

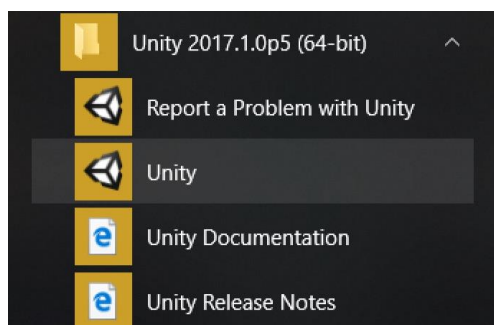
Game Design – Unity Workshop

Activity 4

Goals:

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

Load up unity

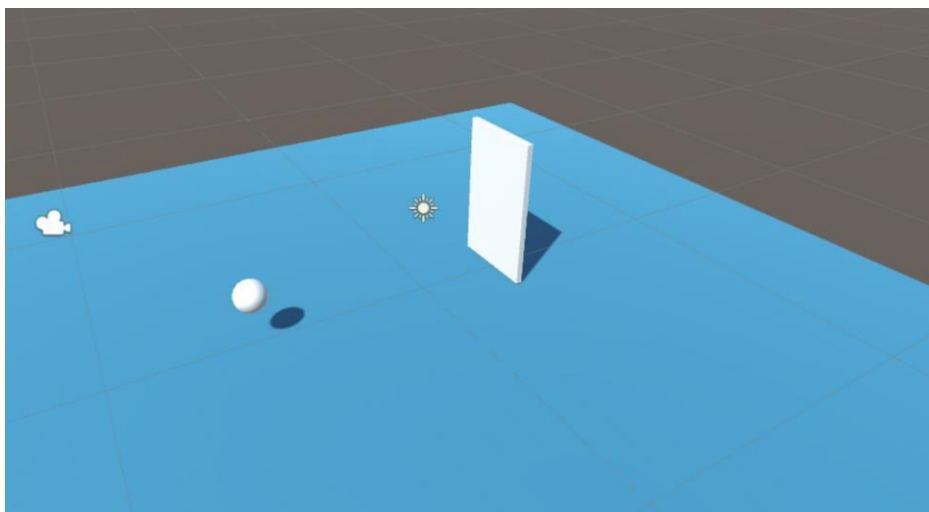


Build Object: Collisions in Unity

Aim: Build various elements and test differing collision types.

Door Open

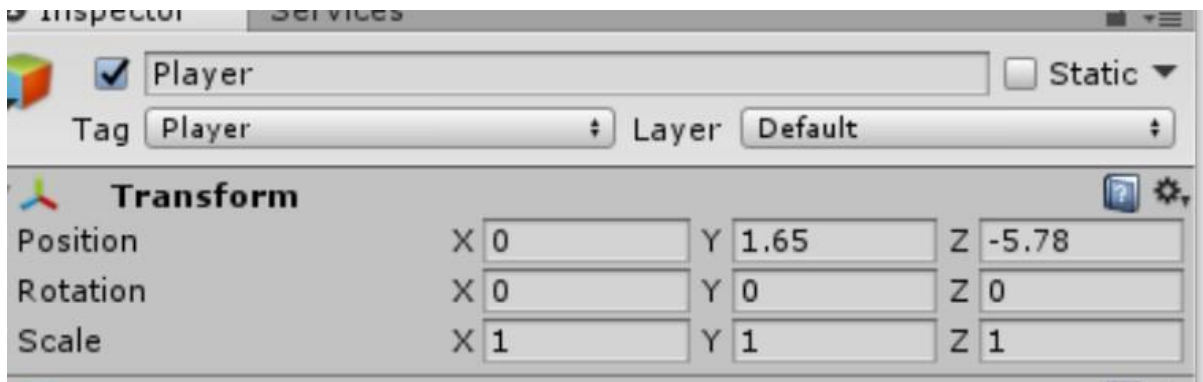
Build the following scene



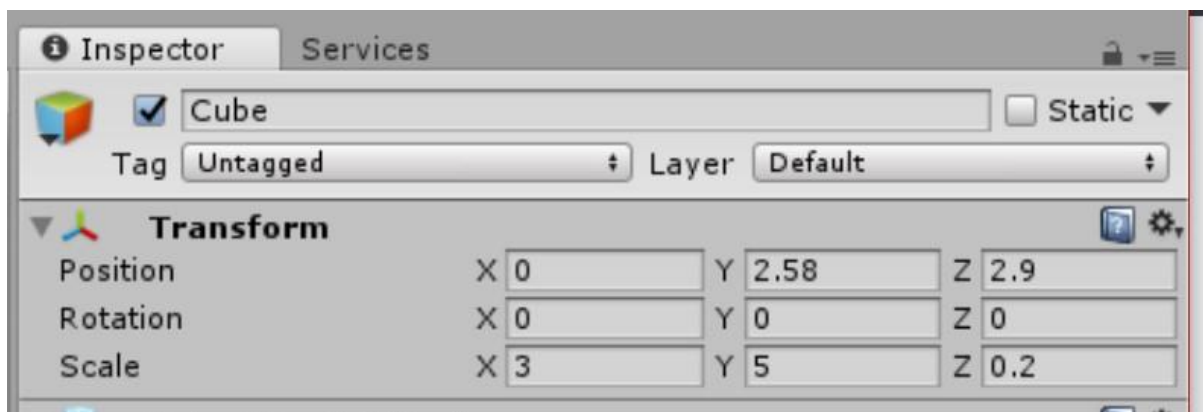
The main camera is lined up like this:



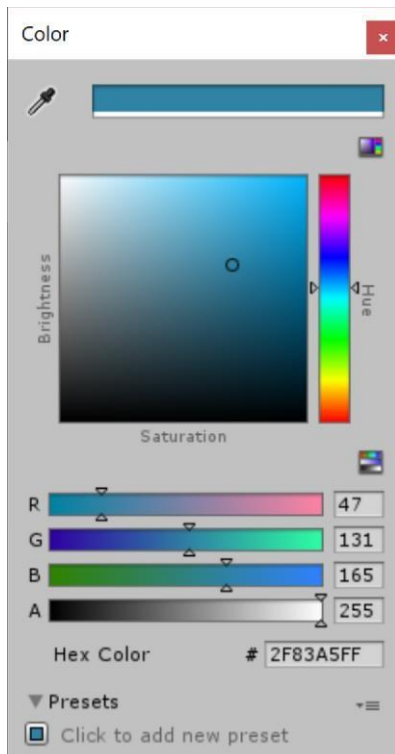
The sphere has been called player and has the following attributes



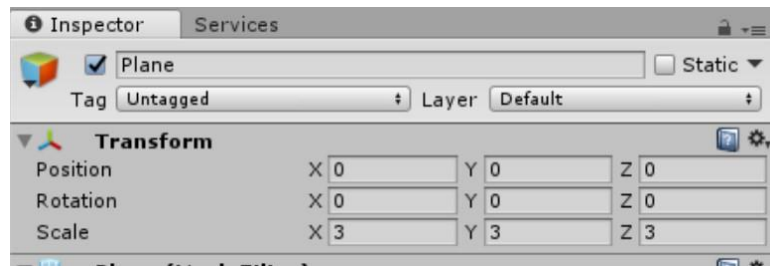
The cube has been shaped like a door, and has the following attributes



The floor plane has the following material applied to it

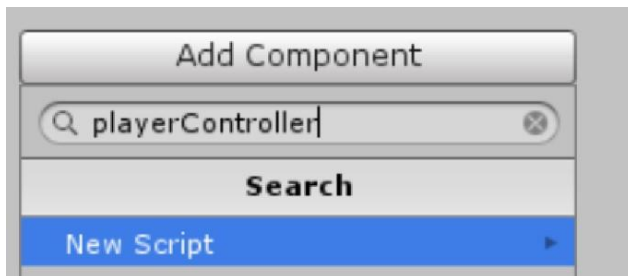


And the following information

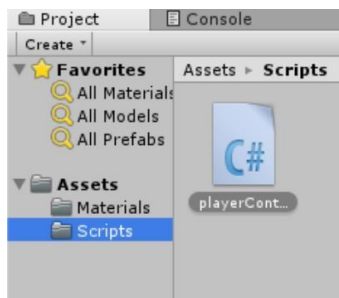


Now that we have the basic layout, we will add the code to move the player around.

Create a new script called playerController and apply it to the player.



Place it in the scripts folder and then open it up in visual studio and apply the following code



```

5 public class playerController : MonoBehaviour {
6
7     private Rigidbody rb;
8     public float speed = 7.0f;
9
10    // Use this for initialization
11    void Start () {
12        rb = GetComponent<Rigidbody>();
13    }
14
15    // Update is called once per frame
16    void FixedUpdate () {
17        float moveHorizontal = Input.GetAxis("Horizontal");
18        float moveVertical = Input.GetAxis("Vertical");
19
20        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
21        rb.AddForce(movement * speed);
22    }
23 }

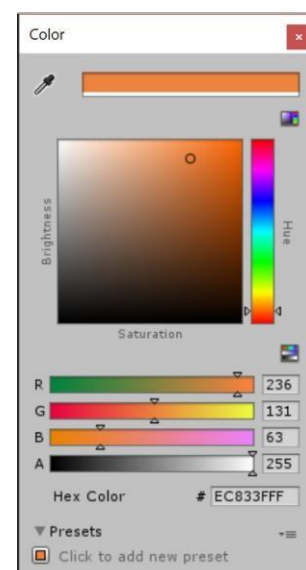
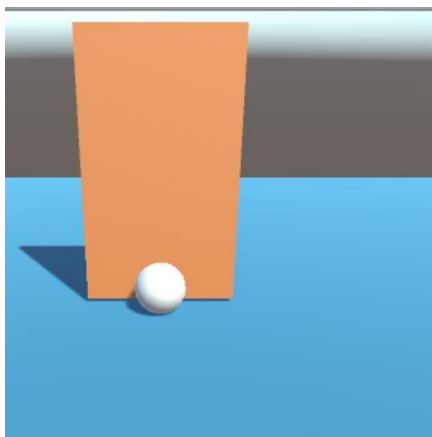
```

Save and test in Unity.

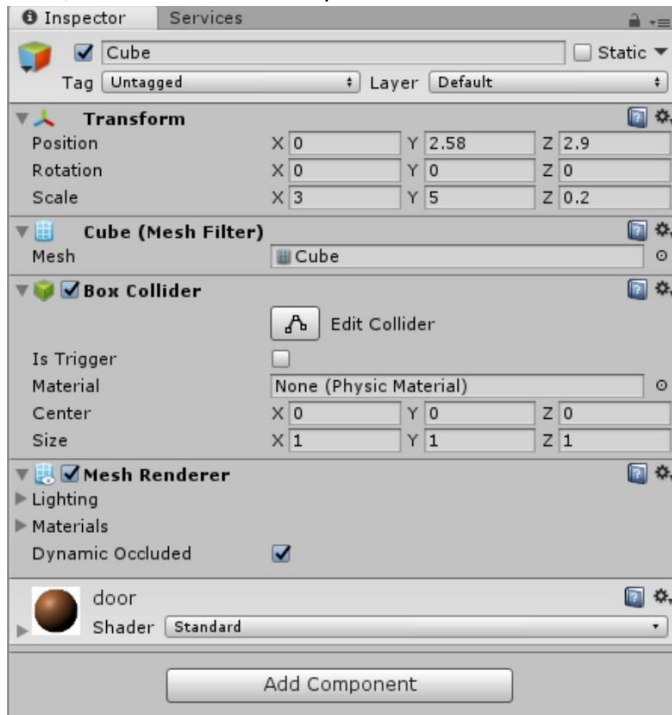
To ensure that the player sphere is visible when heading towards the cube, we'll change the colour of the cube.

The colour selected was this:

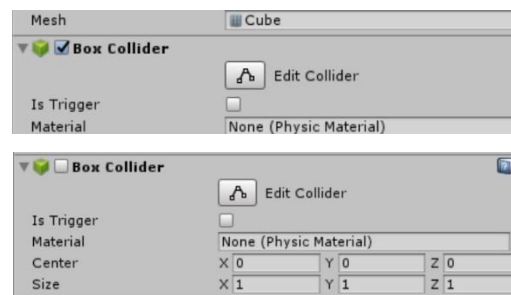
Now that we have the ability to see the sphere when it approaches the door, we need to implement a collision area and an effect that occurs when we get there, in this case, we will just move the door.



Now, let's examine the inspector on the cube.



The transform is as expected, nothing too out of the ordinary, Cube mesh filter is fine, but notice how it automatically has a box collider on it that is active, if you make the box collider inactive, i.e. remove the tick, and play. You will see the player able to move straight through the door object.



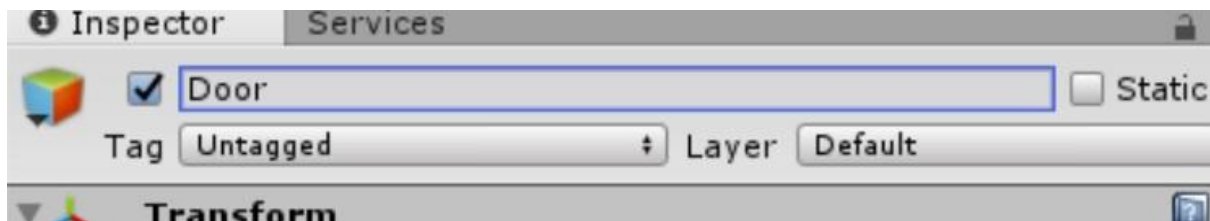
Do this and test.

Turn back on the box collider.

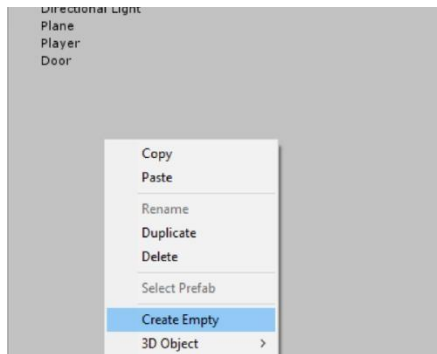
So, the collider is what allows our objects to interact with each other.

The plan is to add an additional collider around the door object (Now is a good time to rename cube to door) and have it trigger an event.

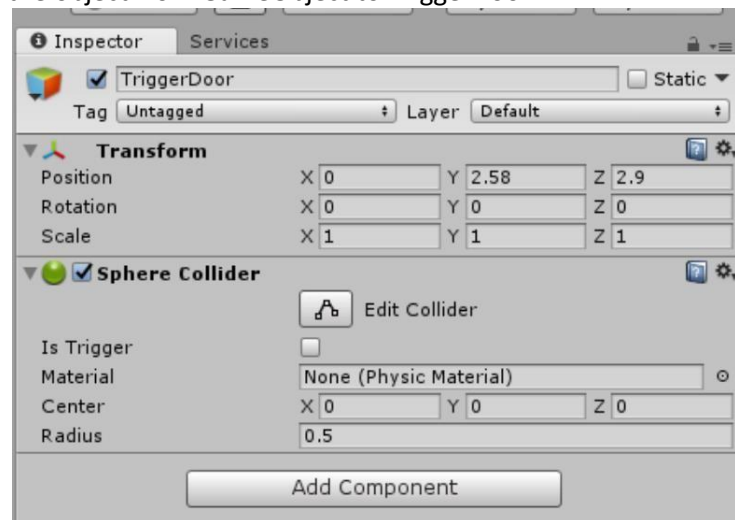
Rename cube to Door.



Right click on the hierarchy and add an empty game object.



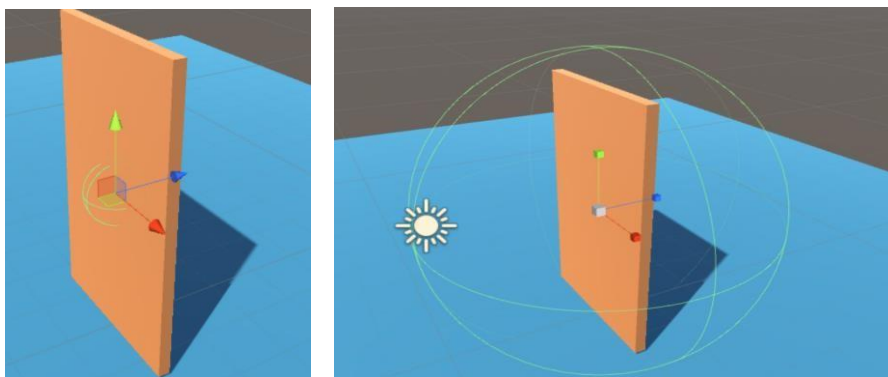
From here, we add a sphere collider onto this object, rename the object from GameObject to TriggerDoor.



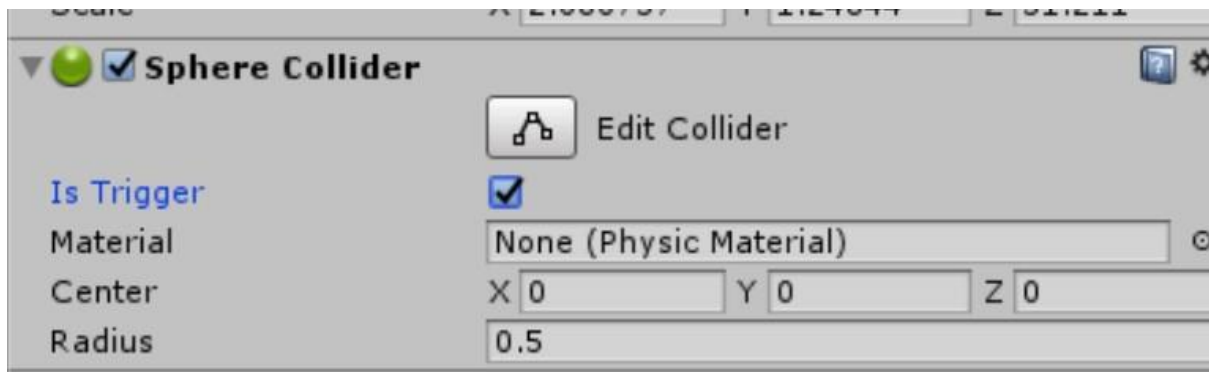
In the hierarchy drag the TriggerDoor on the Door object, in this way creating a parent/Child link.



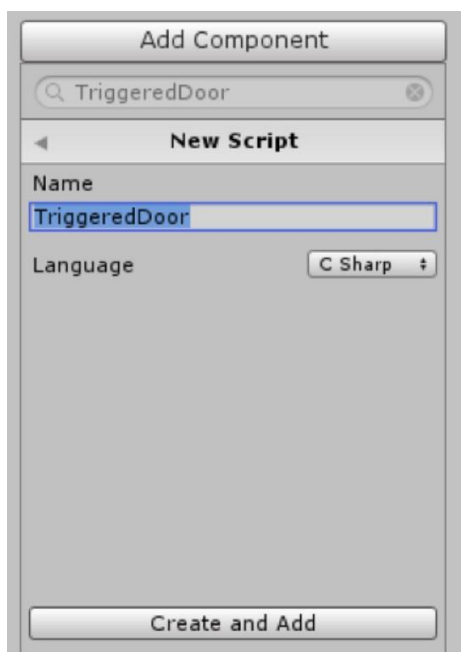
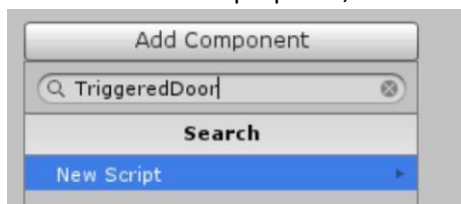
Now, when looking at the scene, the sphere collider is too small to be of use, as we want it to trigger before the player hits the door, as such, scale it up.



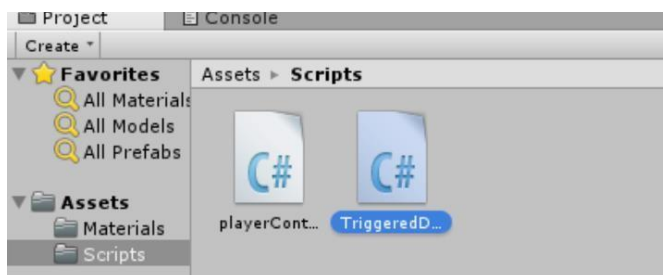
In the inspector, we need to change the sphere collider's attribute, so it works as a trigger and not a collider. Otherwise our player won't be able to pass through the door area. So, on the sphere collider, check the is trigger box.



Now that we have prepared, let's add a script to the TriggerDoor object.



Move the created script into the script file, then open it up in visual studio and add the following code.



```

5 public class TriggeredDoor : MonoBehaviour {
6
7     public Transform door;
8     public Vector3 openDoorPosition = new Vector3(0.0f, 5.0f, 3.0f);
9     public Vector3 closeDoorPosition = new Vector3(0.0f, 2.5f, 3.0f);
10
11     public float openSpeed = 5.0f;
12     public bool open;
13
14     // Use this for initialization
15     void Start () {
16
17     }
18
19     // Update is called once per frame
20     void Update () {
21         if (open)
22         {
23             door.position = Vector3.Lerp(door.position, openDoorPosition, Time.deltaTime * openSpeed);
24         }
25         else
26         {
27             door.position = Vector3.Lerp(door.position, closeDoorPosition, Time.deltaTime * openSpeed);
28         }
29     }

```

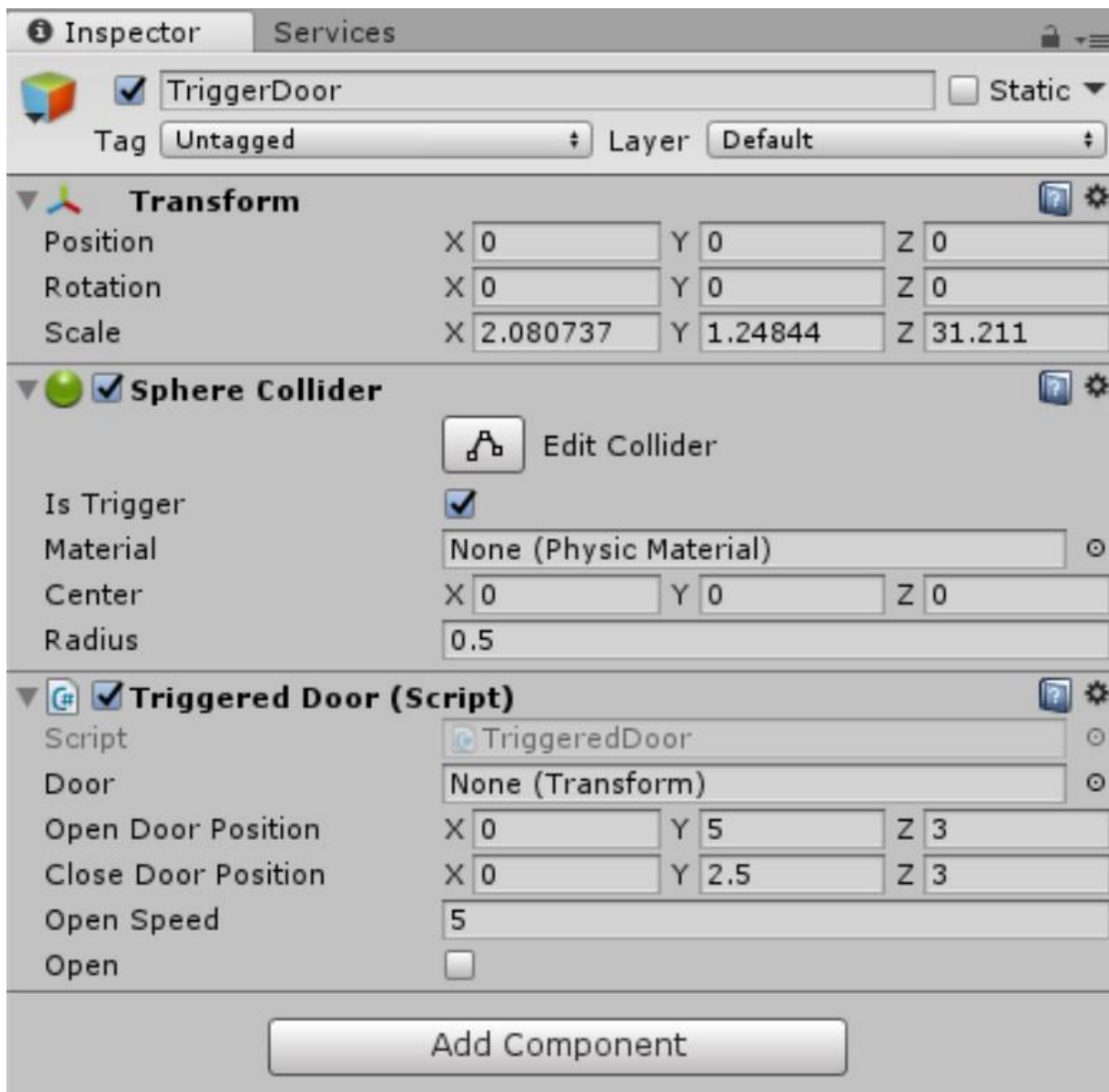
```

30
31     private void OnTriggerEnter(Collider other)
32     {
33         if(other.tag == "Player")
34         {
35             TriggerOpen();
36         }
37     }
38
39     private void OnTriggerExit(Collider other)
40     {
41         if(other.tag == "Player")
42         {
43             TriggerClose();
44         }
45     }
46
47     public void TriggerOpen()
48     {
49         open = true;
50     }
51     public void TriggerClose()
52     {
53         open = false;
54     }
55 }
56

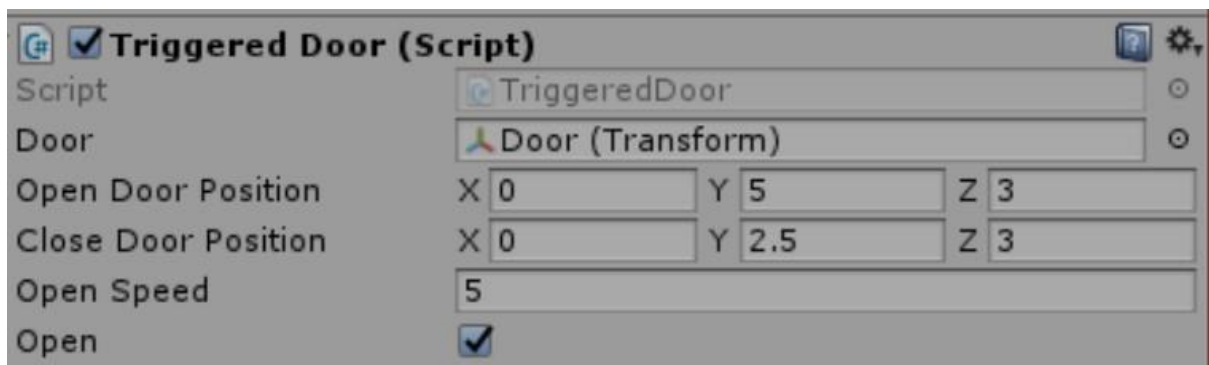
```

Save this and then go back to unity.

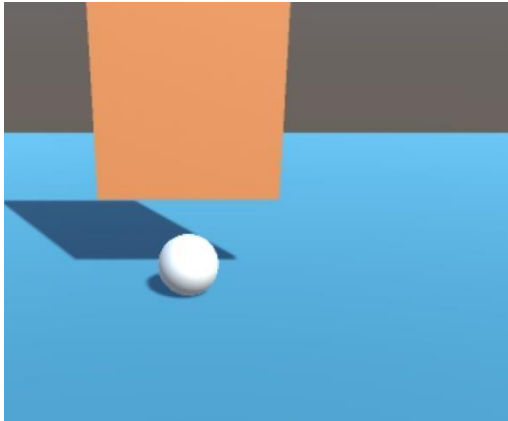
Ensure that the script is attached to the TriggerDoor Object



When looking at the script, notice how the Transform door code, aligns with None, this is awaiting a link to the actual object, drag the door object into it or use the target to locate the object.



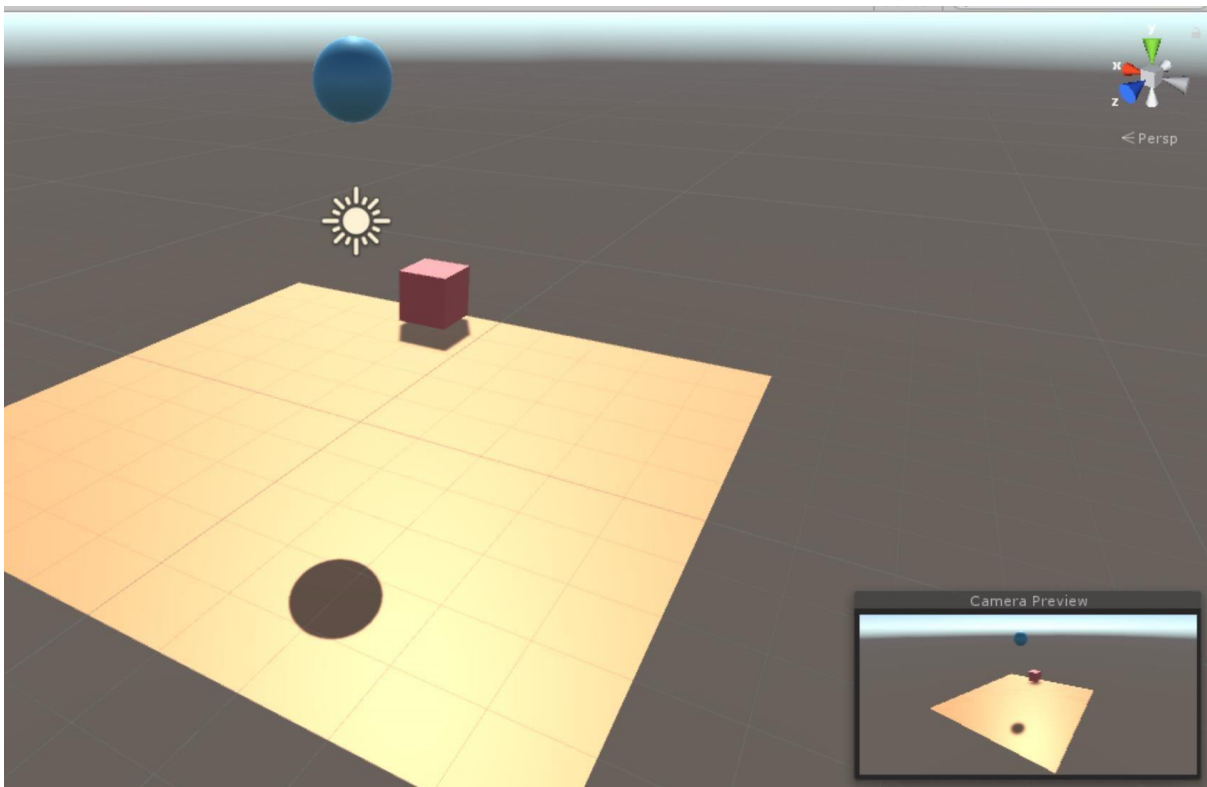
Run the program and you should see the door lifting up to allow the player sphere to travel under it.



During testing the speed of 5 didn't open the door smoothly, when I dropped the Open Speed to 1, the door opening was much smoother.

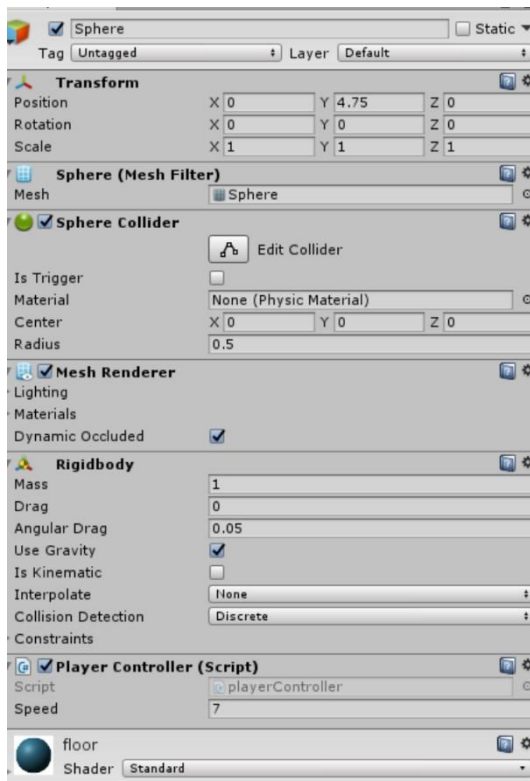
Using a trigger to change scenes

Using the same scene that we have already created, save the scene as level0, then create a new scene as below and save as level1.



The floor material was applied to the player, the door material was applied to the plane, and a simple red was applied to a new cube.

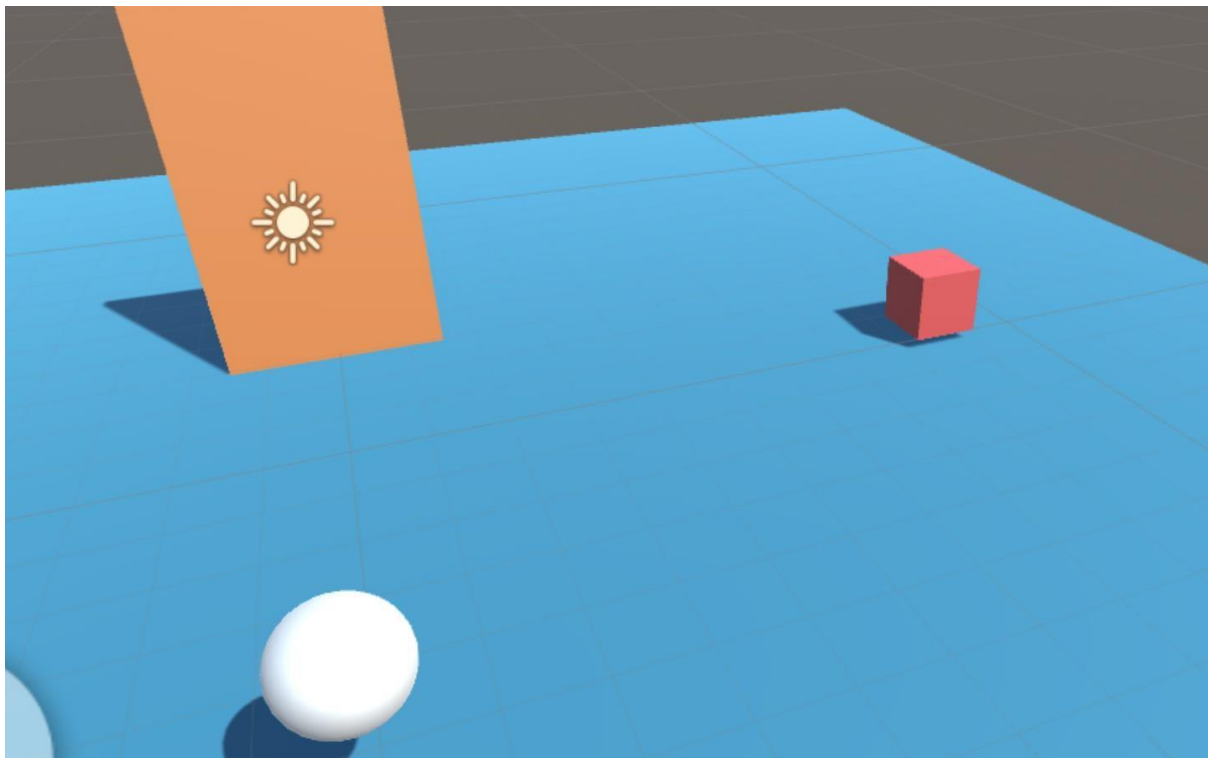
Attach the script, playerController to the player to ensure that we can move the sphere.



Save and test, if the rotation is off for the camera and the controls seem inverted, it's ok, this is just for testing purposes.

Save this scene and go back to level0.

On level0, create a new cube. And position it like so.

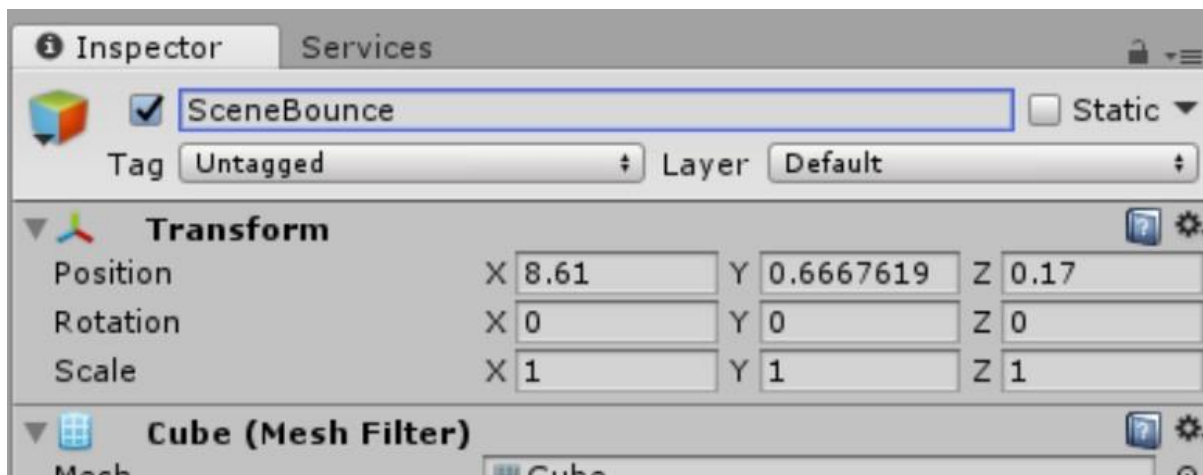


The goal is to assign a collider, just like with the door, except instead of moving the object we will jump between scenes.

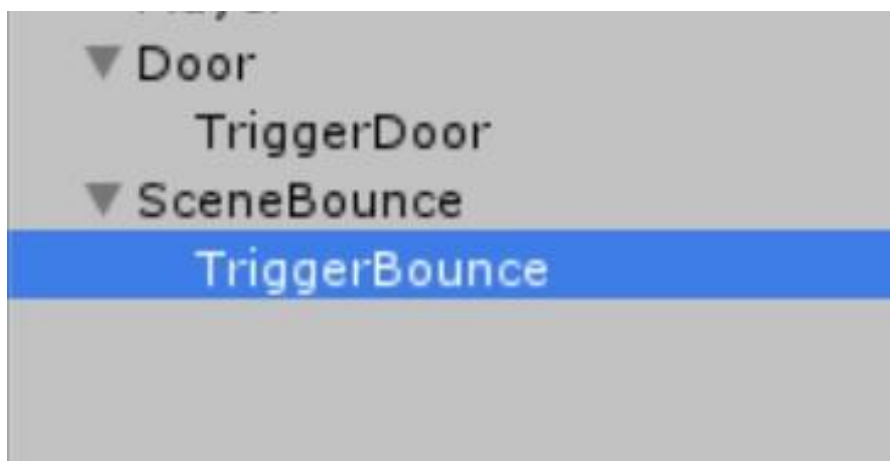
To start with add both scenes to the build settings.



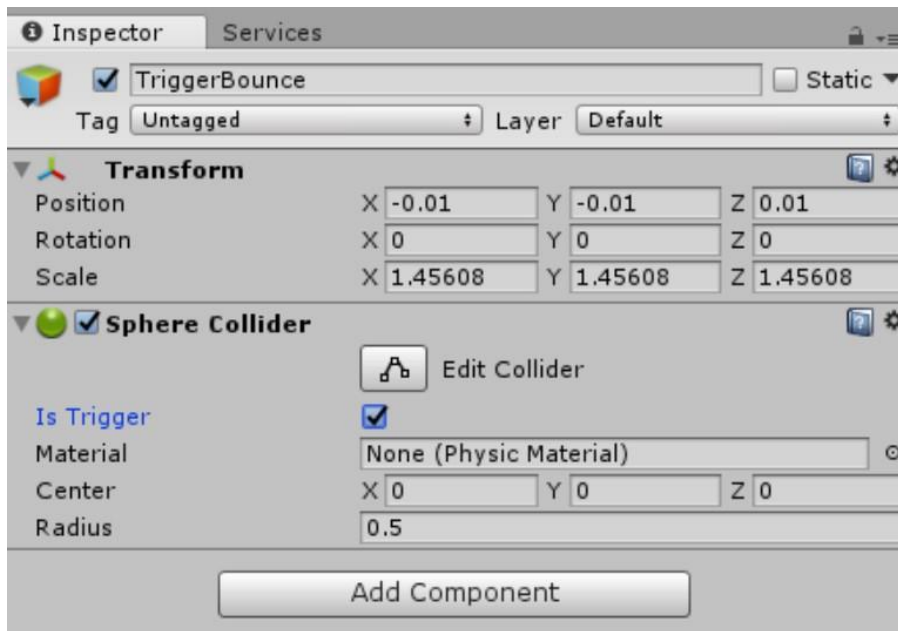
Now, let's create a new Empty Game object as we did before, name it TriggerBounce and place it on the cube, then and add a sphere collider to the object. Rename the cube to SceneBounce.



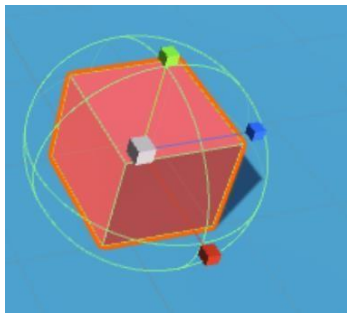
Drag the TriggerBounce onto SceneBounce to make it a child.



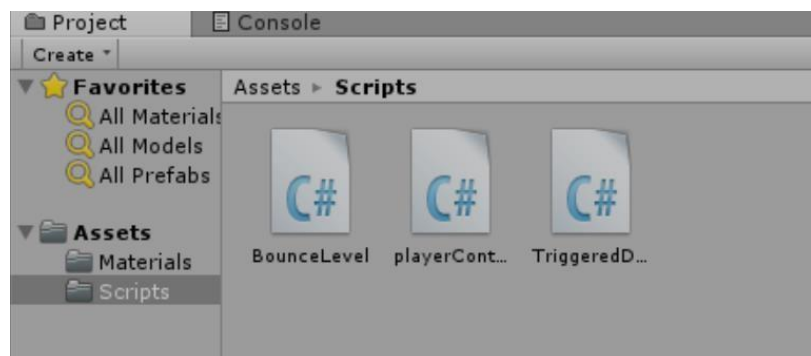
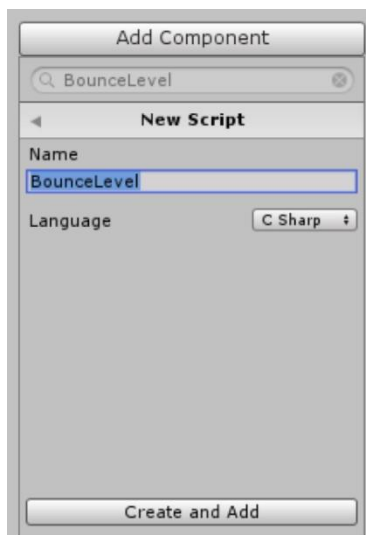
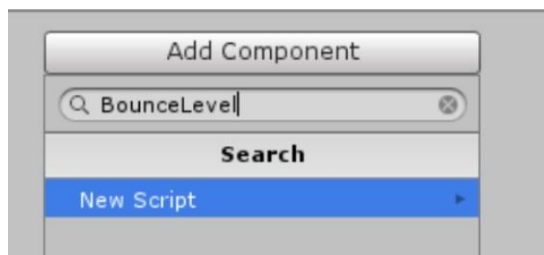
Ensure you have ticked Is Trigger on TriggerBounce



Ensure the sphere collider encapsulates the cube.



On the SceneBounce add a new script called BounceLevel



Once you have done this, move the script into the script folder and then open it up in Visual Studio.

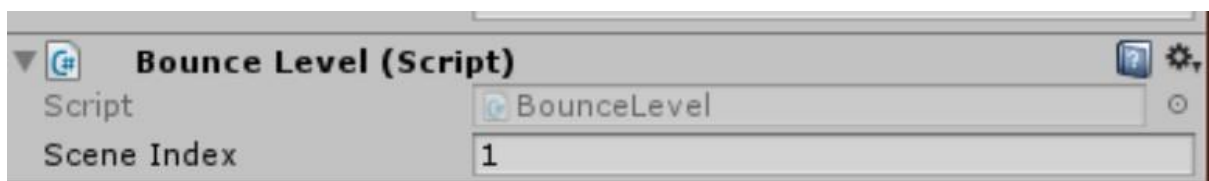
Add the following code.

```

6 public class BounceLevel : MonoBehaviour {
7
8     public int sceneIndex;
9
10    private void OnTriggerEnter(Collider other)
11    {
12        if (other.tag == "Player")
13        {
14            SceneManager.LoadScene(sceneIndex);
15        }
16    }
17 }
18

```

Back in Unity, make a change to the inspector to ensure that we are sending the correct correct scene index to change to.



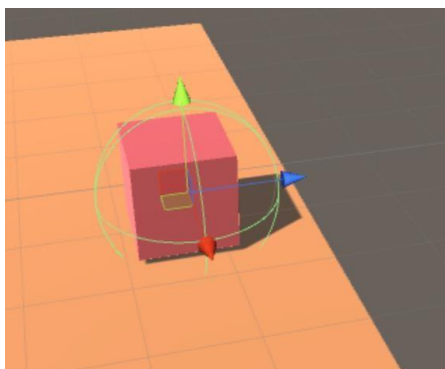
Save and Test, you should be able to walk the player into the cube and bounce to the next scene.

Repeat the process on level1 for the other cube.

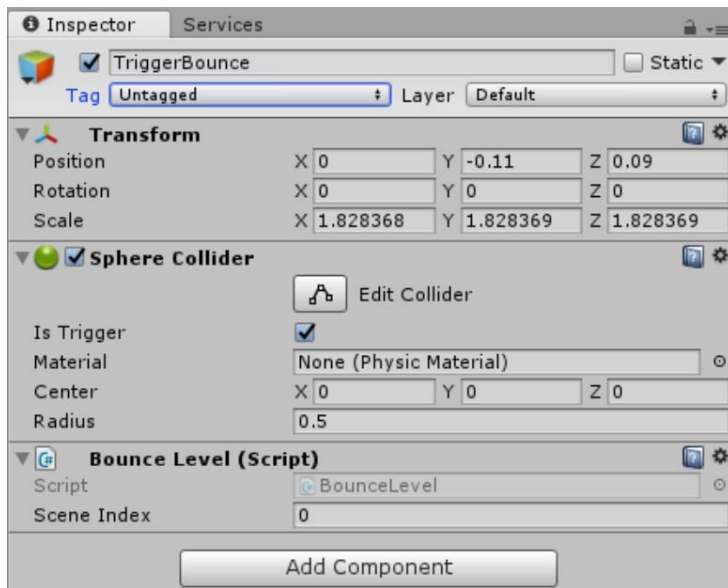
Steps are:

- Empty GameObject
- Apply Sphere collider
- Ensure is trigger is checked
- Link BounceLevel Script
- Ensure the correct Scene Index is being applied.

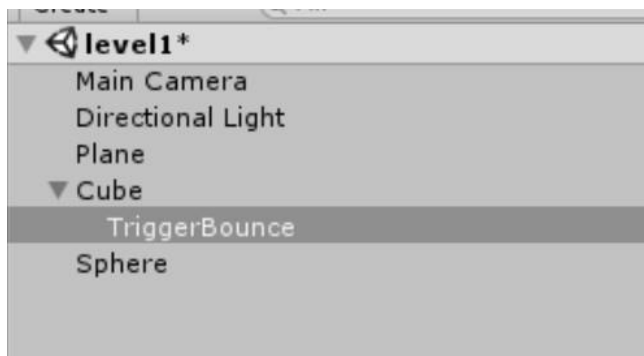
In the scene



In the Inspector



In the hierarchy

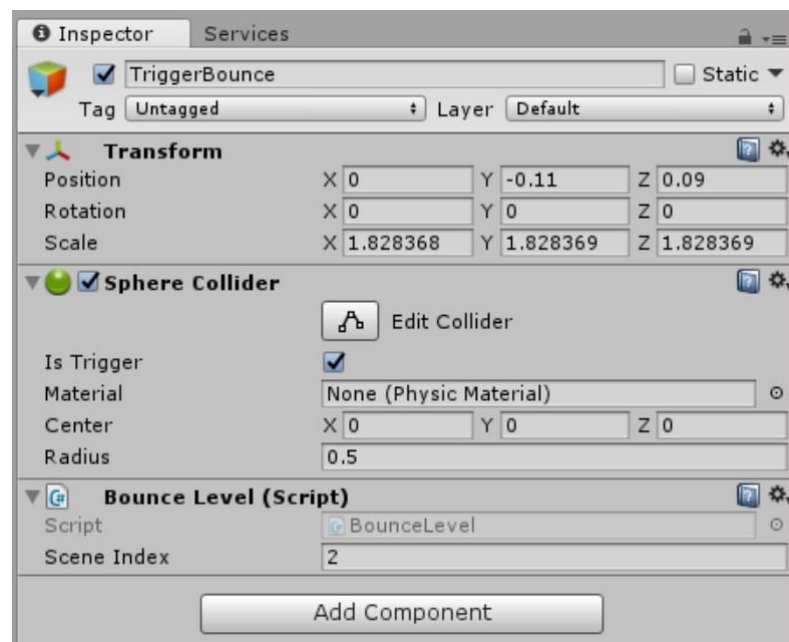


If it doesn't work, ensure that the sphere is tagged as Player.



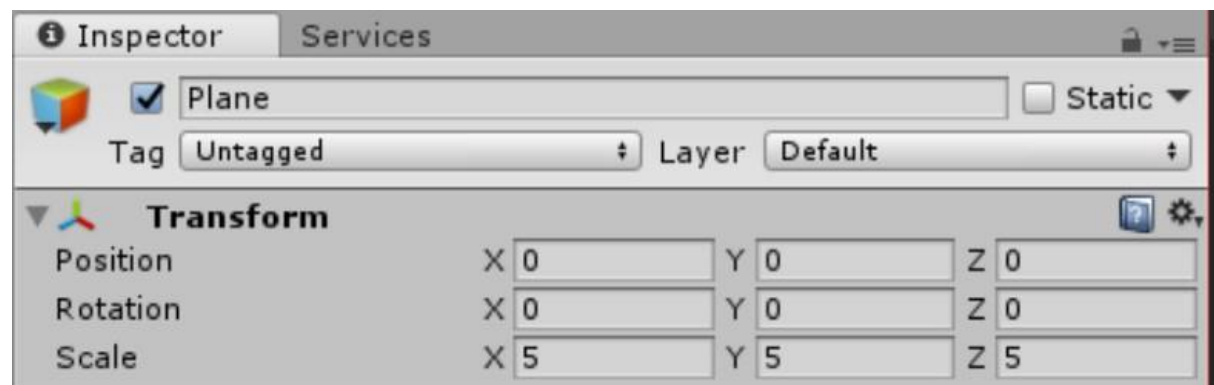
Adding simple NPCs

Using the same project, create a new scene and save it as level 2, then add it to the build settings and change the bounce level from 0 to 2 in the level1 scene. This way we can move from level 0->1>2

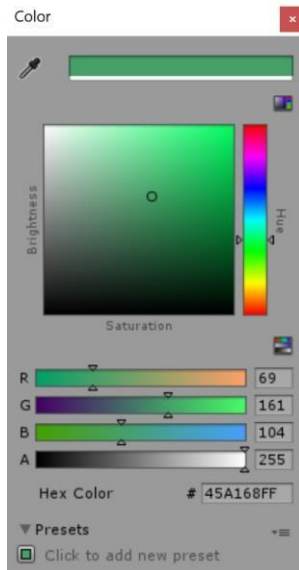


On level2 do the following:

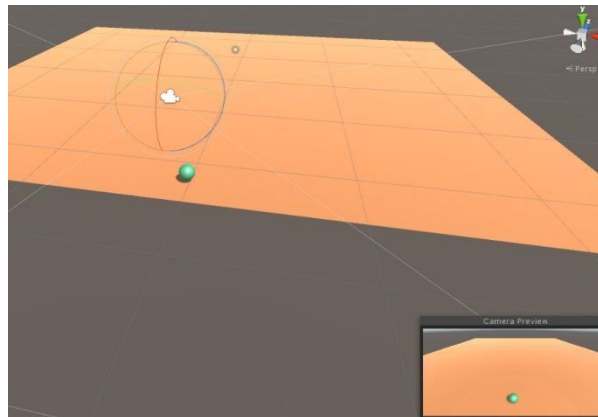
Add a plane, scale it up 5 and use the same door material on it.



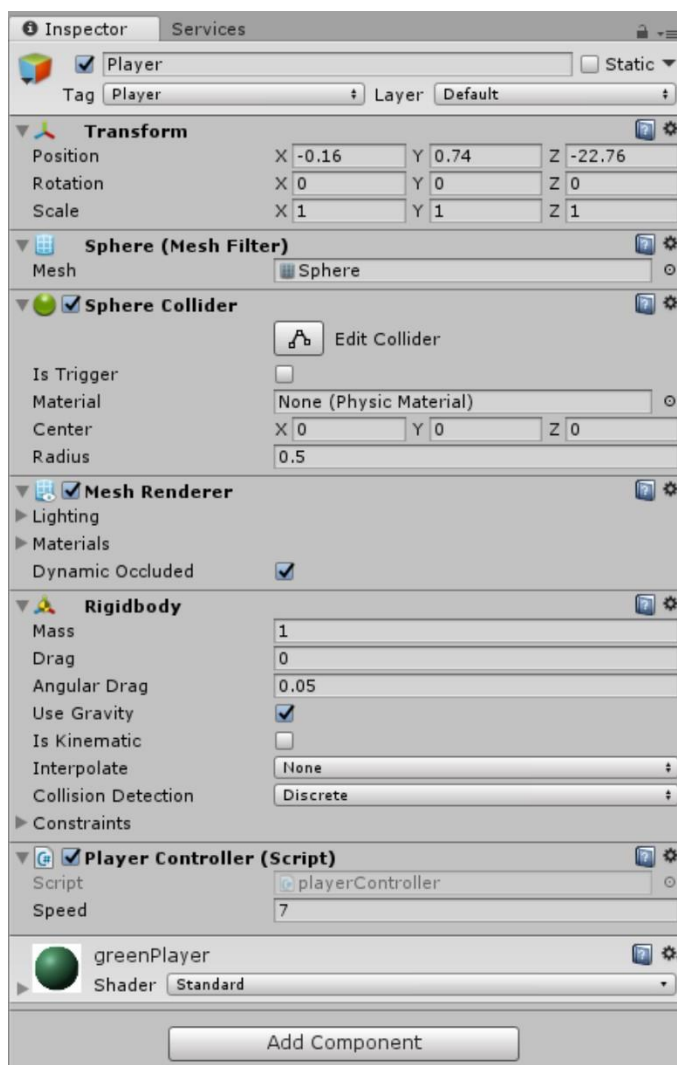
Create sphere, assign it a new green Material.



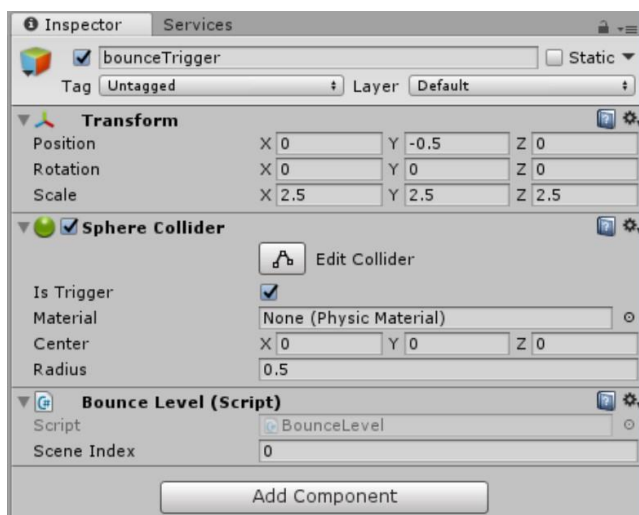
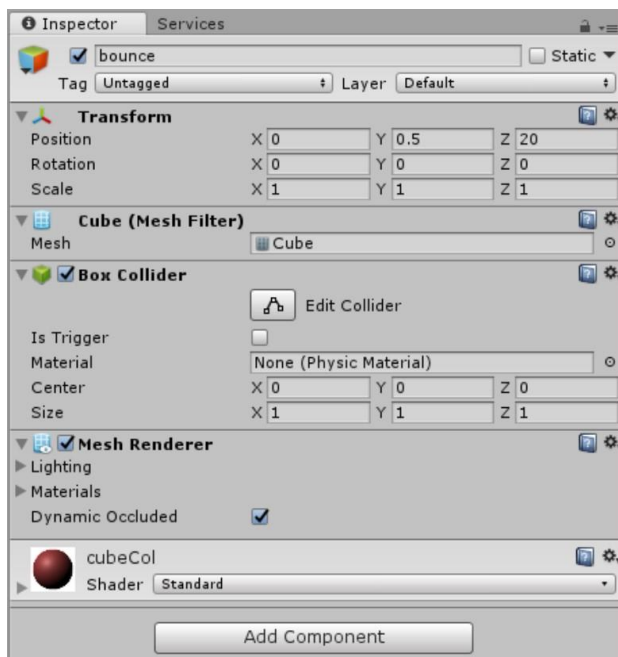
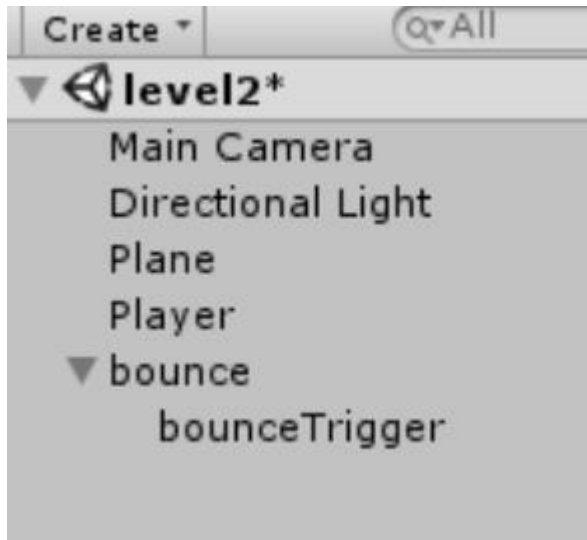
Position the camera above and behind the player so we can see what is going to occur.



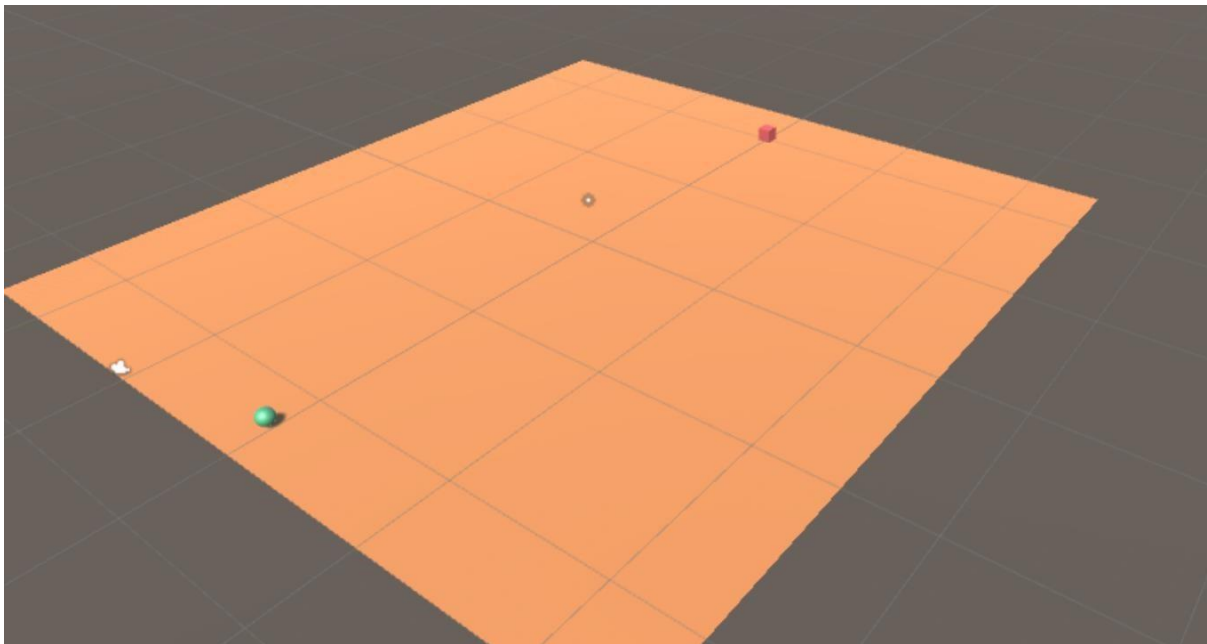
Apply a rigid body and movement script to the player and ensure the sphere is tagged as player.



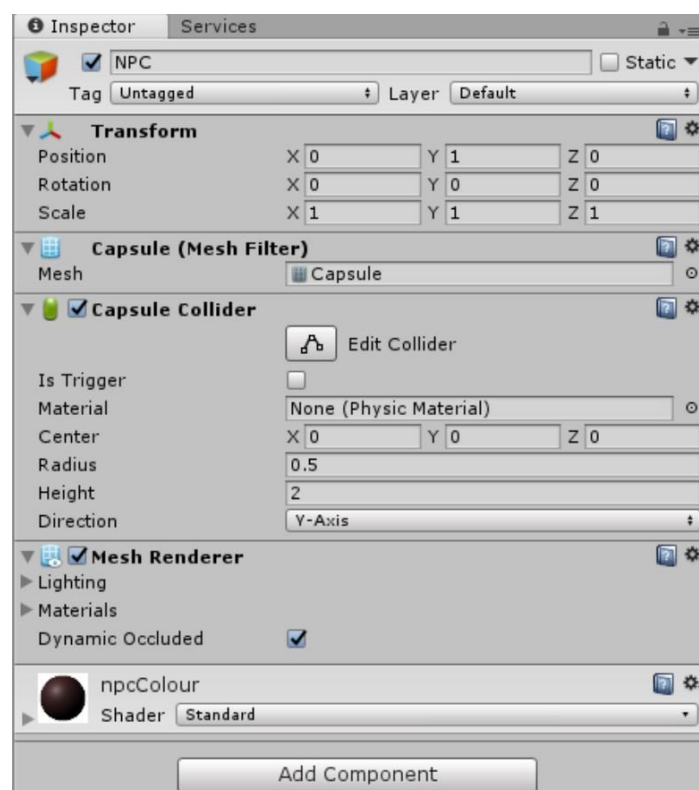
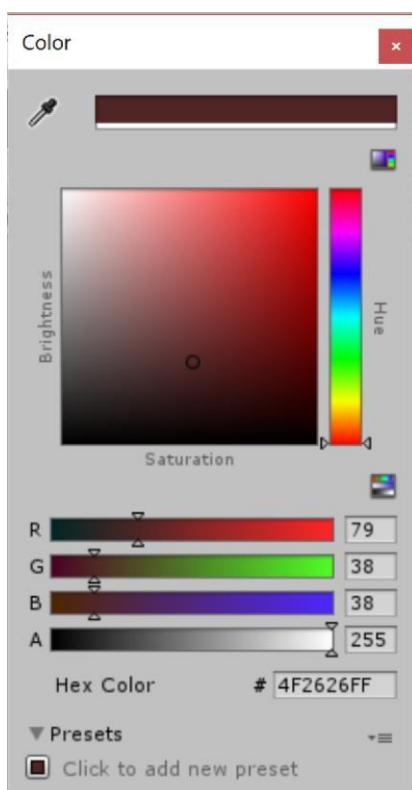
Next create a cube, call it bounce and apply an empty game object to it, apply the scripts to make it so if the player runs into it, it will bounce them to level0. Exactly what we have already done so far.



You have the following scene

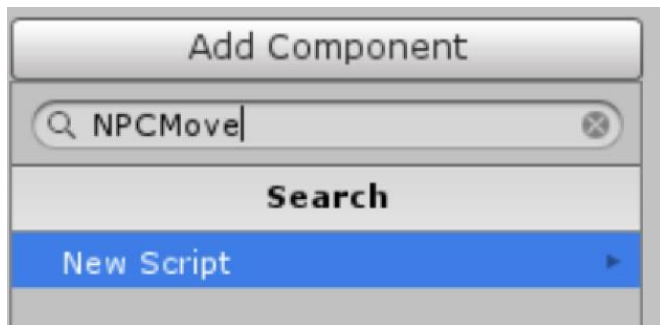


Next, we add a NPC, as we are not bringing in fully fleshed out characters for this, create a capsule, name it NPC and apply a new colour to it.



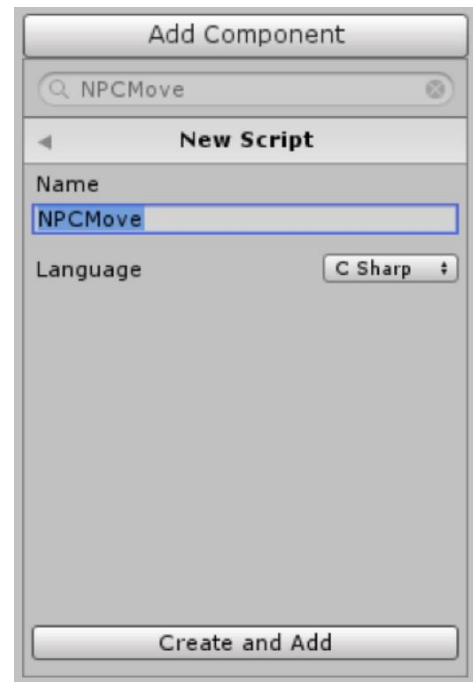
Our NPC will go from side to side and if he collides with the player, the player will be sent back to the start scene.

So, the first thing we need to do, is to create a script in which the NPC can move.



Move it into the scripts folder.

Add the following code



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class NPCMove : MonoBehaviour {
6
7      public Transform ObjectToMove;
8      public Vector3 rightPosition = new Vector3(23.0f, 1.0f, 0.0f);
9      public Vector3 leftPosition = new Vector3(-23.0f, 1.0f, 0.0f);
10
11     public float npcSpeed = 50.0f;
12     private bool moveLeft = true;
13     private bool moveRight = false;
14
15     // Update is called once per frame
16     void Update () {
17         if (moveLeft)
18         {
19             MoveLeftNPC();
20         }
21         else
22         {
23             MoveRightNPC();
24         }
25     }
26 }
```

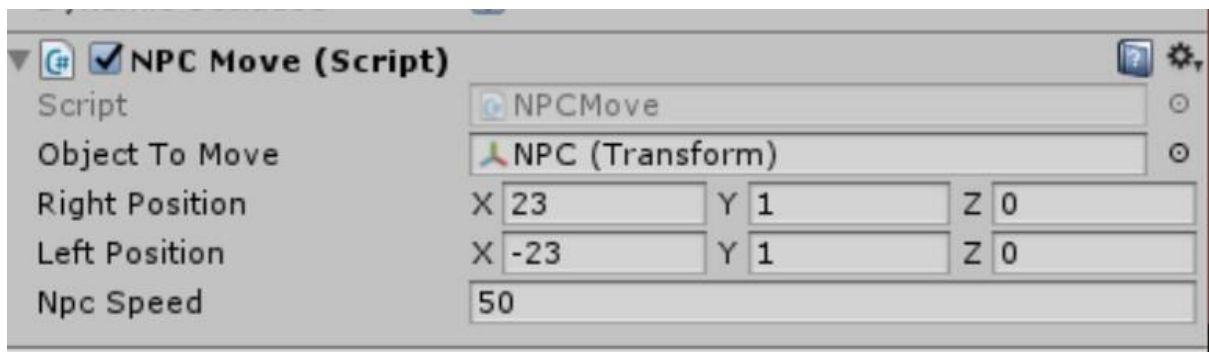
```

26
27 void MoveLeftNPC()
28 {
29     ObjectToMove.position = Vector3.Lerp(leftPosition, ObjectToMove.position, Time.deltaTime * npcSpeed);
30     if(ObjectToMove.position == leftPosition)
31     {
32         moveLeft = false;
33         moveRight = true;
34     }
35 }
36 void MoveRightNPC()
37 {
38     ObjectToMove.position = Vector3.Lerp(rightPosition, ObjectToMove.position, Time.deltaTime * npcSpeed);
39     if (ObjectToMove.position == rightPosition)
40     {
41         moveRight = false;
42         moveLeft = true;
43     }
44 }
45

```

Once done, save and then go back to unity.

In the inspector, assign the NPC object to the Object To Move section of the script and test. You might need to change the speed value.



You should have the capsule moving from side to side.

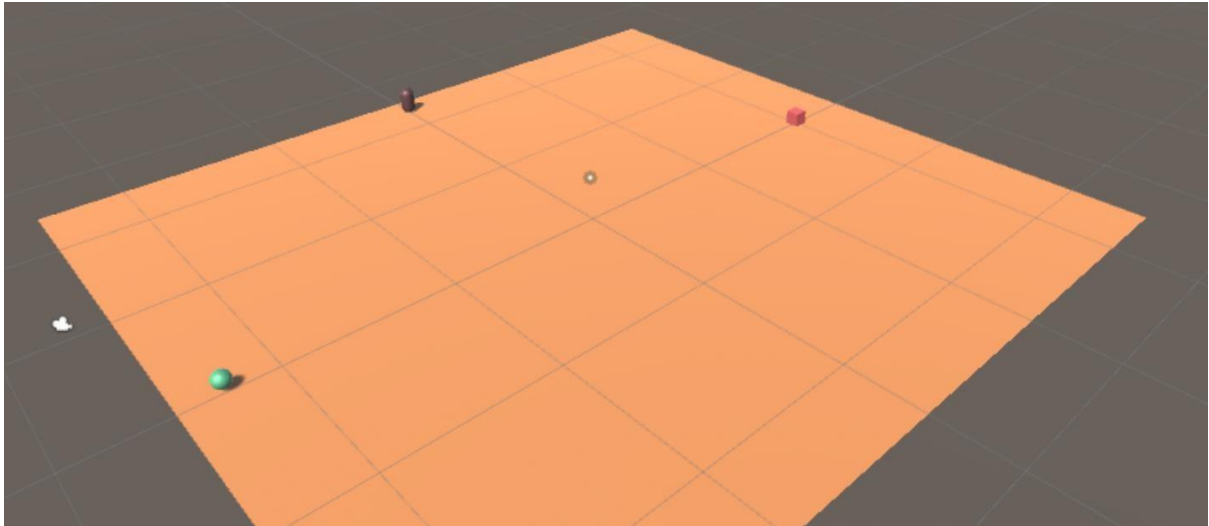
Next, apply the collider to the NPC.

Steps are:

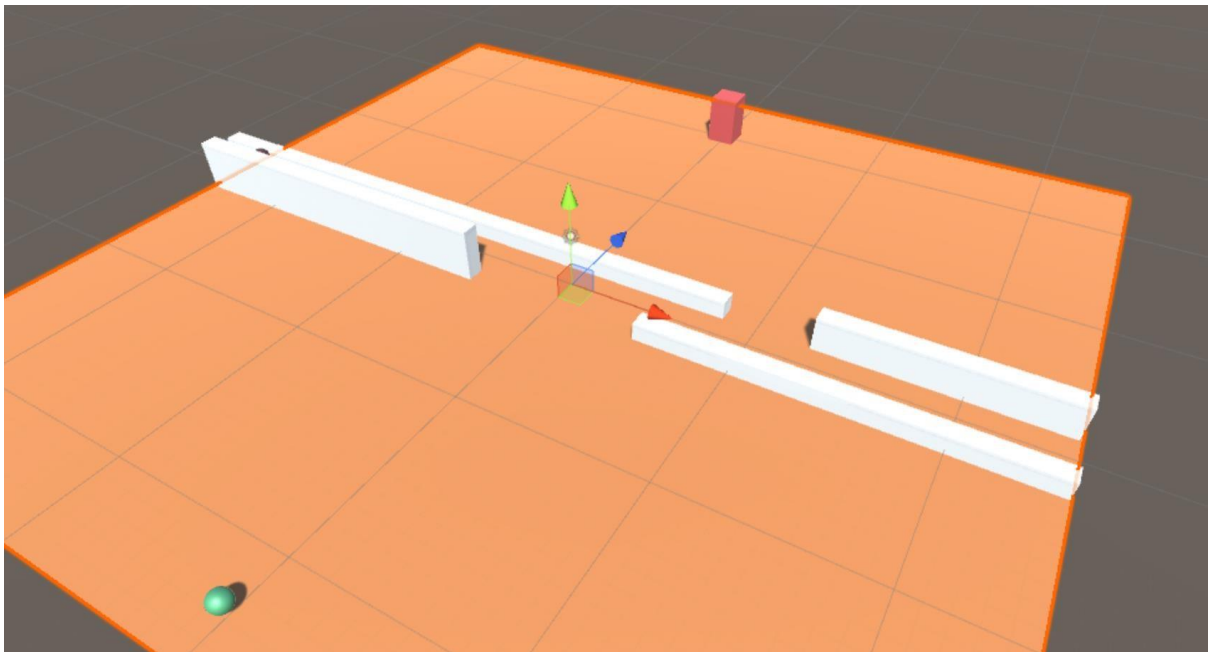
- Empty GameObject
- Apply Sphere collider
- Ensure is trigger is checked
- Link BounceLevel Script
- Ensure the correct Scene Index is being applied.

Run the program, you should be forced back to the main stage if you hit the npc, on either side of the board. You can modify the code, to make the npc move up and down the board as desired, but in this case, just to make it a little more challenging, we'll add some cube walls.

Before walls



After Walls



Play with the speed, the size of the colliders and so forth, this simple layout can stop the user from reaching the final bounce.

Which of course, you could have as taking the player to a new scene which congratulates them on getting through the levels.