Quest 8 - Steps

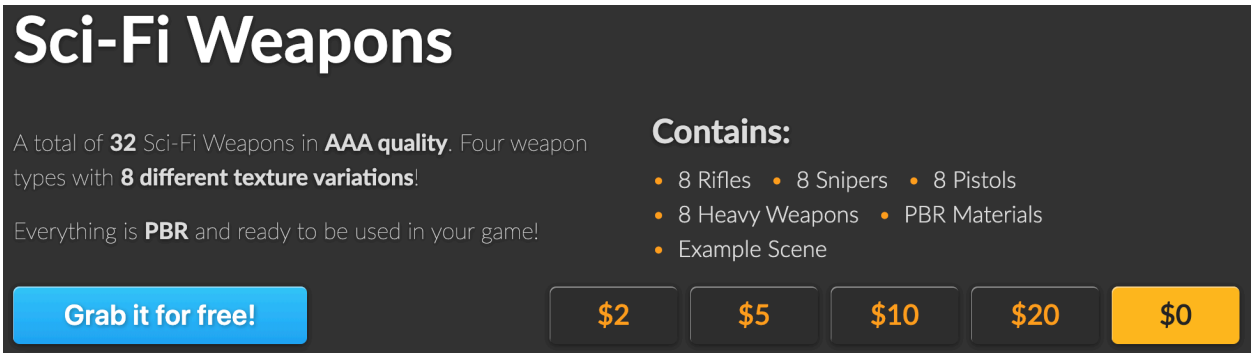
1) Extending Prior Quests

This project requires the First Person Controller, Terrain, and Enemy AI you built in prior assignments. These should be available from your Q6 work. Duplicate the appropriate scene and rename it Q8 for this assignment.

2) Import Assets

We will be using the Sci-Fi Weapons pack from DevAssets. They are available for free:

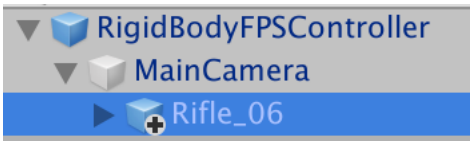
http://devassets.com/assets/sci-fi-weapons/



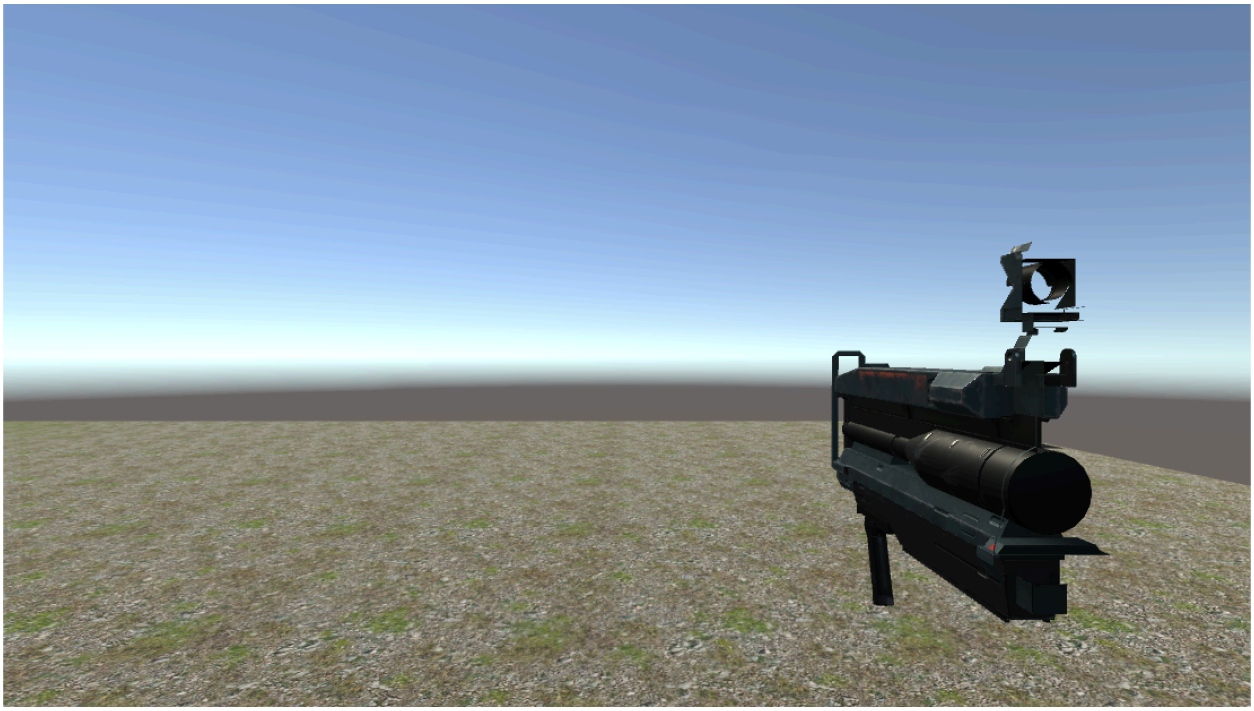
Extract the files and import "Sci-Fi Weapons (Base Pack).unitypackage" into your project.

3) Weapon Setup

Find the Rifle prefab in /Assets/Sci-Fi Weapons/Base Pack/Prefabs/Rifles/ and drag it as a child into the RigidBodyFPSController/MainCamera in your scene.



Align the Rifle so that it appears on camera and turn off the Bullet child object.



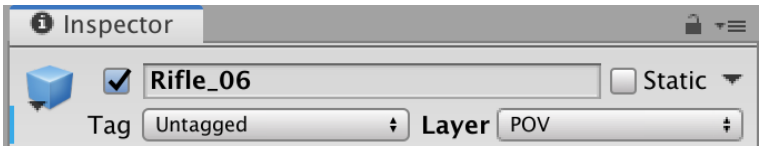
4) Camera Layering

At this point, a few visual artifacts are present when playtesting. The visual of the gun is cut off close to the camera and the gun can pass through the environment.

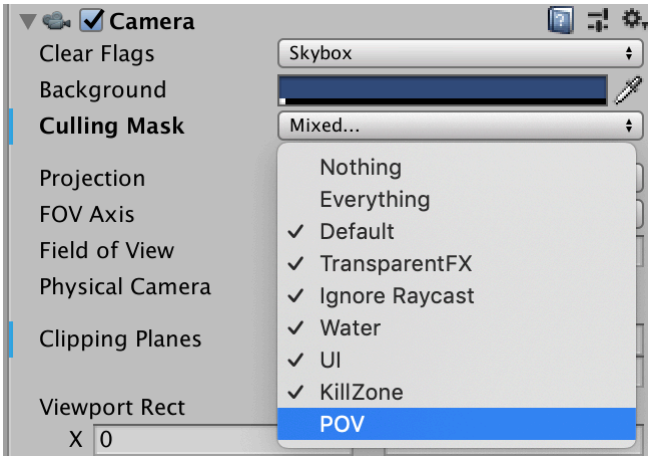
To fix this, we will actually make use of 2 cameras. The existing camera will handle environmental objects, and a new camera will render point-of-view (POV) first-person objects on top like an overlay.

Create a POV layer in Edit->Project Settings->Tags and Layers

Assign the Rifle and all children objects to the POV layer.

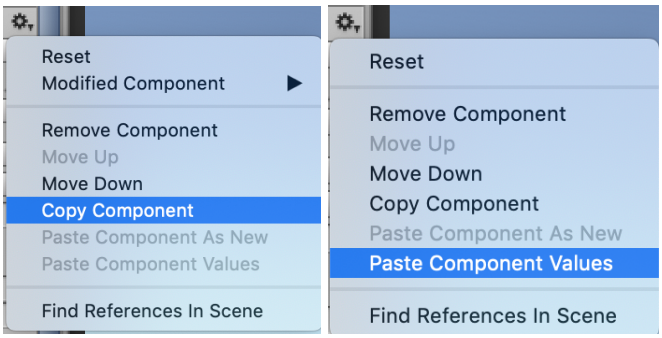


Update the Camera, so its Culling Mask excludes the POV layer. The rifle will disappear from the current game view.

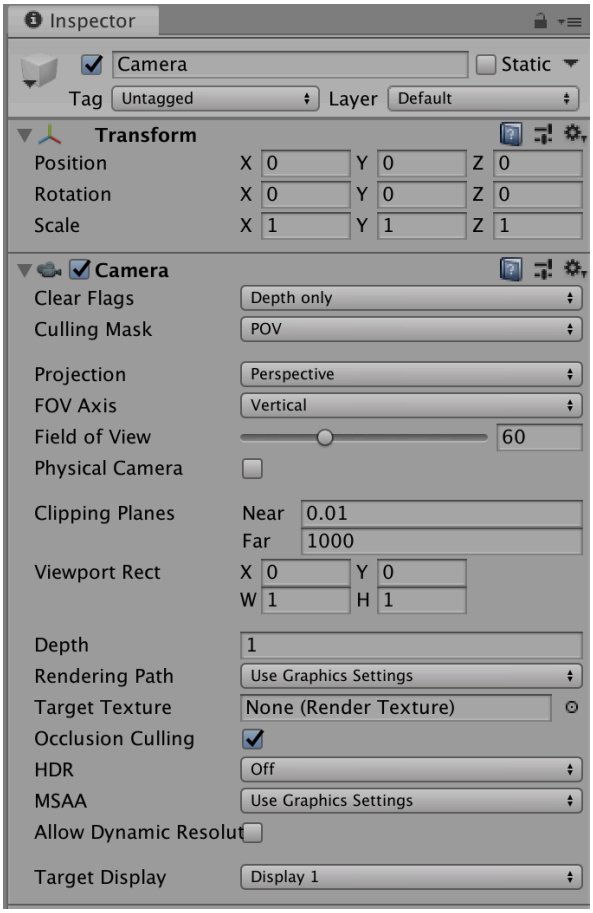


Create a new Camera object and parent it to the original camera.

Zero out the new Camera Transform's position and rotation values. Remove the AudioListener component. Copy the values of the MainCamera's Camera component and paste them into the new Camera's Camera component using the Gear menu.



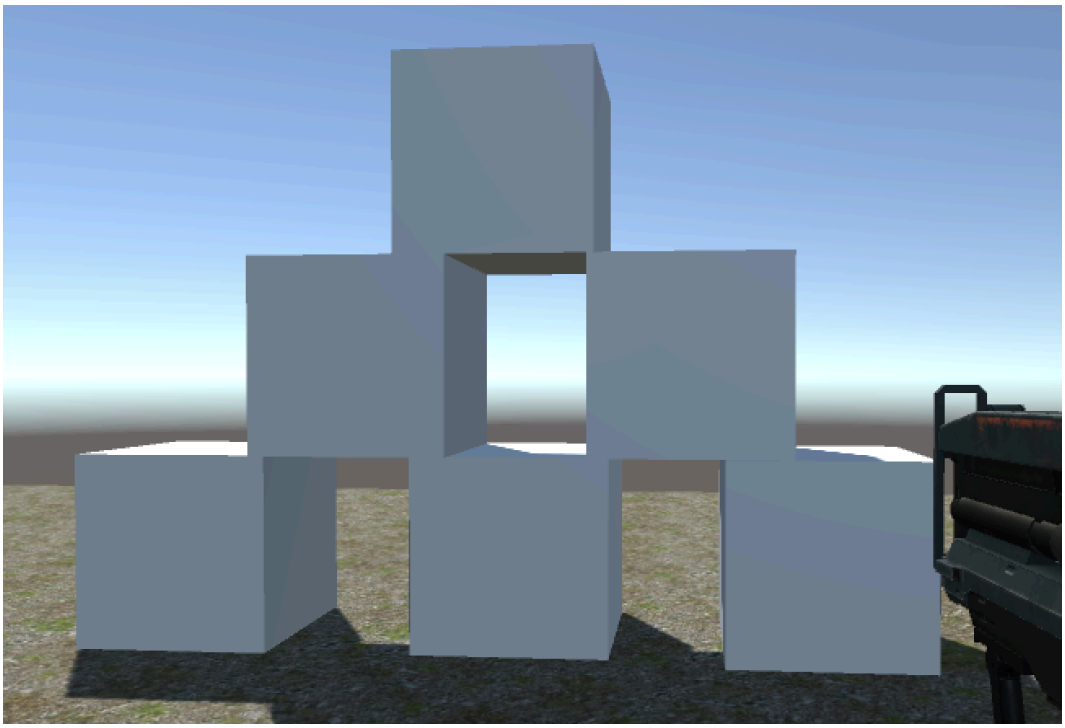
Update the Culling Mask to be POV only. Reduce the Near Clipping Plane and make the new Camera have a depth that is 1 higher than the old camera.



With these settings, the rifle should be completely visible even if you walk up to a wall.

5) Environment Interactables

Create a stack of physics-enabled cubes.



6) Weapon Interactions (Hitscan/Raycast)

Left-clicking will trigger a hitscan attack, which means it is an interaction that uses a raycast and causes an immediate effect.

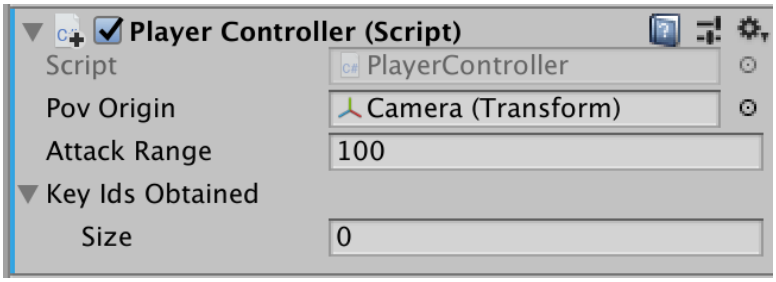
Hitscan attacks must originate from a specific perspective (even though they might appear to come from a handheld weapon. Usually this perspective is the active player's camera view. In PlayerController, we will specify this perspective location with the povOrigin variable. The range of the attack will also be configurable.

```
public class PlayerController : MonoBehaviour {
    public static PlayerController instance;

    // Outlets
    public Transform povOrigin;

    // Configuration
    public float attackRange;
```

In the inspector, povOrigin will be the POV Camera from the previous step. Attack Range will be 100.



The PrimaryAttack function raycasts from the source perspective in the direction faced by the player for the configured range. If an object in range is hit and has a rigidbody, it will be shoved in the direction of our attack.

```
void PrimaryAttack() {
    RaycastHit hit;
    bool hitSomething = Physics.Raycast(povOrigin.position, povOrigin.forward, out hit, attackRange);
    if(hitSomething) {
        Rigidbody targetRigidbody = hit.transform.GetComponent<Rigidbody>();
        if(targetRigidbody) {
            targetRigidbody.AddForce(povOrigin.forward * 100f, ForceMode.Impulse);
        }
    }
}
```

This attack is called by pressing the left mouse button.

```
void Update()
{
    if(Input.GetMouseButtonDown(0)) {
        PrimaryAttack();
    }
```

7) Weapon Interactions (Projectile)

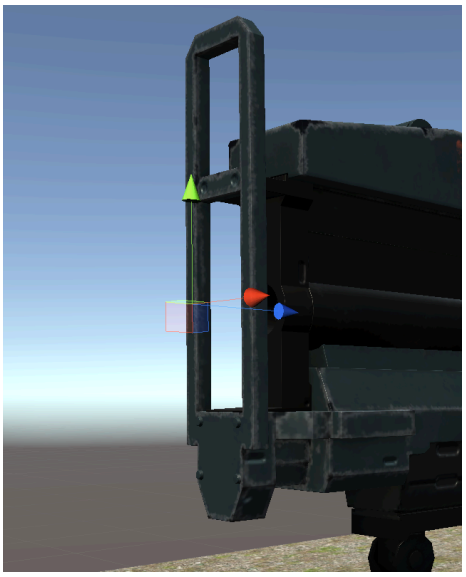
Right-clicking will cause a projectile attack, which will spawn a game object that must travel through the world before it has an effect. Projectile attacks have the advantage of allowing physics and time to influence the outcome of the mechanic.

Projectile attacks usually originate from an object such as a handheld item instead of the character's perspective. We will need to specify the projectile's origin and the prefab to be used as a projectile.

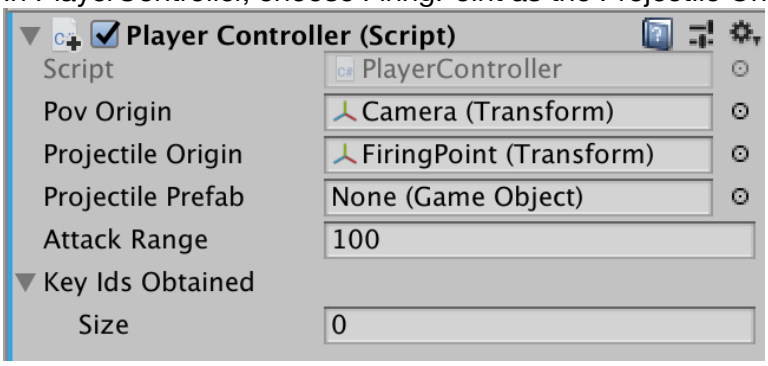
```
public class PlayerController : MonoBehaviour {
    public static PlayerController instance;

    // Outlets
    public Transform povOrigin;
    public Transform projectileOrigin;
    public GameObject projectilePrefab;
```

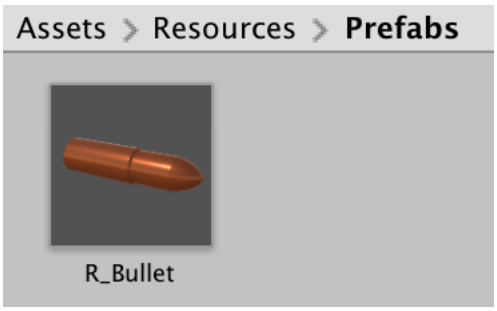
Create an empty child game object under the Rifle and rename it "FiringPoint." Position it just in front of the rifle.



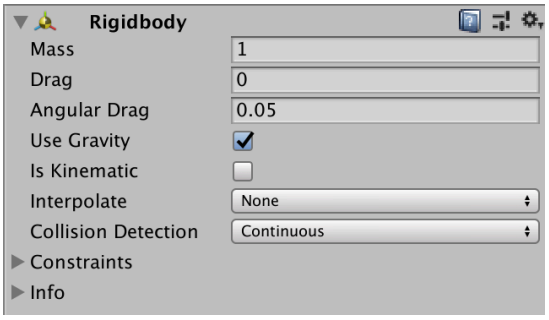
In PlayerController, choose FiringPoint as the Projectile Origin.

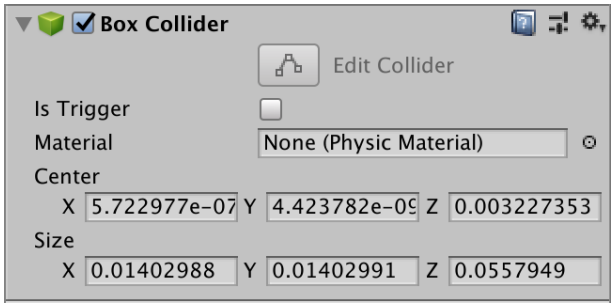


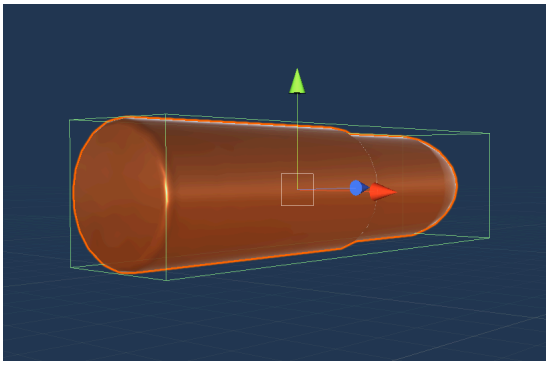
Next, prefab the Rifle's child Bullet game object in /Assets/Resources/Prefabs/

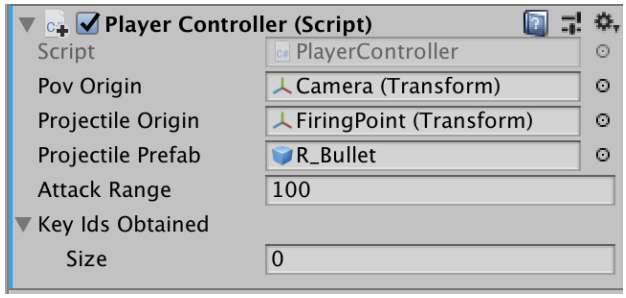


Give it a Rigidbody with Continuous Collision Detection, a BoxCollider, and assign the prefab as the Projectile Prefab on PlayerController.









The SecondaryAttack function in PlayerController will instantiate the prefab at the projectile origin point and send it forward using physics. For demonstration purposes, this attack function exaggerates the size of the projectile by scaling it up.

```
void SecondaryAttack() {
    GameObject projectile = Instantiate(projectilePrefab,
                                        projectileOrigin.position,
                                        Quaternion.LookRotation(povOrigin.forward));
    projectile.transform.localScale = Vector3.one * 5f;
    projectile.GetComponent<Rigidbody>().AddForce(povOrigin.forward * 25f, ForceMode.Impulse);
}
```

This attack function is called from update with the right mouse button.

```
void Update()
{
    if(Input.GetMouseButtonDown(0)) {
        PrimaryAttack();
    }

    if(Input.GetMouseButtonDown(1)) {
        SecondaryAttack();
    }
```

In playtesting, the environment does not react to the shot event, but instead reacts to the projectile game object itself. Over long distances, the projectile even falls due to gravity instead of traveling in a straight line.