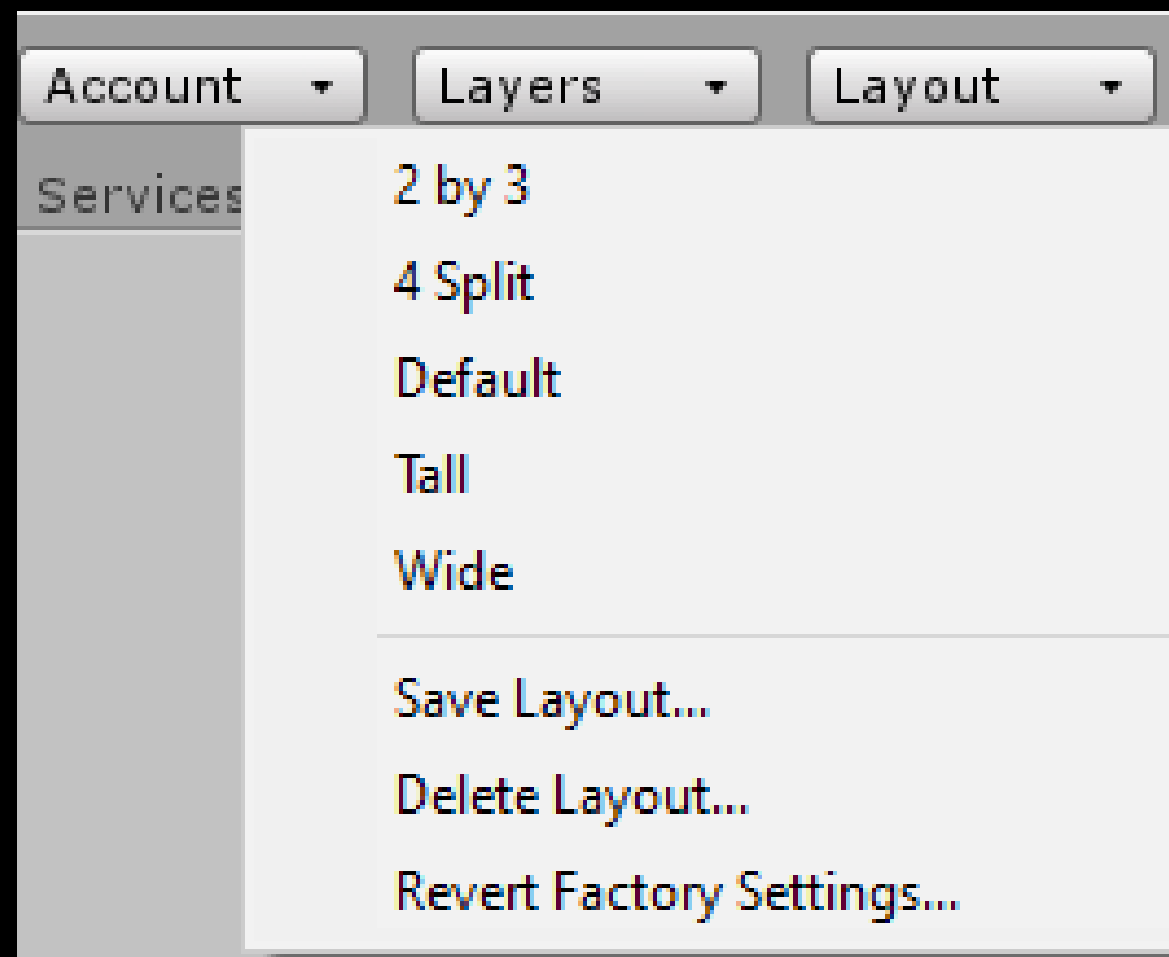


Making of Shoot Out!

Let's start !!!

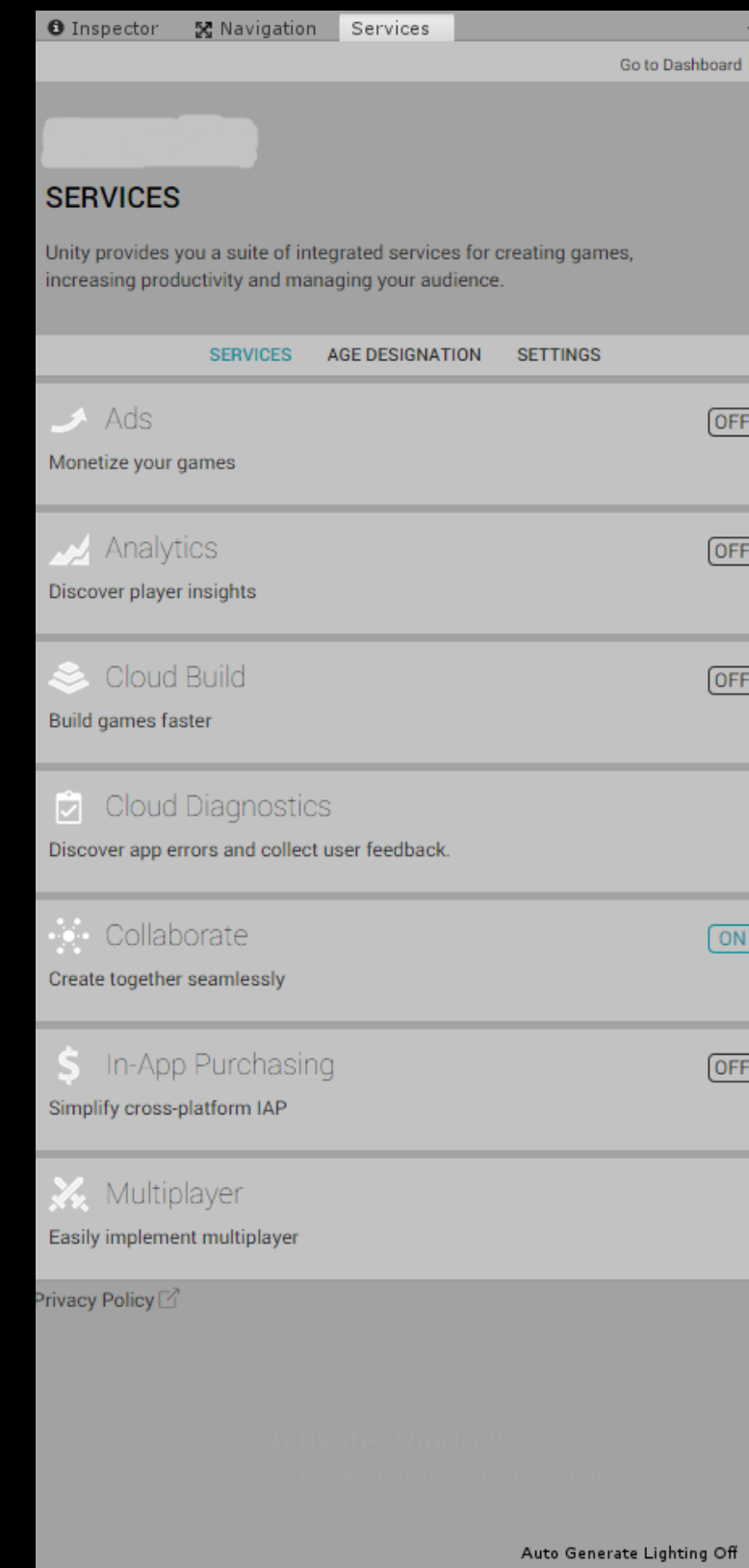
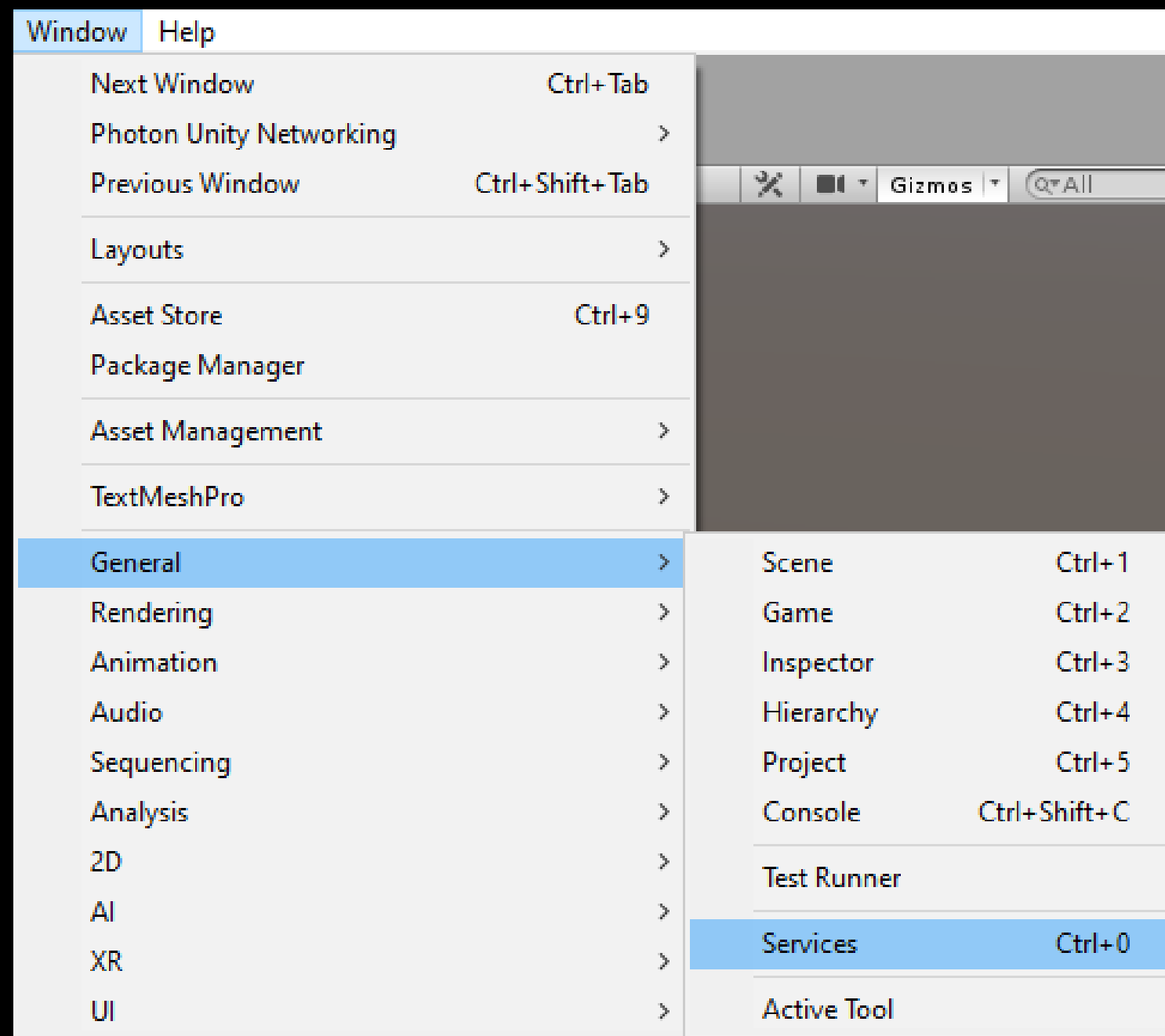
EDITOR LAYOUT

Set the editor layout according to your comfort. You can also make your own custom layout.



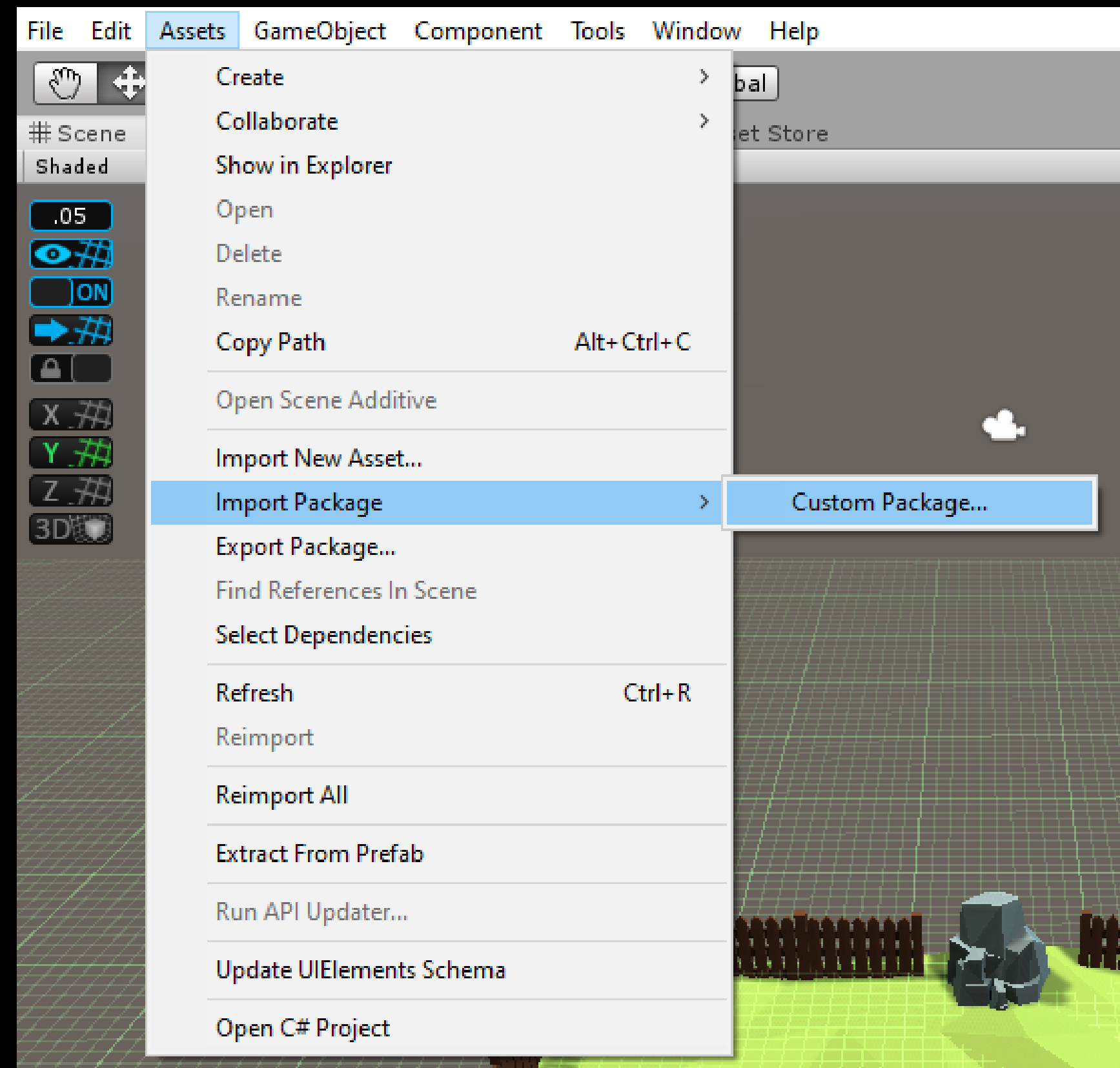
VERSION CONTROL SYSTEM

Turn on Unity's personalised VCS by going to Window -> General -> Services -> Collaborate. Click on the 'Collab' dropdown on the top and click 'Publish' to update your changes to the server.



IMPORTING ASSETS

Assets -> Import Package -> Custom Package... -> *locate the asset package on your system*



CREATING SCENE

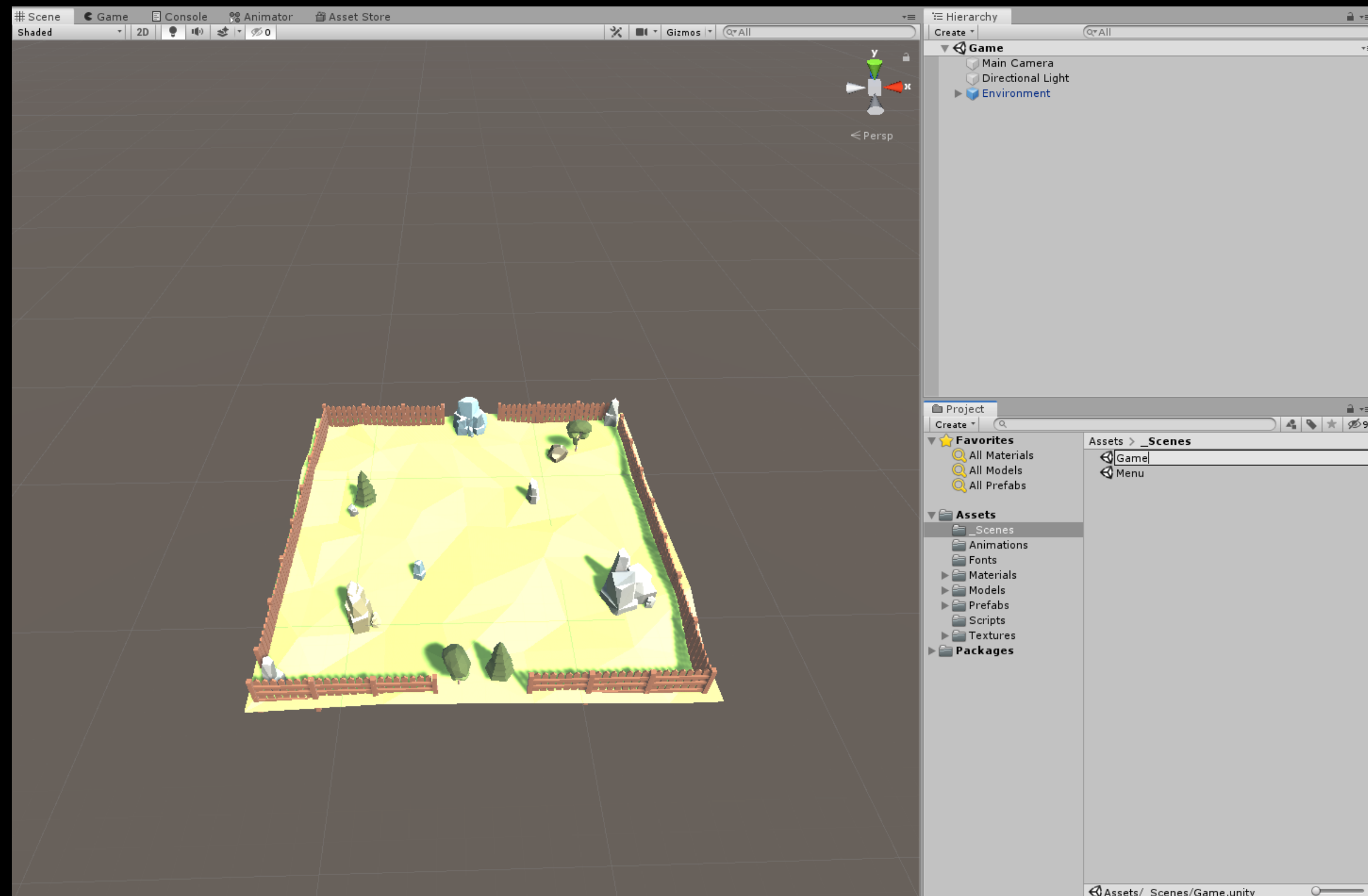
In Project window rename 'Scenes' to '_Scenes' and within _Scenes, rename 'SampleScene' to 'Game'.



LEVEL DESIGN

Design various models such as trees, fencing, rocks, etc. either on Blender, Maya or simply use Pro-Builder and put them together to design an appropriate level.

For this workshop, we have already designed a level. Go to Project -> Prefabs -> Environment. Drag and drop that into Hierarchy.

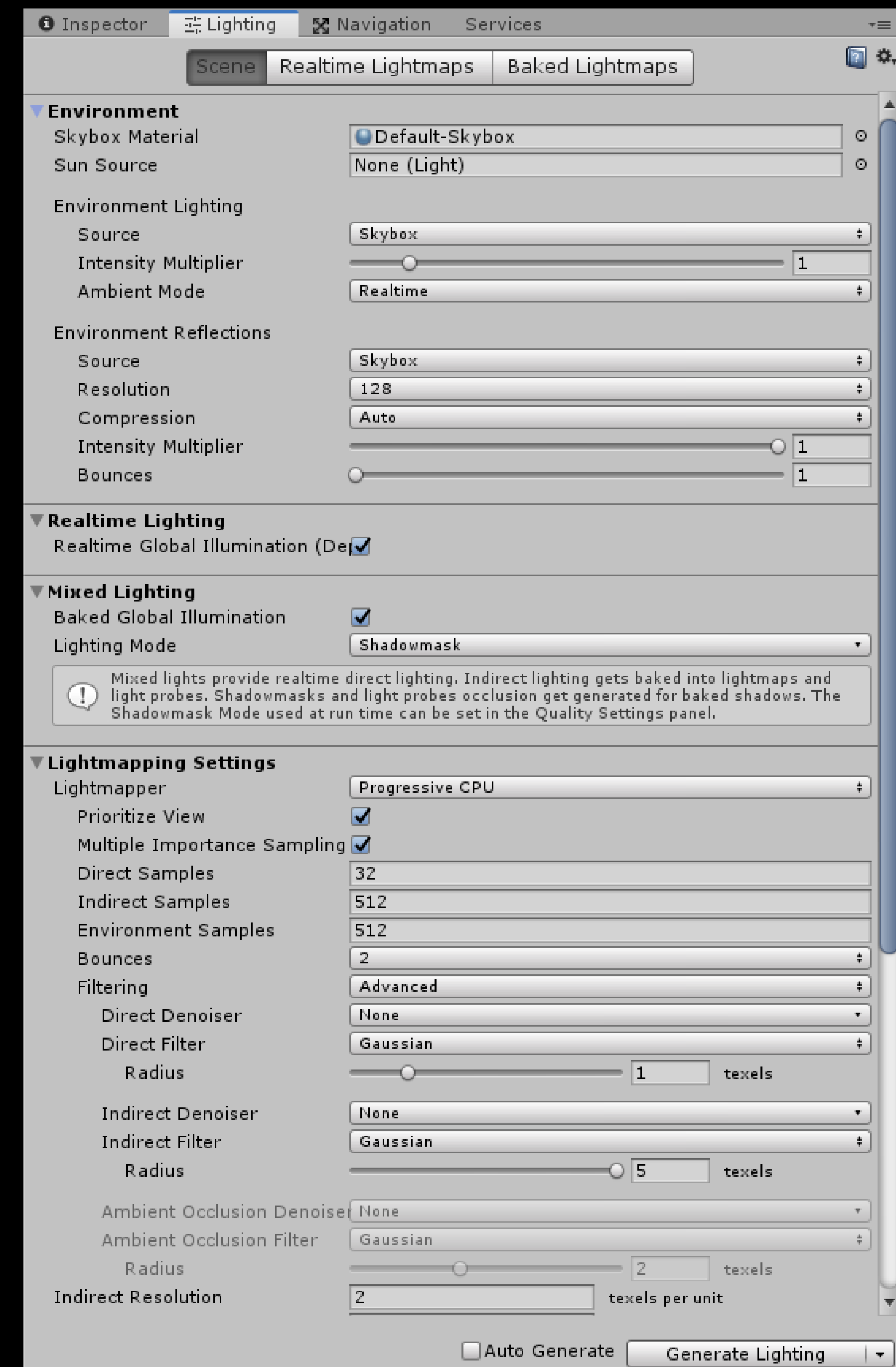
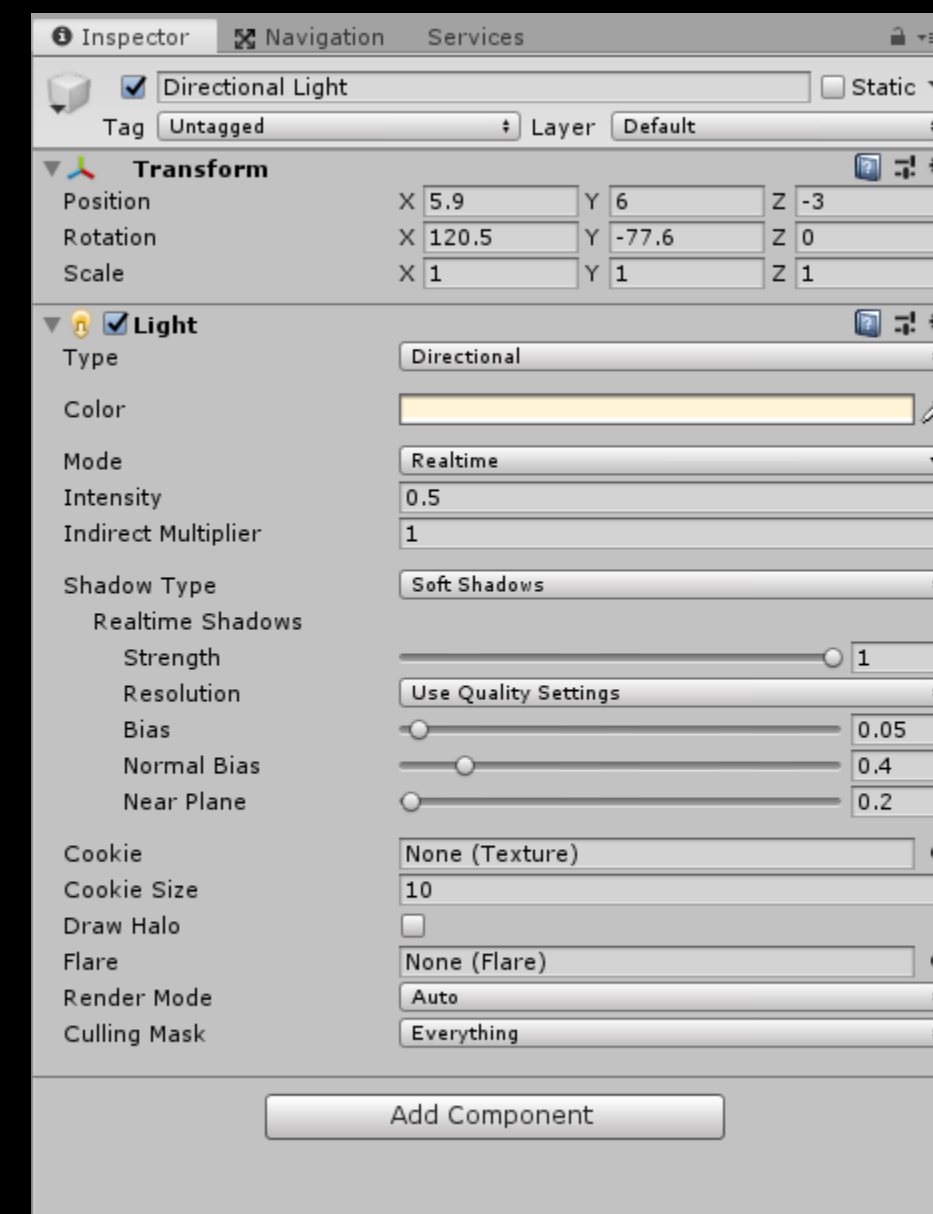


ADJUST LIGHTING

Set the Intensity of the Directional Light game-object in Hierarchy to 0.5 through Inspector.

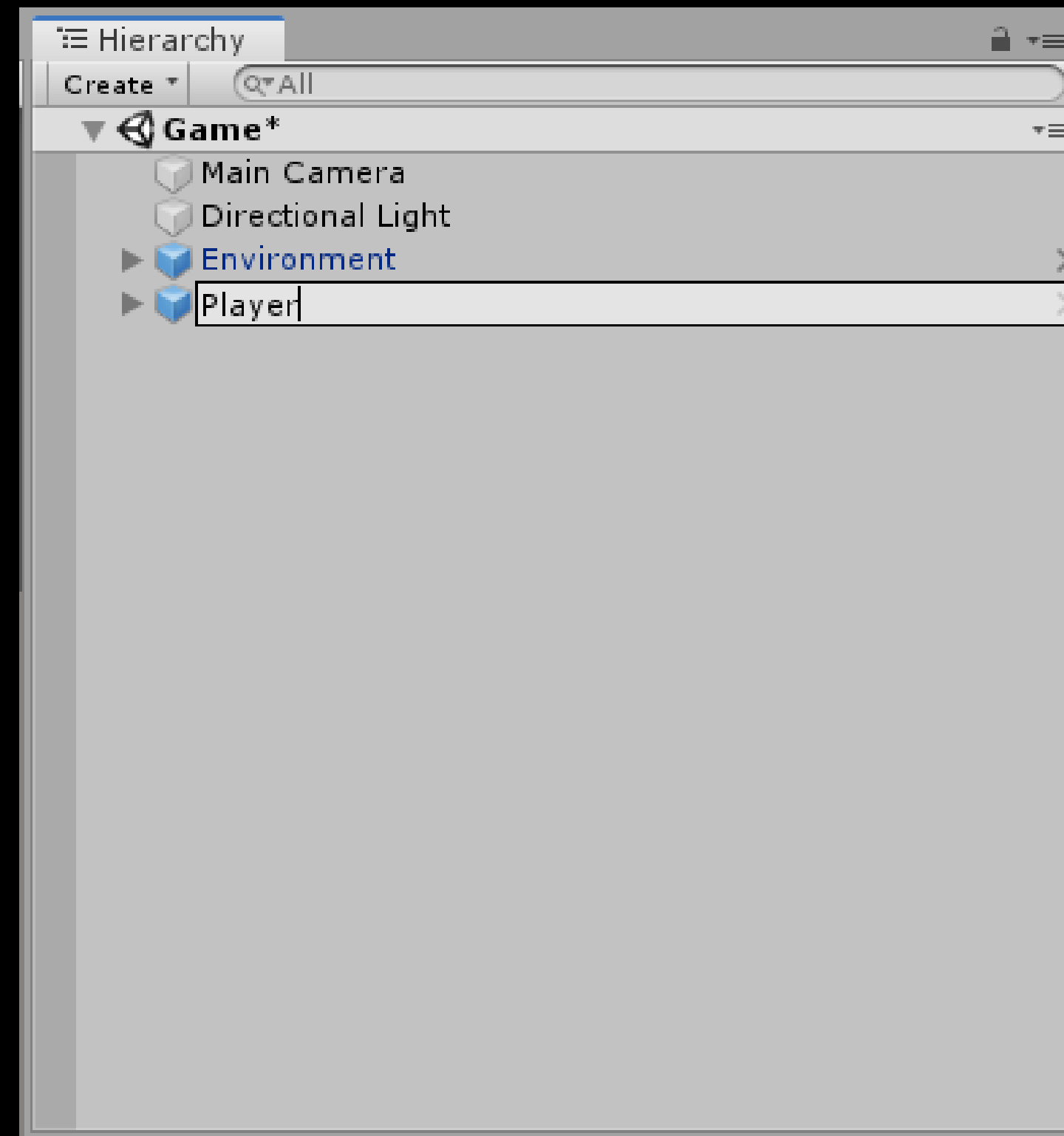
Go to Window -> Rendering -> Lighting Settings. Under Scene tab, set Sun Source to none. Filtering should be set to Advanced with Direct Denoiser as None. Under Bakes Lightmaps tab, set Lighting Data Asset to None. Uncheck Auto Generate and generate lighting.

Restart the editor for changes to take effect.



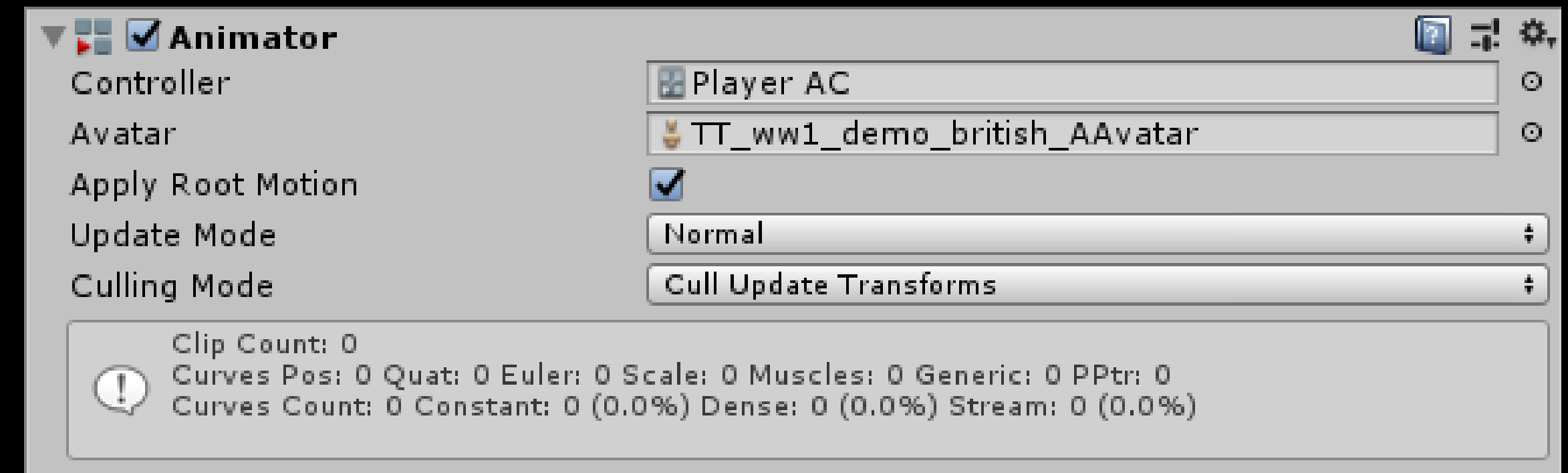
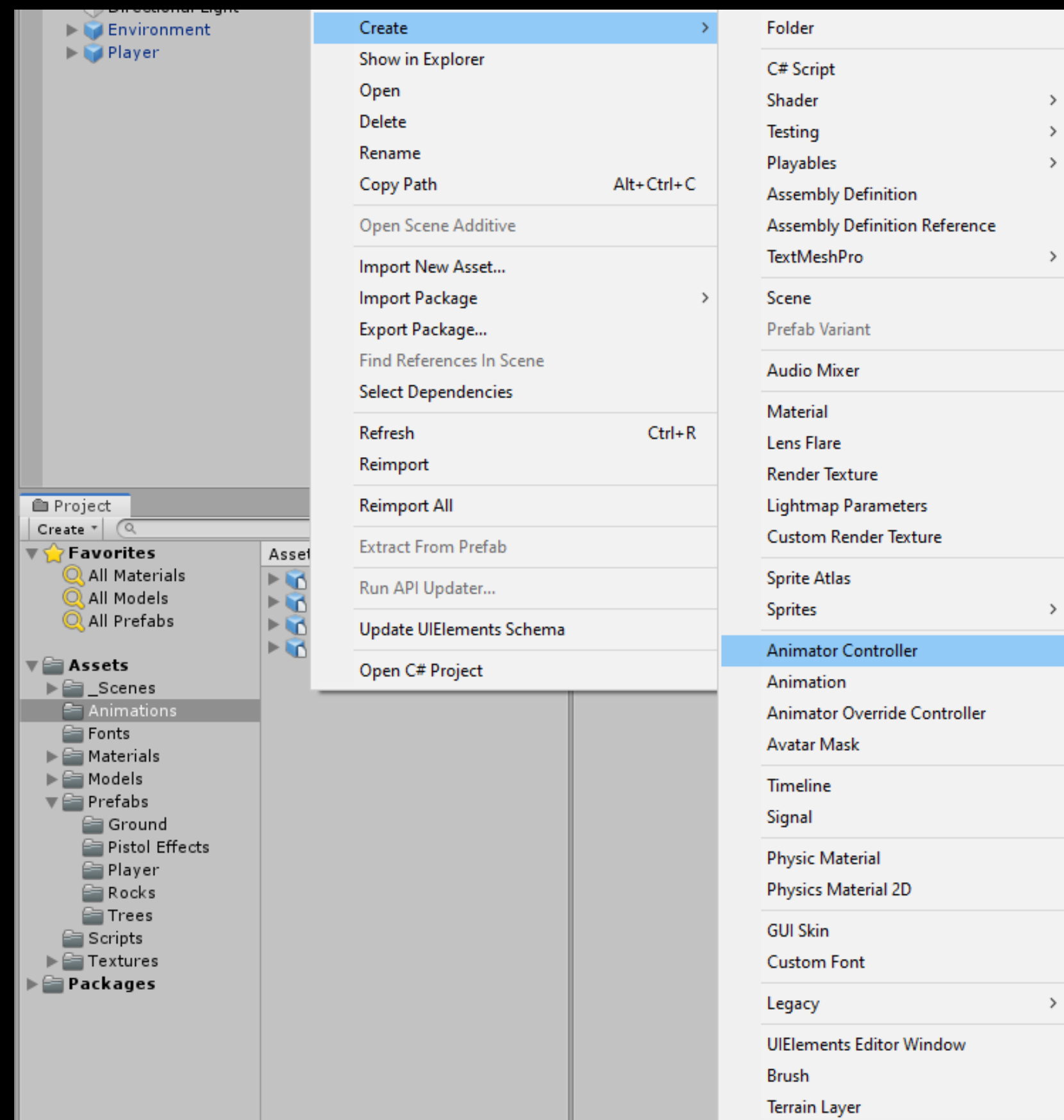
ADDING PLAYER

Go to Prefabs -> Player. Choose a model you'd like to have as your player and drag and drop him in Hierarchy. Rename the game-object as Player.



ANIMATING PLAYER

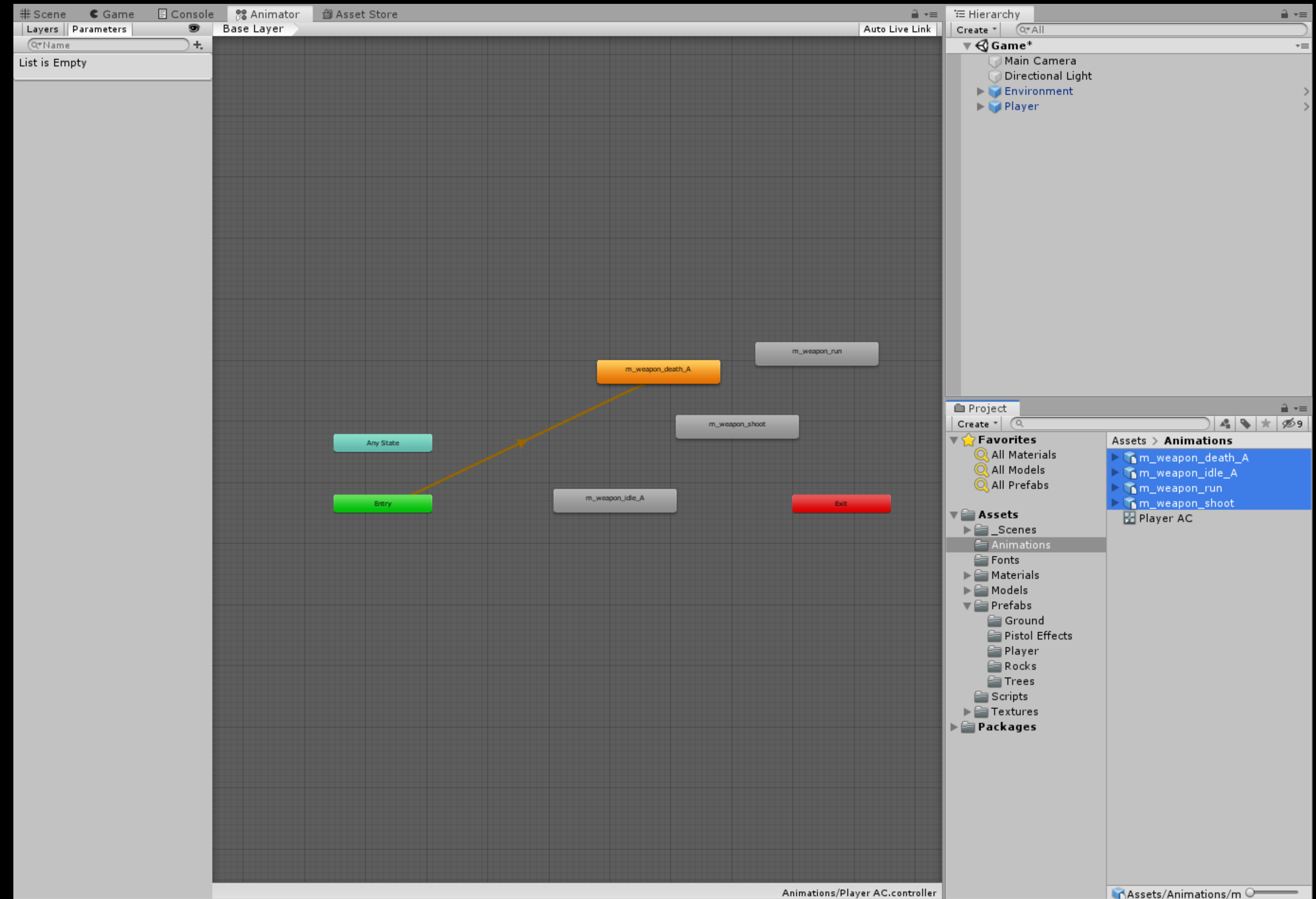
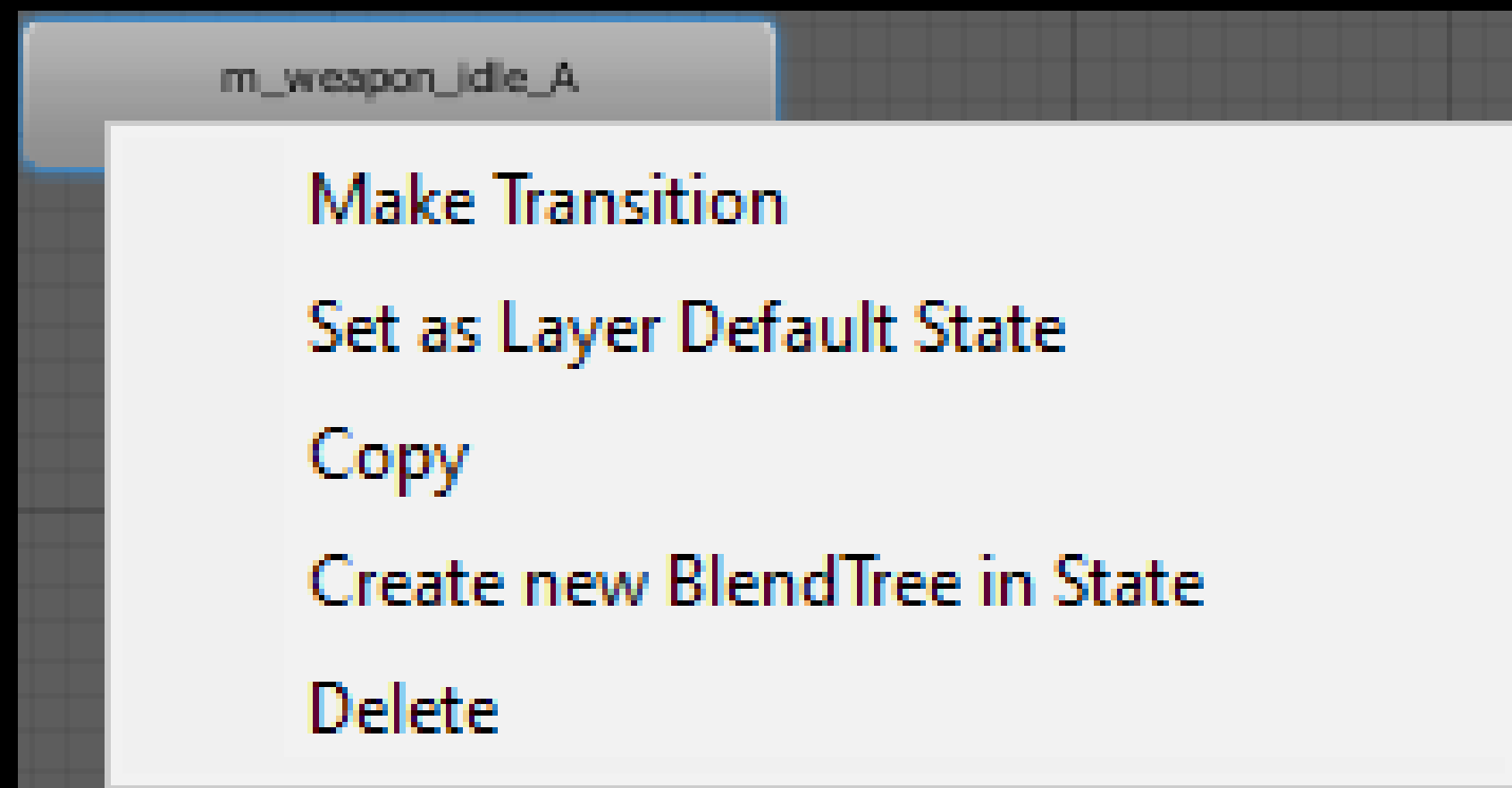
In the Animations folder of Project window, create an Animator Controller called 'Player AC' and assign it to the Animator component of Player in Hierarchy.



ANIMATING PLAYER

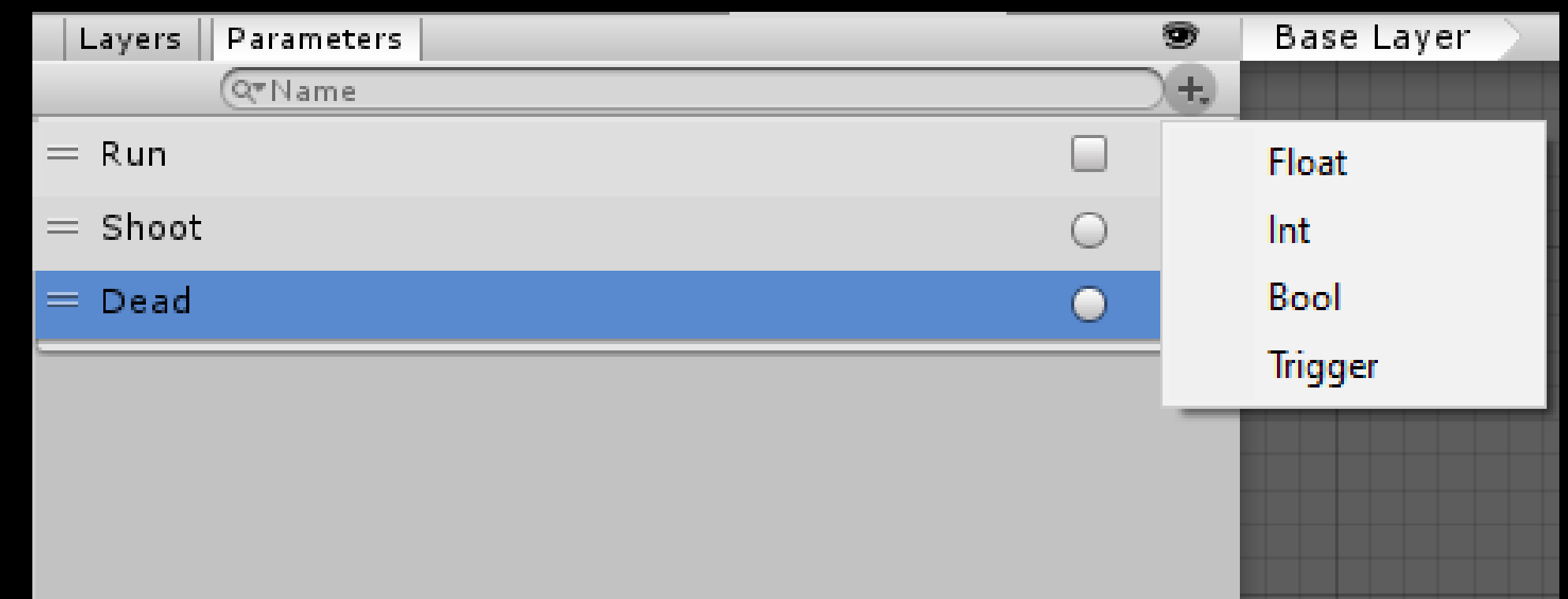
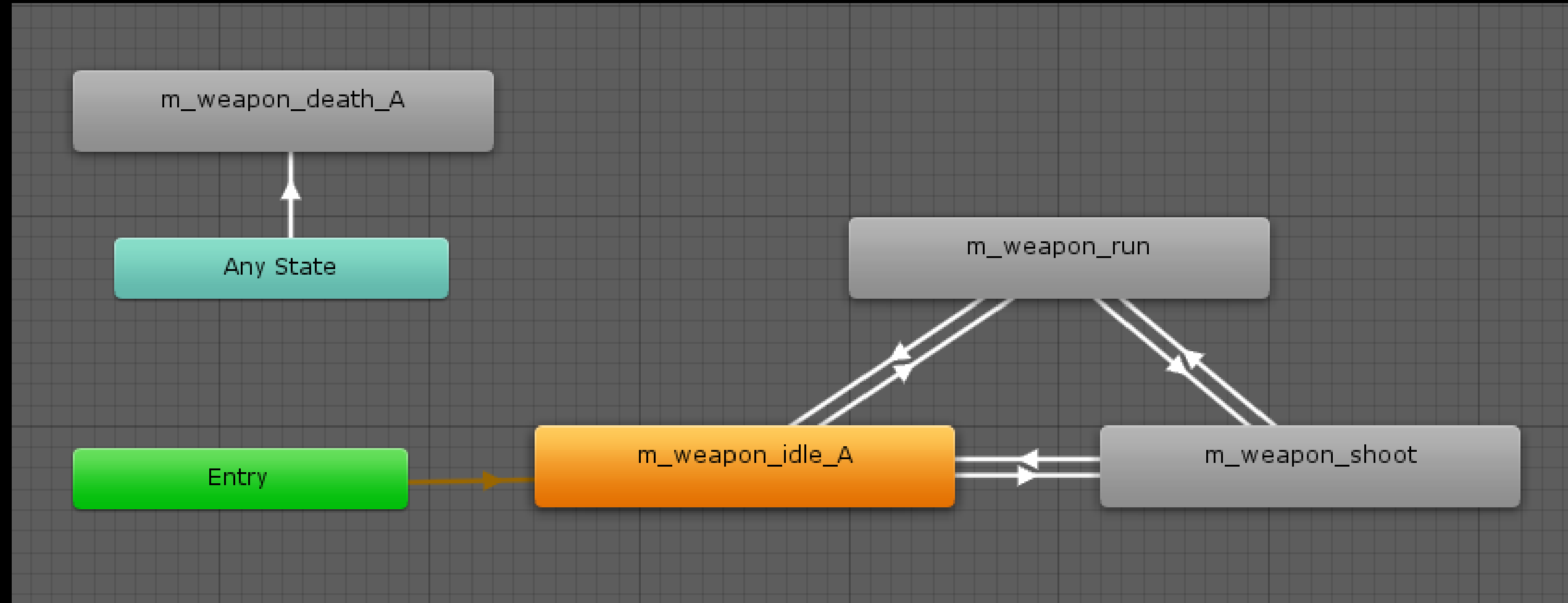
Open 'Player AC' and drag and drop all the animations to the workspace.

Right click on Idle animation and set it as default state.



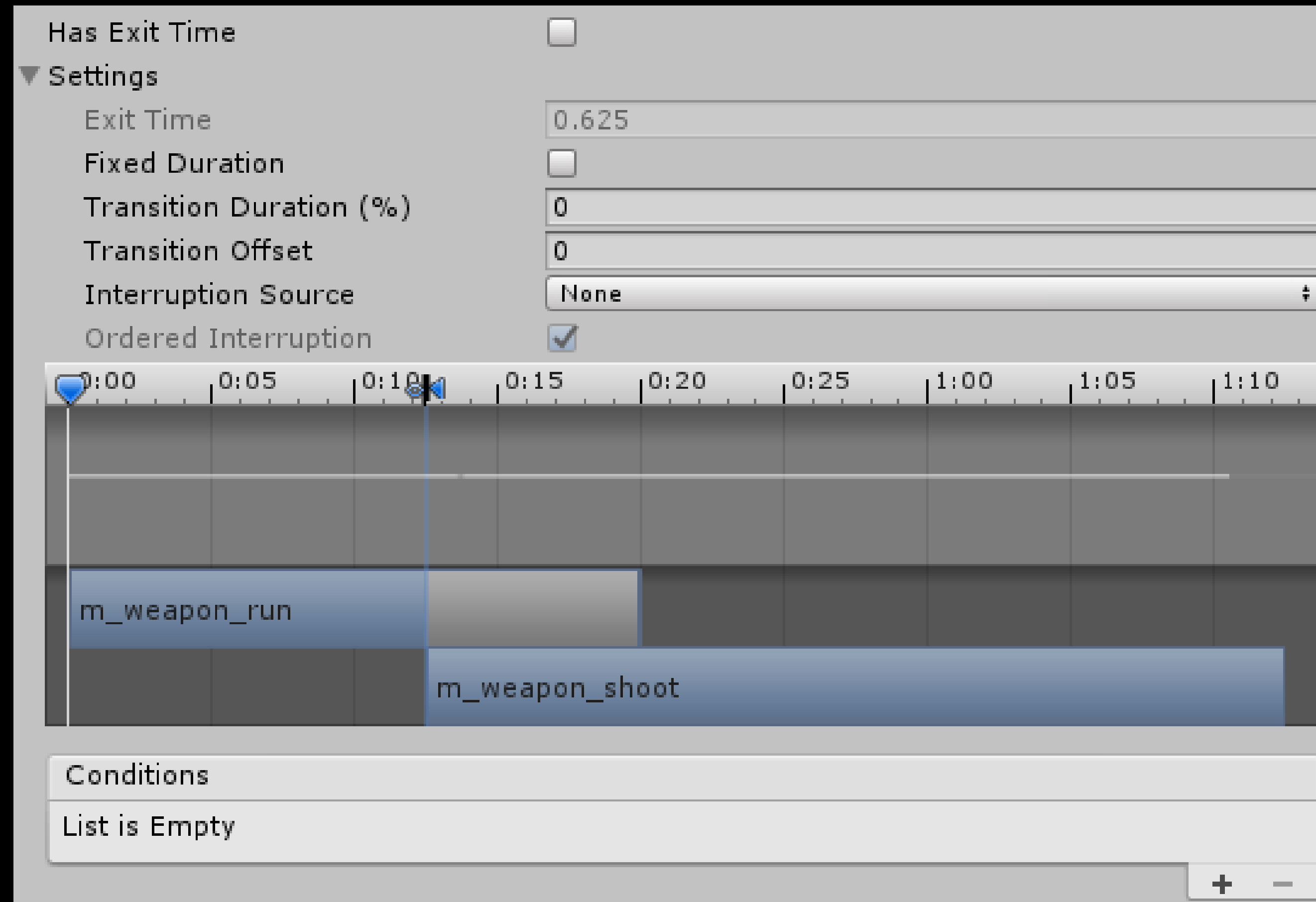
ANIMATING PLAYER

Make the following network between animation states and create the following parameters, one bool called Run and two triggers called Shoot and Dead.



ANIMATING PLAYER

Uncheck 'Has Exit Time' for all transition except for Shoot -> Run and Shoot -> Idle.
For each transition uncheck 'Fixed Duration' and set 'Transition Duration(%)' and 'Transition Offset' to 0.



ANIMATING PLAYER

Set the Conditions for each transition accordingly.

Any State -> Death : Dead

Idle -> Shoot : Shoot

Run -> Shoot : Shoot

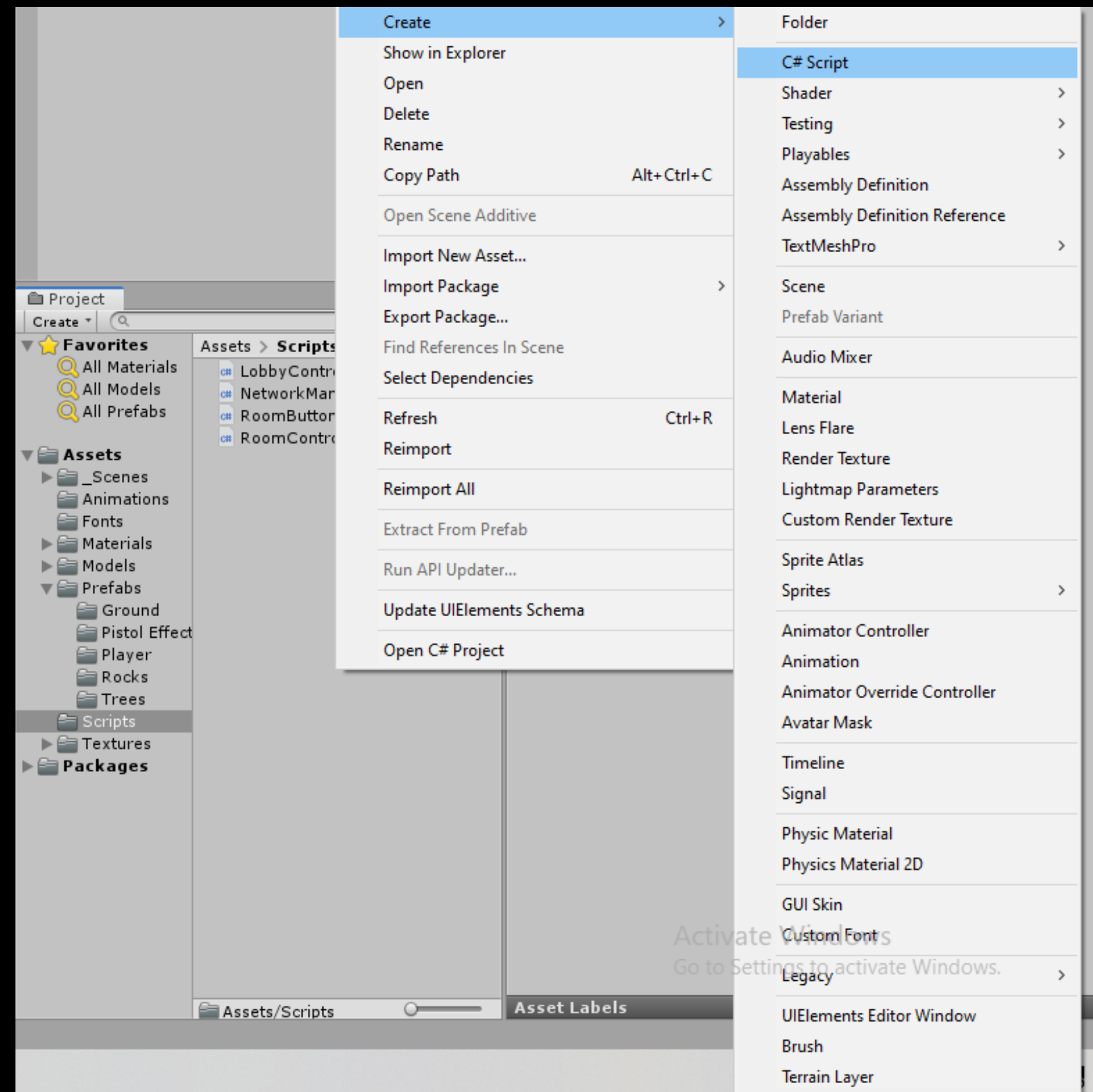
Idle -> Run : Run (True)

Run -> Idle : Run (False)

Let's do some programming !!!

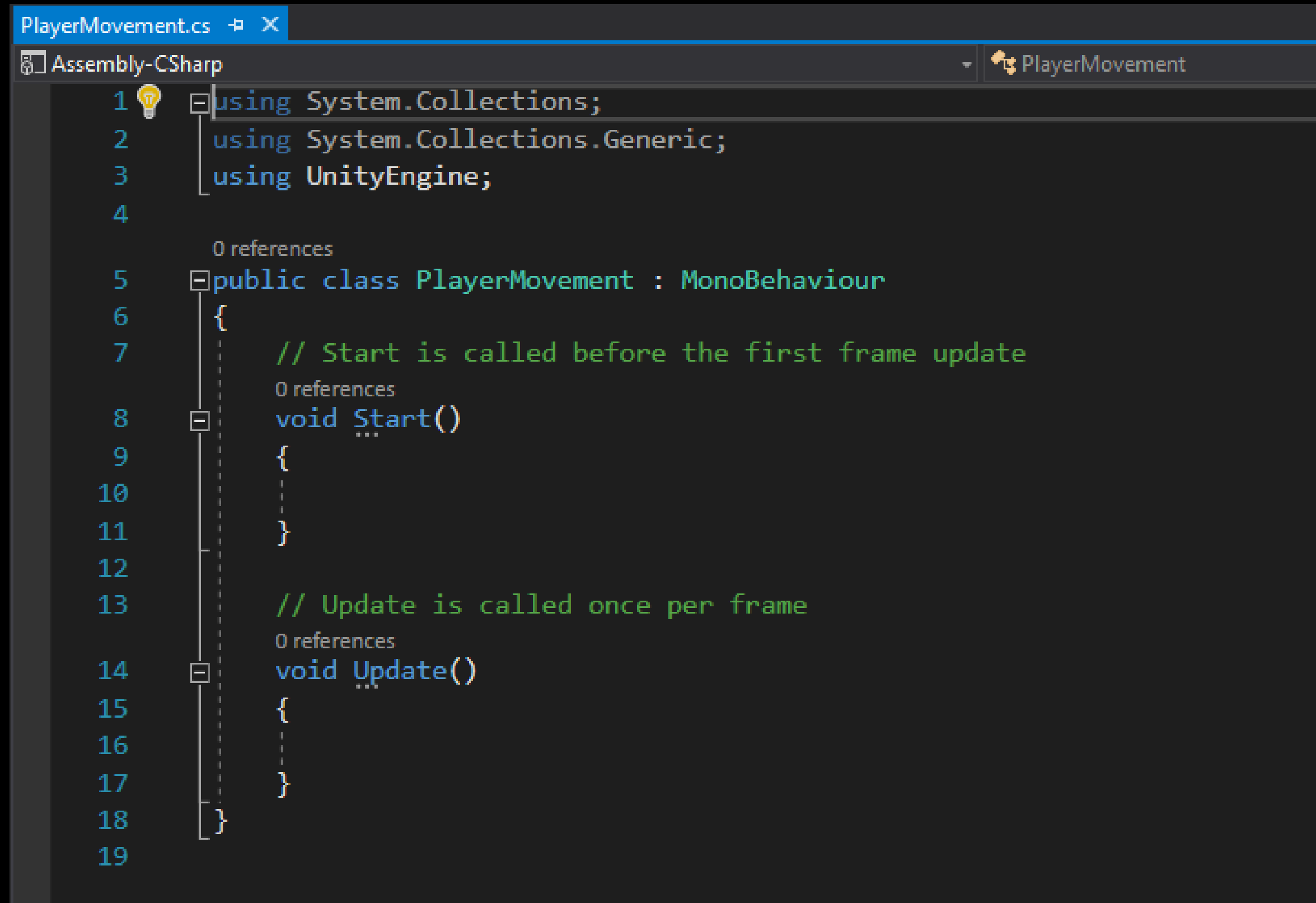
PLAYER MOVEMENT

In the Scripts folder of Project window, create a C# script and name it 'PlayerMovement'. Make sure there is no space in the script's name. Attach the script to player.



PLAYER MOVEMENT

Open the script in any editor of your choice, preferably Visual Studio or Visual Studio Code.



The screenshot shows a code editor window titled "PlayerMovement.cs". The editor displays the following C# code:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class PlayerMovement : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```

The code is formatted with syntax highlighting. The editor interface includes a toolbar at the top with icons for undo, redo, and search, and a dropdown menu showing "Assembly-CSharp" and "PlayerMovement".

PLAYER MOVEMENT

Code the player to move around the level.

```
public class PlayerMovement : MonoBehaviour
{
    public float moveVelocity = 5f;

    private float moveHorizontal;
    private float moveVertical;

    0 references
    private void FixedUpdate()
    {
        moveHorizontal = Input.GetAxisRaw("Horizontal");
        moveVertical = Input.GetAxisRaw("Vertical");

        Move(moveHorizontal, moveVertical);
    }

    1 reference
    private void Move(float moveHorizontal, float moveVertical)
    {
        transform.Translate(Vector3.forward * moveVertical * moveVelocity * Time.deltaTime);
        transform.Translate(Vector3.right * moveHorizontal * moveVelocity * Time.deltaTime);
    }
}
```

PLAYER MOVEMENT

Code the player to look around the game area using mouse.

```
public class PlayerMovement : MonoBehaviour
{
    public float moveVelocity = 5f;
    public LayerMask floorMask;

    private float moveHorizontal;
    private float moveVertical;
    private Rigidbody playerRigidbody;

    0 references
    private void Start()
    {
        playerRigidbody = GetComponent<Rigidbody>();
    }

    0 references
    private void FixedUpdate()
    {
        moveHorizontal = Input.GetAxisRaw("Horizontal");
        moveVertical = Input.GetAxisRaw("Vertical");

        Move(moveHorizontal, moveVertical);
        TurnAround();
    }

    1 reference
    private void Move(float moveHorizontal, float moveVertical)
    {
        transform.Translate(Vector3.forward * moveVertical * moveVelocity * Time.deltaTime);
        transform.Translate(Vector3.right * moveHorizontal * moveVelocity * Time.deltaTime);
    }

    1 reference
    private void TurnAround()
    {
        Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit floorHit;

        if (Physics.Raycast(camRay, out floorHit, 100f, floorMask))
        {
            Vector3 playerToFocussedPoint = floorHit.point - this.transform.position;
            playerToFocussedPoint.y = 0f;
            Quaternion newRotation = Quaternion.LookRotation(playerToFocussedPoint);
            playerRigidbody.MoveRotation(newRotation);
        }
    }
}
```

PLAYER MOVEMENT

Control the animations of the player movement.

```
public class PlayerMovement : MonoBehaviour
{
    public float moveVelocity = 5f;
    public LayerMask floorMask;

    private float moveHorizontal;
    private float moveVertical;
    private Rigidbody playerRigidbody;
    private Animator animator;

    0 references
    private void Start()
    {
        playerRigidbody = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();
    }

    0 references
    private void FixedUpdate()
    {
        moveHorizontal = Input.GetAxisRaw("Horizontal");
        moveVertical = Input.GetAxisRaw("Vertical");

        Move(moveHorizontal, moveVertical);
        TurnAround();
        AnimatePlayer(moveHorizontal, moveVertical);
    }

    1 reference
    private void Move(float moveHorizontal, float moveVertical)
    {
        transform.Translate(Vector3.forward * moveVertical * moveVelocity * Time.deltaTime);
        transform.Translate(Vector3.right * moveHorizontal * moveVelocity * Time.deltaTime);
    }

    1 reference
    private void TurnAround()
    {
        Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit floorHit;

        if (Physics.Raycast(camRay, out floorHit, 100f, floorMask))
        {
            Vector3 playerToFocussedPoint = floorHit.point - this.transform.position;
            playerToFocussedPoint.y = 0f;
            Quaternion newRotation = Quaternion.LookRotation(playerToFocussedPoint);
            playerRigidbody.MoveRotation(newRotation);
        }
    }

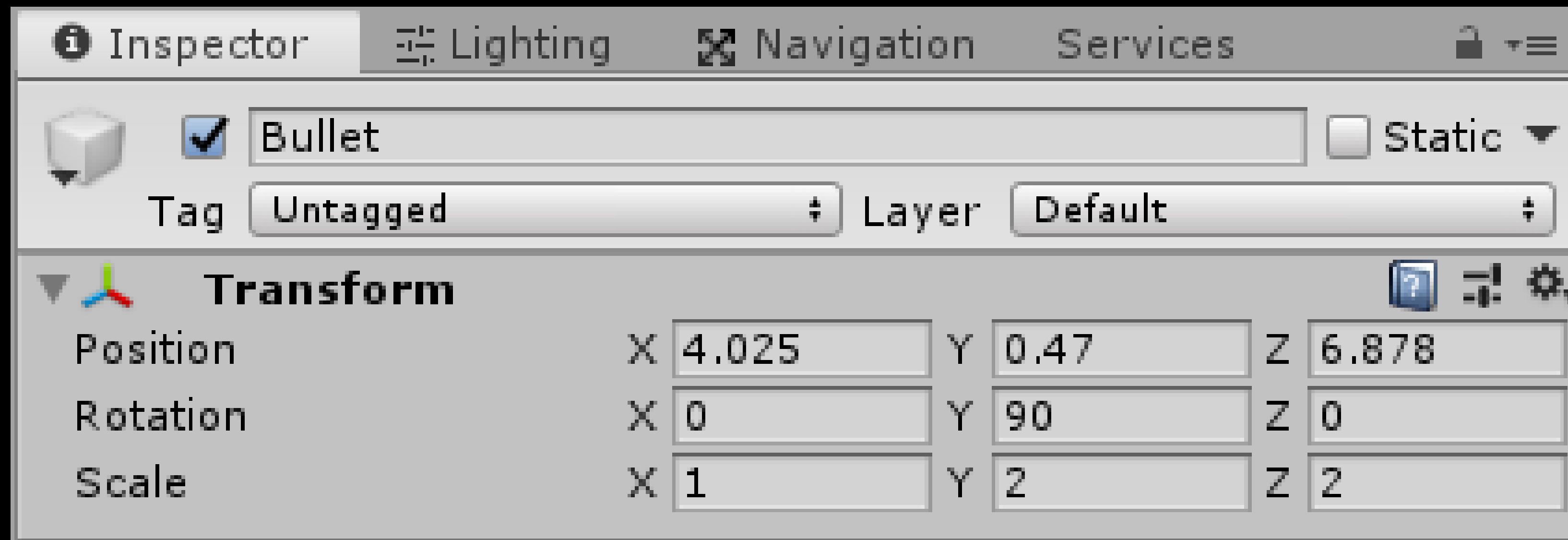
    1 reference
    private void AnimatePlayer(float moveHorizontal, float moveVertical)
    {
        bool walking = (moveHorizontal != 0f) || (moveVertical != 0f);

        animator.SetBool("Run", walking);
    }
}
```

Time for our first test run !!!

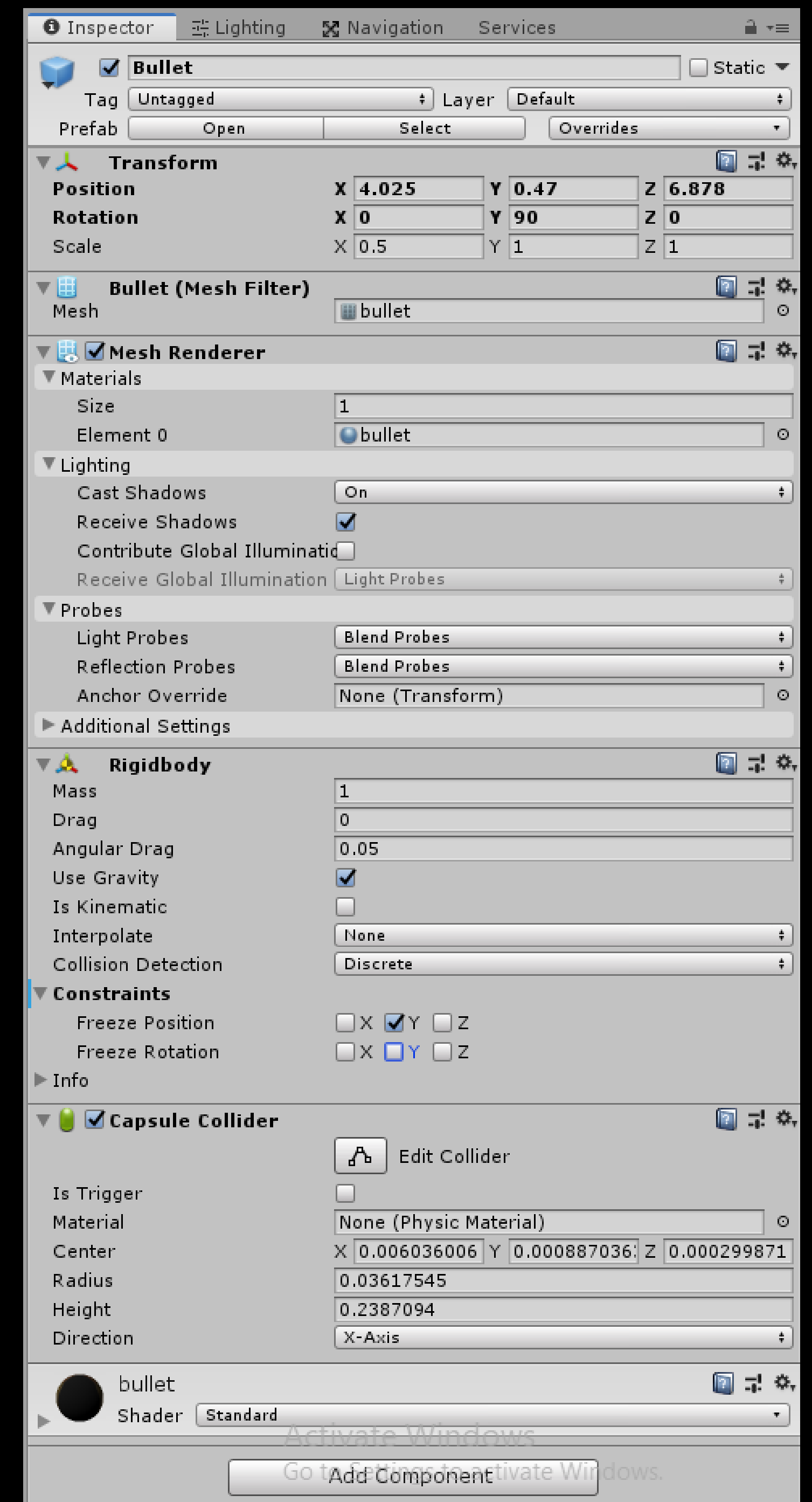
BULLET CHARACTERISTICS

Go to Prefabs -> Bullet -> Bullet. Drag and drop Bullet into the Hierarchy. Set the rotation and scale of bullet as follows.



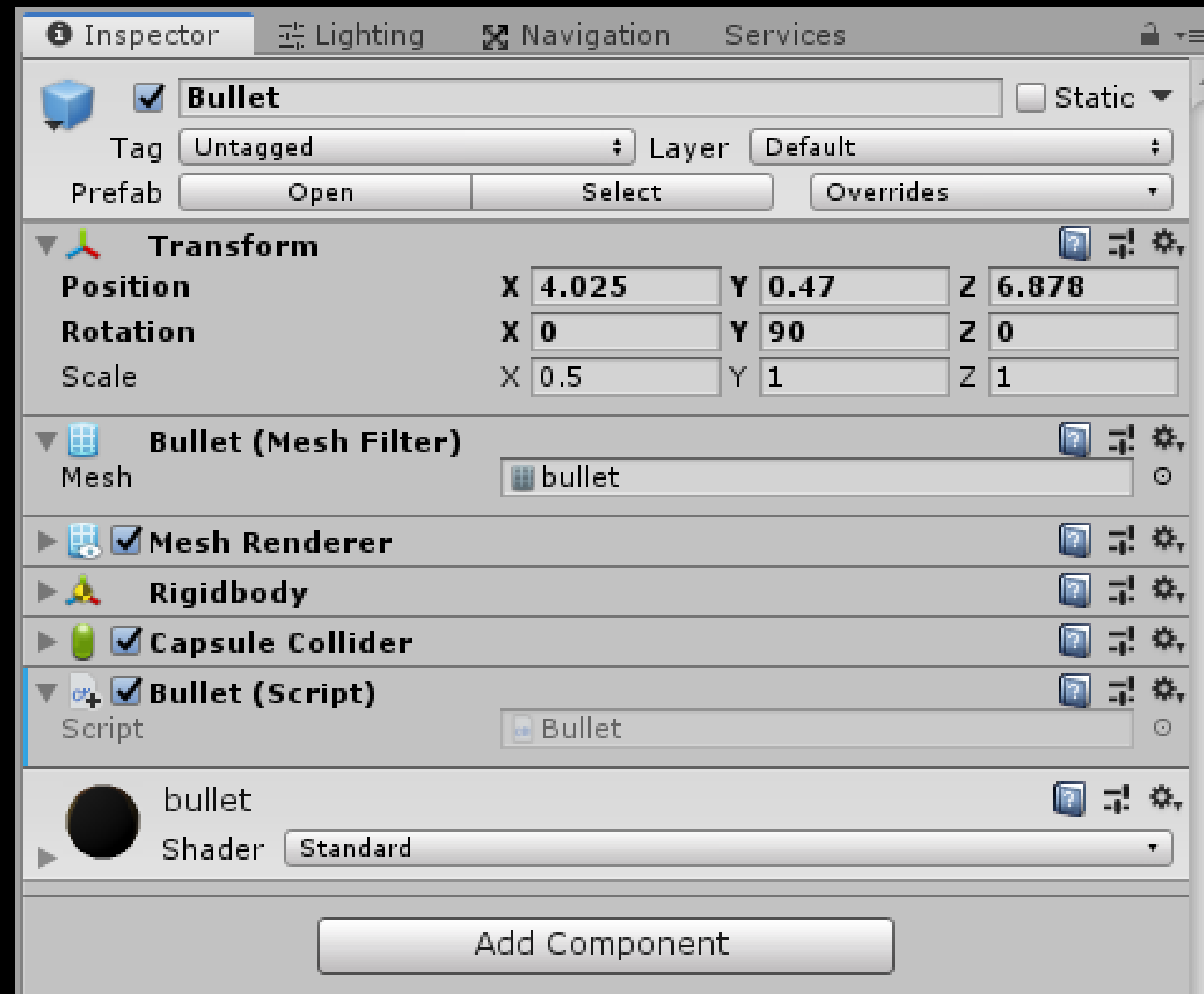
BULLET CHARACTERISTICS

Add Rigidbody and Capsule Collider components to the bullet to give it volume in world space. Adjust the radius and height of capsule collider. Free the y position of the Bullet in its Rigidbody component.



BULLET CHARACTERISTICS

In the Scripts folder, create a script called Bullet and attach it to the Bullet game-object.



BULLET CHARACTERISTICS

Add force to the Bullet through script at the time of its instantiation to get it moving.

```
public class Bullet : MonoBehaviour
{
    public float speed = 10f;

    0 references
    private void Start()
    {
        GetComponent<Rigidbody>().AddForce(transform.forward * speed, ForceMode.VelocityChange);
    }
}
```

BULLET CHARACTERISTICS

Handle the behaviour of the bullet when it collides with any object from the environment and also trigger player's death animation accordingly.

```
public class Bullet : MonoBehaviour
{
    public float speed = 10f;

    0 references
    private void Start()
    {
        GetComponent<Rigidbody>().AddForce(transform.forward * speed, ForceMode.VelocityChange);
    }

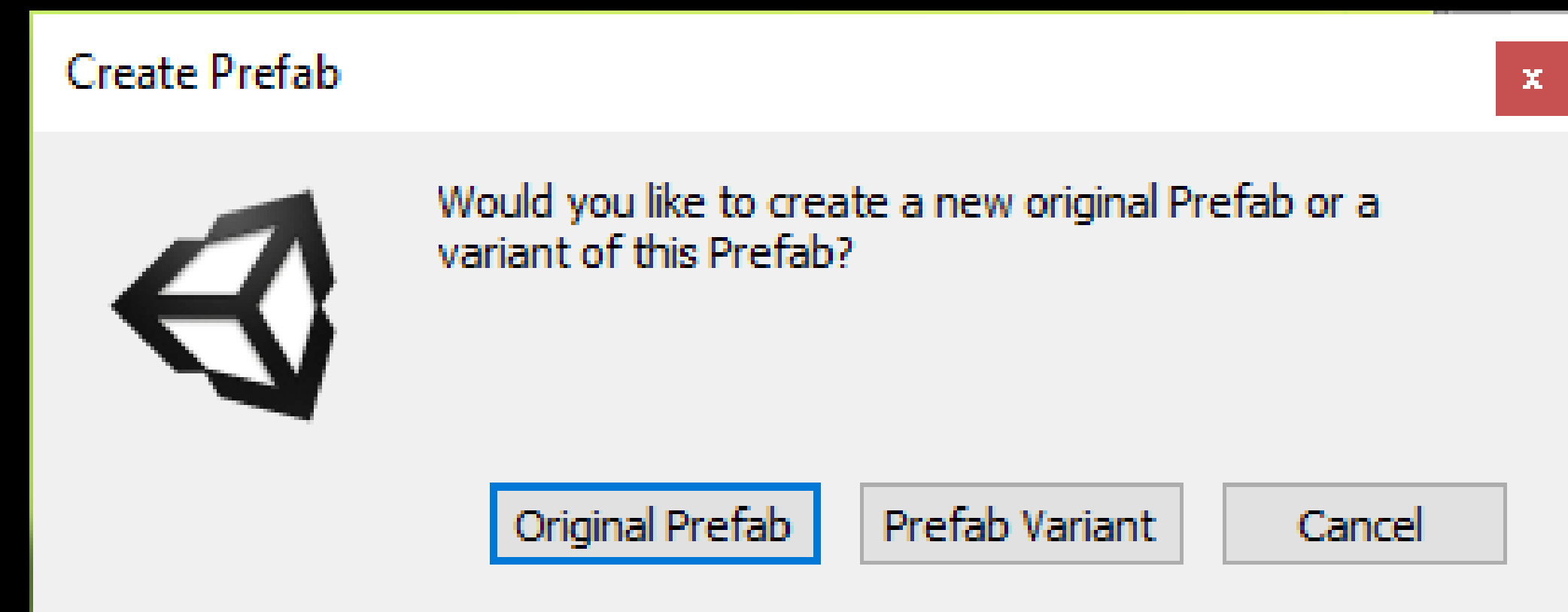
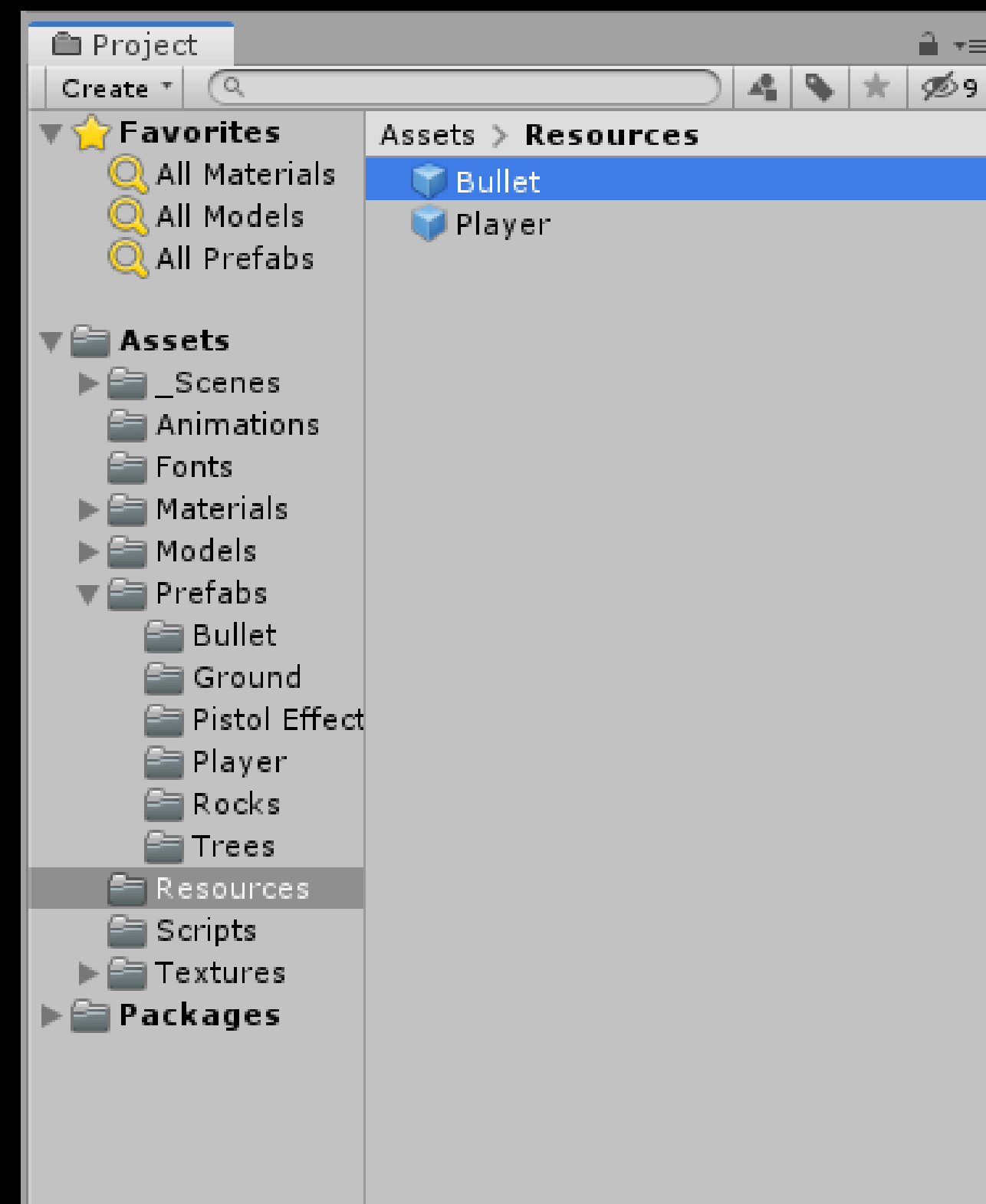
    0 references
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            collision.gameObject.GetComponent<PlayerMovement>().enabled = false;
            collision.gameObject.GetComponent<Animator>().SetTrigger("Dead");
        }

        Destroy(this.gameObject);
    }
}
```

SAVING BULLET & PLAYER AS PREFABS

In Project window, create a new folder and name it Resources. Drag and drop the Player and Bullet from Hierarchy into Resources folder to save them as Prefabs.

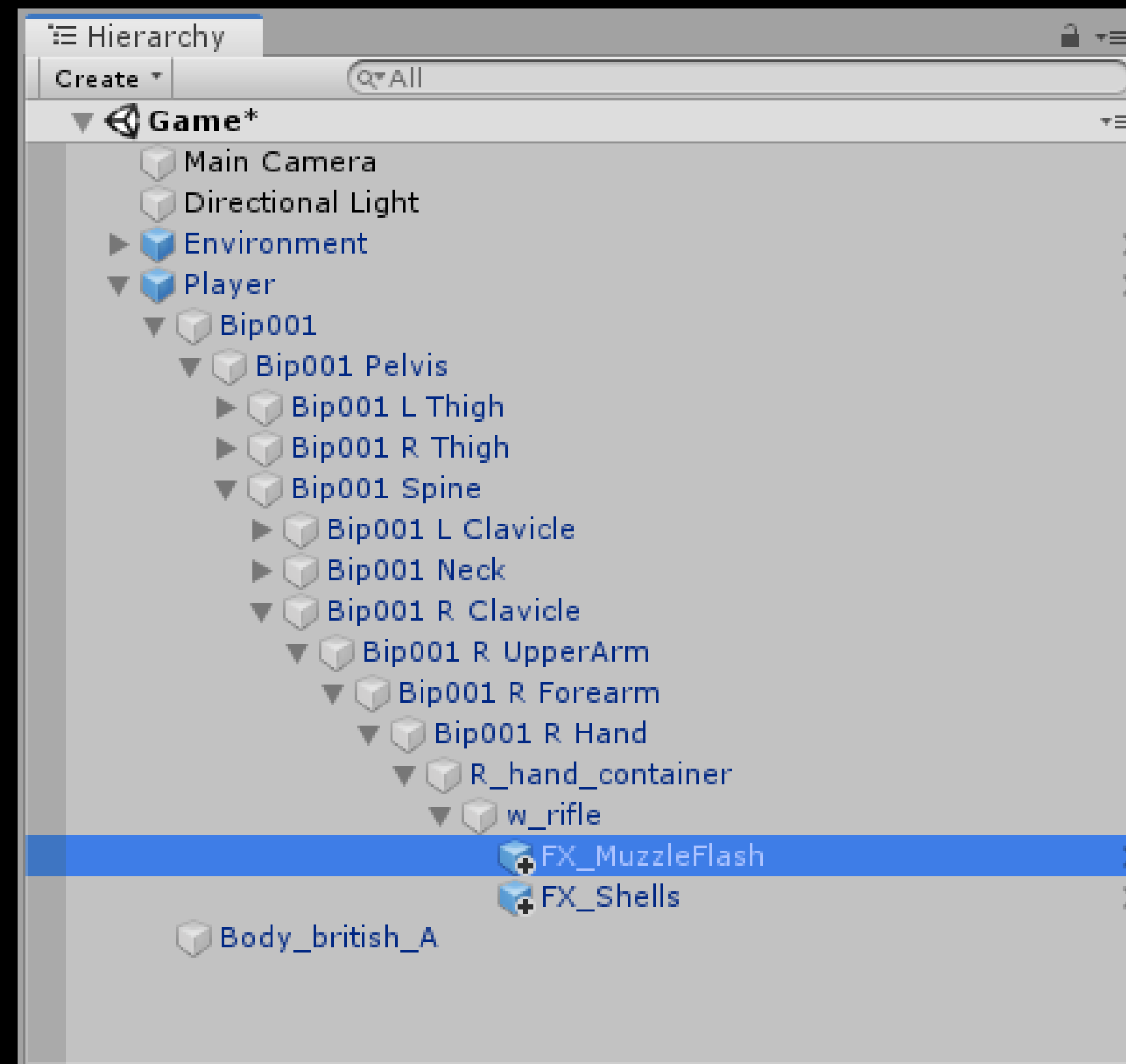
In the dialog box that pops up, select Original Prefab.



PLAYER SHOOTING

We need some effects to act as feedback for shooting, a muzzle flash and shells.

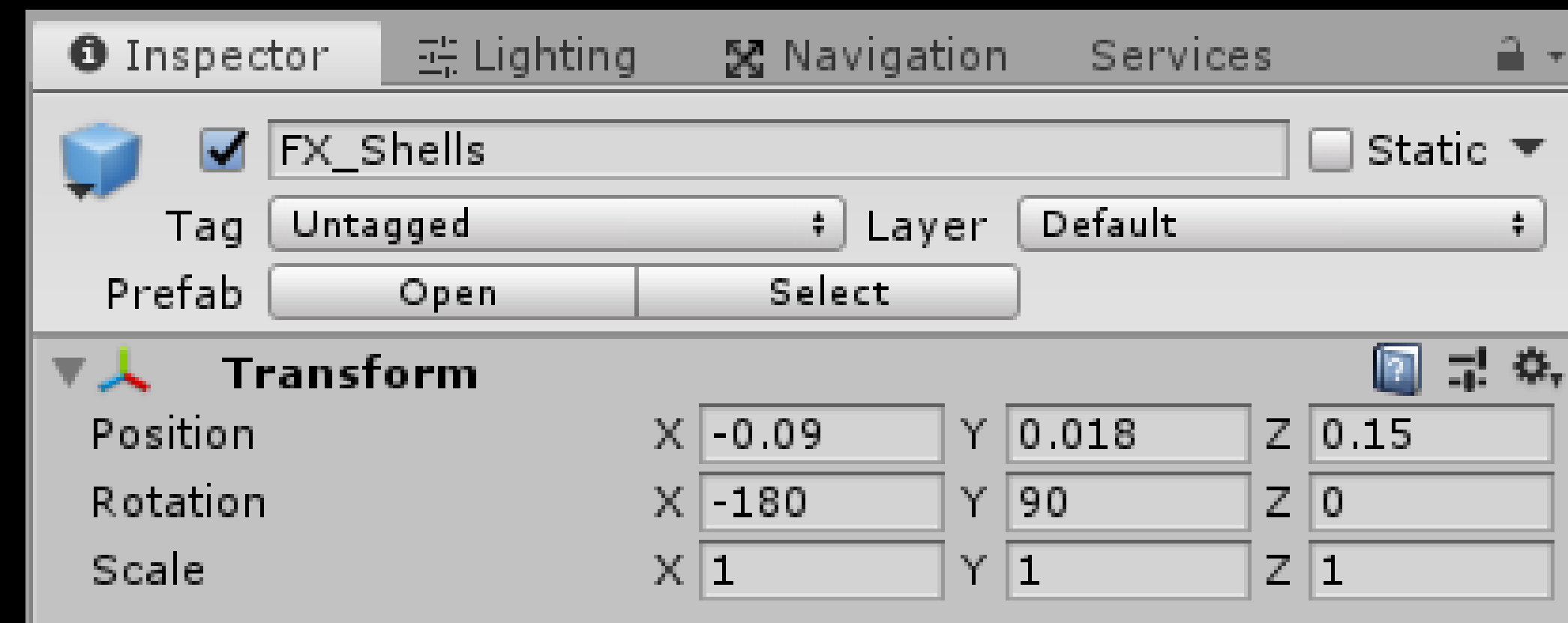
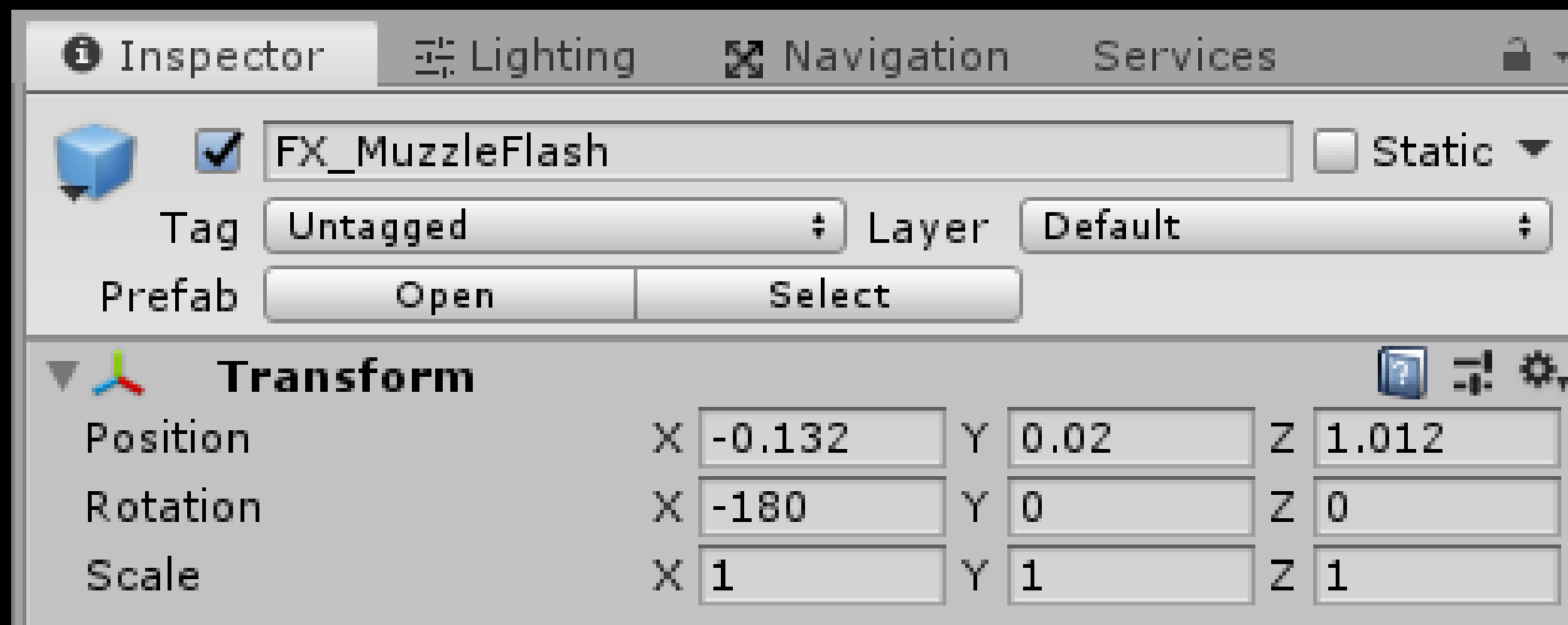
Go to Prefabs -> Pistol Effects -> FX_MuzzleFlash & FX_Shells. Add these game-object as a child to the player in the path shown.



PLAYER SHOOTING

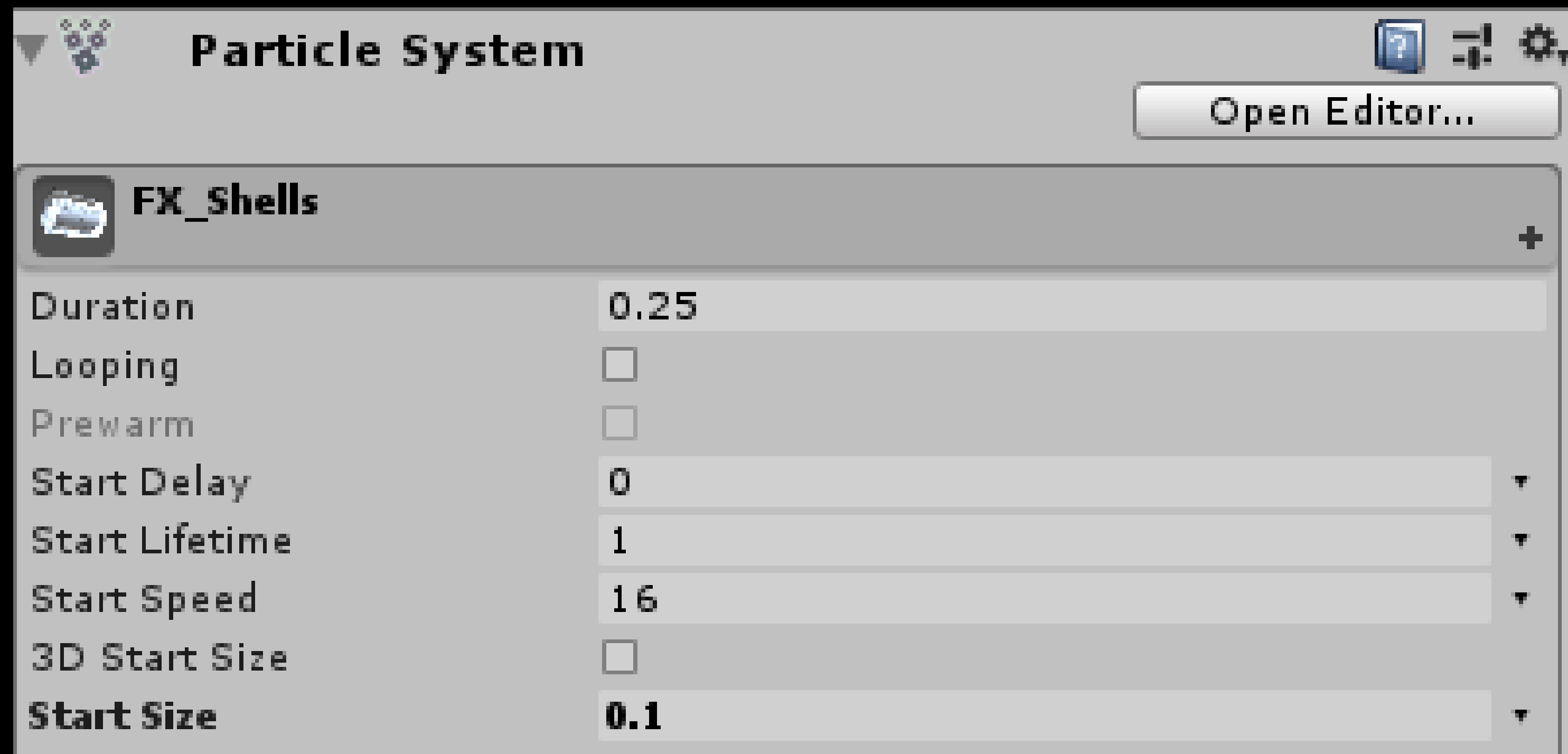
Set the transform component of FX_MuzzleFlash and FX_Shells as follows.

Apply all changes to the player prefab.



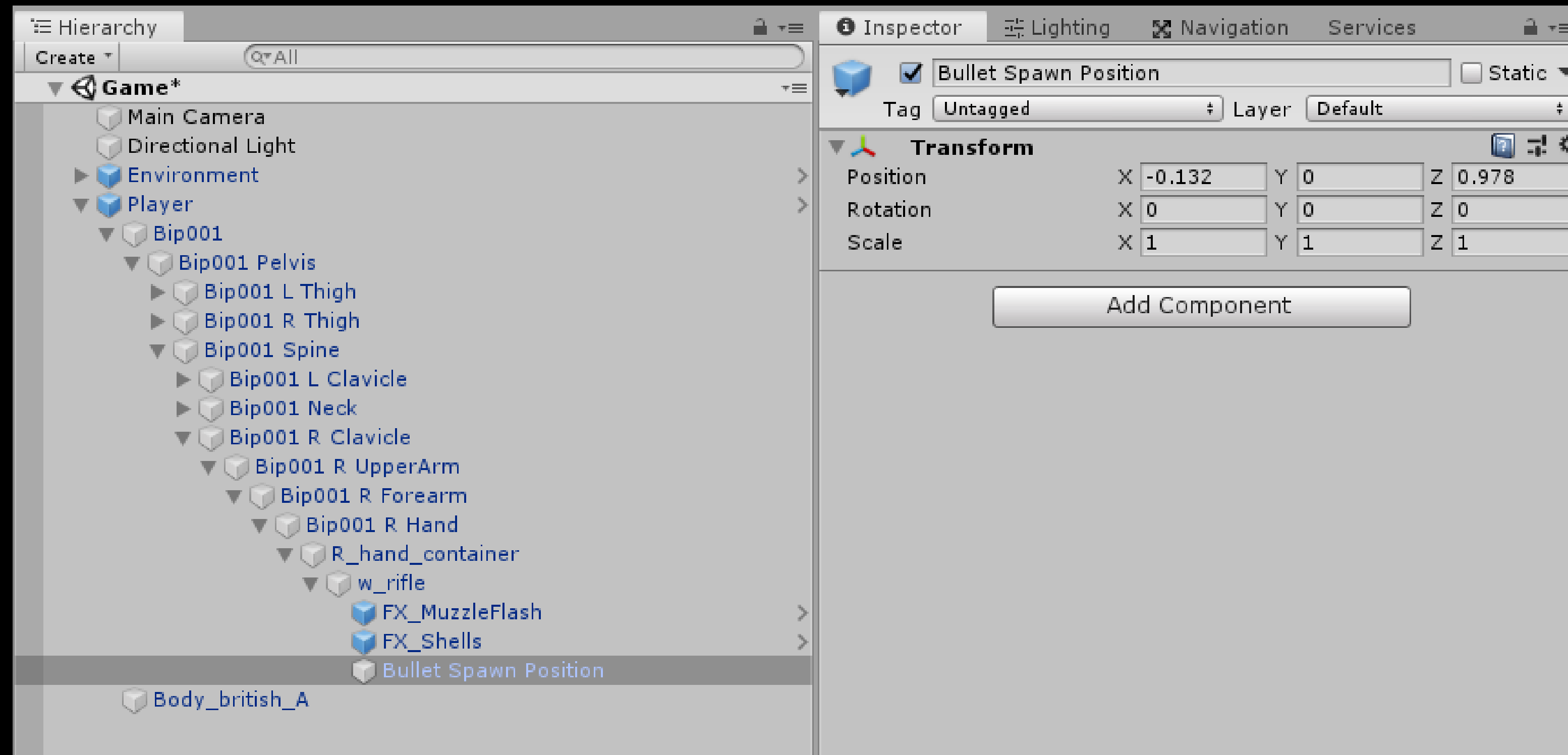
PLAYER SHOOTING

Set the start size of FX_Shells to 0.1



PLAYER SHOOTING

Create an empty game-object to refer to bullet's spawn position at the nozzle of the rifle and set its transform to follows.



PLAYER SHOOTING

Create a script called 'PlayerShooting' and add it to the Player.

```
public class PlayerShooting : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    void Start()
    {
    }

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


PLAYER SHOOTING

Instantiate the bullet at its spawn position and play the particle effects on mouse click.

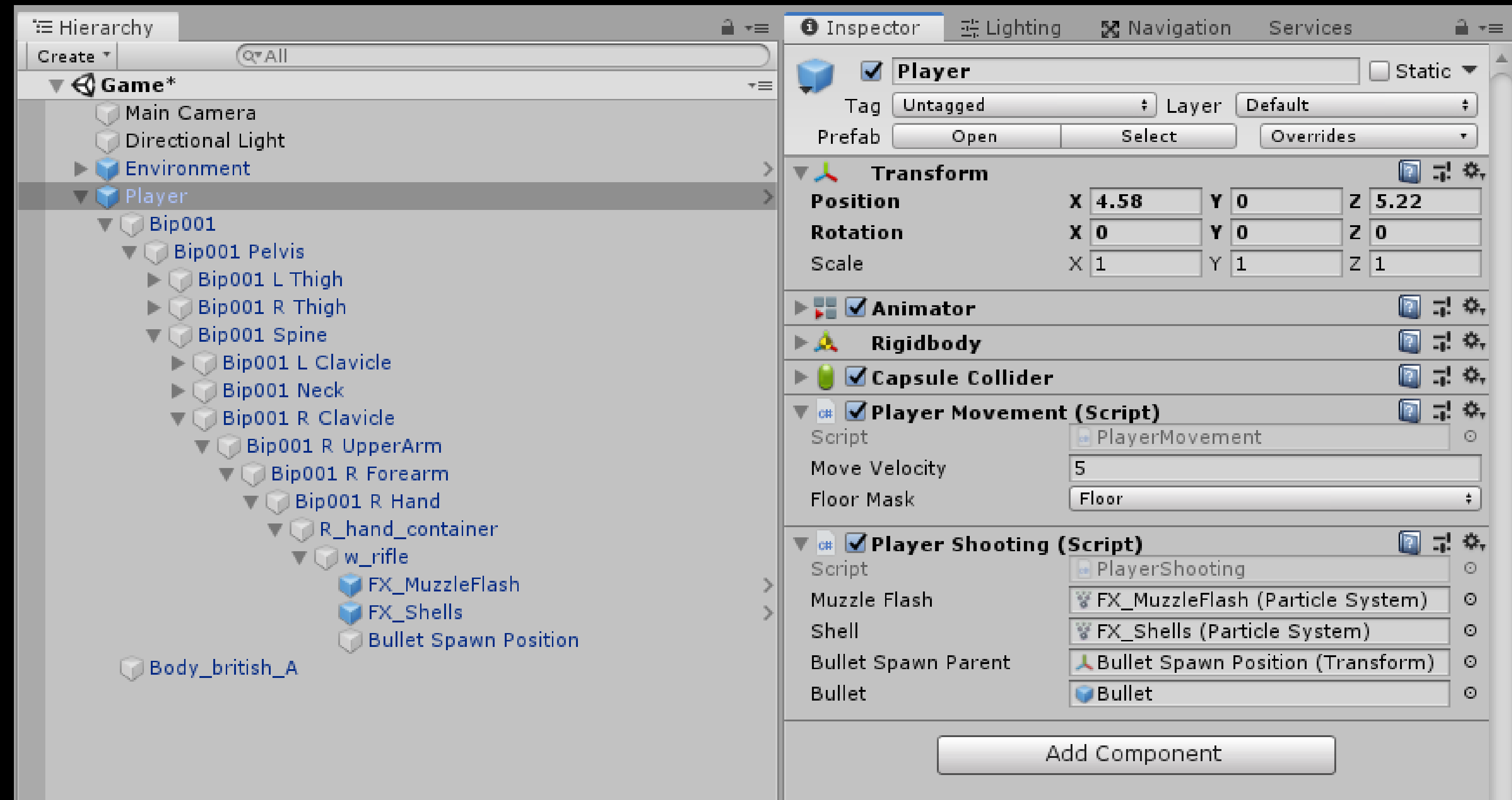
```
public class PlayerShooting : MonoBehaviour
{
    public ParticleSystem muzzleFlash;
    public ParticleSystem shell;
    public Transform bulletSpawnParent;
    public GameObject bullet;

    0 references
    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            PlayShootEffect();
            Instantiate(bullet, bulletSpawnParent.position, this.transform.localRotation);
        }
    }

    1 reference
    public void PlayShootEffect()
    {
        this.muzzleFlash.Play();
        this.shell.Play();
    }
}
```

PLAYER SHOOTING

Assign the public variables as follows in the Inspector window of the Player and apply the changes to the prefab.

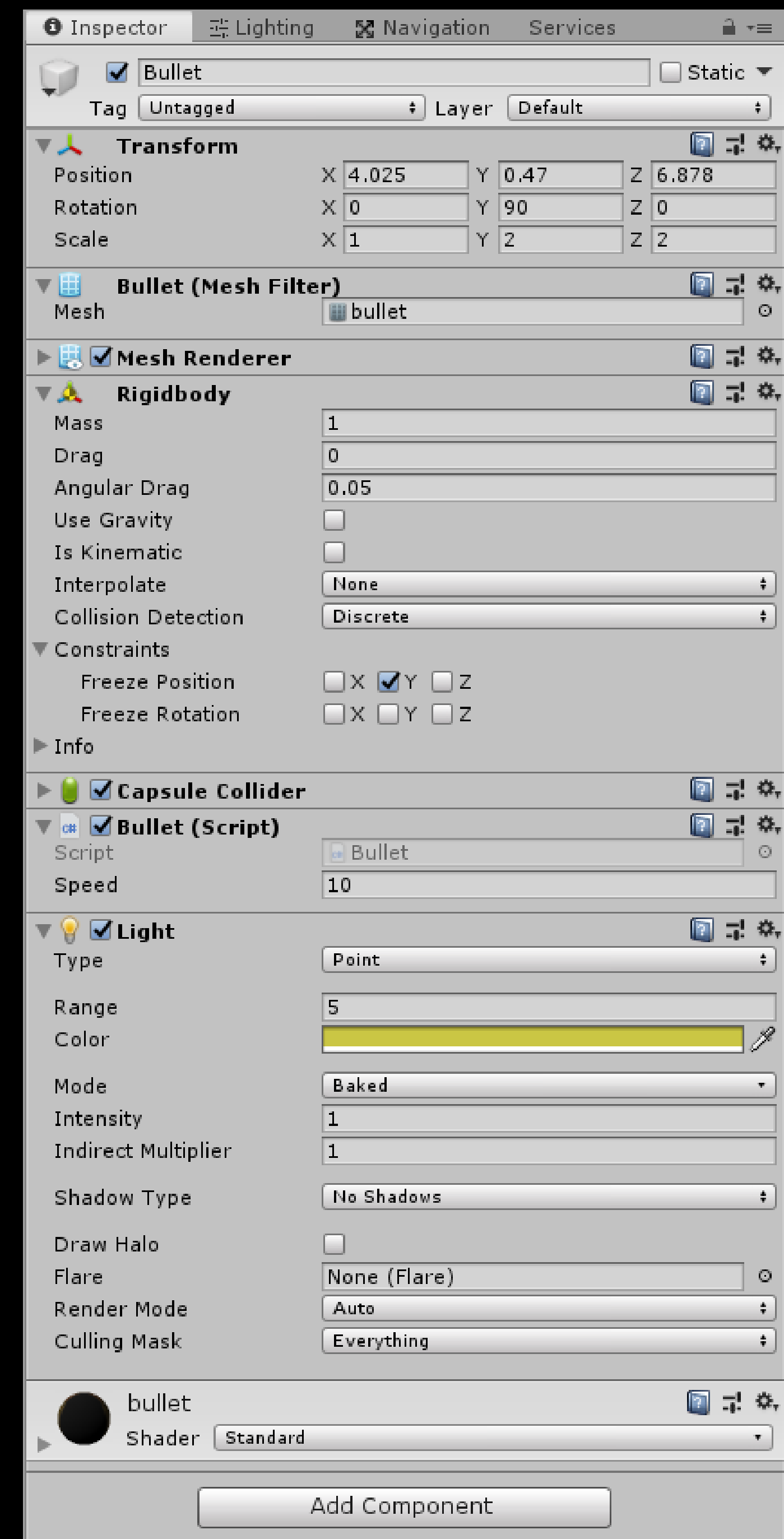


Time for our another test run !!!

PLAYER SHOOTING

The bullet looks too subtle right?
How about adding a point light?

Color code: CAC644

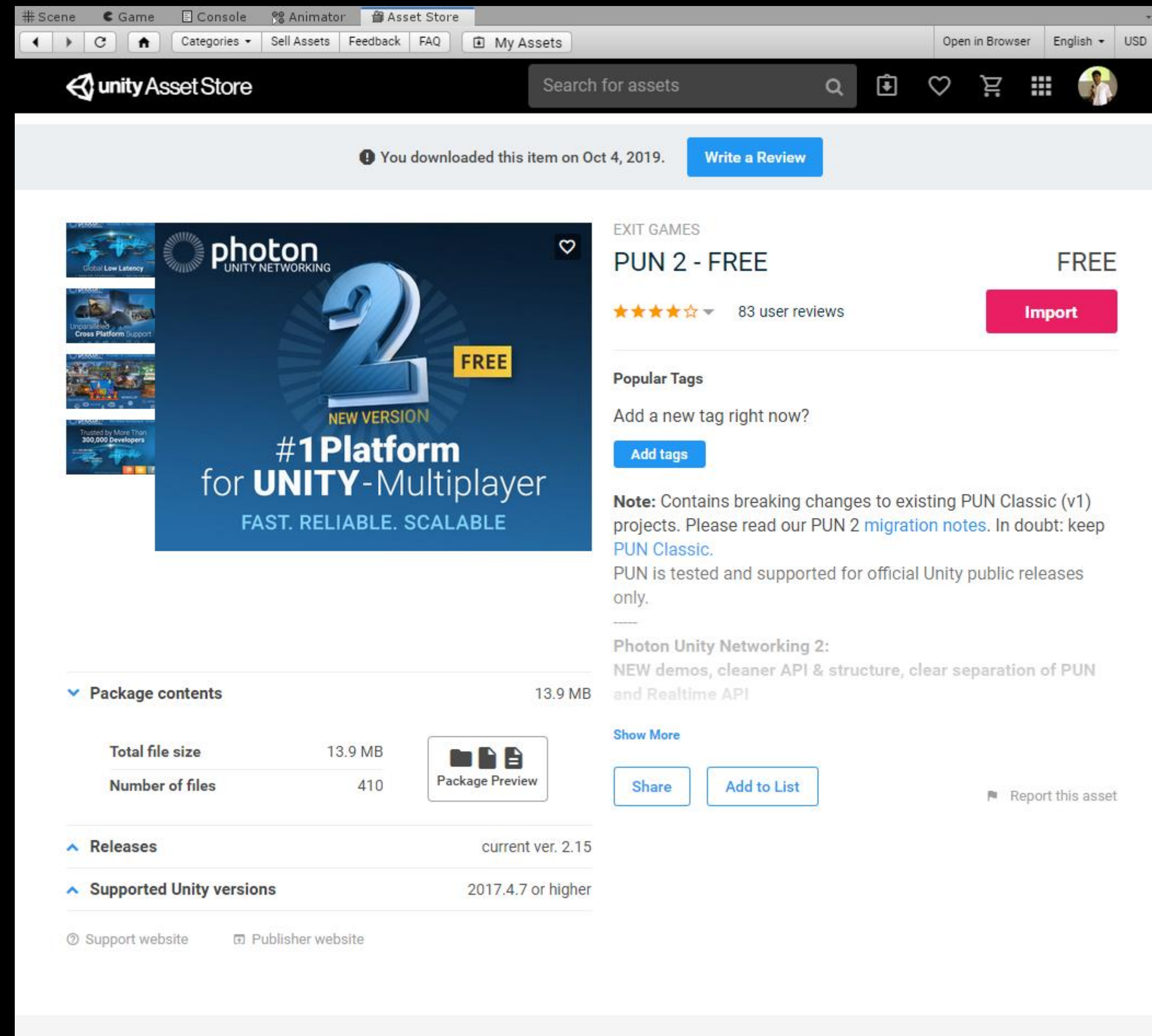


**And now we have a complete single
player game !!!**

Time for making it multiplayer !!!

PUN2

Go to Window -> Asset Store. Search for 'PUN2 – FREE'. Download and import the asset.



PUN2

Add the AppID from your Photon dashboard.



MULTIPLYAER SCRIPTS

Uncomment the code from LobbyController, RoomController, RoomButton & NetwrokManager.

```
/*using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

public class NetworkManager : MonoBehaviourPunCallbacks
{
    private void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    public override void OnConnectedToMaster()
    {
        Debug.Log("Connected to " + PhotonNetwork.CloudRegion + " server!");
    }
}*/
```

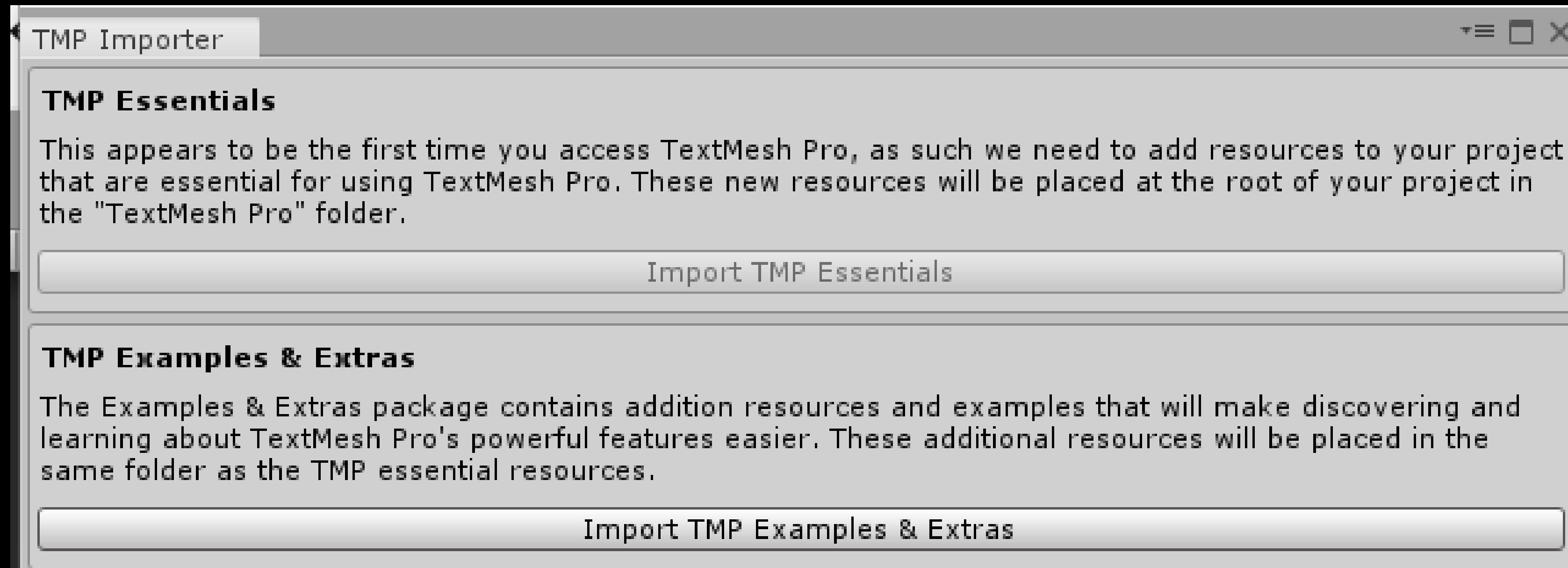
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;

0 references
public class NetworkManager : MonoBehaviourPunCallbacks
{
    0 references
    private void Start()
    {
        PhotonNetwork.ConnectUsingSettings();
    }

    14 references
    public override void OnConnectedToMaster()
    {
        Debug.Log("Connected to " + PhotonNetwork.CloudRegion + " server!");
    }
}
```

MENU SCENE

Import TMP Essentials when prompted. Reload the scene for changes to take effect.

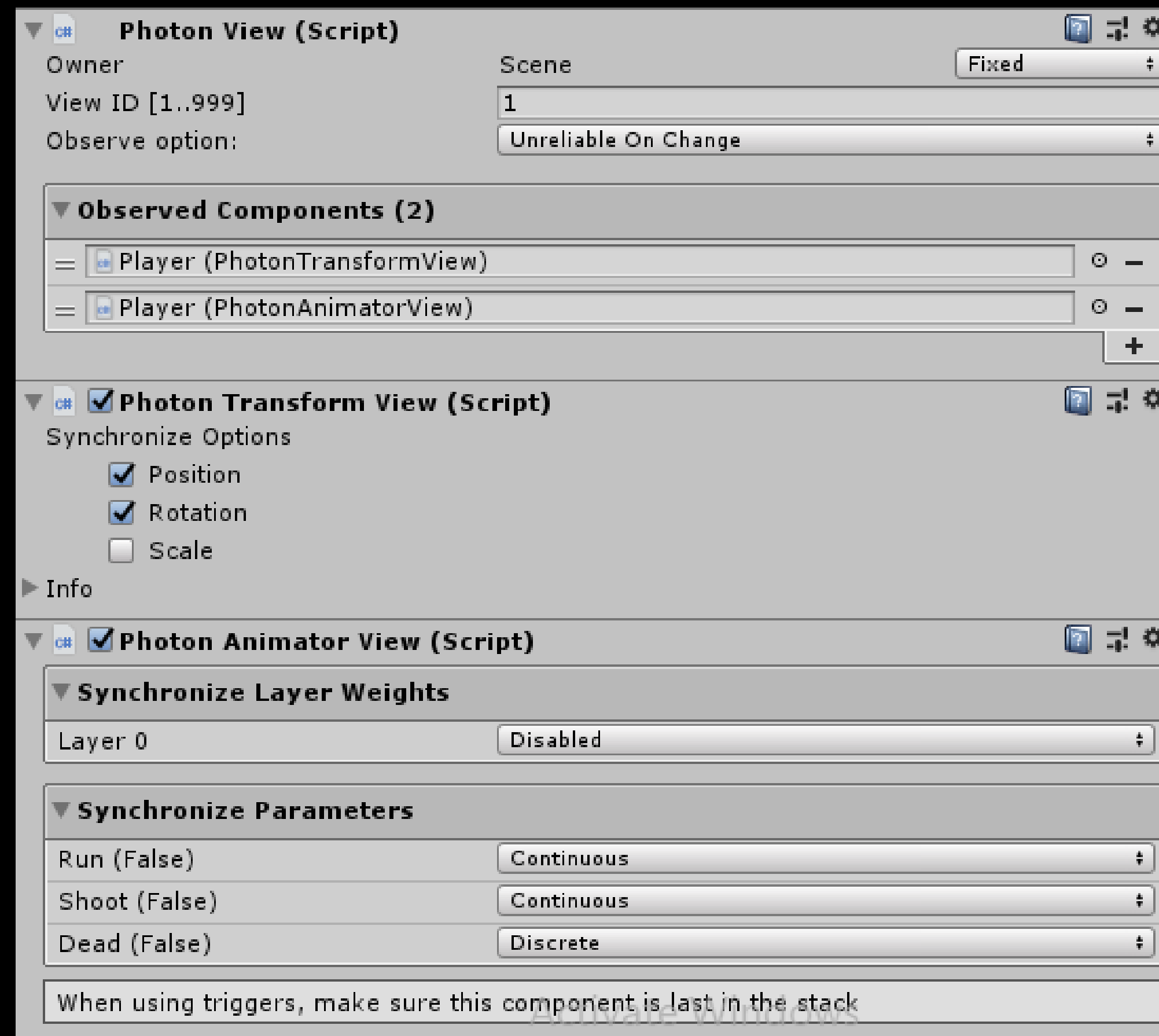


MULTIPLAYER FUNCTIONALITY

We need to send out the in-game changes in one instance of the game over the local area network to another instance of the game. These changes include position of player, bullet, their animations, effects and all others. For achieving that, let us add some predefined scripts to the game-objects.

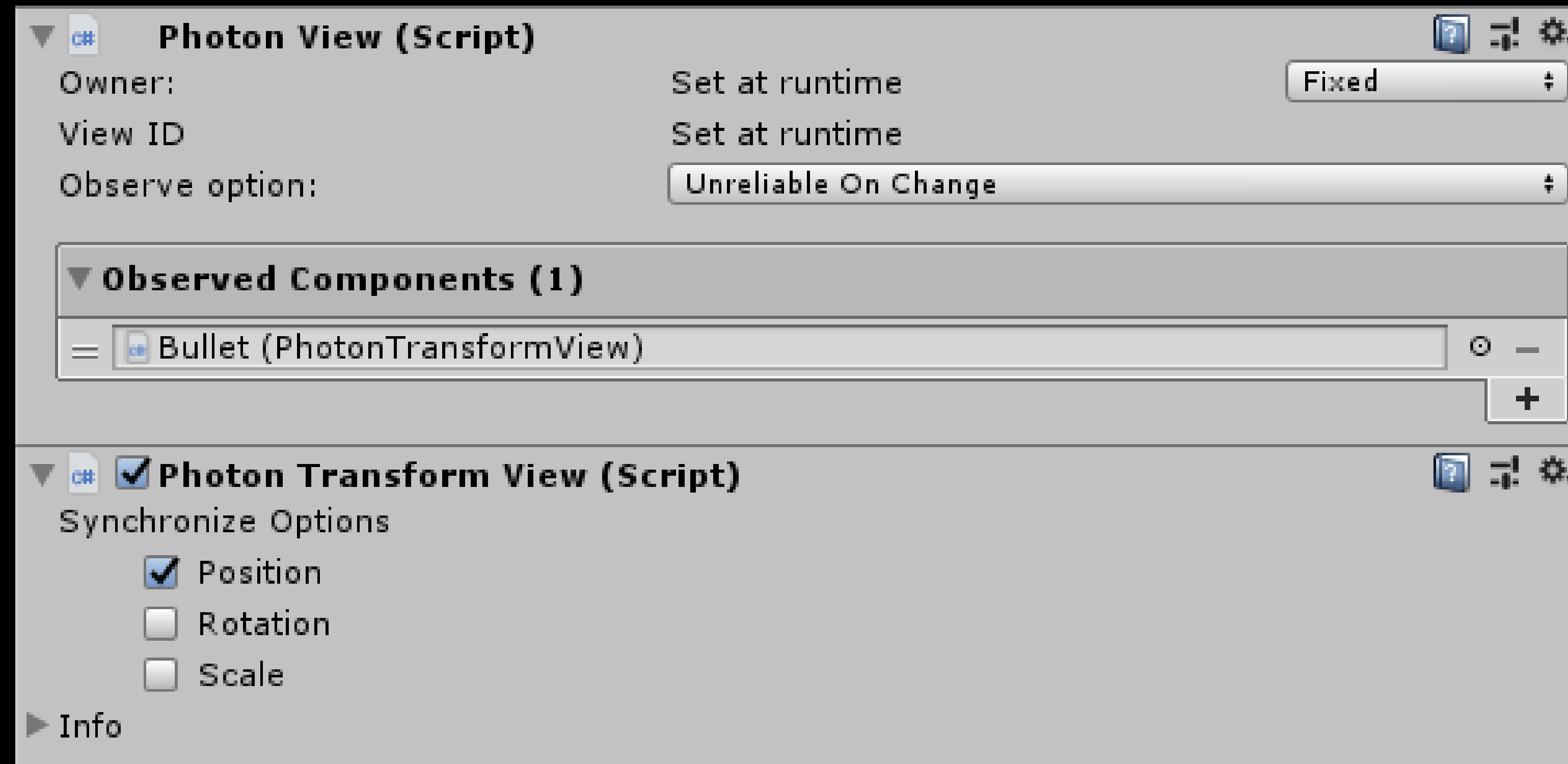
MULTIPLAYER FUNCTIONALITY

To the player prefab, add the following scripts through inspector, Photon View, Photon Transform View, Photon Animator View. Drag and drop Photon Transform View and Photon Animator View into the Observed Components field of Photon View.



MULTIPLAYER FUNCTIONALITY

To the bullet prefab, add the following scripts through inspector, Photon View and Photon Transform View. Drag and drop Photon Transform View into the Observed Components field of Photon View.



MULTIPLAYER FUNCTIONALITY

To send information about changes you're doing to game-object through code, you need to define the change as [PunRPC] categorised function.

Also, to instantiate game-object in all instances across the network, we use predefined method 'PhotonNetwork.Instantiate()'.

If in case you are running two instance of the game on the same machine for test purposes, we need to make sure that changes we make in a particular instance are not recorded by the other instance as well. So we need to run the code only if the are in our specific instance only and also only if it is still connected to photon network.

Making all the necessary changes, the final scripts shall be as follow.

PLAYER SHOOTING

```
using UnityEngine;
using Photon.Pun;

1 reference
public class PlayerShooting : MonoBehaviourPun
{
    public ParticleSystem muzzleFlash;
    public ParticleSystem shell;
    public Transform bulletSpawnParent;

    private GameObject bullet;

    0 references
    private void Update()
    {
        if (photonView.IsMine == false && PhotonNetwork.IsConnected == true)
        {
            return;
        }

        if (Input.GetMouseButtonDown(0))
        {
            GetComponent<PhotonView>().RPC("PlayShootEffect", RpcTarget.AllBuffered);
            bullet = PhotonNetwork.Instantiate("Bullet", bulletSpawnParent.position, this.transform.localRotation);
        }
    }

    [PunRPC]
    0 references
    public void PlayShootEffect()
    {
        this.muzzleFlash.Play();
        this.shell.Play();
    }
}
```

PLAYER MOVEMENT

```
using UnityEngine;
using Photon.Pun;

1 reference
public class PlayerMovement : MonoBehaviourPun
{
    public float moveVelocity = 5f;
    public LayerMask floorMask;

    private float moveHorizontal;
    private float moveVertical;
    private Rigidbody playerRigidbody;
    private Animator animator;

    0 references
    private void Start()
    {
        playerRigidbody = GetComponent<Rigidbody>();
        animator = GetComponent<Animator>();
    }

    0 references
    private void FixedUpdate()
    {
        if (photonView.IsMine == false && PhotonNetwork.IsConnected == true)
        {
            return;
        }

        moveHorizontal = Input.GetAxisRaw("Horizontal");
        moveVertical = Input.GetAxisRaw("Vertical");

        Move(moveHorizontal, moveVertical);
        TurnAround();
        AnimatePlayer(moveHorizontal, moveVertical);
    }
}
```

```
1 reference
private void Move(float moveHorizontal, float moveVertical)
{
    transform.Translate(Vector3.forward * moveVertical * moveVelocity * Time.deltaTime);
    transform.Translate(Vector3.right * moveHorizontal * moveVelocity * Time.deltaTime);
}

1 reference
private void TurnAround()
{
    Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit floorHit;

    if (Physics.Raycast(camRay, out floorHit, 100f, floorMask))
    {
        Vector3 playerToFocussedPoint = floorHit.point - this.transform.position;
        playerToFocussedPoint.y = 0f;
        Quaternion newRotation = Quaternion.LookRotation(playerToFocussedPoint);
        playerRigidbody.MoveRotation(newRotation);
    }
}

1 reference
private void AnimatePlayer(float moveHorizontal, float moveVertical)
{
    bool walking = (moveHorizontal != 0f) || (moveVertical != 0f);

    animator.SetBool("Run", walking);
}
}
```


BULLET

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

0 references
public class Bullet : MonoBehaviourPun
{
    public float speed = 10f;

    0 references
    private void Start()
    {
        GetComponent<Rigidbody>().AddForce(transform.forward * speed, ForceMode.VelocityChange);
    }

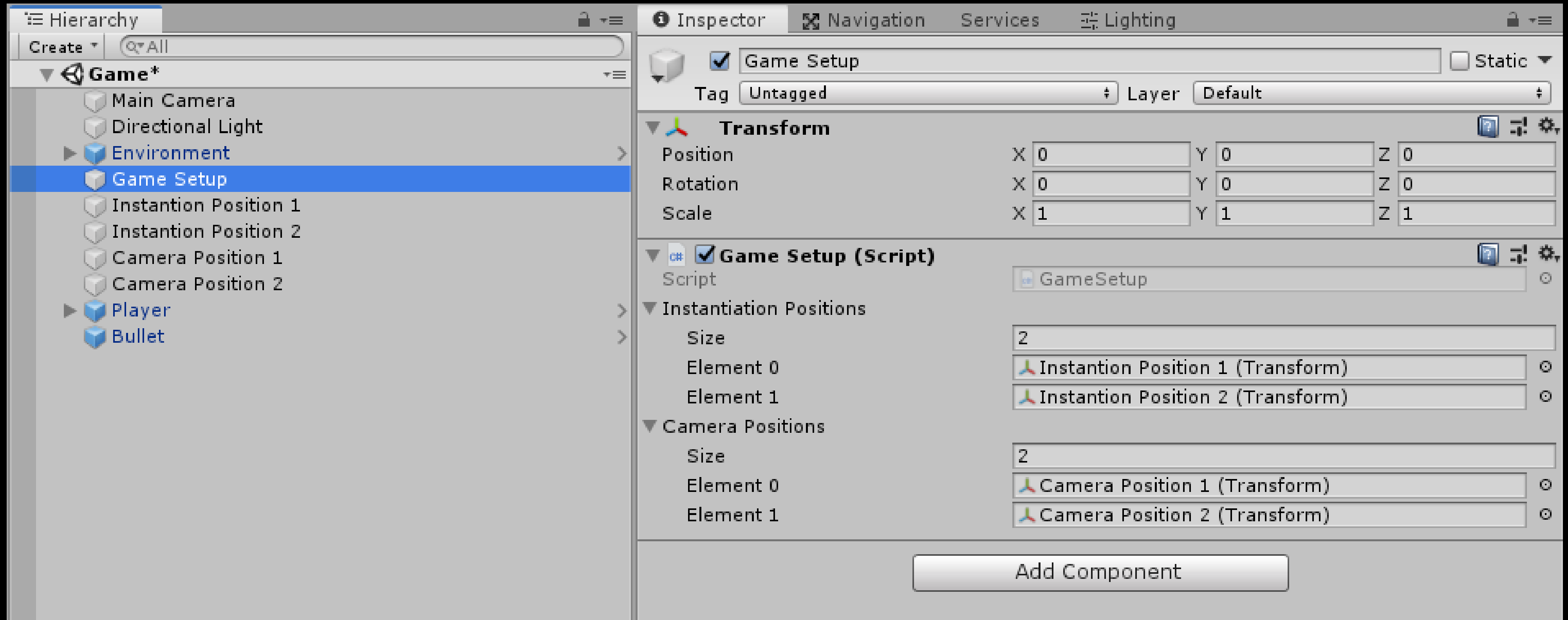
    0 references
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            collision.gameObject.GetComponent<PlayerMovement>().enabled = false;
            collision.gameObject.GetComponent<PlayerShooting>().enabled = false;

            collision.gameObject.GetComponent<Animator>().SetTrigger("Dead");
        }

        Destroy(this.gameObject);
    }
}
```

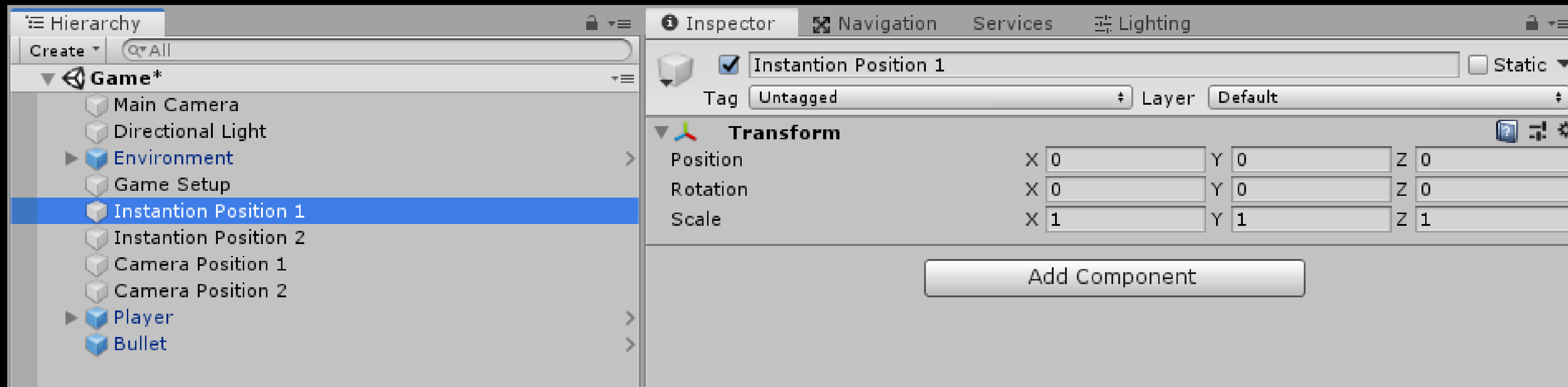
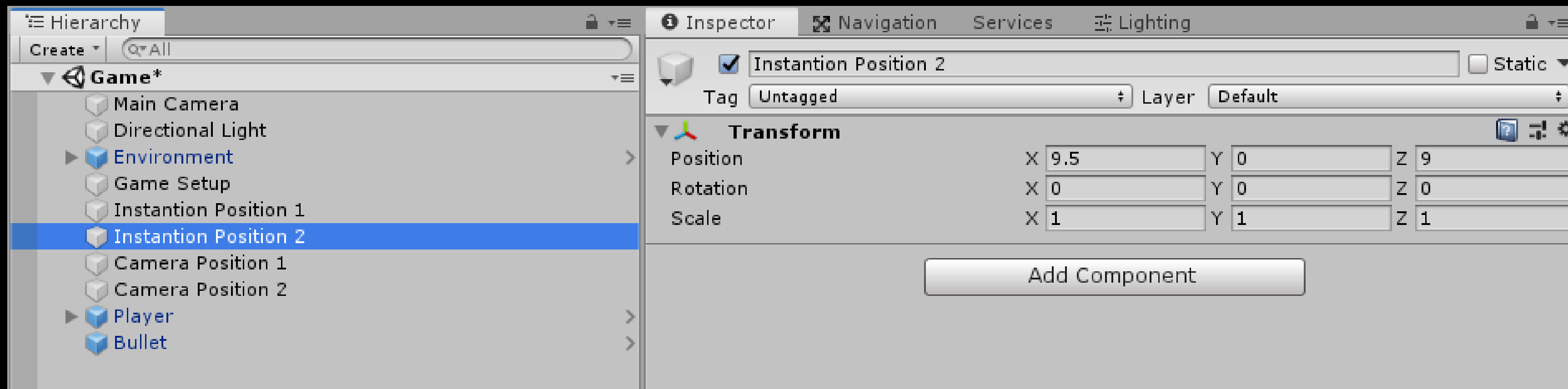
GAME SETUP

Let us not hard code the player's initial position in game. Create an empty game-object and name it Game Setup. Create a C# script called 'GameSetup' and attach it to Game Setup game-object.



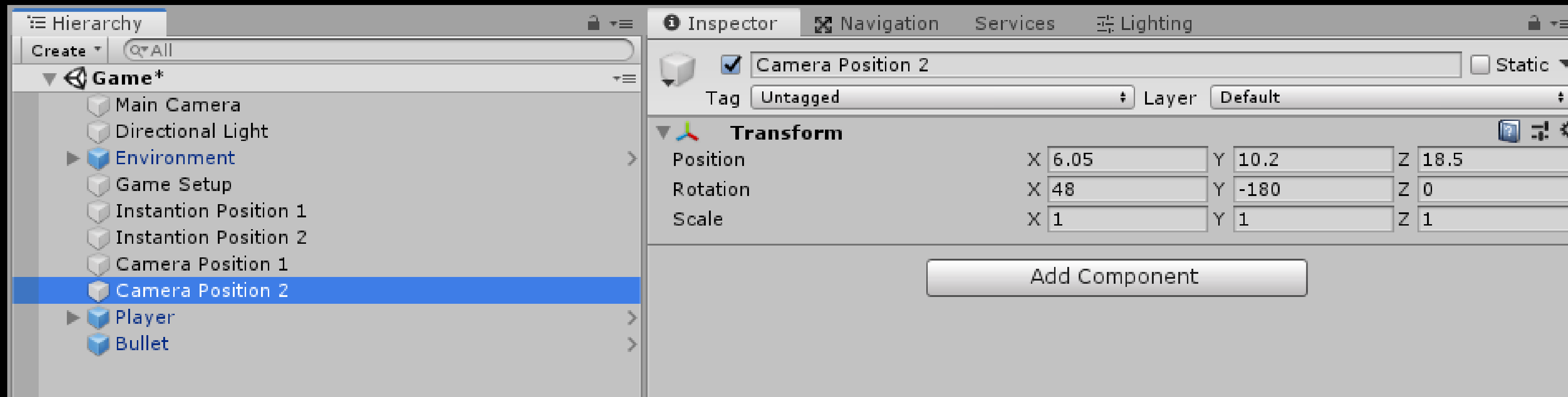
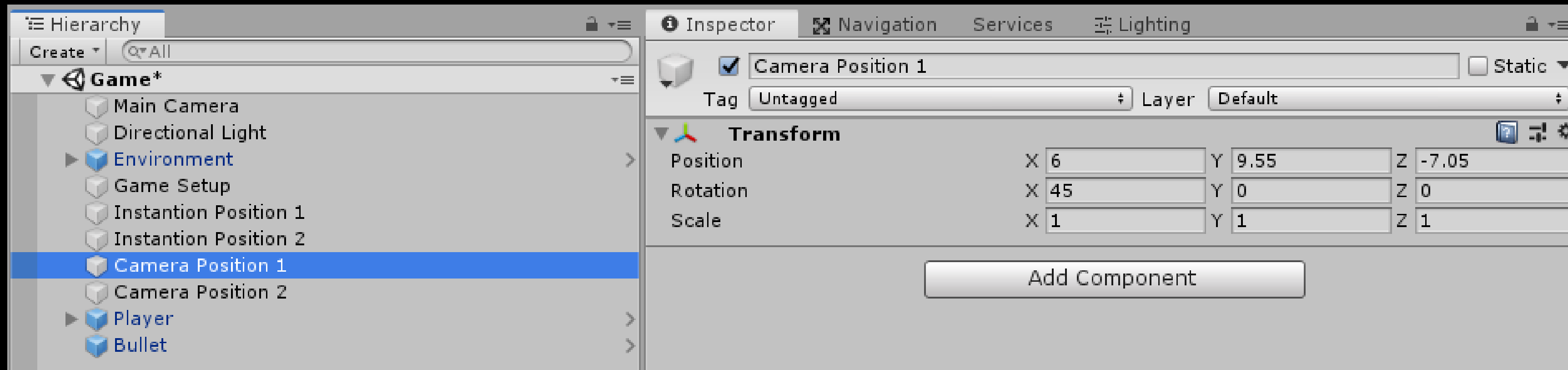
GAME SETUP

Decide as to how many random instantiation position you want to have in your game and create equal number of empty game object and place them in the desired location.



GAME SETUP

Similarly have a corresponding camera position for each initial position of the player and create empty game-objects to refer to the position and rotation.



GAME SETUP

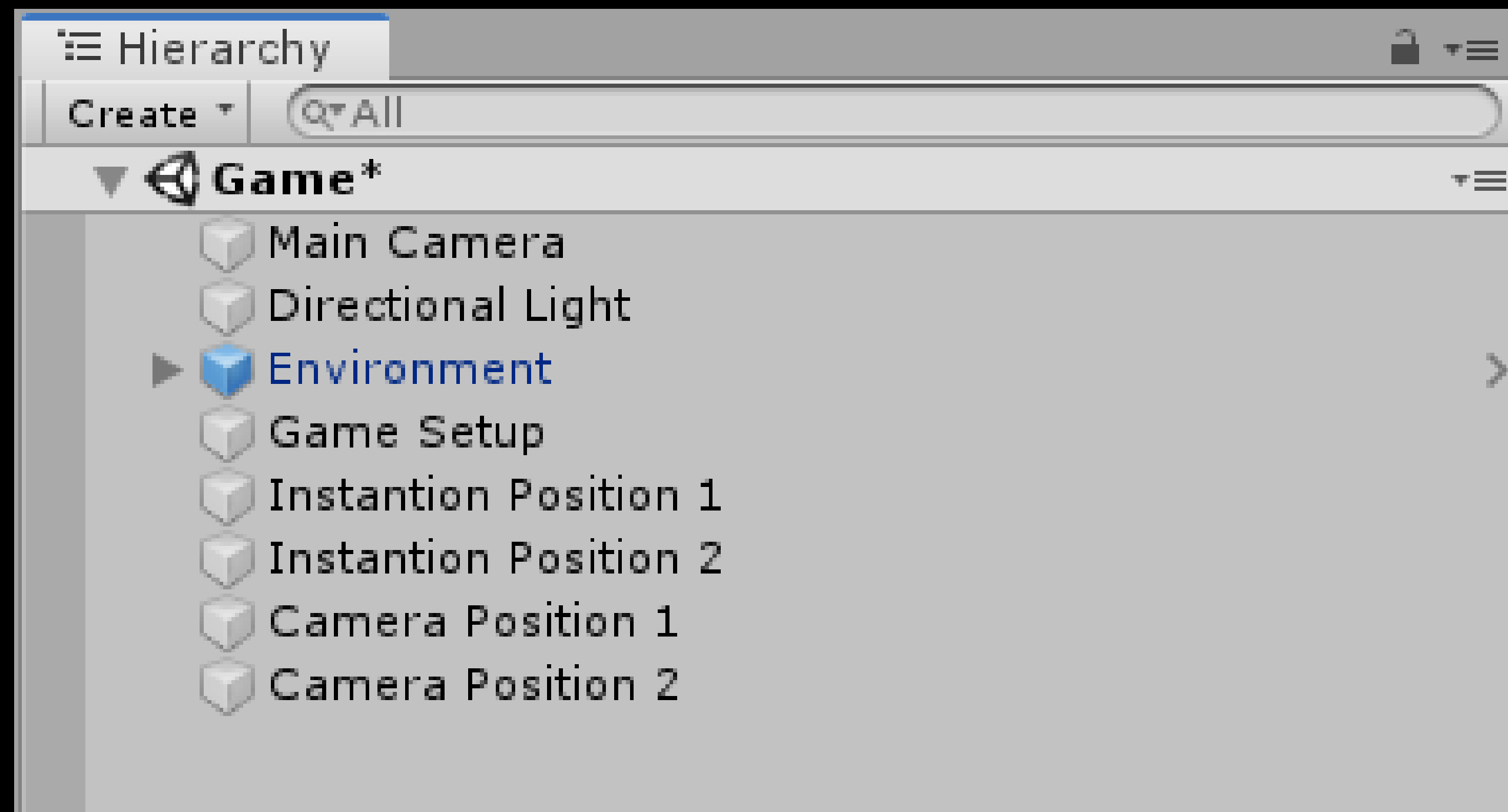
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Photon.Pun;
using System.IO;

0 references
public class GameSetup : MonoBehaviour
{
    public Transform[] instantiationPositions;
    public Transform[] cameraPositions;

    0 references
    private void Start()
    {
        Debug.Log("Creating Player");
        int randomNumber = Random.Range(0, instantiationPositions.Length);
        Debug.Log(randomNumber);
        Camera.main.transform.position = cameraPositions[randomNumber].position;
        Camera.main.transform.localRotation = cameraPositions[randomNumber].localRotation;
        PhotonNetwork.Instantiate("Player", instantiationPositions[randomNumber].position, Quaternion.identity);
    }
}
```

CLEAN THE HIERARCHY

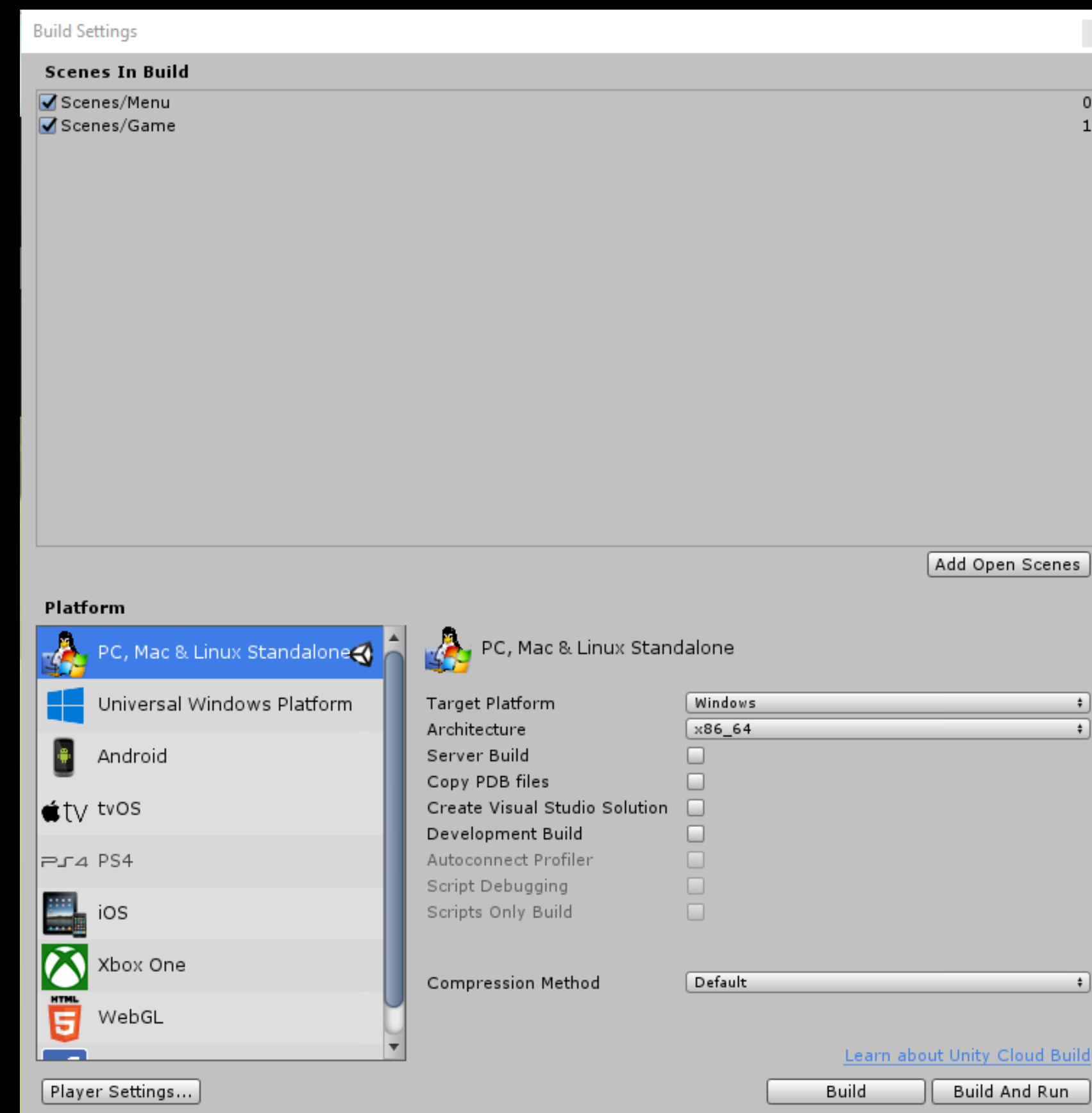
Delete the Player and Bullet game-objects from Hierarchy of Game scene and change to Menu scene.



BUILD THE GAME

Time to finally build and run the game !

Go to File -> Build Settings. Add the scenes in the given order, make sure the build platform is set to Windows and click on Build.



And we're done !!!