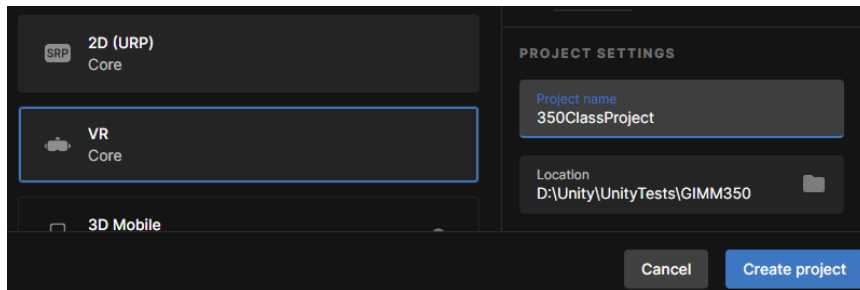


VR Class Project – Part 1

Create a new project.

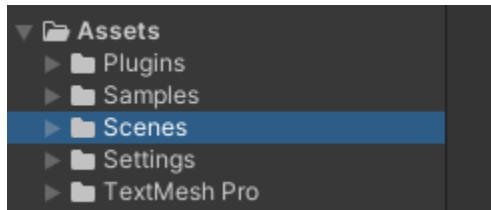
1. Create new VR Core project and name it 350ClassProject.



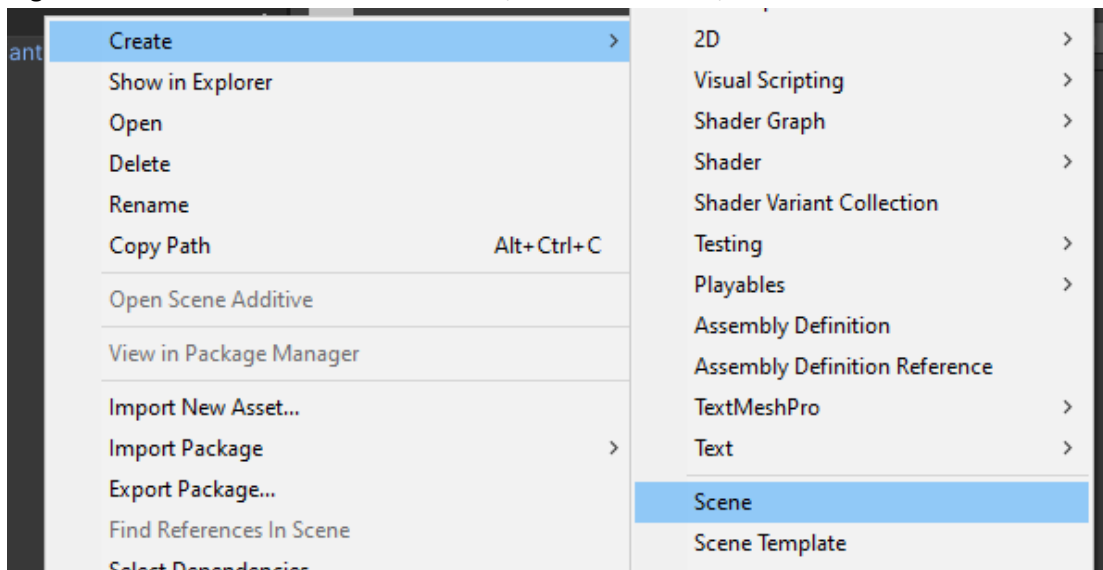
2. Complete Unity Setup for VR from the “Setting Up Unity for VR Development” document.

Creating a new scene

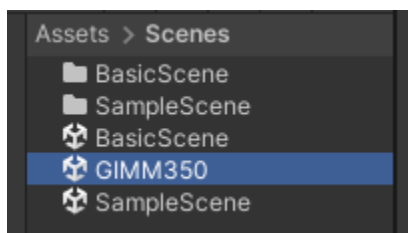
1. Double click on the Scenes folder under Assets



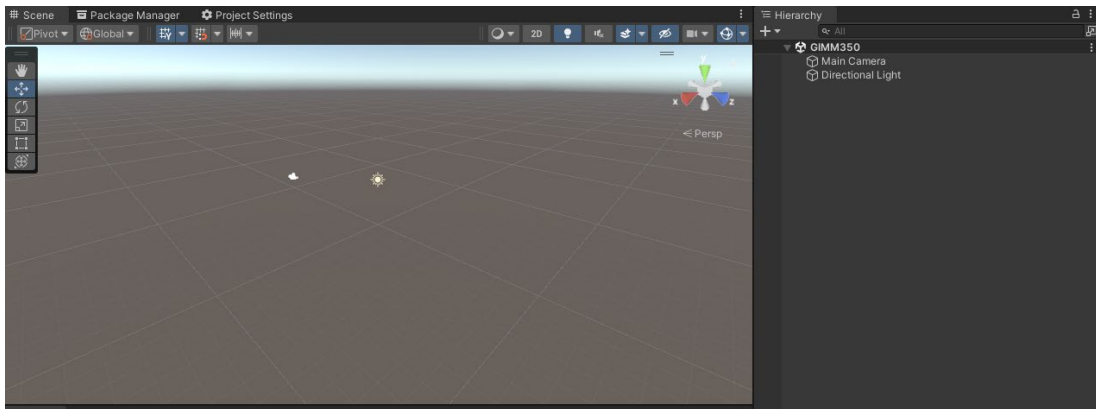
2. Right click with cursor in the Scenes folder, then click Create, then Scene.



3. Rename the “New Scene” GIMM350

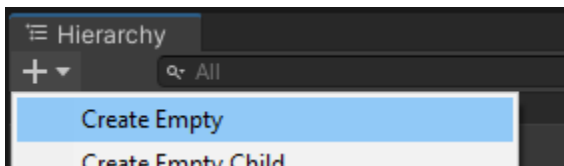


4. Double click on the GIMM350 scene to open a fresh scene.

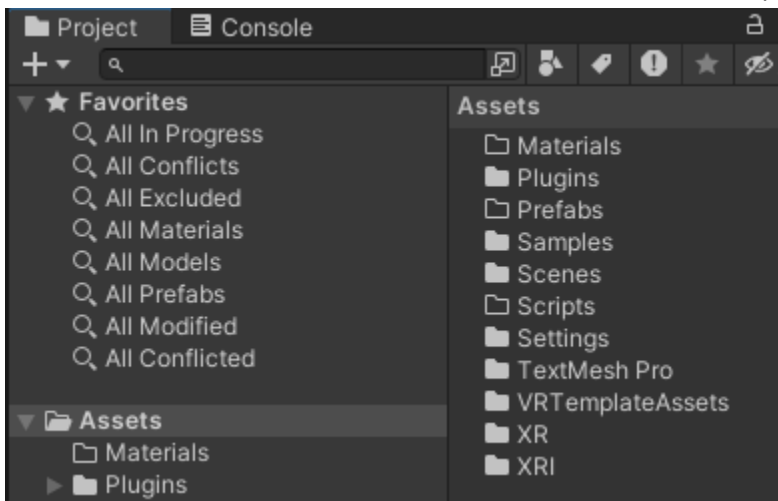


Building a new scene

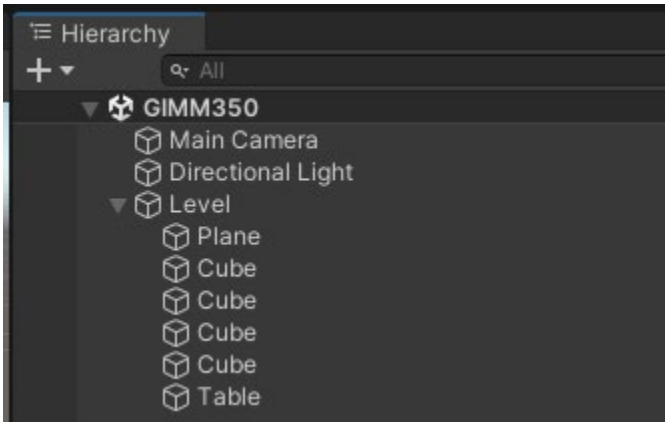
1. Create a new empty game object and rename it Level. Click on the plus sign with dropdown arrow on the Hierarchy and select Create Empty.



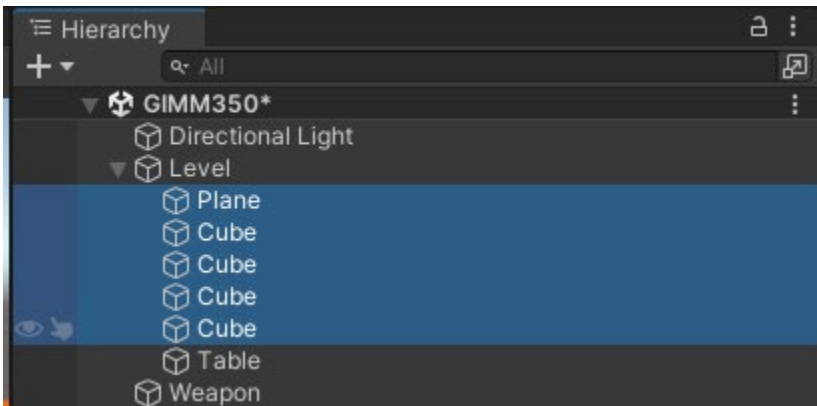
2. Create three new Assets subfolders: Materials, Prefabs, and Scripts.



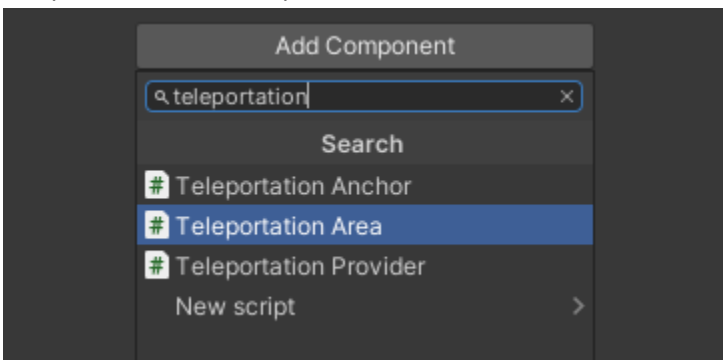
3. Create the following 3D objects as children of Level:
Plane (Position 0, 0, 0 | Rotation 0, 0, 0 | Scale 2, 1, 2)
Cube (Position -20, 3, 0 | Rotation 0, 0, 0 | Scale 8, 0.1, 8)
Cube (Position 20, 3, 0 | Rotation 0, 0, 0 | Scale 8, 0.1, 8)
Cube (Position 0, 1, -20 | Rotation 0, 0, 0 | Scale 8, 0.1, 8)
Cube (Position 0, 1, 20 | Rotation 0, 0, 0 | Scale 8, 0.1, 8)
Cube (Position 0, 1.55, -20 | Rotation 0, 0, 0 | Scale 1, 1, 1) – Rename to Table



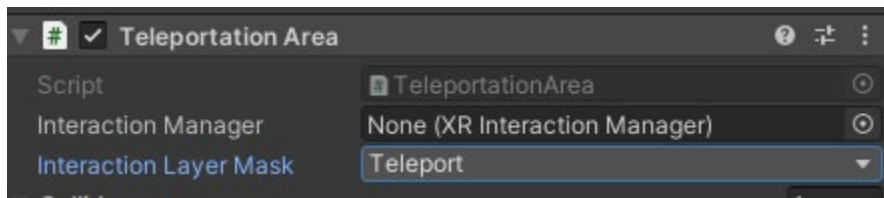
4. Click on Plane and then press shift and click on the bottom cube to select all of these GameObjects in the Hierarchy.



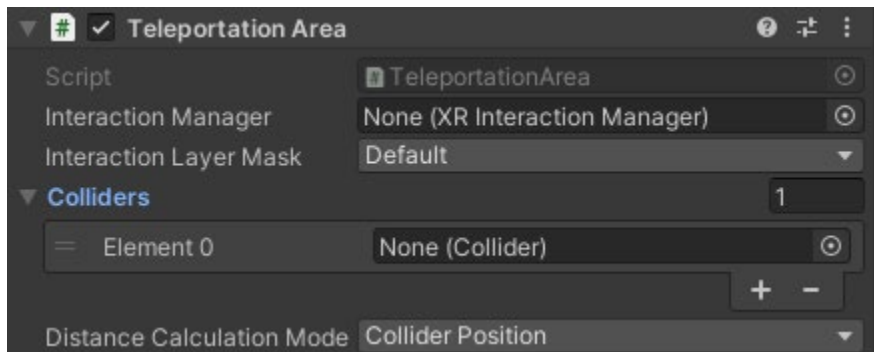
5. Then click on Add Component. Type "teleportation" into the search bar and click on the Teleportation Area component to add it to each of the selected GameObjects.



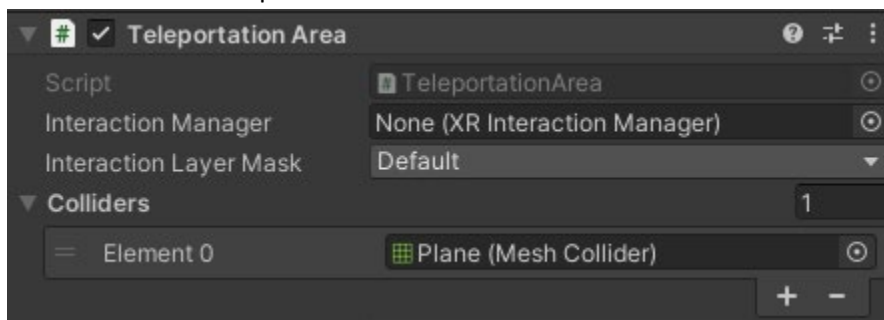
6. With the Plane and four Cube GameObjects still selected, set the Interaction Layer Mask to Teleport.



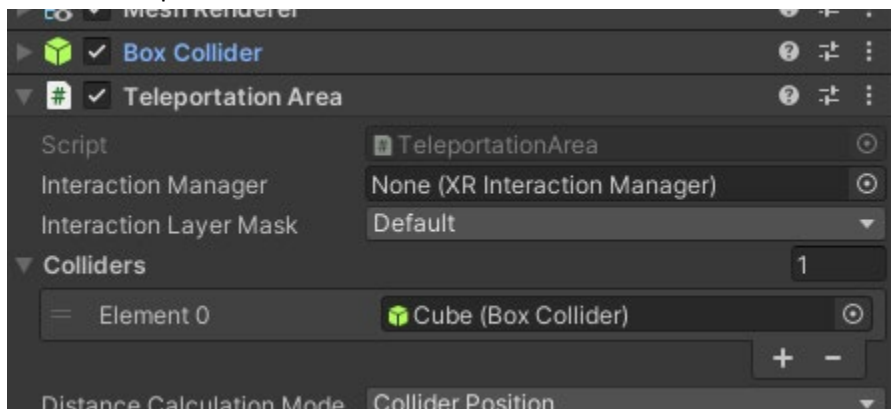
7. With the Plane and four Cube GameObjects still selected, in the new Teleportation Area component change then number of Colliders from 0 to 1. Then click on the triangle next to Colliders to open the variable.



8. Click on the Plane GameObject. Then in the Inspector, drag the Mesh Collider component into Element 0 of the Teleportation Area Colliders.

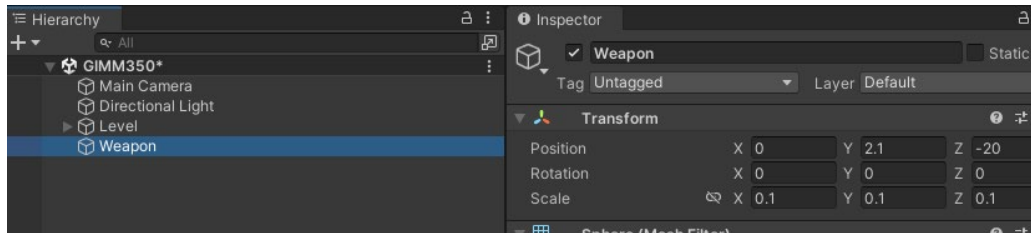


9. Do the same thing for each of the Cubes, however instead of the Mesh Collider, drag in the Box Collider Component.

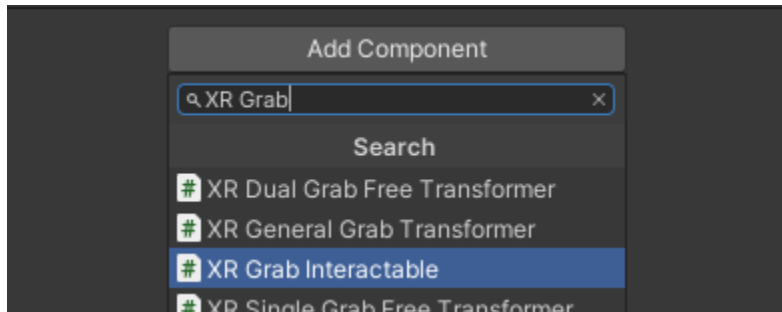


10. Add a Sphere to the Hierarchy.

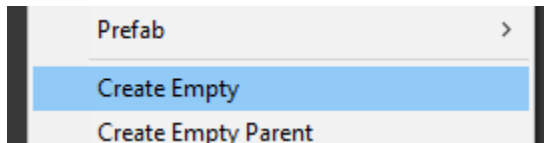
Rename it to Weapon (Position 0, 2.1, -19.65 | Rotation 0, 180, 0 | Scale 0.1, 0.1, 0.1)



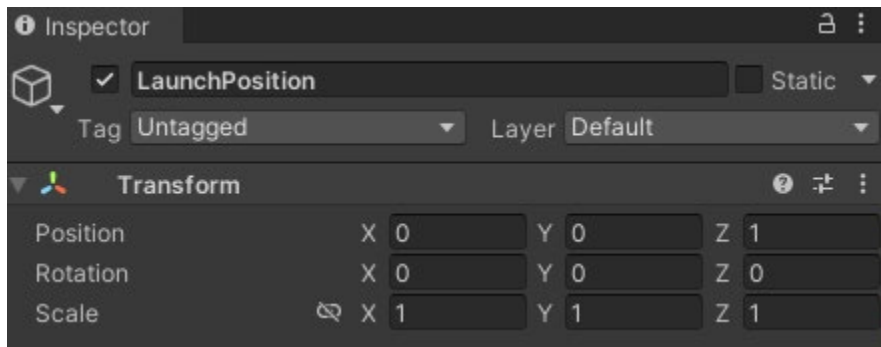
11. Click on the Weapon GameObject, then click on Add Component and in the search box type “XR Grab” and click on XR Grab Interactable to add it to the Weapon’s Inspector.



12. Right click on the Weapon GameObject and select Create Empty.

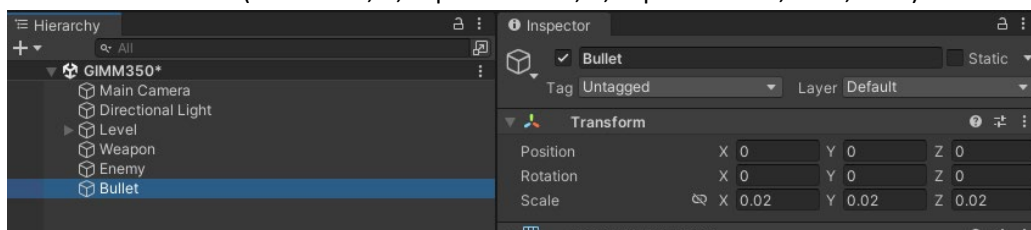


13. Rename the new Empty GameObject to LaunchPosition, and set the transforms as follows:

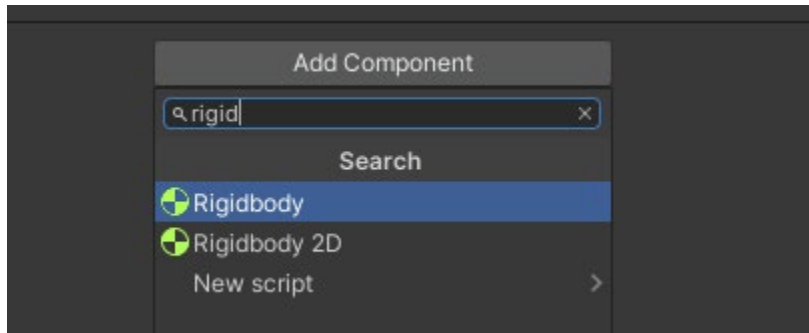


14. Add a Sphere to the Hierarchy.

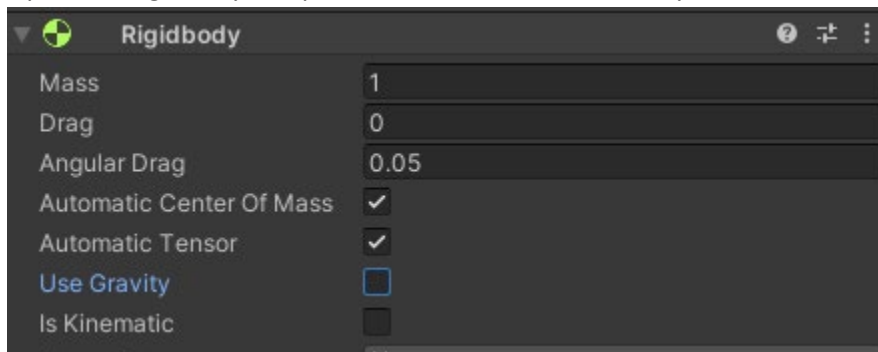
Rename it to Bullet (Position 0, 0, 0 | Rotation 0, 0, 0 | Scale 0.02, 0.02, 0.02)



15. Add a Rigidbody component to the Bullet GameObject. Click on Add Component, type in “rigid,” and click on Rigidbody to add it to the Bullet’s Inspector.

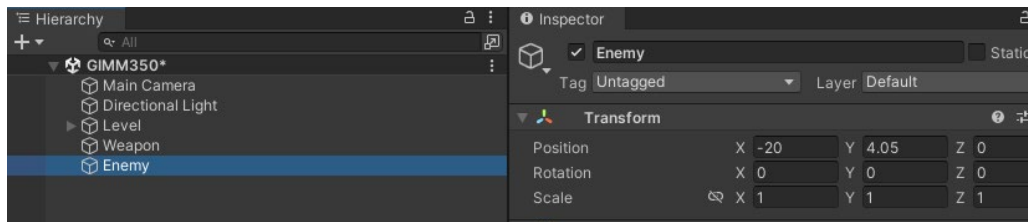


16. Open the Rigidbody component and deselect Use Gravity.



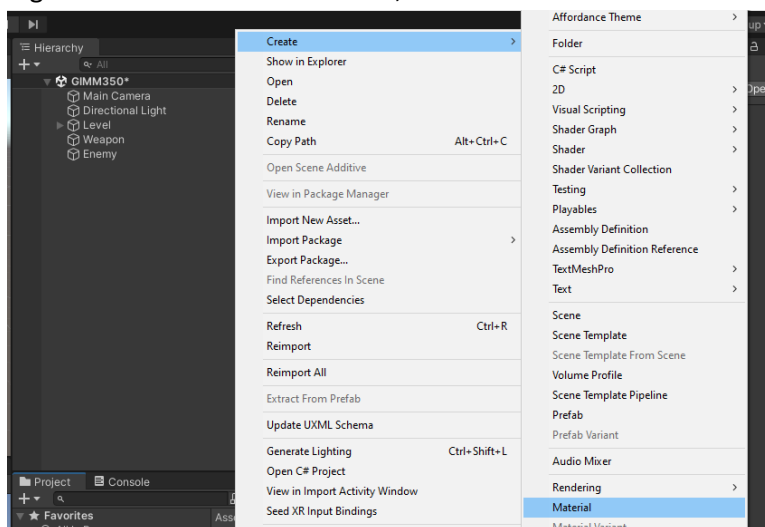
17. Add a Cylinder to the Hierarchy.

Rename it to Enemy (Position -20, 4.05, 0 | Rotation 0, 0, 0 | Scale 1, 1, 1)

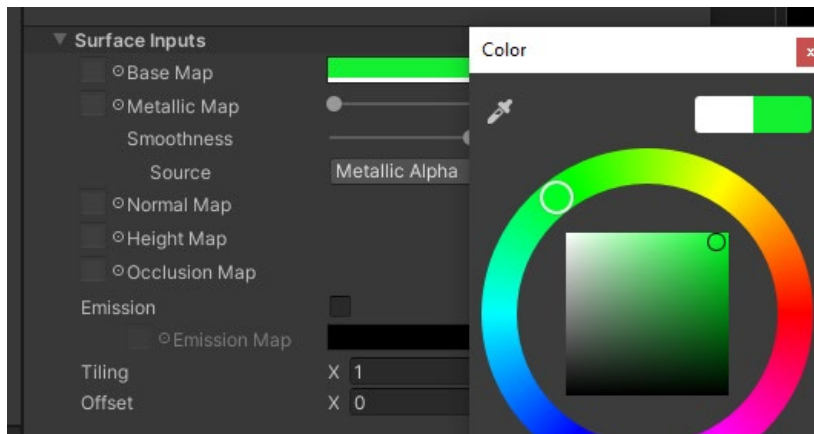


18. Create a new material within the Materials folder:

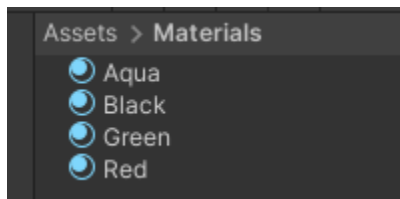
Right click in the Materials folder, click on Create -> Material



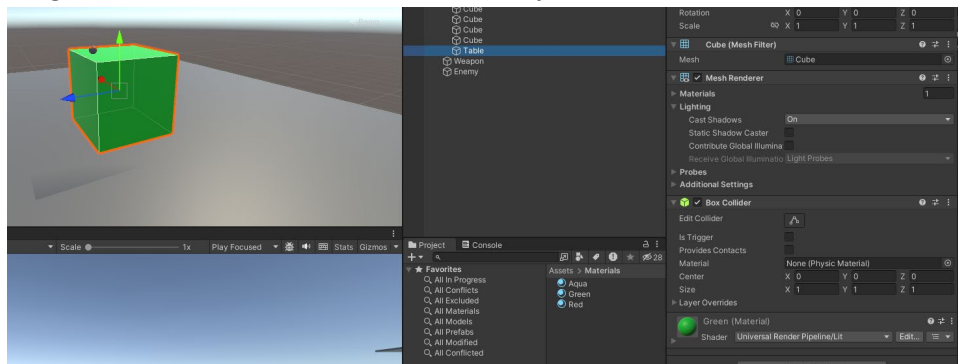
19. Rename to new material to Green. Click on the white color bar next to Base Map under Surface Inputs, then click in the green area of the outer ring and then the green area of the inner square.



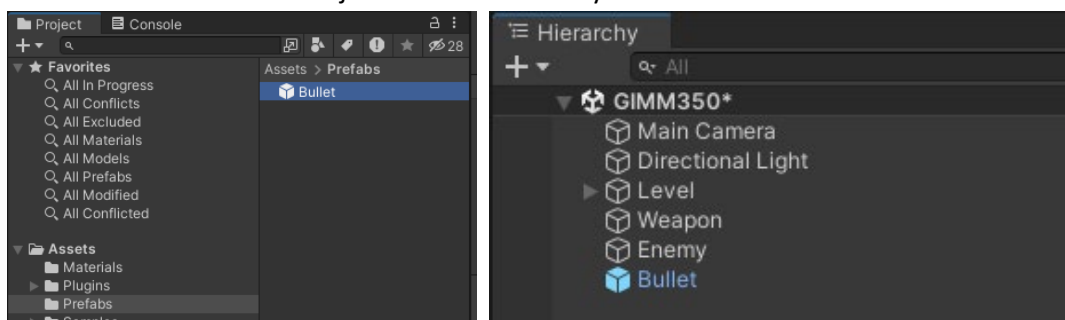
20. Do the same three more times, creating Aqua, Black, and Red materials



21. Drag the Green material onto Table GameObject.



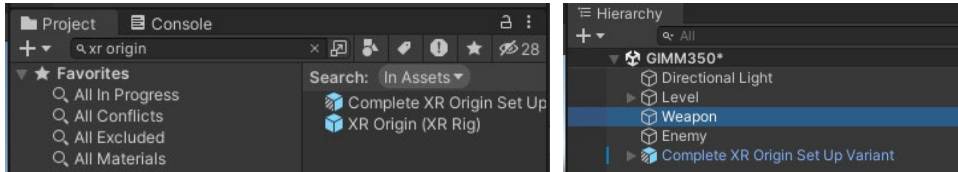
22. Drag the Red material onto the Weapon GameObject.
23. Drag the Black material onto the Enemy GameObject.
24. Drag the Aqua material onto the Bullet GameObject.
25. Drag the Bullet GameObject into the Prefabs folder (which makes it a prefab and changes the name of the Bullet GameObject into the Hierarchy to blue).



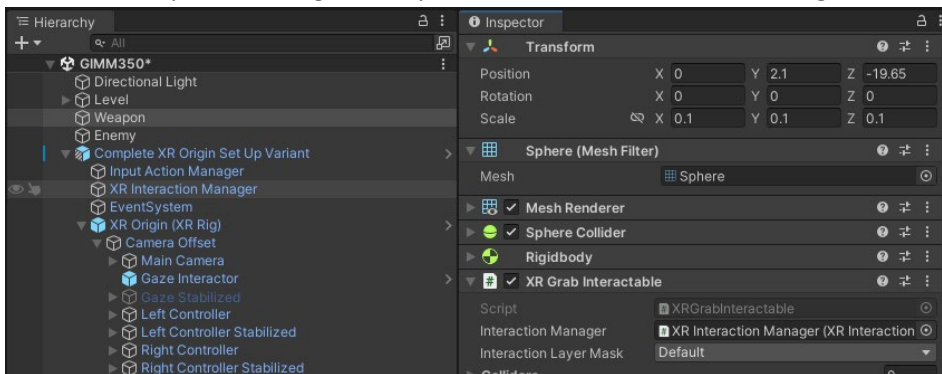
26. Delete the Bullet GameObject from the Hierarchy. (We will spawn these as needed via script)

Create the VR Player

1. Now that the scene is all set up, it's time to add the player.
2. Remove the Main Camera from the Hierarchy.
3. In the Project search bar, type in "xr origin." Drag the Complete XR Origin Set Up Variant GameObject into the Hierarchy.



4. Open the Complete XR Origin Set Up Variant Game Object. Then click on the Weapon GameObject. In the Weapon XR Grab Interactable component, drag the XR Interaction Manager from the Complete XR Origin Set Up Variant to the Interaction Manager.

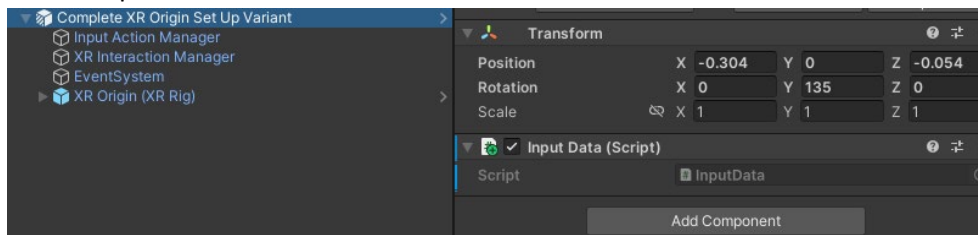


5. In the Scripts subfolder, right click in the folder, then select Create -> C# Script. Rename the script to InputData.

6. Open the InputData.cs script and write the following code:

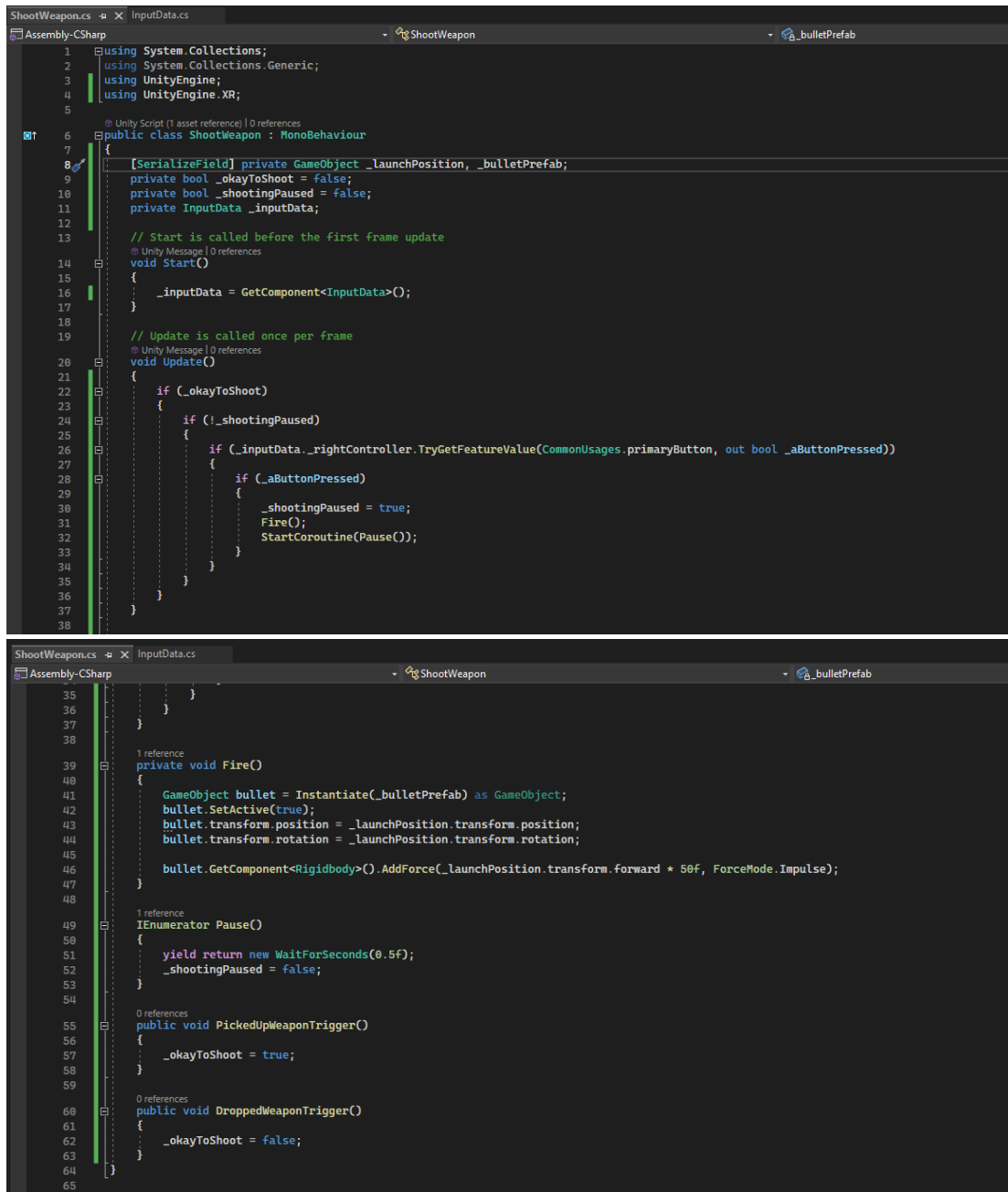
```
InputData.cs* [X]
Assembly-CSharp [X] InputData [X] Update()
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.XR;
5
6 @ Unity Script | 0 references
7 public class InputData : MonoBehaviour
8 {
9     public InputDevice _rightController;
10    public InputDevice _leftController;
11    public InputDevice _HMD;
12
13    // Update is called once per frame
14    @ Unity Message | 0 references
15    void Update()
16    {
17        if (!_rightController.isValid || !_leftController.isValid || !_HMD.isValid)
18            InitializeInputDevices();
19    }
20
21    1 reference
22    private void InitializeInputDevices()
23    {
24        if (!_rightController.isValid)
25            InitializeInputDevice(InputDeviceCharacteristics.Controller | InputDeviceCharacteristics.Right, ref _rightController);
26        if (!_leftController.isValid)
27            InitializeInputDevice(InputDeviceCharacteristics.Controller | InputDeviceCharacteristics.Left, ref _leftController);
28        if (!_HMD.isValid)
29            InitializeInputDevice(InputDeviceCharacteristics.HeadMounted, ref _HMD);
30    }
31
32    3 references
33    private void InitializeInputDevice(InputDeviceCharacteristics inputCharacteristics, ref InputDevice inputDevice)
34    {
35        List<InputDevice> devices = new List<InputDevice>();
36        InputDevices.GetDevicesWithCharacteristics(inputCharacteristics, devices);
37
38        if (devices.Count > 0)
39        {
40            inputDevice = devices[0];
41        }
42    }
43 }
```

7. Add the InputData.cs script to the Complete XR Origin Set Up Variant GameObject by dragging it to the inspector.



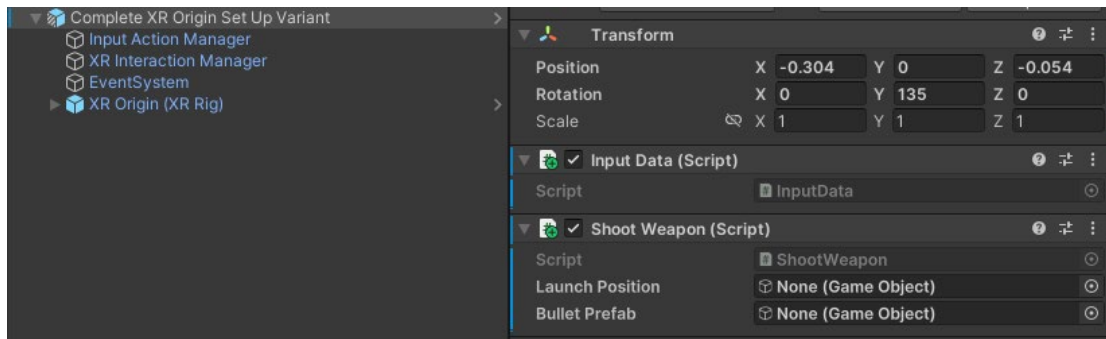
8. Create another new script in the Scripts folder and rename it to ShootWeapon.

9. Open the ShootWeapon.cs script and add the following code:

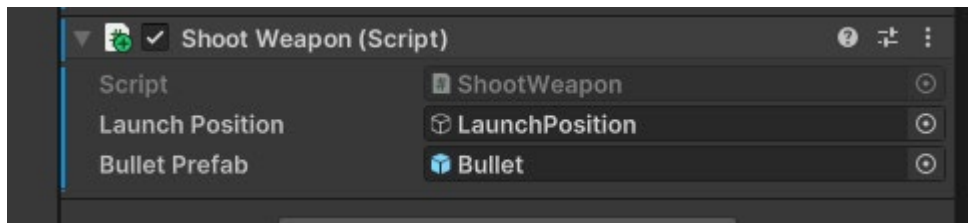


```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.XR;
5
6 public class ShootWeapon : MonoBehaviour
7 {
8     [SerializeField] private GameObject _launchPosition, _bulletPrefab;
9     private bool _okayToShoot = false;
10    private bool _shootingPaused = false;
11    private InputData _inputData;
12
13    // Start is called before the first frame update
14    void Start()
15    {
16        _inputData = GetComponent<InputData>();
17    }
18
19    // Update is called once per frame
20    void Update()
21    {
22        if (_okayToShoot)
23        {
24            if (!_shootingPaused)
25            {
26                if (_inputData._rightController.TryGetFeatureValue(CommonUsages.primaryButton, out bool _aButtonPressed))
27                {
28                    if (_aButtonPressed)
29                    {
30                        _shootingPaused = true;
31                        Fire();
32                        StartCoroutine(Pause());
33                    }
34                }
35            }
36        }
37    }
38
39    private void Fire()
40    {
41        GameObject bullet = Instantiate(_bulletPrefab) as GameObject;
42        bullet.SetActive(true);
43        bullet.transform.position = _launchPosition.transform.position;
44        bullet.transform.rotation = _launchPosition.transform.rotation;
45
46        bullet.GetComponent<Rigidbody>().AddForce(_launchPosition.transform.forward * 50f, ForceMode.Impulse);
47    }
48
49    1 reference
50    IEnumerator Pause()
51    {
52        yield return new WaitForSeconds(0.5f);
53        _shootingPaused = false;
54    }
55
56    0 references
57    public void PickedUpWeaponTrigger()
58    {
59        _okayToShoot = true;
60    }
61
62    0 references
63    public void DroppedWeaponTrigger()
64    {
65        _okayToShoot = false;
66    }
67 }
```

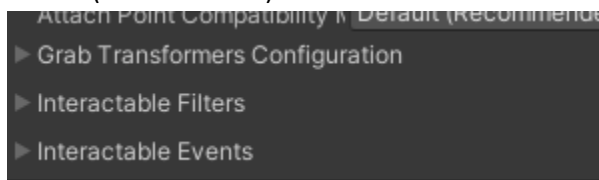
10. Add the ShootWeapon.cs script to the Complete XR Origin Set Up Variant GameObject by dragging it to the inspector.



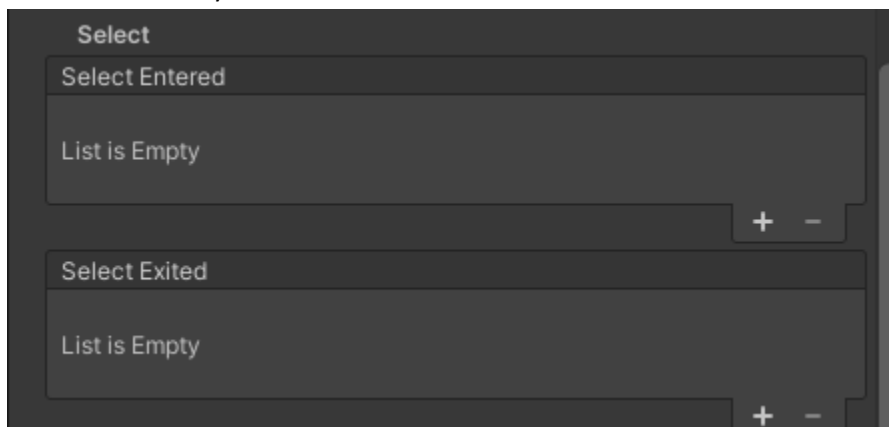
11. In the ShootingWeapons script component, drag the LaunchPosition from the Weapon GameObject to the Launch Position variable. Also drag the Bullet from the Prefabs folder to the Bullet Prefab variable.



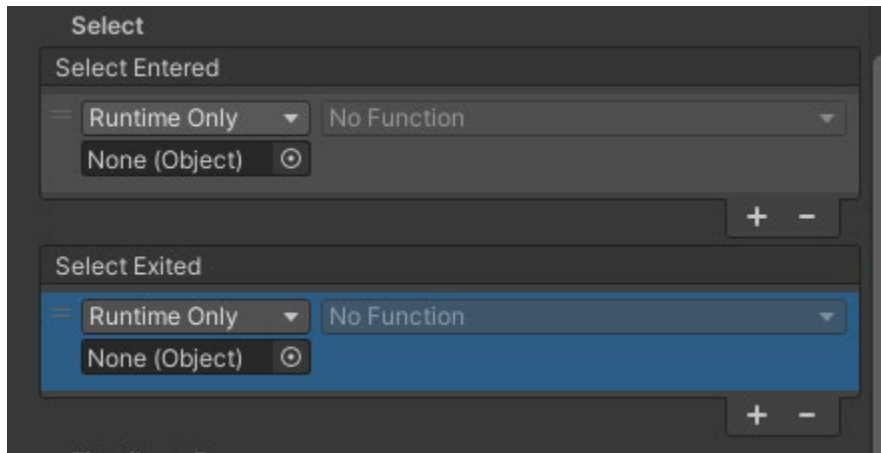
12. Click on the Weapon GameObject. In the XR Grab Interactable component, expand Interactable Events (at the bottom).



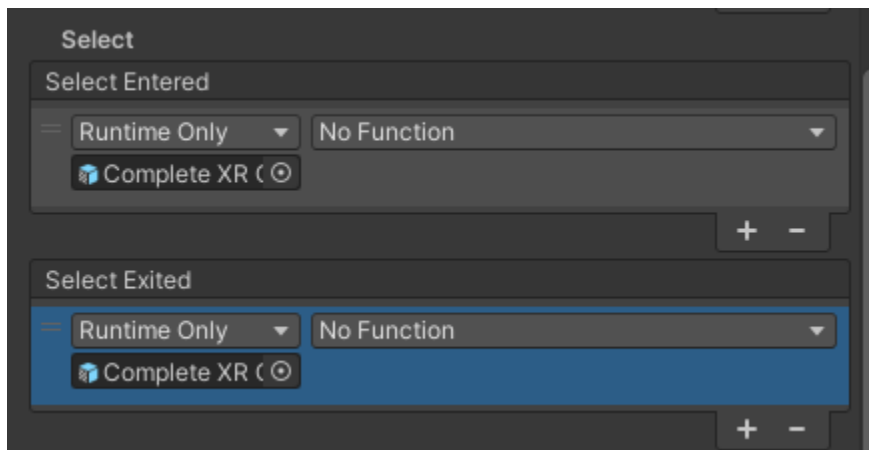
13. Scroll down until you see Select with Select Entered and Select Exited.



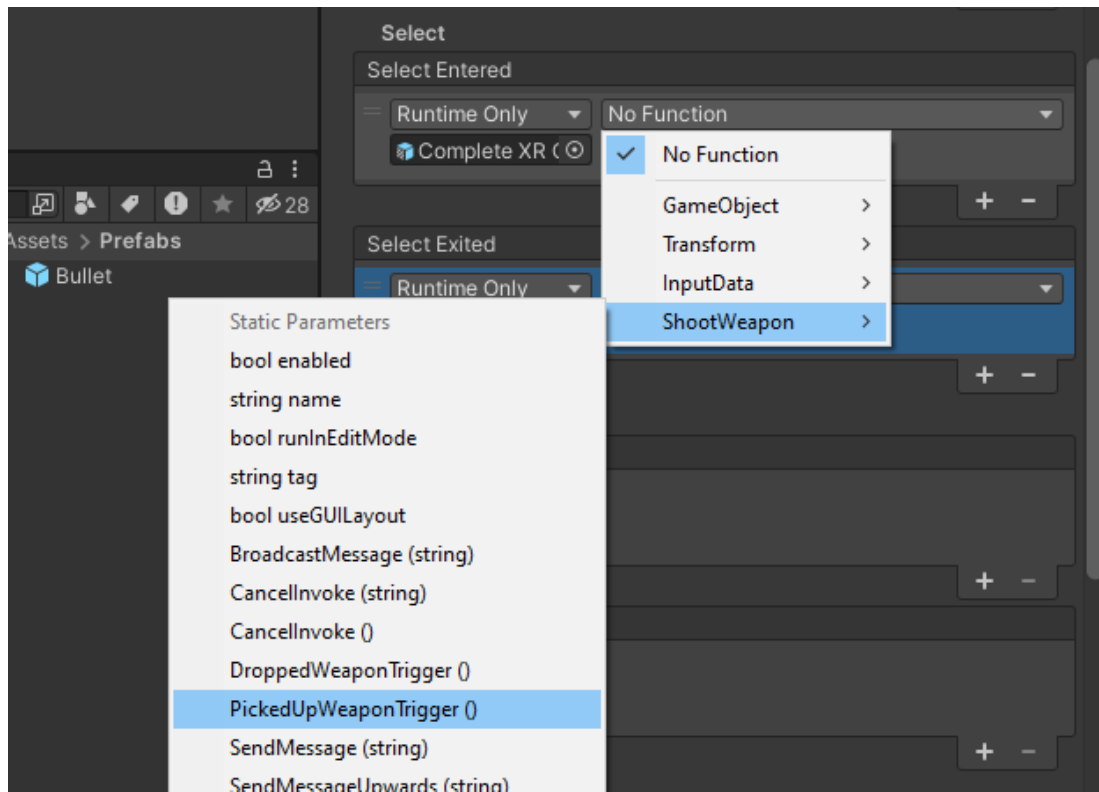
14. Click on the plus symbol on both Select Entered and Select Exited.



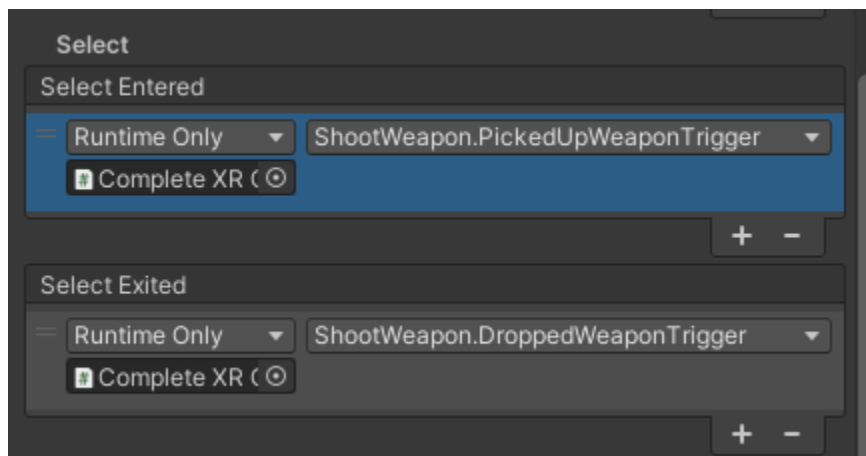
15. Drag the Complete XR Origin Set Up Variant GameObject to the “None (Object)” selectors for each.



- Click on No Function for Select Entered, then click on ShootWeapon (which calls to functions in the ShootWeapon.cs script) and then click on the PickedUpWeaponTrigger() function to select it.



- For Select Exited, do the same thing except choose DroppedWeaponTrigger(). The two should now look like this.

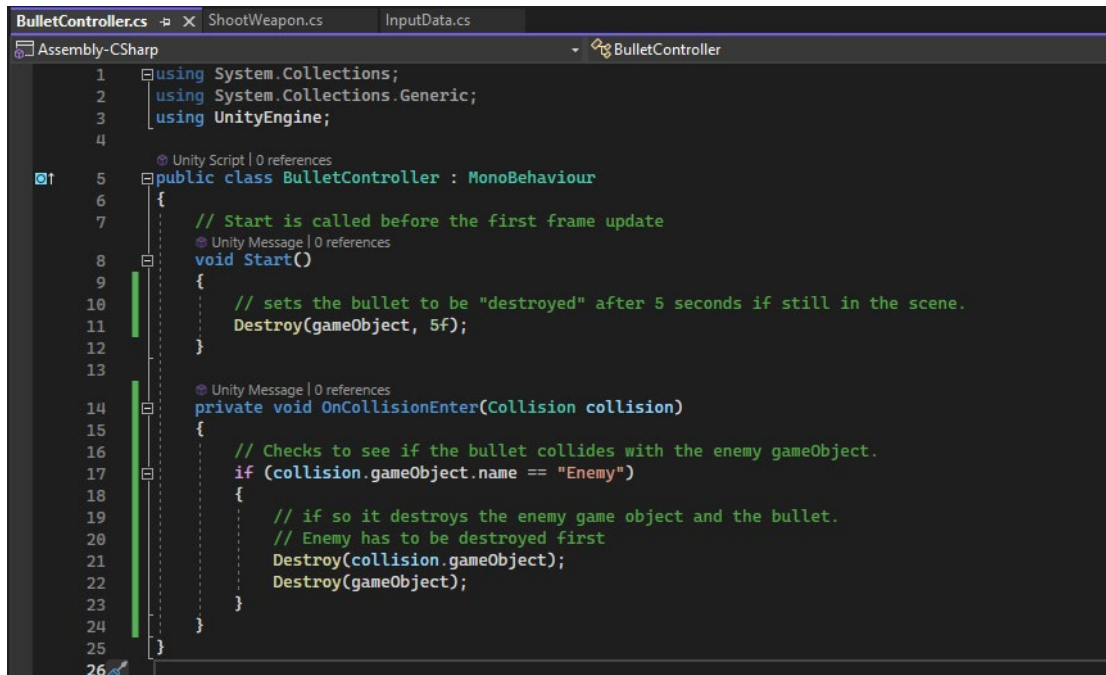


This will now only allow the player to shoot the weapon while it is actively being “held” by the player.

Bullet Destroys the Enemy.

- Now that we have a “functional” weapon we need to make the bullet interact with the Enemy GameObject.
- Create a new script in the Scripts subfolder and rename it BulletController. Then open the script.

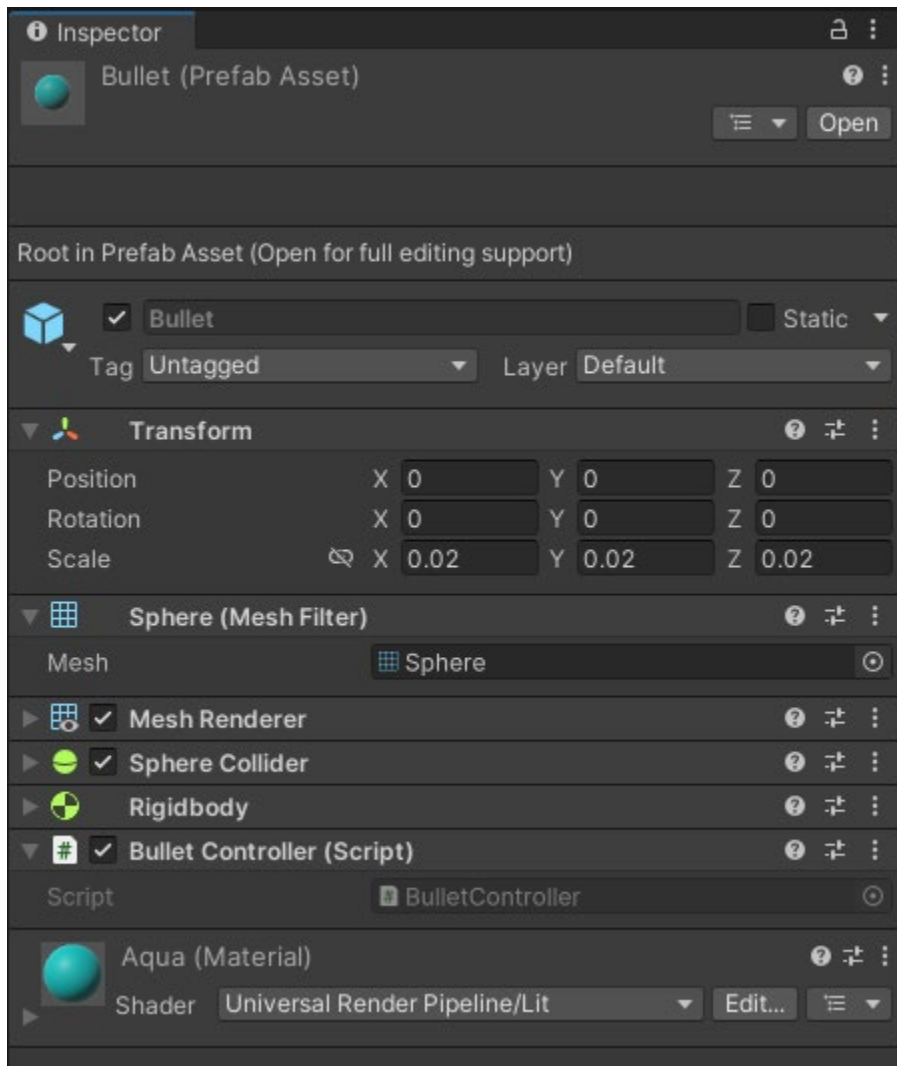
3. Write the following code into the BulletController.cs script:



The screenshot shows a code editor with the following tabs: BulletController.cs, ShootWeapon.cs, and InputData.cs. The BulletController.cs script is selected and contains the following C# code:

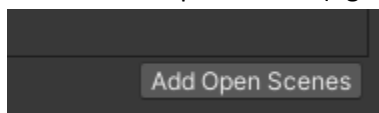
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class BulletController : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10         // sets the bullet to be "destroyed" after 5 seconds if still in the scene.
11         Destroy(gameObject, 5f);
12     }
13
14     private void OnCollisionEnter(Collision collision)
15     {
16         // Checks to see if the bullet collides with the enemy gameObject.
17         if (collision.gameObject.name == "Enemy")
18         {
19             // if so it destroys the enemy game object and the bullet.
20             // Enemy has to be destroyed first
21             Destroy(collision.gameObject);
22             Destroy(gameObject);
23         }
24     }
25 }
26
```

4. Click on the Prefabs subfolder, then click on the Bullet Prefab GameObject. Then click on the Scripts subfolder and then drag the BulletController.cs script onto the Bullet Prefab Inspector.

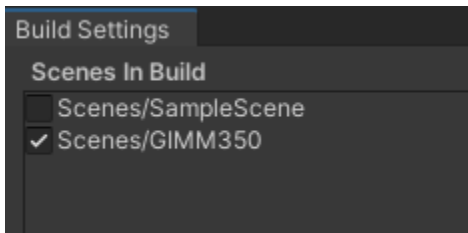


Build the new scene.

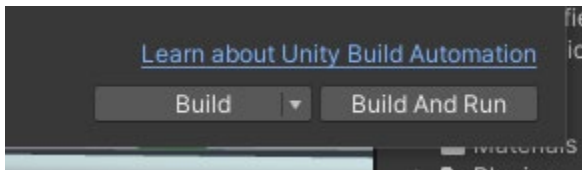
1. Click on File then click on Build Settings.
2. Click on Add Open Scenes (right hand side, near the top of the window)



3. Then deselect the Scenes/SampleScene.



4. Finally, click Build and Run.



Play.

Use the right controller joystick to teleport and the left controller joystick to move continuously. The red sphere weapon can be grabbed with the middle finger grip button on the side of the controllers. Once picked up the weapon can be fired by pressing the A button on the right controller. Try and destroy the enemy!