;
    private float zOffset = 0.2f; //distance from player
    private float yOffset = 0.08f; //height above player

    // Update is called once per frame
    void Update () {
        float interpolation = speed * Time.deltaTime;
        Vector3 position = this.transform.position;

        //positioning of camera
        position.z = Mathf.Lerp(this.transform.position.z - zOffset, objectToFollow.transform.position.z, interpolation);
        position.y = Mathf.Lerp(this.transform.position.y + yOffset, objectToFollow.transform.position.y, interpolation);
        position.x = Mathf.Lerp(this.transform.position.x, objectToFollow.transform.position.x, interpolation);
        this.transform.position = position;
    }
}
```

Save this code and then go back to unity.

With the main camera selected, examine the Inspector panel.

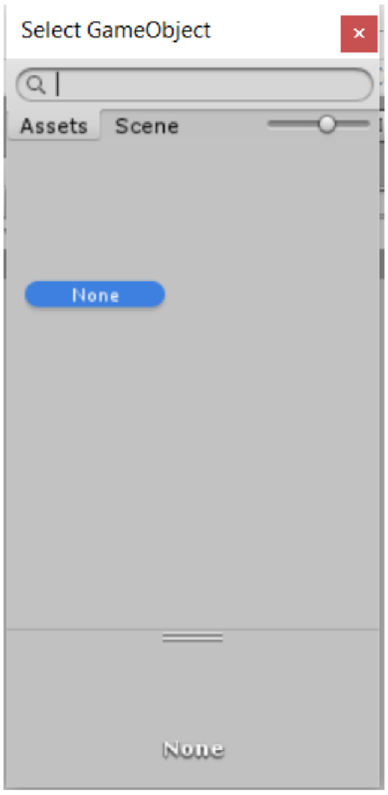

Chase Cam (Script)

Script: chaseCam

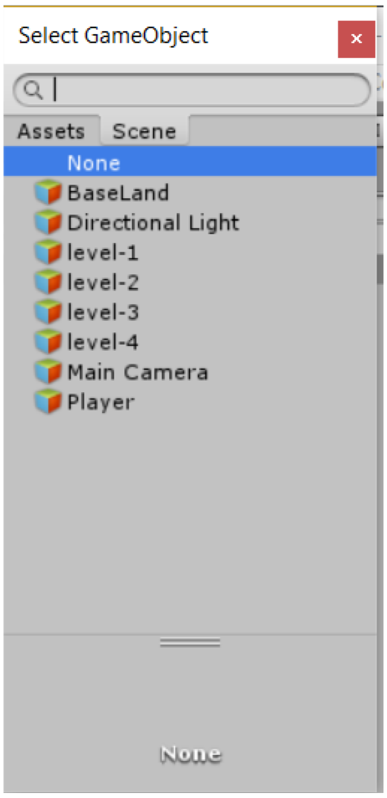
Object To Follow: None (Game Object)

Speed: 2

Notice we have a default speed, which we can change, and nothing set to follow. To fix this, click on the target to the right of Object to follow, this will bring up the Select GameObject panel.

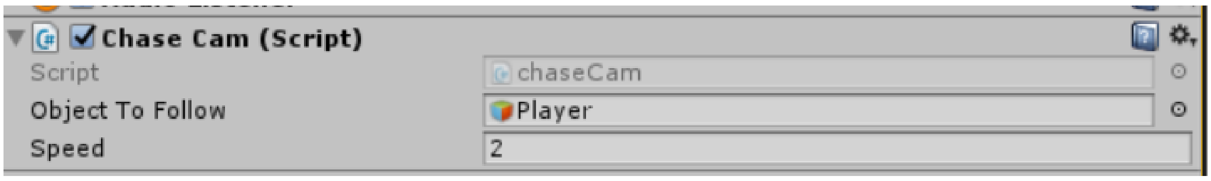


From here, click on the scene tab, this will provide a list of objects to select.



BaseLand
Directional Light
level-1
level-2
level-3
level-4
Main Camera
Player

From this list of objects in the game select Player. It will appear in the Inspector Panel.


Chase Cam (Script)

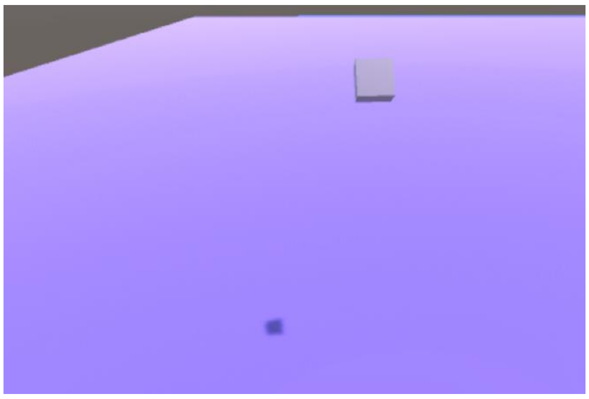
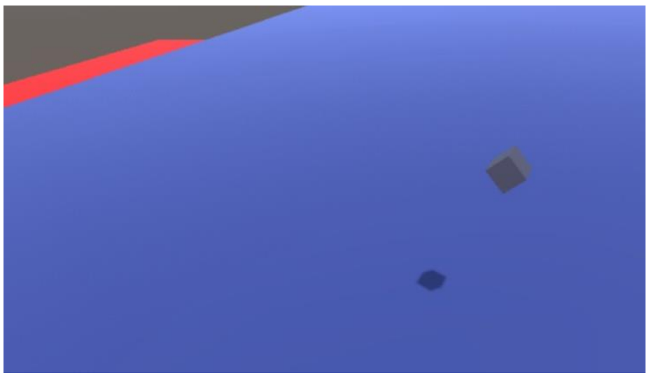
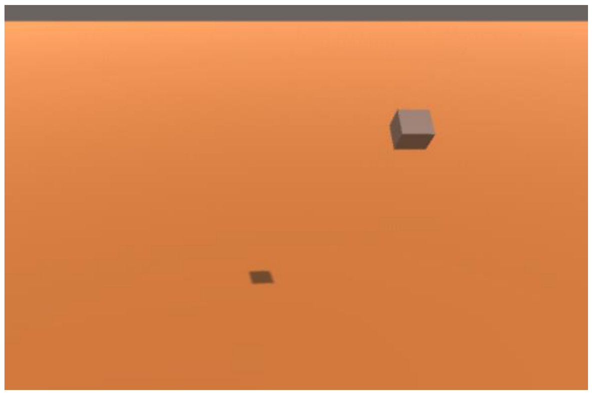
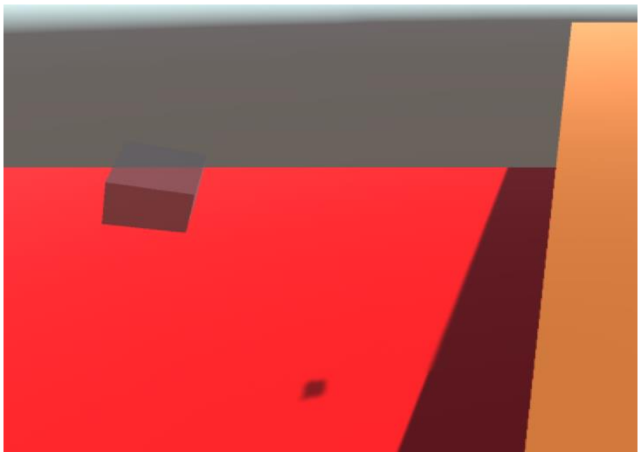
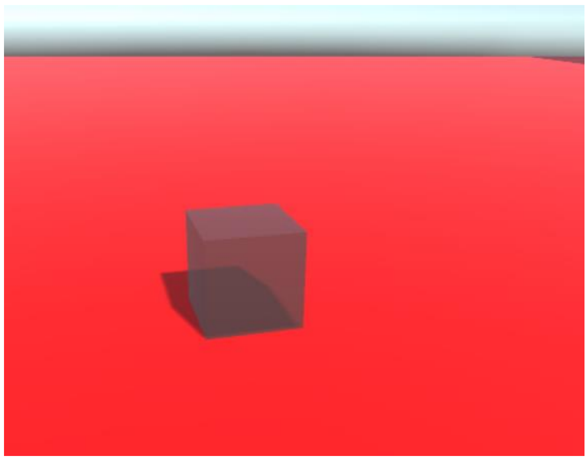
Script: chaseCam

Object To Follow: Player

Speed: 2

Now hit play to test the game.

By using the keys, you should be able to move the player from the green BaseLand object through to level 4, the red base.

As you can see, the camera follows the player over each of the levels. There are some unique elements such as the player being able to speed away from the camera, though when we slow down the camera catches up. This is based off not having limits on the speed the player can get to. Feel free to play with the settings.

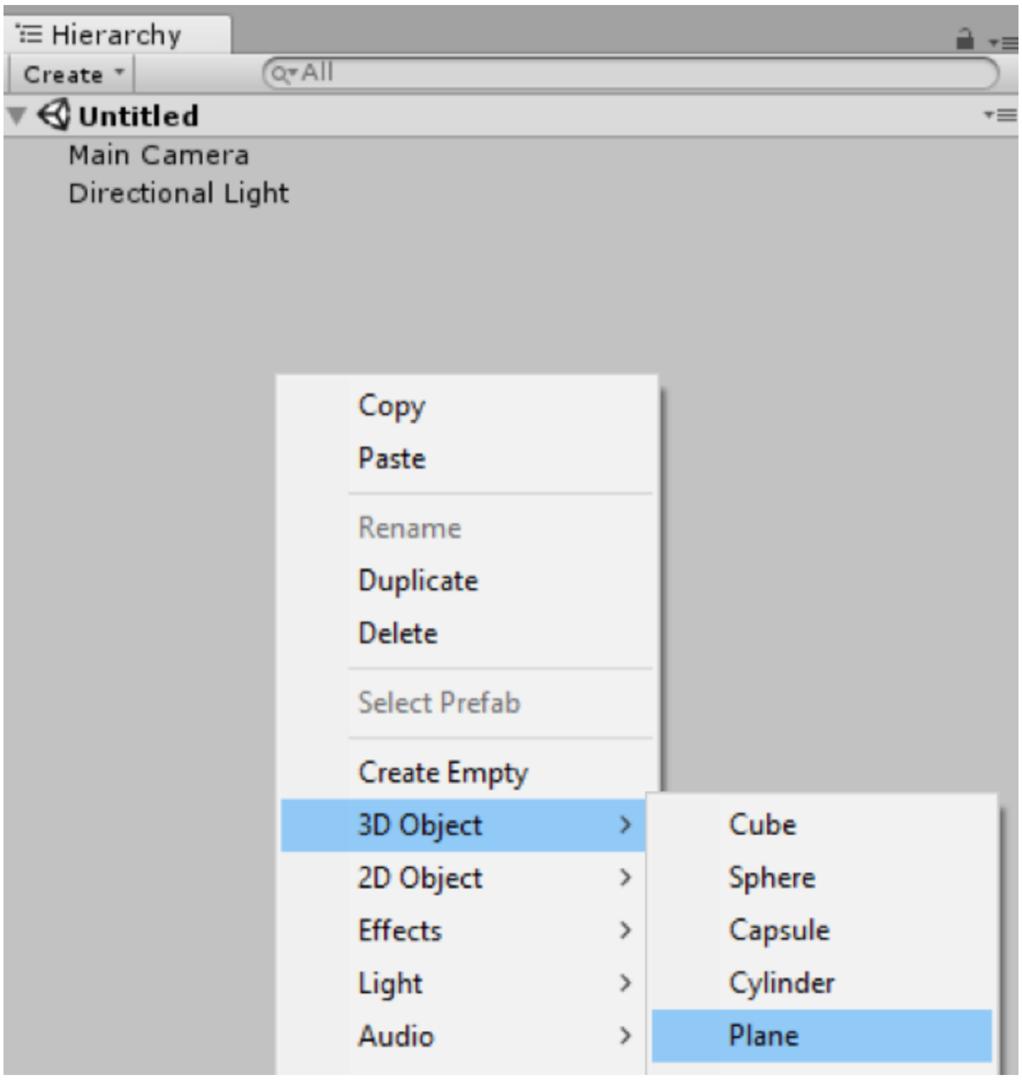
Script 2

There are multiple scripts that can be used when using unity, they all rely upon modifying the state of the camera. As such, here is another script for a chase cam.

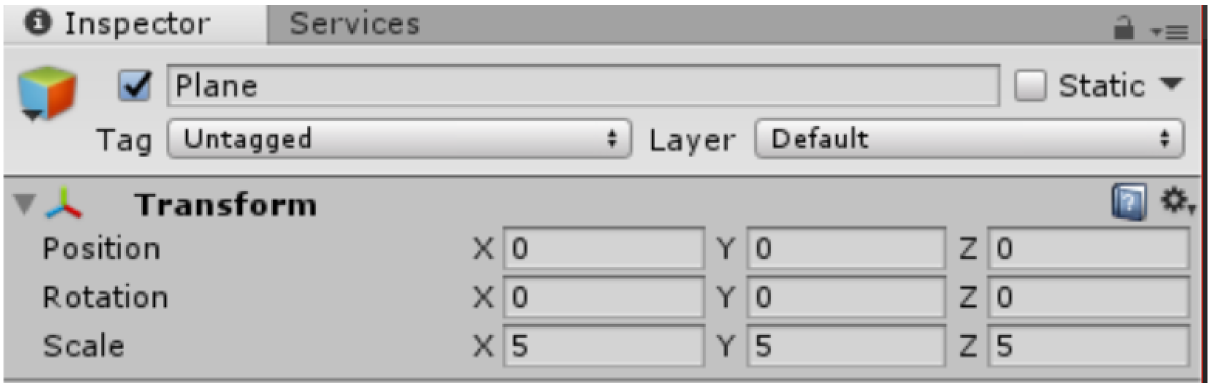
Start a new project and call it CC2.

Add a plane and a sphere to the world.

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

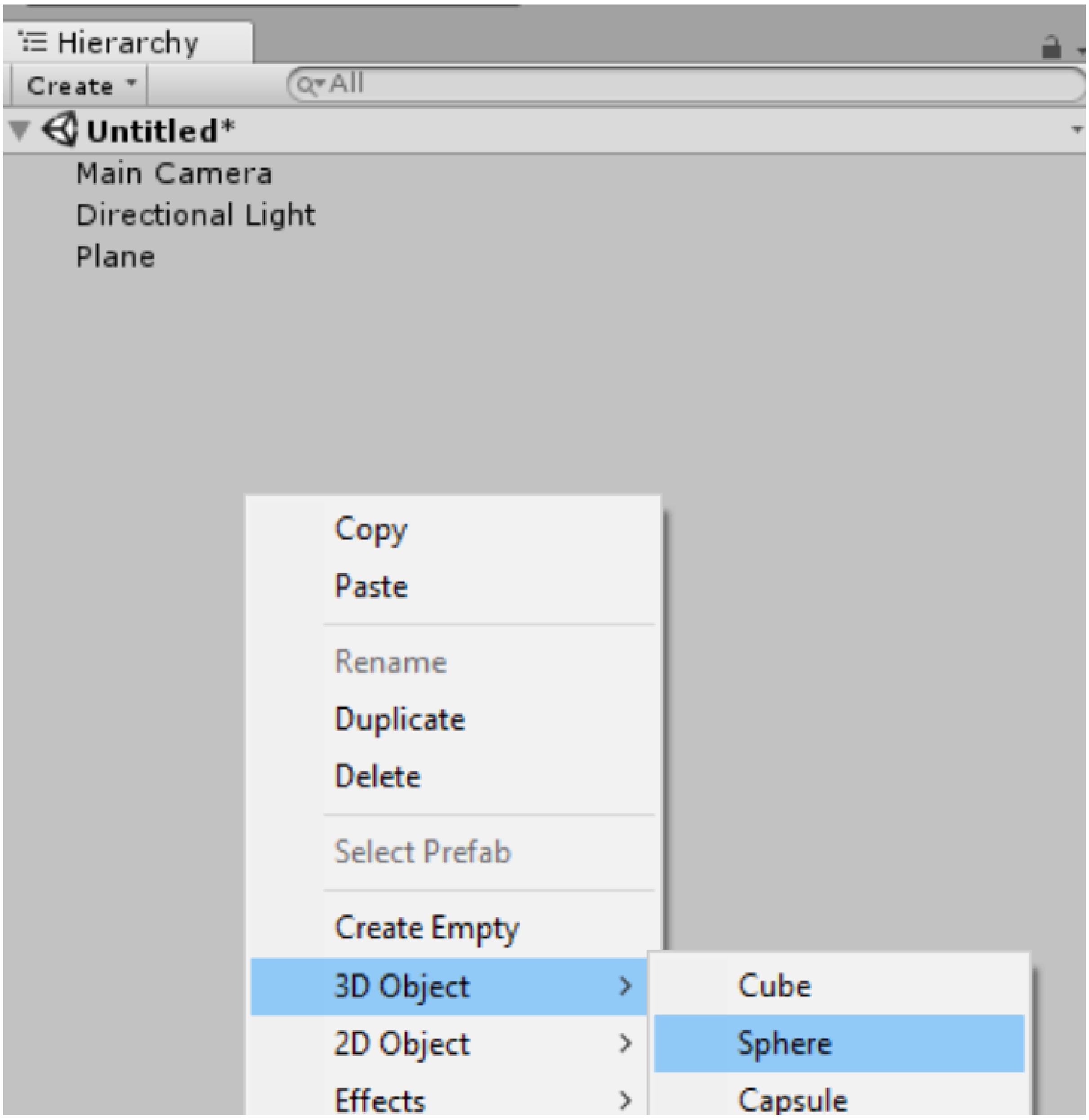


In the inspector side of the plane, boost the scale to 5.

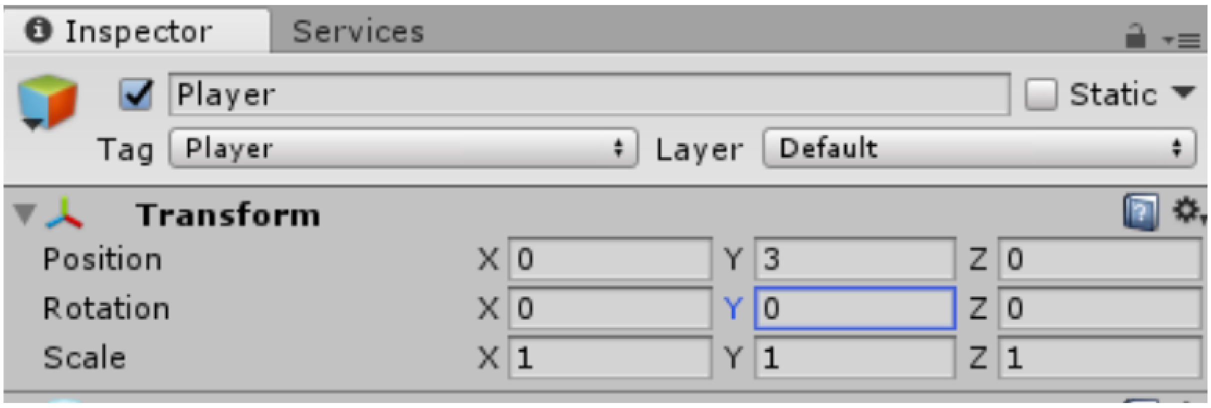


	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	5	5	5

Next right click in the hierarchy and add a sphere.



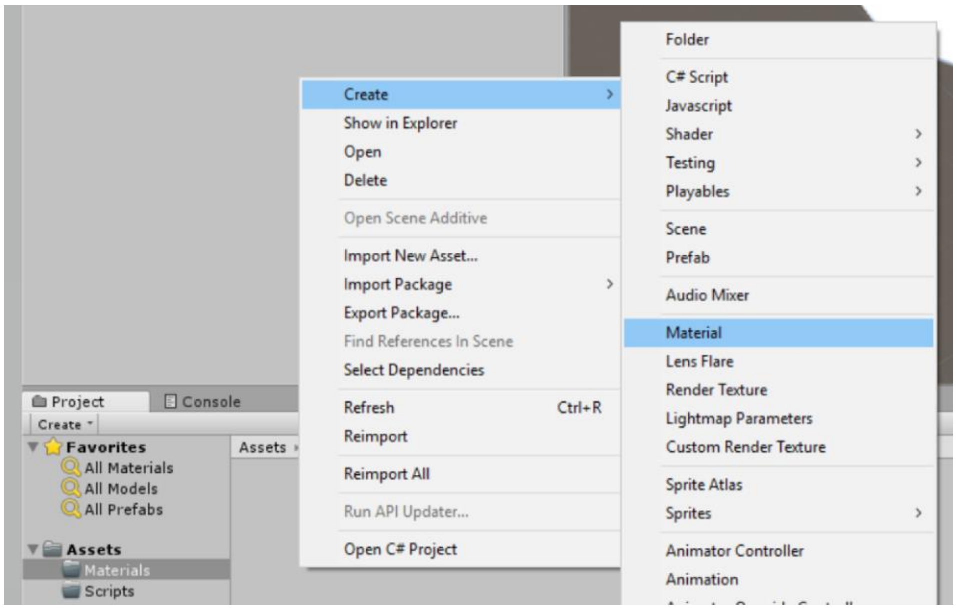
In the inspector properties, name the sphere player, ensure it is tagged as player and raise the sphere's Y axis so it is above the plane.

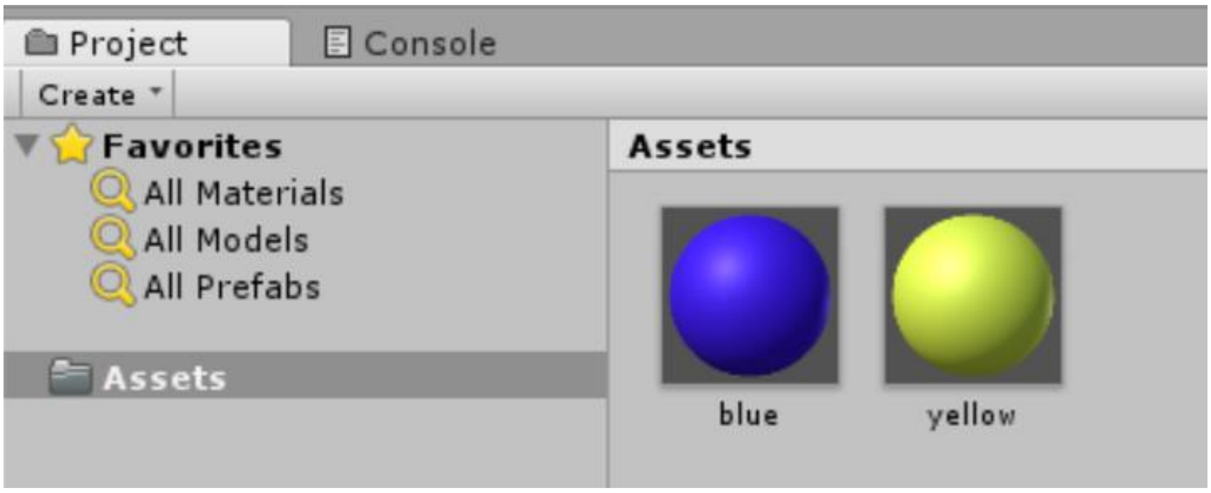


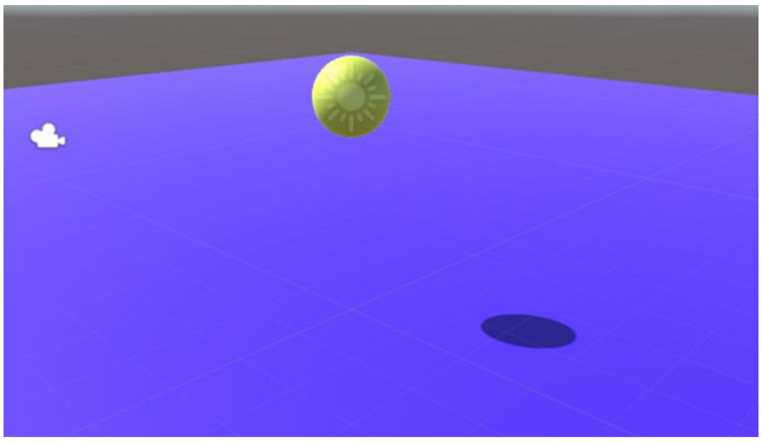
Player
Tag Player Layer Default

Position	X	0	Y	3	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Create 2 new materials and drag onto the plane and sphere, in this manner we will be able to distinguish the differences between the objects.

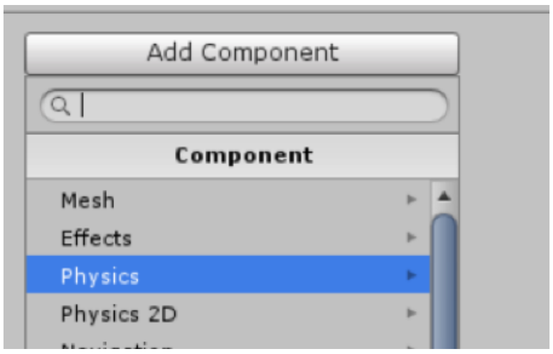


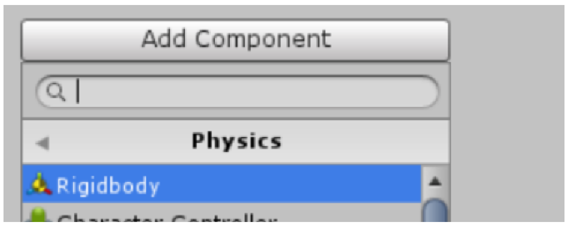




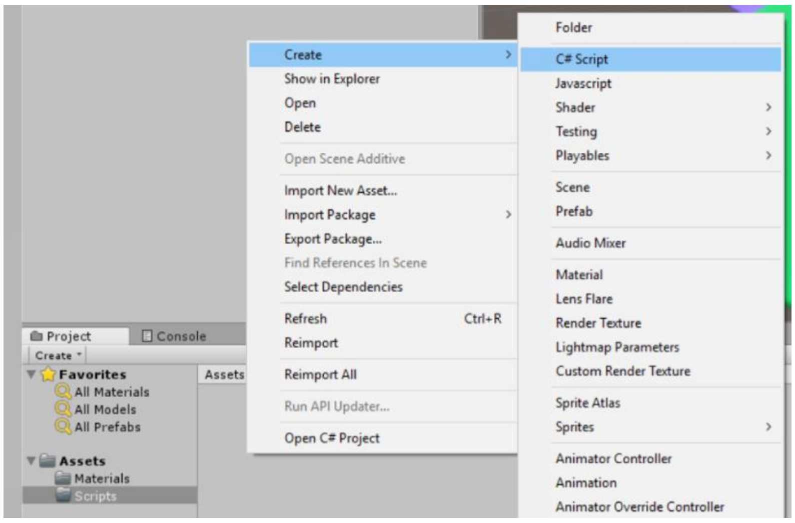
Once this is done, we need to apply a rigid Body to the Player so it will be able to fall and then the move code to the player as well.

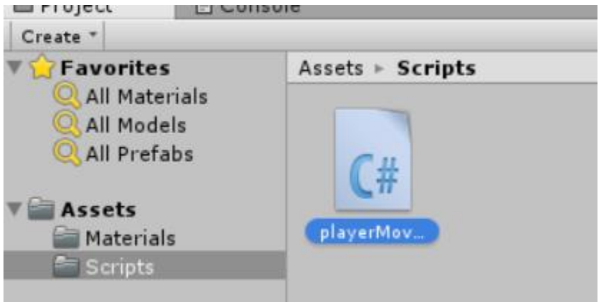
. This can be done by clicking on Add Component->Physics->RigidBody



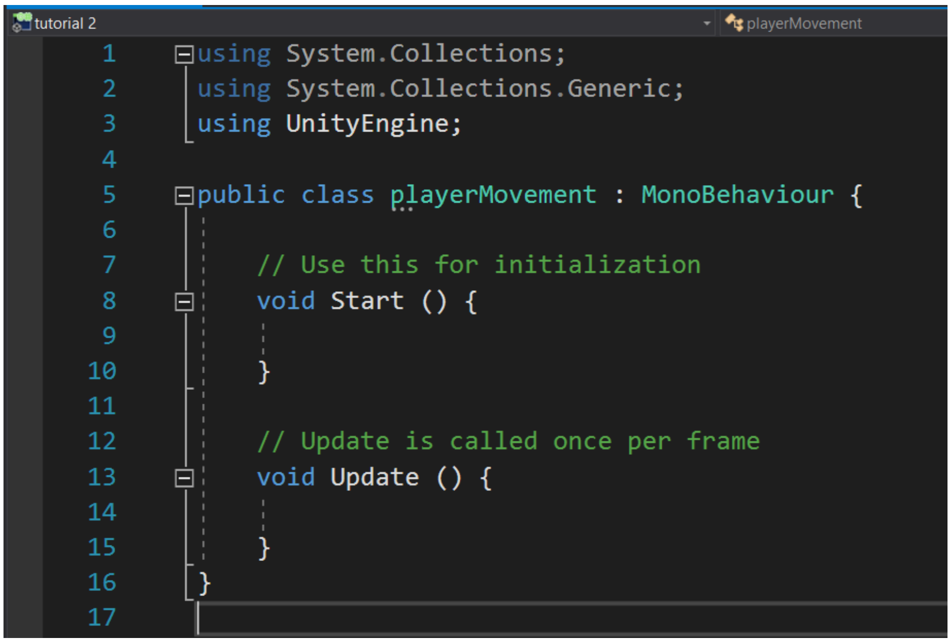


From here we add a new script to the player object, in the Assets folder, click into Scripts, then right click and create new C# Script, call this script playerMovement.





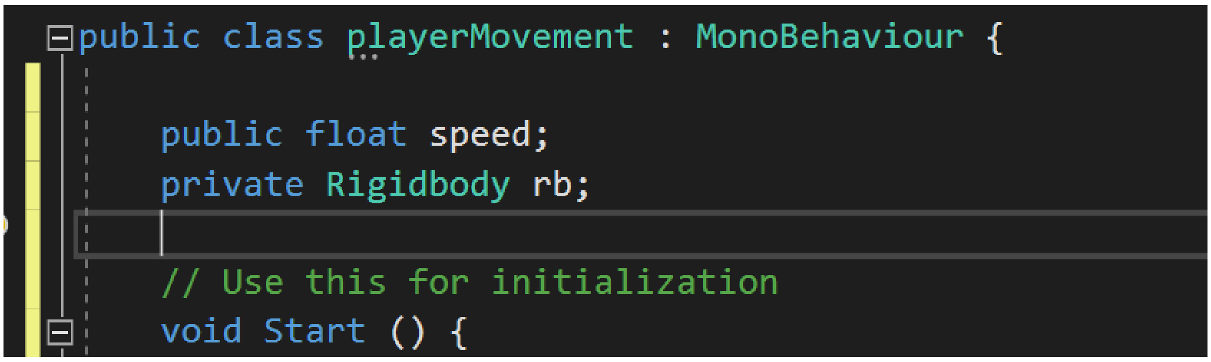
Now, double click the script, it should open in visual studio.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class playerMovement : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}

```

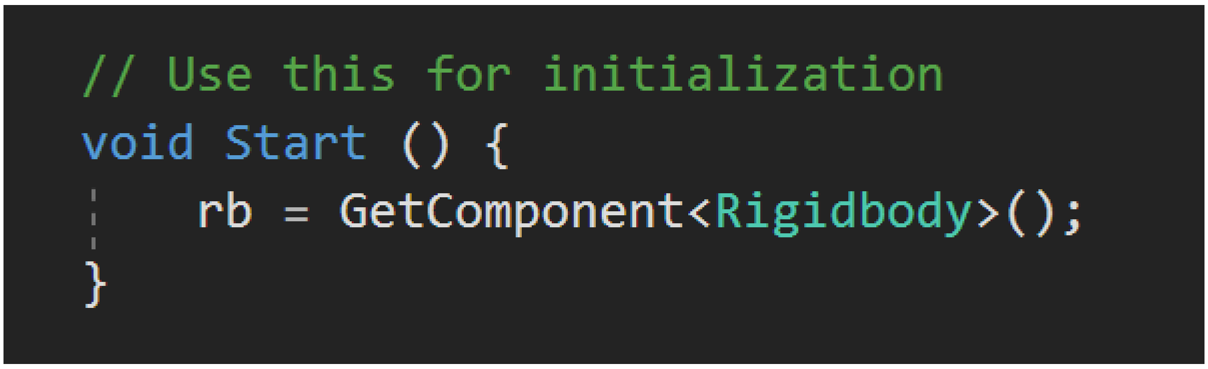
As we did in the last tutorial, we need to be able to programmatically access the player object, to do this we will create a couple of variables above the void start, this will be the speed (used for speed of movement) and rigidbody (Allow the object to be interacted with).

The code looks like this



```
public class playerMovement : MonoBehaviour {

    public float speed;
    private Rigidbody rb;

    // Use this for initialization
    void Start () {
```

Next, we initialise the access to the rigidbody this is done in the start function. We get all of the components of the rigidbody and make then accessible via the rb variable

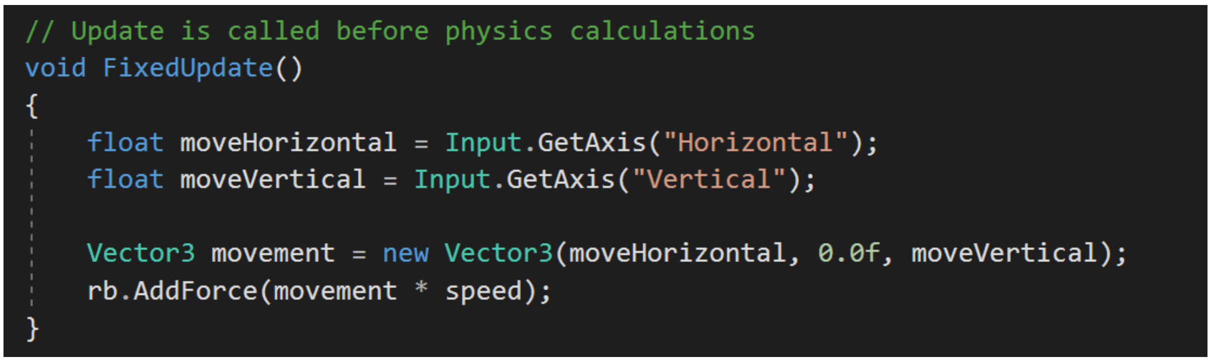


```
// Use this for initialization
void Start () {
    rb = GetComponent<Rigidbody>();
}
```

Next, we add in the ability to capture the movement via input, i.e. a key is pressed (WASD or Cursor keys). This is done by creating a couple of float variables called moveHorizontal and moveVertical.

From here, we implement a Vector3 variable (x, y, z). This variable holds the horizontal, vertical and Y position of the player object.

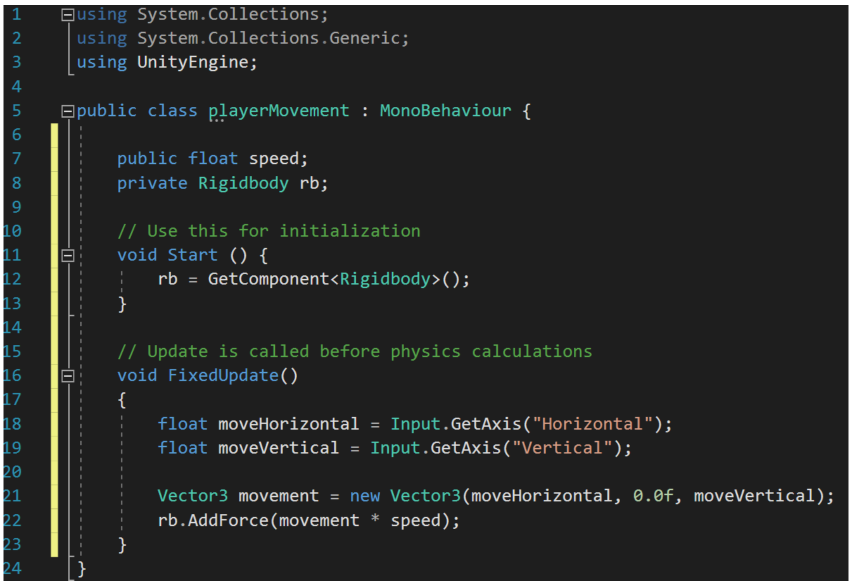
Finally, we apply force to the rigidbody to make it move when the keys are pushed. The force is calculated by movement * speed. The code below is what should be created.



```
// 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);
}
```

Notice that the Update function is replaced with the FixedUpdate function.

The overall code for player movement is:



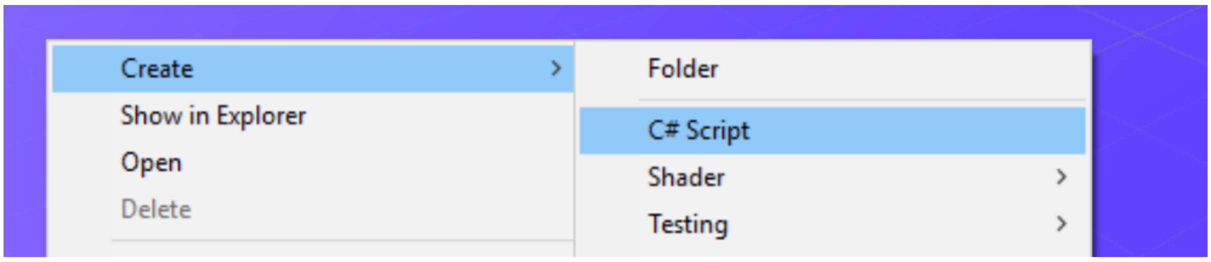
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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 (Ctrl+S or File -> Save), then return to unity.

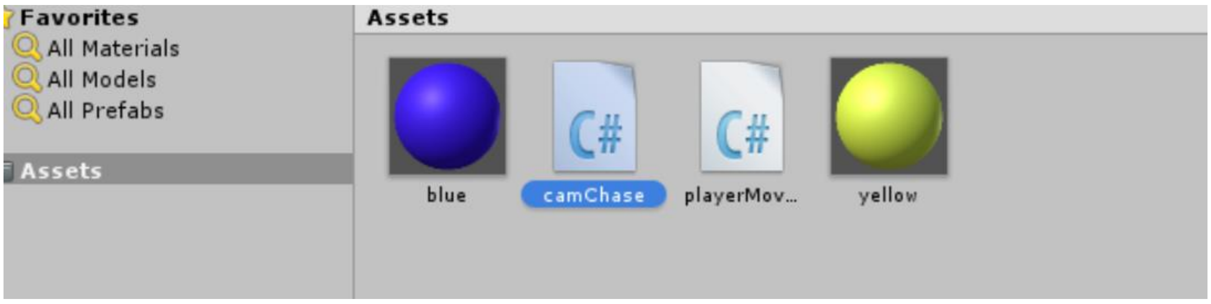
Now if we test it, we should be able to see the player drop onto the plane and be moved around by using the WASD keys.

Now we will create a new scrip to go on the main camera.

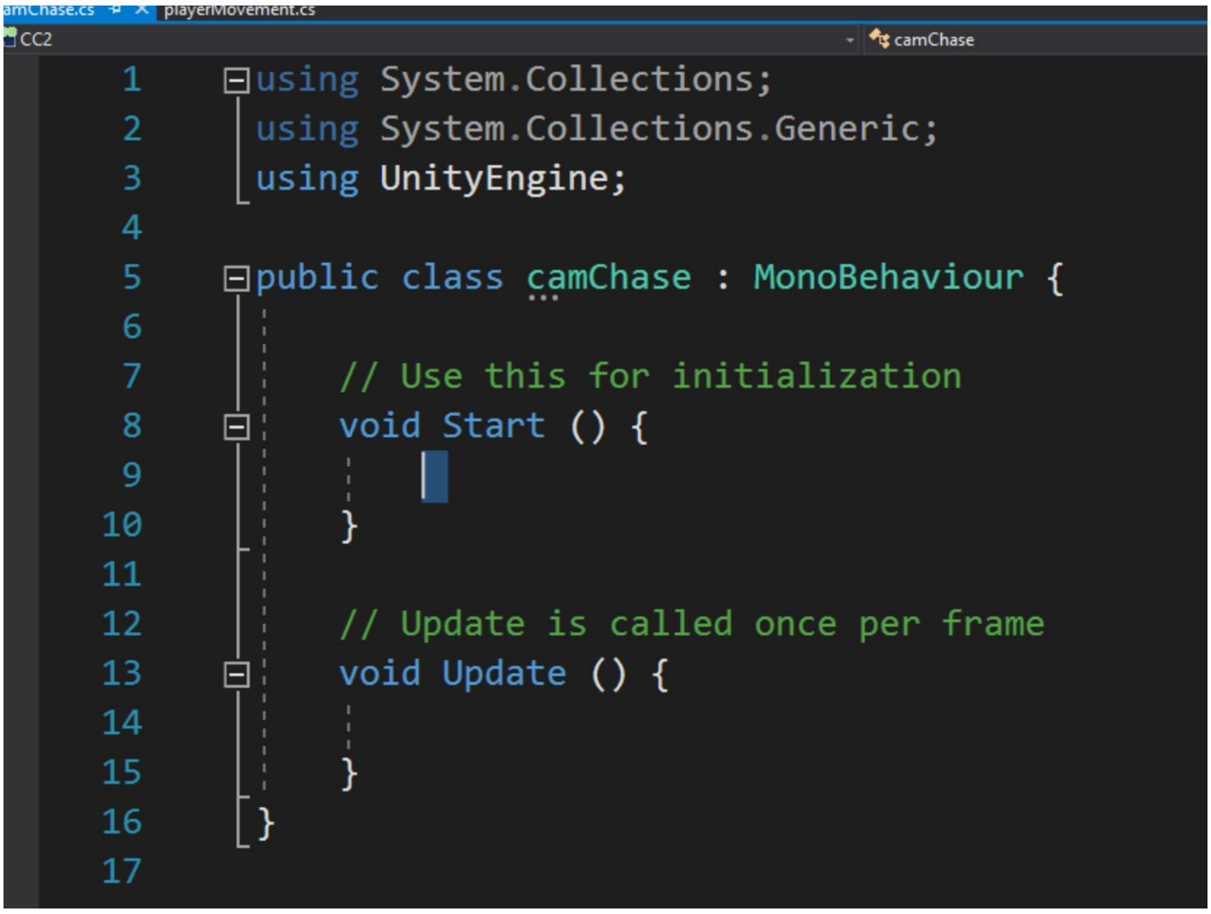
Right click in the scripts folder and create a new C# script.



Call this Script chaseCam

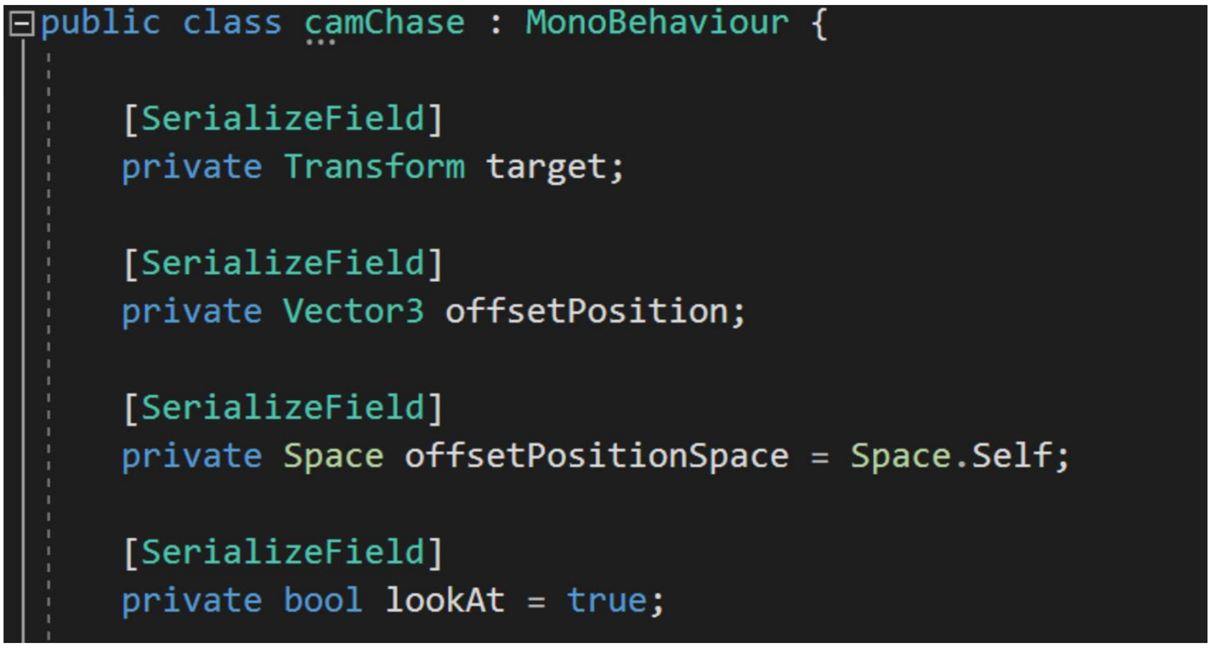


Double click the file and open up visual explorer



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class camChase : MonoBehaviour {

	// Use this for initialization
	void Start () {
		
	}

	// Update is called once per frame
	void Update () {

	}
}

```

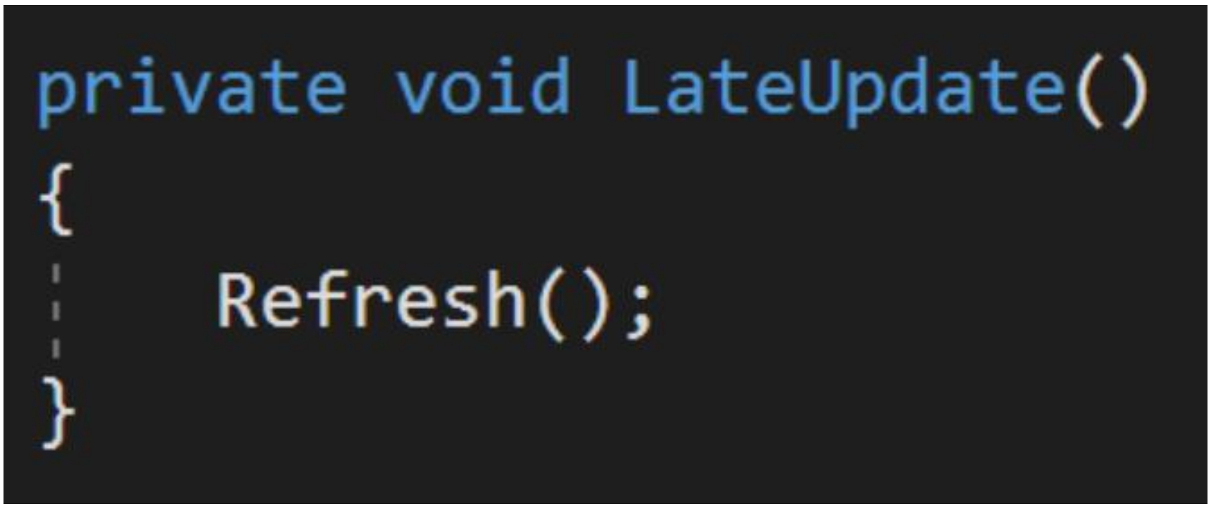
Change the code in the following manner



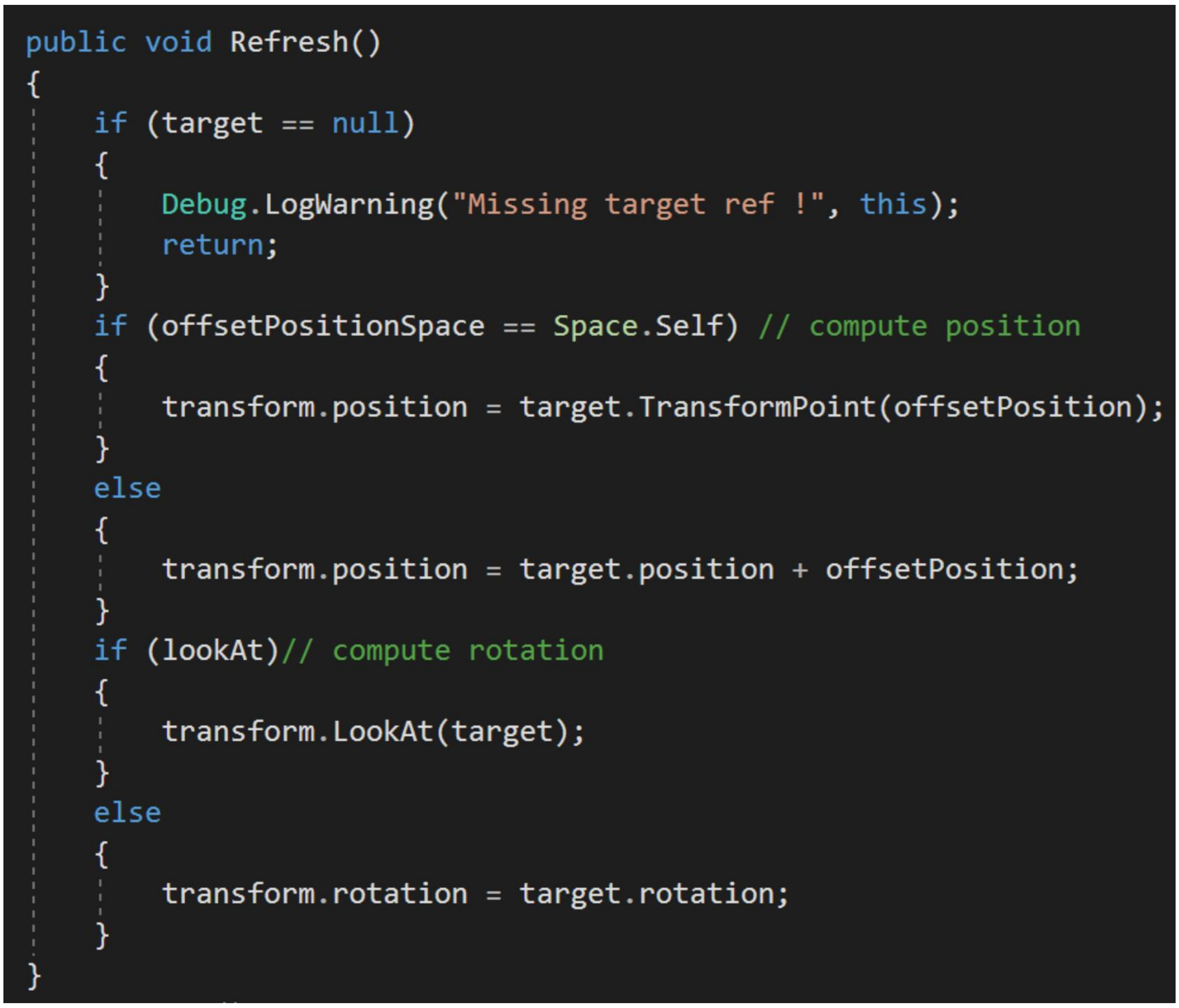
```
public class camChase : MonoBehaviour {

    [SerializeField]
    private Transform target;

    [SerializeField]
    private Vector3 offsetPosition;

    [SerializeField]
    private Space offsetPositionSpace = Space.Self;

    [SerializeField]
    private bool lookAt = true;
```

Side Note: a serialized field makes a private variable accessible in unity, this can be achieved with a public variable, but it does appear in unity code, hence the introduction to it.

After the variables, add the following:



```
private void LateUpdate()
{
    Refresh();
}
```

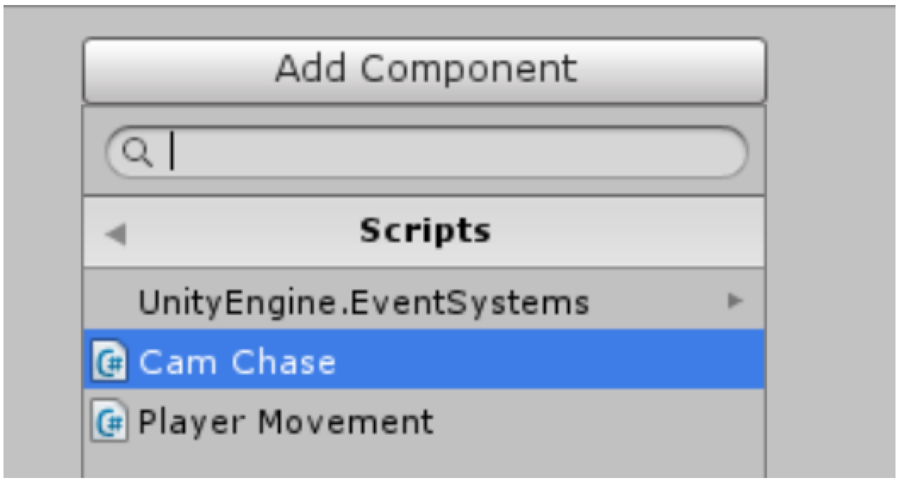


```
public void Refresh()
{
    if (target == null)
    {
        Debug.LogWarning("Missing target ref !", this);
        return;
    }
    if (offsetPositionSpace == Space.Self) // compute position
    {
        transform.position = target.TransformPoint(offsetPosition);
    }
    else
    {
        transform.position = target.position + offsetPosition;
    }
    if (lookAt)// compute rotation
    {
        transform.LookAt(target);
    }
    else
    {
        transform.rotation = target.rotation;
    }
}
```

Ensure you do not miss the closing } curly brace.

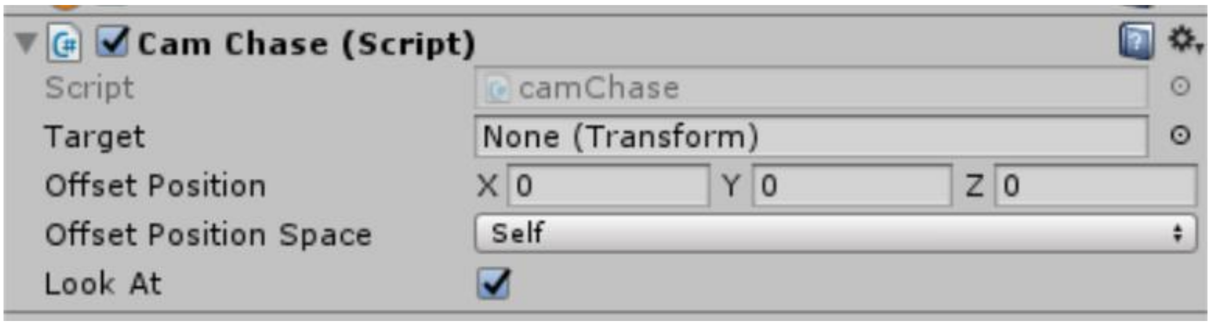
Save and go back to unity.

Once in unity, highlight the main camera in the hierarchy panel and then in the inspector panel , add the camChase script.



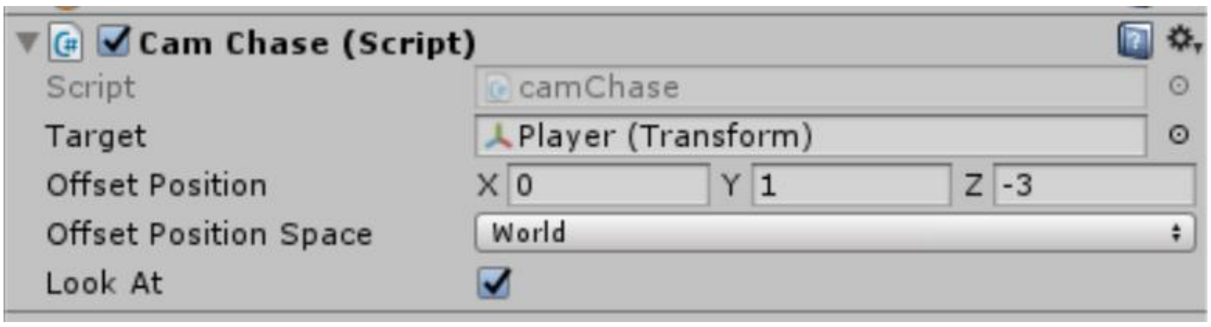
Once the script has been added, you will need to set the properties. In the inspector.

Original


Cam Chase (Script)

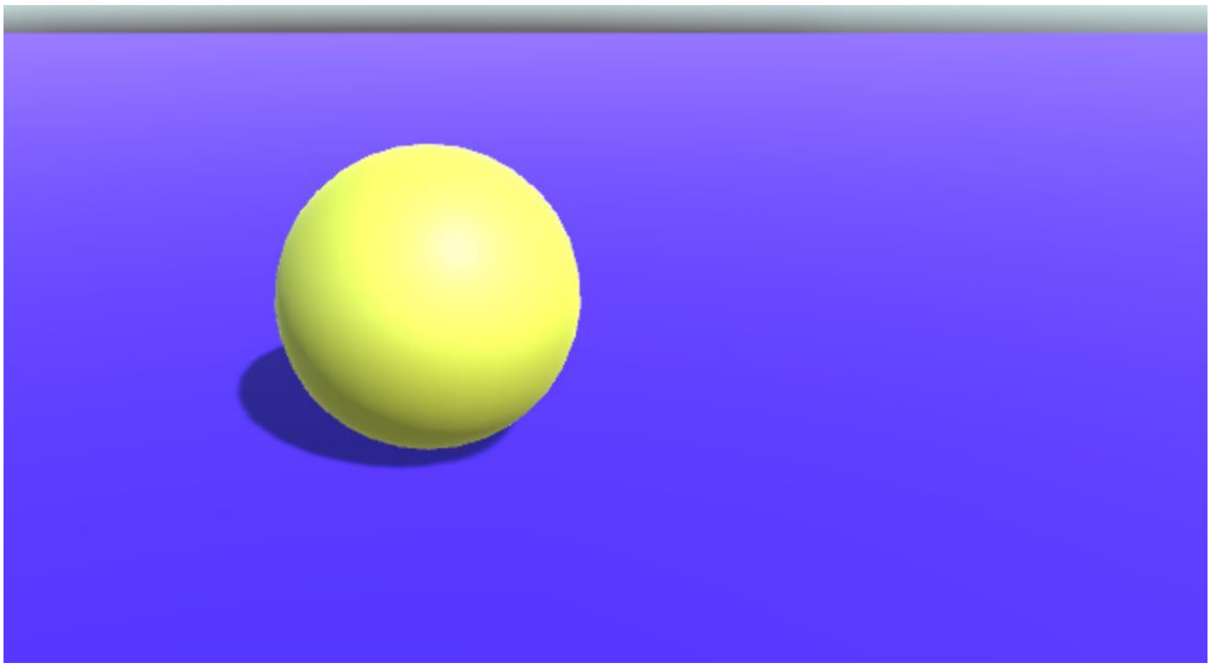
Script	camChase
Target	None (Transform)
Offset Position	X 0 Y 0 Z 0
Offset Position Space	Self
Look At	✓

Modified


Cam Chase (Script)

Script	camChase
Target	Player (Transform)
Offset Position	X 0 Y 1 Z -3
Offset Position Space	World
Look At	✓

Save and test.



The camera will position itself behind the player and when you move the player, the camera will follow as well.