# Advanced FPS Multiplayer Template

# Inhoudsopgave

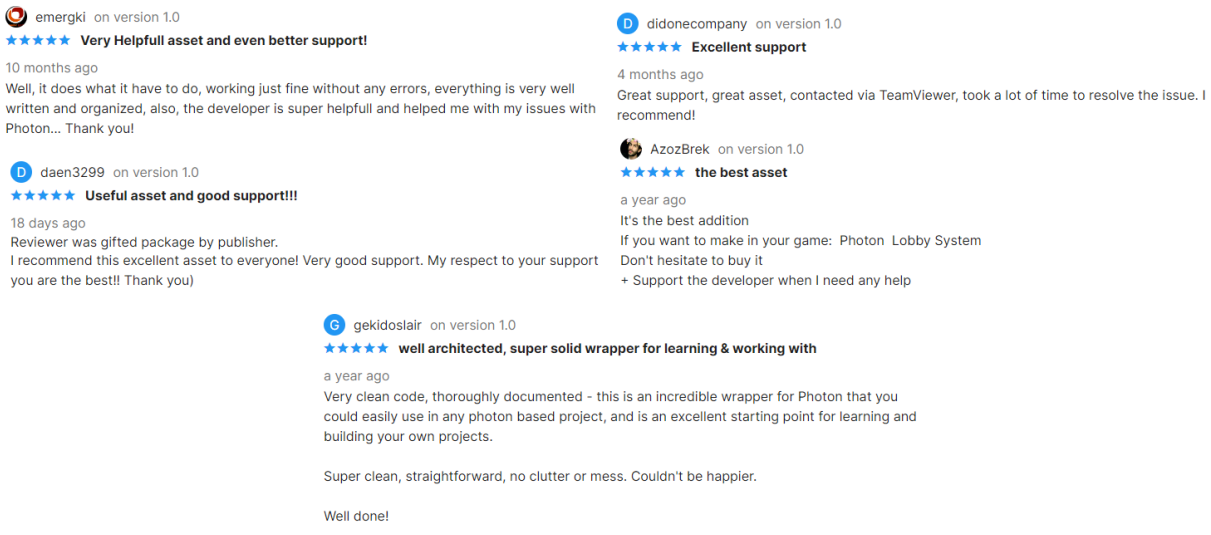
# 1 Introduction

First of all we would like to thank you for purchasing our Multiplayer FPS Template. This document provides information about the project and accompanying explanation. It is important that you read this document carefully as it contains useful information to get you started.

**ForceCode** is a small Unity developing team with almost 7 years' experience. We have been working on 2D and 3D customer projects. Our goal is to help people who want to make their own game. We try to keep our codes understandable for our customers with explanation comments.

We strive to continuously improve our structure of working. Our template is provided with dynamic scripts and structured folders. The features are divided in the *Core* folder.

ForceCode is known for good support. Please don't doubt to ask for help!



# 2 Multiplayer solution

Advanced FPS Template is created using Photon 2 networking. Photon is a real-time multiplayer game development framework. Photon allows it to create multiplayer projects easily and get members playing online. This solution is free to use, with a CCU capped depending on the license type you hold.

For adding custom networking to Advanced FPS template please check the official Photon website for documentation: https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro.

## 2.1 Matchmaking

We have created an custom matchmaking system to find available rooms. When the player is searching for an available room using matchmaking/pre-game lobby, the system is waiting a few seconds before it tries to join a random available room. If there are no rooms available, the system is creating a room automatically in a few seconds. The game will start in a few seconds when there are enough players. To get in a room without waiting on other players, you can create manually a room.

## 2.2 General Photon features

This system will be used to create and join rooms. You can make a room manually or you can use the matchmaking system. When you are creating a room manually, you have to setup the room name, max players, game mode and the map yourself. If you choose the matchmaking, the system will apply random room options. It is possible to see all available rooms in the room browser. On each room browser entry object, there is an join button to get in the room.

# 3 Project Structure

We try to keep our projects as tidy as possible. For example, we create divisions within our folders and scripts to separate categories. This will make it easier for our users to understand and heal any extensions.

## 3.1 Scriptable objects

We are using scriptable objects to architect our project in a clean way. It is very common to retain data/settings. It is even possible to store data in the editor in runtime. The scriptable object script is named `r_ScriptName` + `Base`, for example `r_PlayerControllerBase`

## 3.2 Input Manager

The input for the player is saved as struct type in the `r_InputManager` script. This struct is used in the `r_PlayerController` script. You can read for example the horizontal and vertical input.

## 3.3 Instances

We are making some scripts static named instance, so it is easier to call methods from other scripts independent. Below you will find an example of how you can create an instance yourself.

```
public class r_Manager : MonoBehaviour
{
    public static r_Manager Instance;

    Unity Message | 0 references
    private void Awake()
    {
        if (Instance)
        {
            //Destroy old instance
            Destroy(Instance);
            Destroy(Instance.gameObject);
        }

        //Attach new instance
        Instance = this;
    }
}
```

Now its possible to call a method another script using `r_Manager.Instance.Method();`

## 3.4 Grouping

In our scripts you will find several groupings with #region - #endregion. We do this to distinguish between methods and callbacks. This will give you an overview of what these methods are for.

#region Public Variables => Here you will find all declared variables that can be seen in the inspector or for other scripts

#region Private Variables => Here you will find all declared variables that are not visible in the inspector or runtime data

#region Functions => Here you will find the in-built functions such as Update() and Start() from which all other methods are called

#region Actions => Here you will find methods that take a one-time effect based on Start() function or player's input

#region Handling => Here you will find the methods constantly called by FixedUpdate or Update()

#region Set => Here you will find the methods where you can, for example to save data

#region Get => Here you will find the methods from which you can read or receive data

#region Custom => Here you will find the methods related to Unity, for example OnControllerColliderHit

## 3.5 Instances

In our approach we have divided the prefix in our scripts over 3 types:

r_ClassName => the r_ prefix is used with classes of monobehaviours (Uppercase)

m_VariableName => the m_ prefix is used with public and private variables that are declared (uppercase)

_variable_name => the _ prefix is used with variables that are not declared, for example temporary variables in methods (lowercase)

# 4 Main Menu Features

The main menu is the first scene where you end up. Here you will automatically connect to Photon 2. The main menu has modern functions such as:

- Making connection
- Simple main menu
- Simple networked pre-lobby system
- Call of duty style matchmaking
- Creating room / pre-lobby with extra options
- Joining and leaving rooms
- Room browser
- Simple audio controller
- Loadout manager
- Shop Manager
- Clean and simple UI

With the time and support we receive, we will expand this project even more by implementing additional features as requested by our users.

## 4.1 Room creation and scene manager explained

The rooms can be created manually or automatically by pre-lobby. When creating a room, a game map and mode are selected. The game scene will load automatically. The name of the scene is determined by the `game map` + `_` + `game mode`. It is therefore important that the scene name is the same, for example `Nuketown_FFA`.

## 4.2 Pre-lobby explained

A pre-lobby is also called a waiting room. This system is used to search for other players so there are enough players before joining the game scene.

The pre-lobby system is working when the player clicks on the "**Play**" button in the main menu. The system is automatically looking for available rooms to join with a delay. If there are no rooms available, an new room will be created automatically with random options. When the current player count is equal to the required, the game will start in a few seconds and the game scene will be loaded. The trick for a waiting room using Photon 2 is to join a room while you are in the main menu and display it as a waiting room.

In addition to the standard features, we have created a loadout and a shop system that you often encounter in today's games.

## 4.3 Shop Manager

The products added to the shop manager will be displayed as product cards. In the inspector you can assign a value to give one-time bonus to the players. With this bonus, for example, weapons can be bought if the balance is sufficient. The purchased weapons are displayed in the inventory.

## 4.4 Loadout Manager

In the loadout manager you can create different classes with weapons you want to play with. The created loadout is displayed in the game as a selection if it has been added to the game scene. The chosen loadout weapons are given to the player using the `r_WeaponManager` script on deploy.

The names of the scripts related to the main menu are listed below.

**Main Connection**

`r_MenuManager`
`r_PhotonHandler`
`r_RoomBrowserController`
`r_CreateRoomController`
`r_CreateRoomControllerUI`
`r_LobbyController`
`r_LobbyControllerUI`
`r_LobbyEntry`
`r_AudioController`

**Loadout Menu**

`r_LoadoutManagerMenu`
`r_LoadoutWeapon`
`r_LoadoutWeaponClass`

**Loadout Game**

`r_LoadoutManagerGame`
`r_LoadoutGameEntry`

**Shop**

`r_ShopManager`
`r_ShopItemUI`
`r_ShopItemConfig`

# 5 In-Game Features

There are many in game features. These are divided into multiple scripts to make it easy to export to a desired project.

- Player deployment
- Simple UI pause menu with settings and loadout selection
- Different game states such as (Waiting, Starting, Playing, Ending, Ended)
- Game mode (FFA)
- Scoreboard
- Loadout in-game
- Game timer
- Kill feed system
- Game chat
- Graphic options
- Player spectate system

## 5.1 In-Game Manager

The `r_InGameManager` script is the main manager which handles the room core and main user interface. By keeping track of the current state of the room, we can perform actions at a specific time. The order of the game starting can be easily changed in the script.

## 5.2 In-Game Mode

The `r_InGameMode` script is used to hold different game modes information like game duration and winning score. This script is dynamic and other game modes can be easily added. This information is passed back to the r_InGameManager so that the information from the mode can be passed to the `r_InGameTimer` script. The current game state will be changed when the current countdown is ended.

## 5.3 In-Game Timer

The in game timer is created using `PhotonNetwork.ServerTimeStamp` and gets saved in the room properties. The master client saves the start time of the timer. The other players receives the start time from the room properties and automatically calculate the current timer value.

## 5.4 In-Game Scoreboard

The scoreboard is displaying all players in the room. A game object will be instantiated with r_InGameScoreboardEntry script to show player information like kills, deaths and ping.

## 5.5 In-Game Loadout

The r_LoadoutManagerGame is holding the scriptable loadout classes. The loadout classes will be instantiated as a game object with the r_LoadoutGameEntry script on it. This script holds information to apply the correct data when the player gets deployed. We added four default classes as example. Make sure you add the scriptable loadout class manually to the inspector on creating a new class.

## 5.6 In-Game Kill Feed

The kill feed manager is used to display whose killing each other. This way everyone is aware of what is currently happening. The current local player and enemy player is highlighted with a specific color.

## 5.7 In-Game Chat

A simple modern chat system is created using simple RPC callbacks. The current local player and enemy player is highlighted with a specific color. A great way for communication in game.

## 5.8 In-Game Spectate

The spectate system is made to see which player killed you with information. Each player has a spectate holder transform linked to them. When a player gets killed, the local spectate camera will move to the other player's spectate holder transform. After a few seconds, the local game settings will be reset in the r_InGameManager so the player can redeploy again. The settings are easy to change via the inspector.

**Easy implement**

All in-game managers with accompanying UI is saved to the prefabs folder. This prefab can be imported into a new scene. The names of the scripts related to the game are listed below.

| **Main scripts** | **Spectator scripts** | **UI scripts** |
|---|---|---|
| r_InGameManager<br>r_InGameMode<br>r_InGameTimer<br>r_InGameOptions<br>r_ChatManager | r_SpectateController<br>r_SpectateHolder | r_KillfeedManager<br>r_InGameScoreboard<br>r_InGameScoreboardEntry<br>r_LoadoutManagerGame<br>r_LoadoutGameEntry |

# 6 Network Player Explanation

Since this project is a multiplayer project, there is more than just the First Person view, namely the third person side. Because of networking things have been coded on a more complex way.

First of all we have our player prefab which we are instantiating in the rooms to actually play. This player prefab has different scripts on it. The scripts associated with the player are:

**First Person (Main Transform)**

r_PlayerController
r_PlayerCamera
r_PlayerHealth
r_PlayerAudio
r_PlayerUI
r_PlayerConfig
r_InputManager

**Third Person (Character Model)**

r_ThirdPersonManager
r_ThirdPersonAimManager
r_ThirdPersonBodyPart
r_ThirdPersonCamera

## 6.1 Player view

The main script which is attached to the player to handle networking is r_PlayerConfig. This script ensures that first person and third person are distinguished from each other.

When the player is instantiated, the SetupLocalPlayer() method is called in the r_PlayerConfig. This method checks whether we are the local player. If so, the objects such as camera for first person view will be turned on. The character model mesh is hidden via the script r_ThirdPersonManager for first person and will only be visible to other players.

The player's mesh is hidden by changing the skinned mesh renderer cast shadow option on the character model to shadows-only. This way it is possible to see the shadow of your player in first person mode.

## 6.2 Player ragdoll

When adding a character model, it is necessary that the ragdoll is made in advance. The character model will need to be added under the main player transform. Those body parts that contain colliders will need to be provided with the `r_ThirdPersonBodyPart` script.

The ragdoll is activated when the player is killed. The first person and third person are getting separated from each other. The parent of the third person character model will be removed from the main player transform and all monobehaviours will be removed. Then the first person related objects will get destroyed.

## 6.3 Player locomotion

Movement is a crucial aspect of game design, as it impacts the player's experience. By providing custom and smooth movement mechanics, game developers can create engaging and immersive games that keep players coming back for more. The first person camera gives the player actually control which can feel very realistic with our features. We will keep improving the player controls for the best feeling in game.

The position and rotation of the player is synchronized over the network using `PhotonView` and `PhotonTransformView` scripts on the main player transform. The current state of the player is declared as `m_MoveState` in the `r_PlayerController` script.

To make the player's settings dynamically, we have used serializable classes in a list. From there it is possible to apply settings based on each player state, like the movement speed is used. This method is also used with the first person camera with head bob effect.

## 6.4 Player Health

The `r_PlayerHealth` script is handles the player damage. The amount of damage applied to the player depends on the weapon and body part. The character model ragdoll collider parts should have `r_ThirdPersonBodyPart` script on it, to detect head, body, arms and legs. Specific damage for each body part is set in the weapon configuration.

## 6.5 Player Audio

The `r_PlayerAudio` script handles all the audio for the player. `AudioSource.PlayClipAtPoint` is mostly used for playing an audio over network. This makes it possible for the player to listen where the sound is coming from.

# 7 Weapon Explanation

When creating a new weapon, a few configurations have to be made manually. The first time doing it will take a bit time. But once you understand it, it is actually really easy!

There are three weapon prefabs which have to be created. These weapons are used to instantiate in runtime while playing. The first person, third person and physical pickup able weapon prefab with rigid body. The player is not holding any weapon prefab. The scripts attached to the weapon prefabs are:

- First person weapon prefab => r_WeaponController
- Third person weapon prefab => r_ThirdPersonWeapon
- Physic pickup weapon prefab => r_WeaponPickup and Photon View

It is useful to know that configurations are attached to the added scripts. To speed up the process, it is possible to duplicate the configurations we have made and adjust them to your own taste.

## 7.1 Weapon Configurations

r_WeaponControllerBase is the main weapon control scriptable configuration. Here you can adjust the values for the weapon like fire rate, mode and more. Be sure that the weapon name and other information entered for this configuration matches other configurations you make for this weapon.

r_ThirdPersonWeapon holds information for the third person weapon like the hand positions for the character model. This script is more intended for the third person side.

r_WeaponPickupBase contains important information. Like the weapon ID. It is necessary to assign a unique ID to each new weapon you create, which does not yet exist. This ID is used when picking up the weapon and distinguishing it in the third person animator substates. This scriptable configuration holds the three weapon prefabs, first person, third person and physic pickup prefab.

# 8 Get started

Follow the steps below to get started with the project!

## 8.1 Setup project

1. Open the unity package manager, and import our package.
2. Import Pun 2 – Networking package to the project.
3. Setup Photon 2 using your Photon 2 ID.

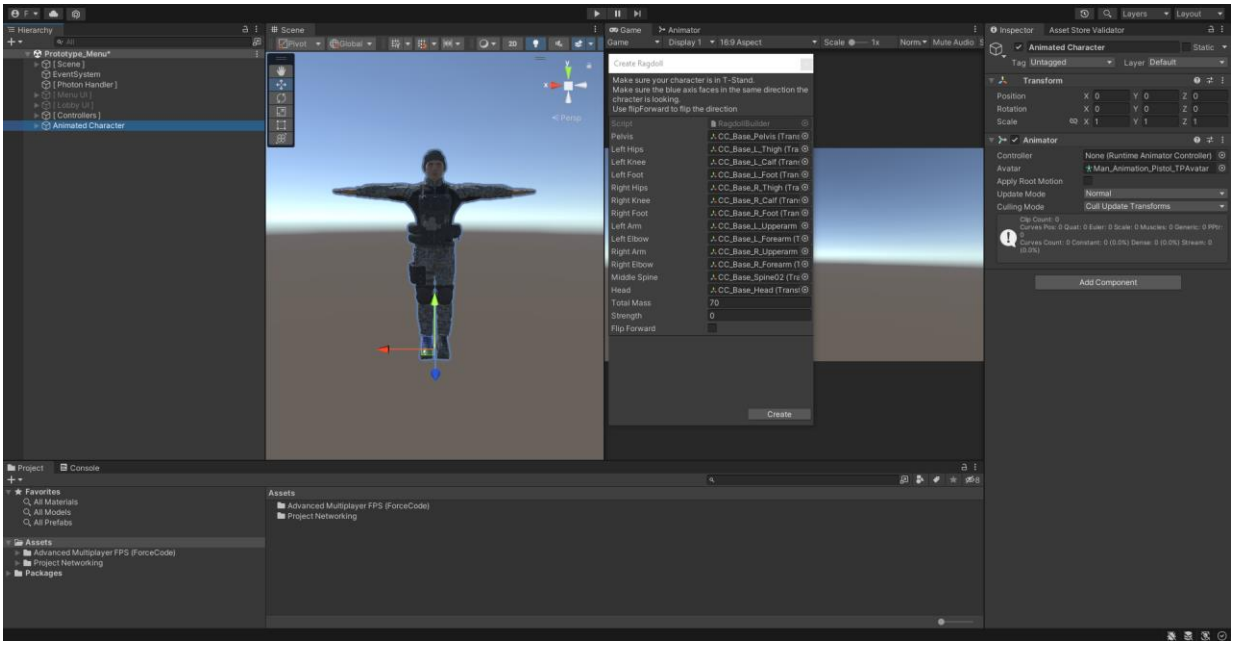
Post processing is added extra to the project to make the project graphically more beautiful. You can choose to download this or do without.

**Adding post processing to your project**

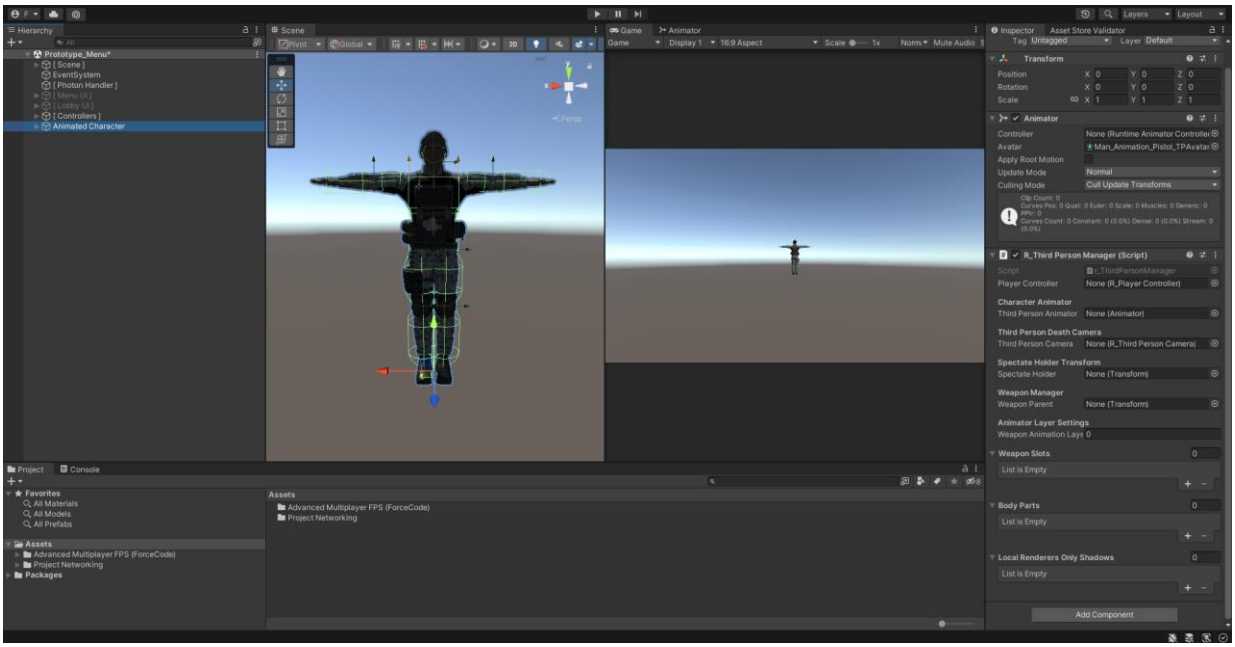
Open the Unity package manager -> Select unity packages : **Unity Registry** -> Search Post Processing and install/import it to your project.

## 8.2 Add character model

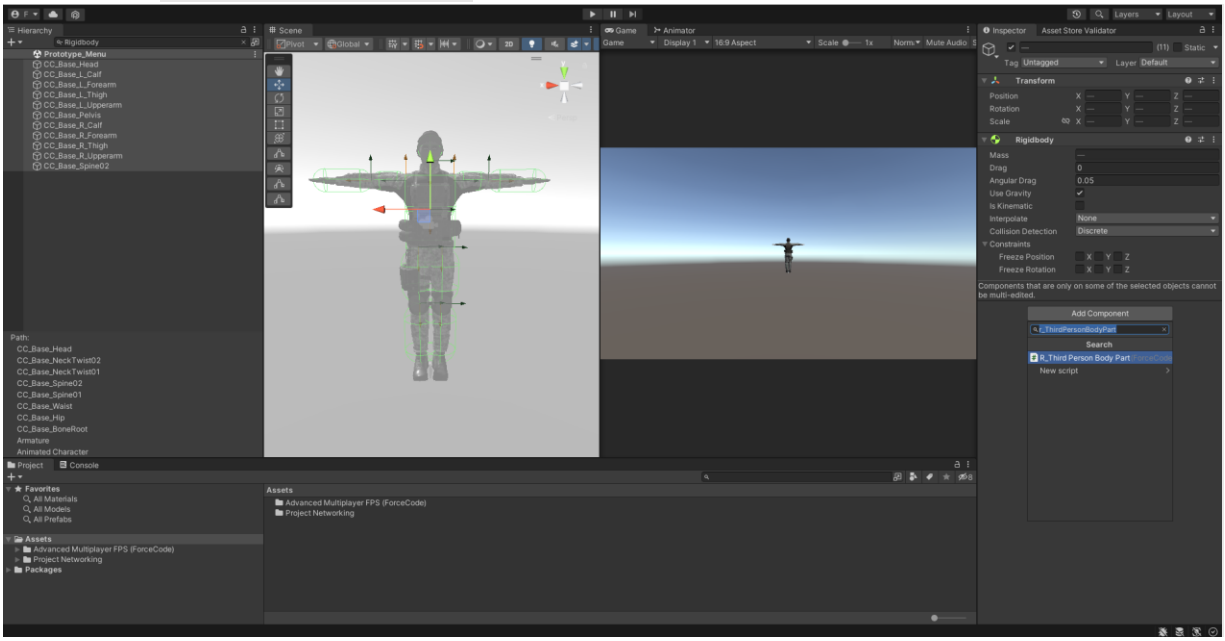
1. Add your character model to the scene, unpack your prefab and create a new ragdoll. Go to the tab section -> Game object -> 3D -> select Ragdoll.



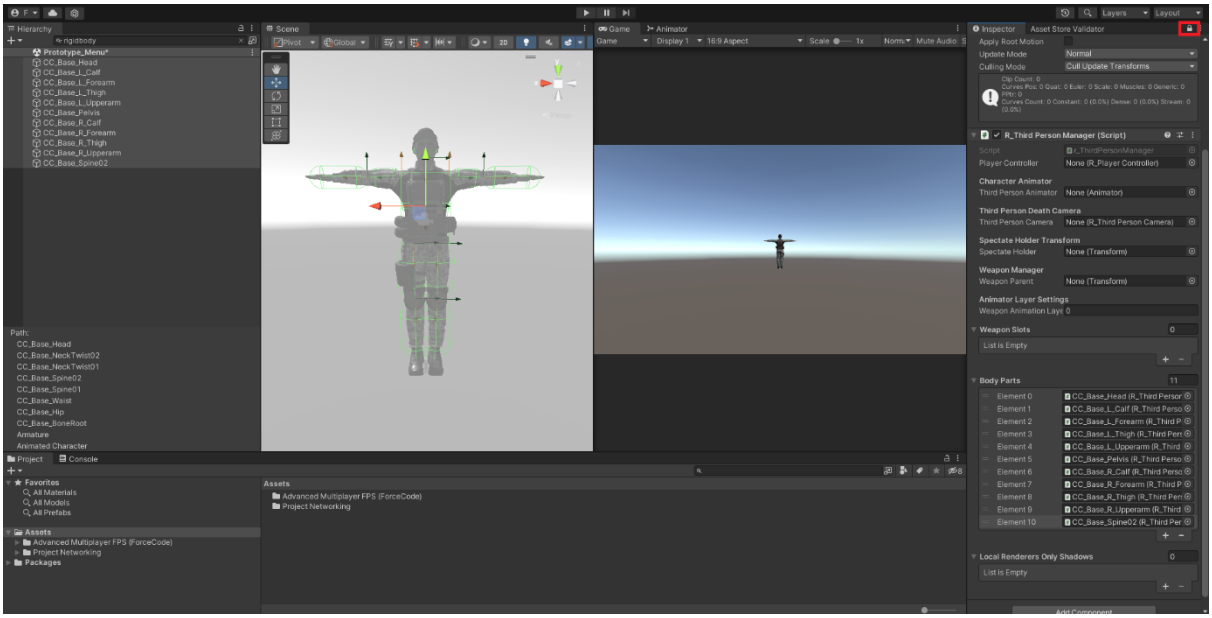
2. Select the character model and add component r_ThirdPersonManager.



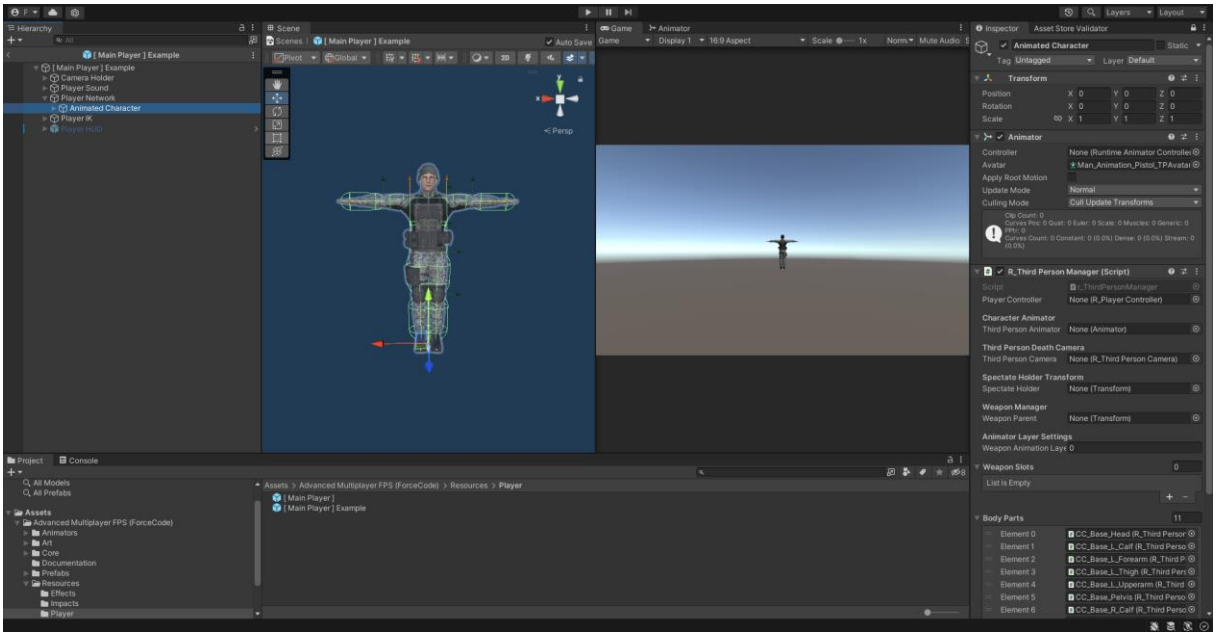
3. Go to the hierarchy tab and search for “**Rigidbody**”. Select all results and add component r_ThirdPersonBodyPart.



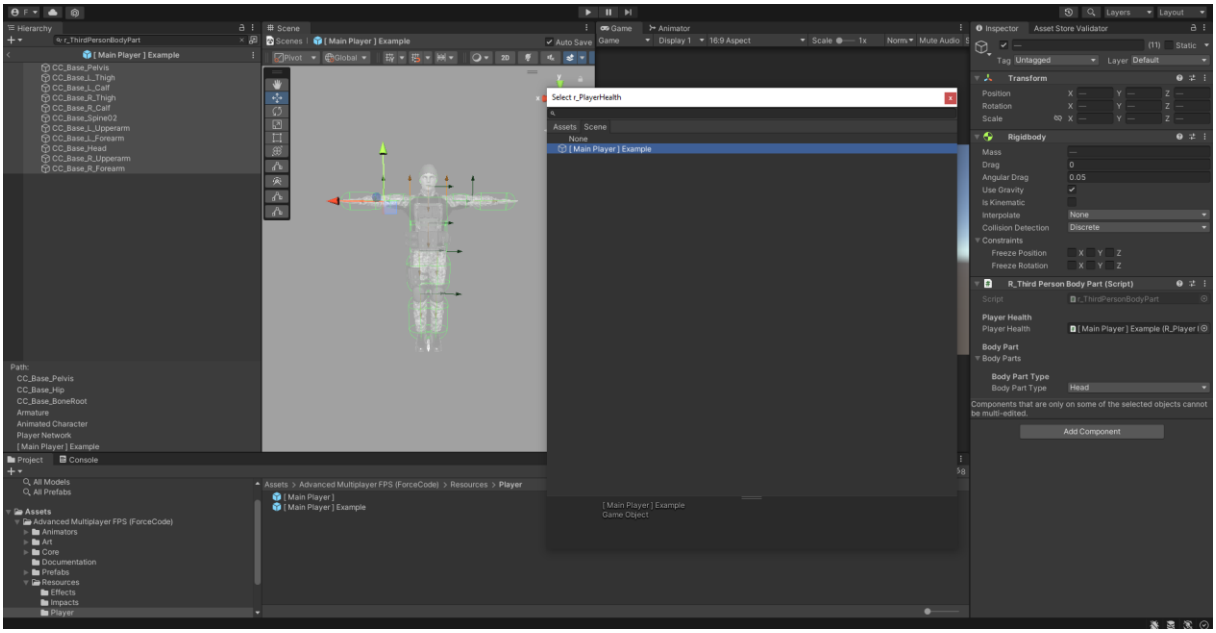
4. Lock your inspector. Go to the hierarchy tab and search for “**Rigidbody**”. Select all results and drag it to “**Body Parts**” in the inspector on r_ThirdPersonManager.



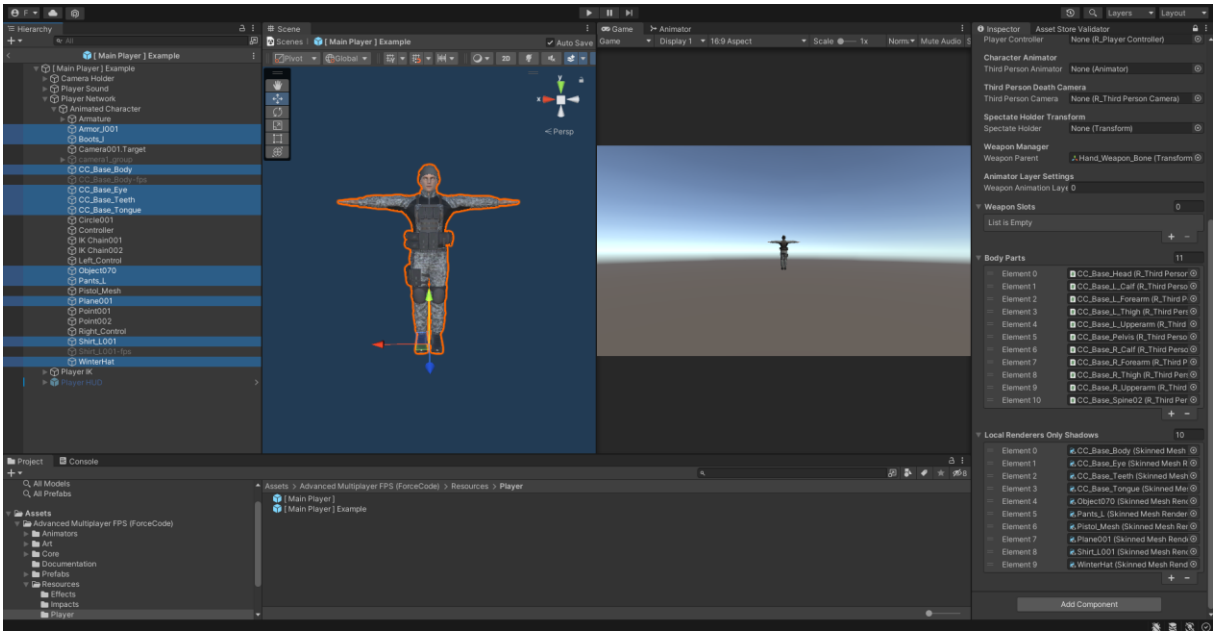
5. Copy your character model and open your main player prefab. Paste the character model under the parent “**Player Network**”.



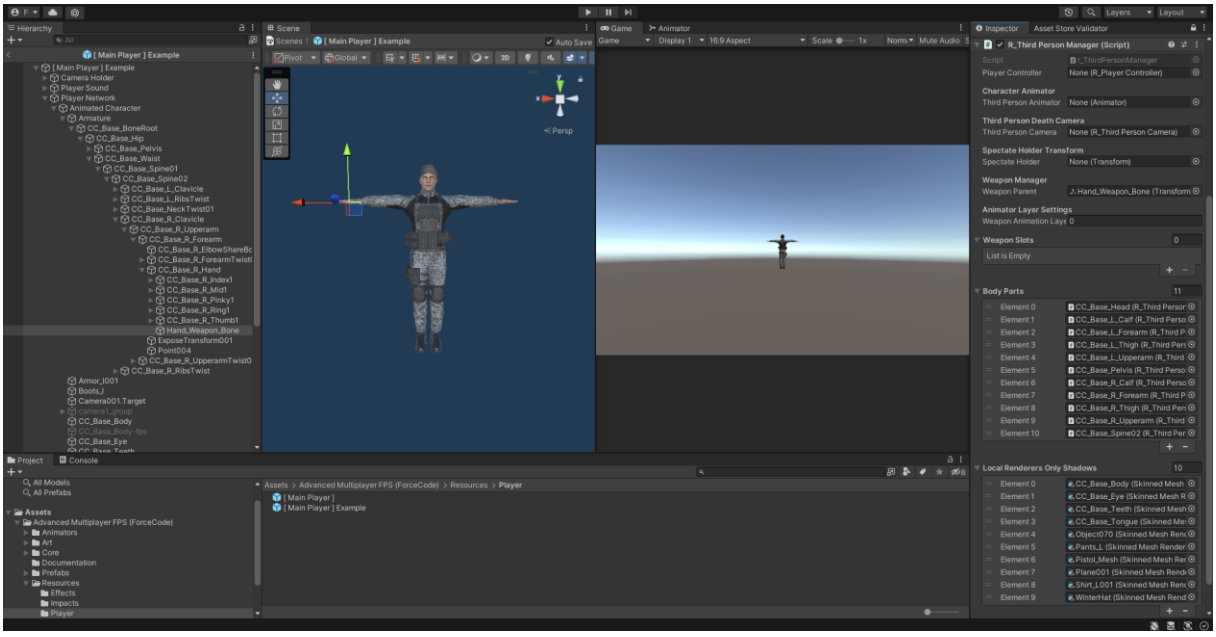
6. Search in the hierarchy for r_ThirdPersonBodyPart, select all and set the type of your body on all parts and select the player health script from your main player transform.



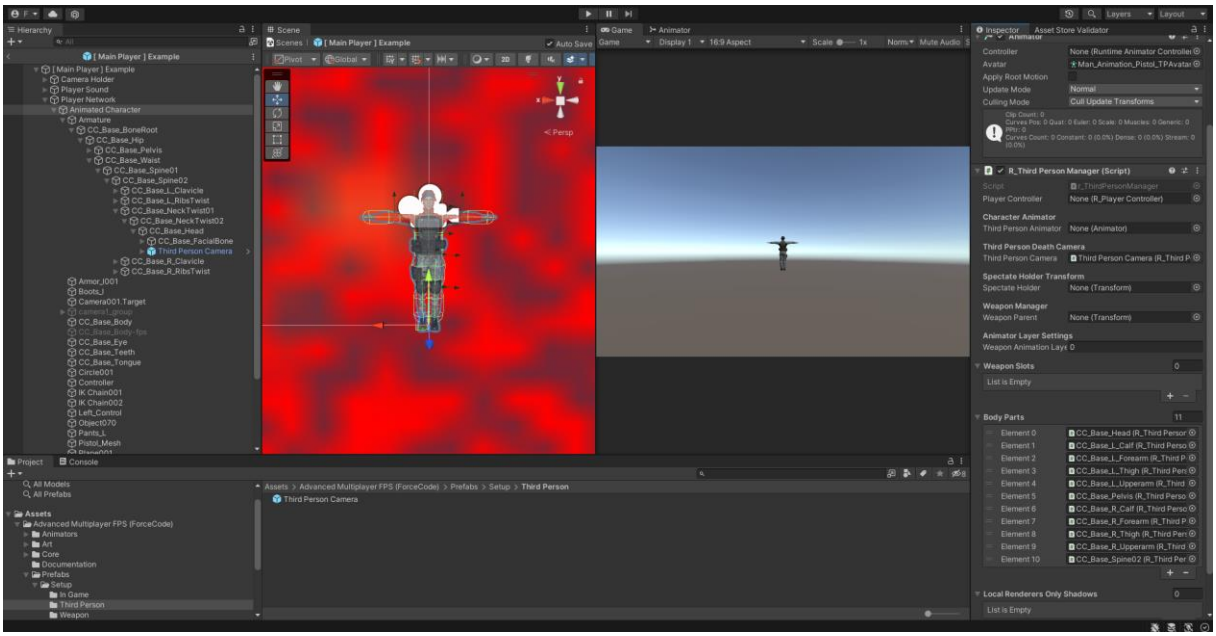
7. Lock your inspector. Select all your mesh in the hierarchy and drag it to “**Local renderer shadows**” in the inspector.



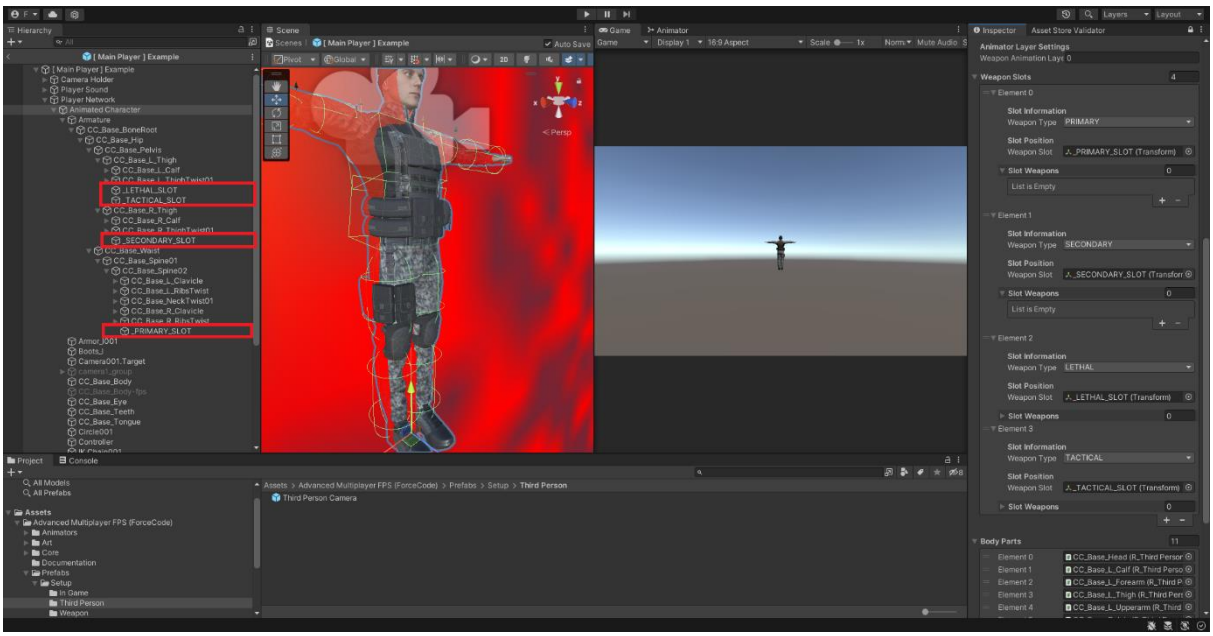
8. Open the armature and go to the right hand transform. Create a new empty transform to hold third person weapons. Attach the created transform in the inspector on “**Weapon parent**”.



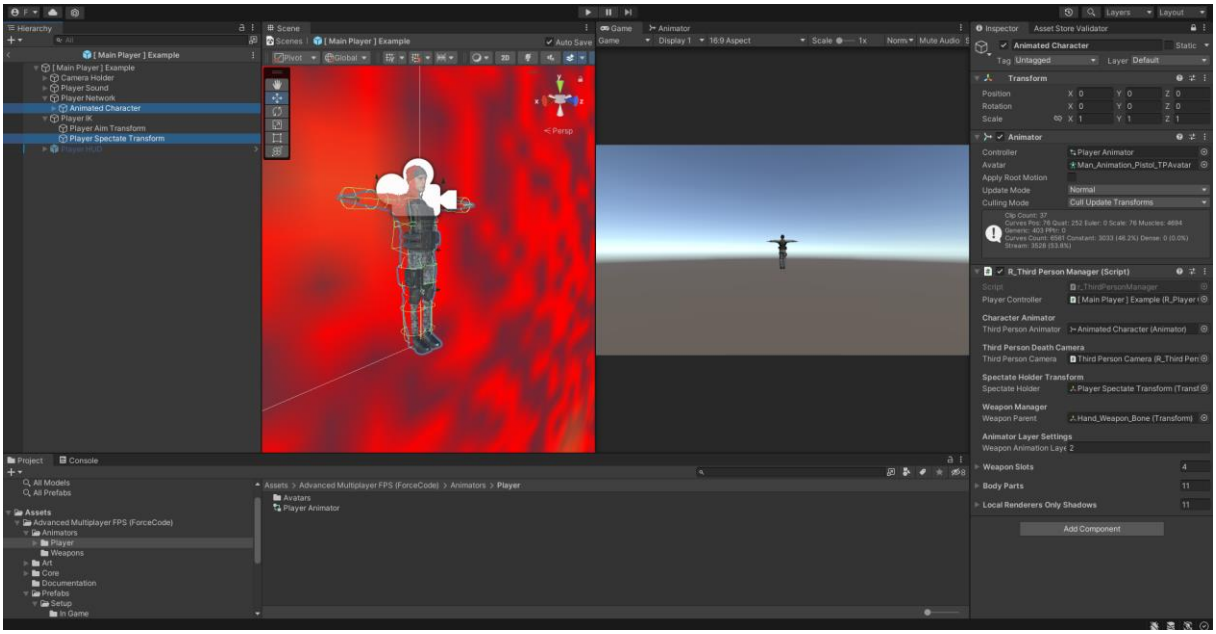
9. Open the armature and go to the “**head**” transform. Go to the prefabs folder -> Setup -> Third Person and drag the **third person camera** prefab to the head. Select the added camera in your inspector.



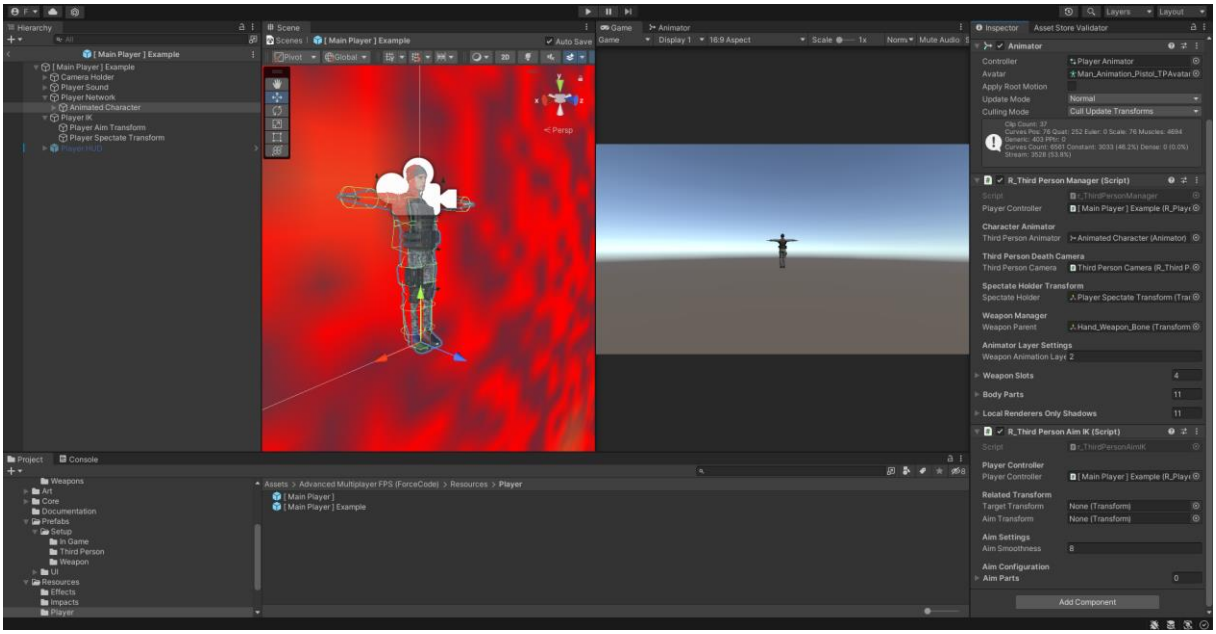
10. Create 4 new slot types in the r_ThirdPersonManager. Create empty transforms attached on the bones of the character model and set it in the inspector. The third person weapons will be set on those positions. You can position and rotate it to your own taste.



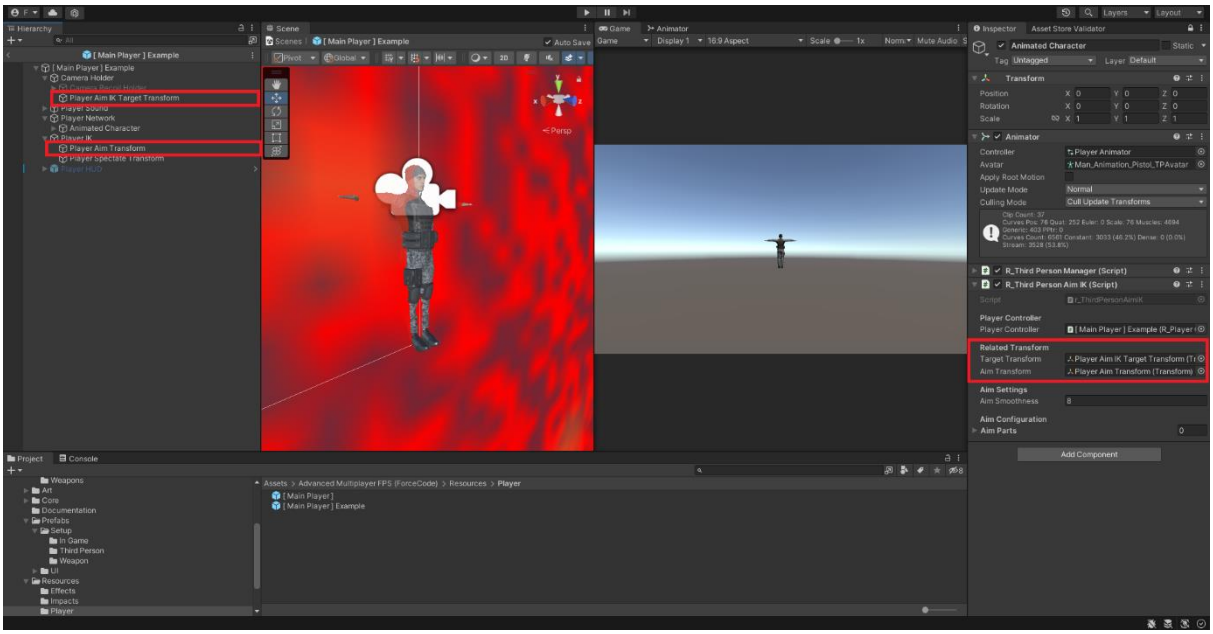
11. Go to your hierarchy and set the “**Player Spectate Position**” transform under “**Player IK**” parent to the `r_ThirdPersonManager` in your inspector. Make sure you set all other references like player controller and animator. The default animator weapon layer is 2.



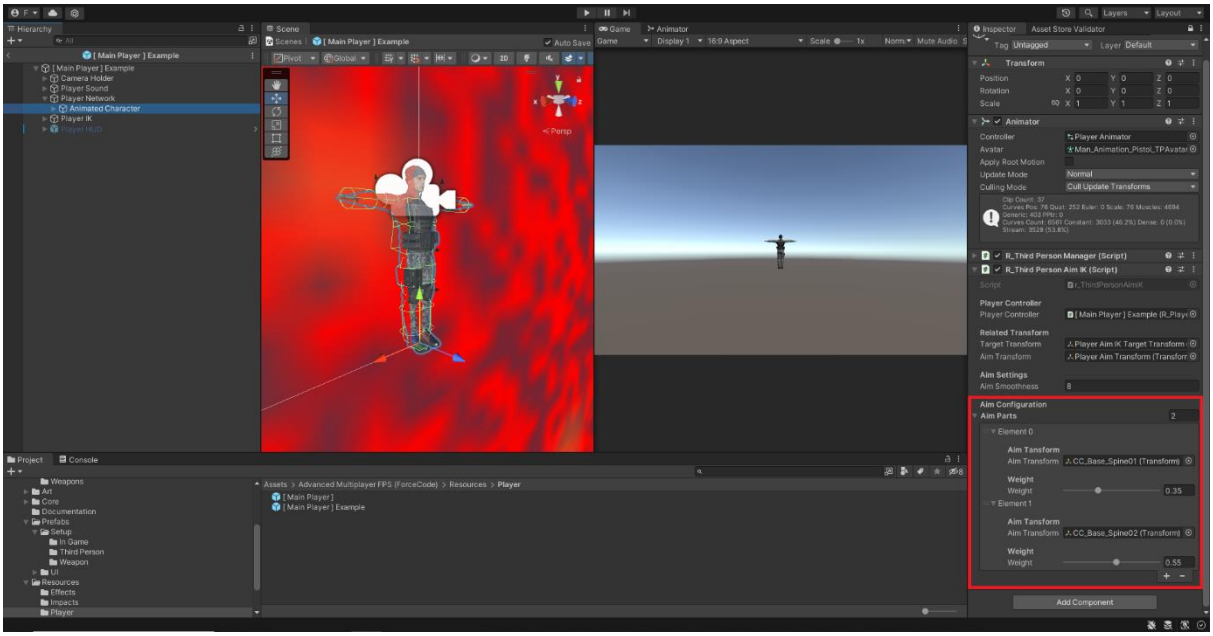
12. Add `r_ThirdPersonAimIK` script to the character model. Set what you can set already like the player controller and smoothness on for example 8.



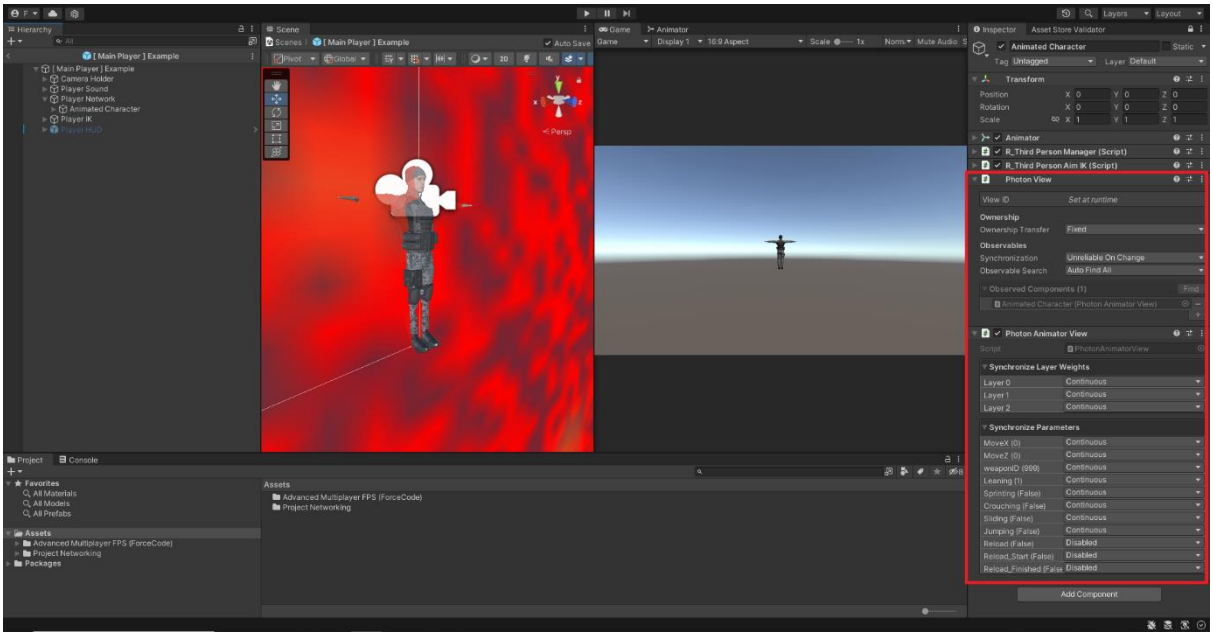
13. Select “**Player Aim Transform**” in the hierarchy under parent “**Player IK**”, and drag it to your inspector on “**Aim Transform**”. This is where the spine will aim from. You can rotate this transform if your character model doesn’t face forward. The “**Target Transform**” is under “**Camera Holder**”. The spine will aim the direction of first person camera.



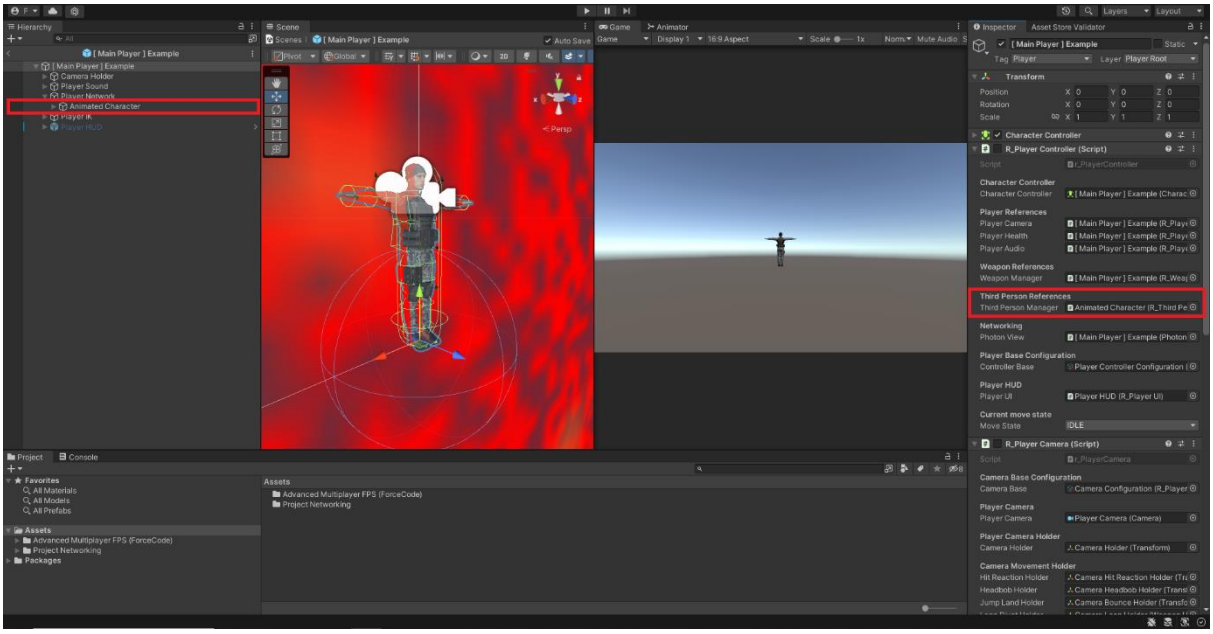
14. Create new aim part configurations for the spines. Open your armature and add it to the aim transform. Set the weight from 0 - 1 for the working.



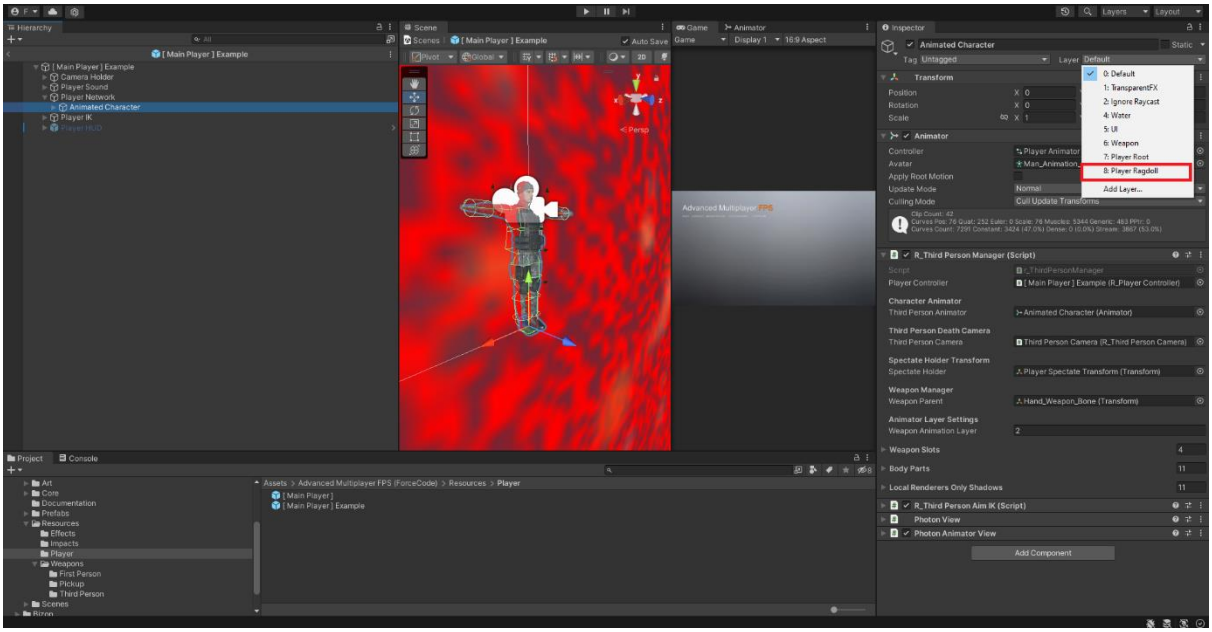
15. Add `Photon View` and `Photon Animator View` to the character model. Repeat the same settings as showed on the screenshot below.



16. Select your main player transform with the `r_PlayerController` script on it. Drag your character model to the Third Person Manager in the inspector.



17. Select your character model and make sure the selected layer is “**Player Ragdoll**” for children as well, and main player transform layer is “**Player Root**” with this object only.



## 8.3 Add weapon

First of all start with creating the configurations for the new weapon or duplicate one of the examples. The configurations can be found in `Project/Core/Configurations/Weapon`

- Create weapon controller configuration -> `r_WeaponControllerBase`
- Create weapon pickup configuration -> `r_WeaponPickupBase`

After created the weapon configurations, duplicate the first person weapon animator, and set your animations. If you don't have walk/run animation, you can use the motion system in the `r_WeaponController`. You can set motions using the created weapon configuration.

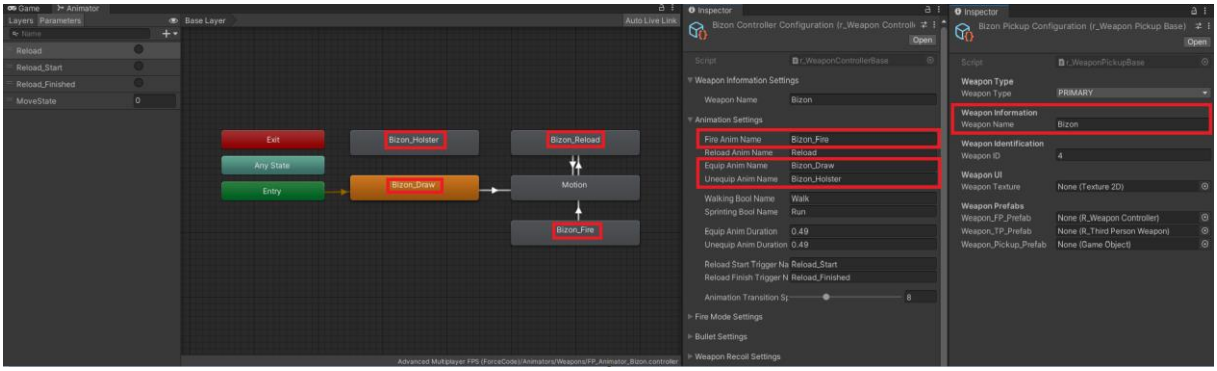
**Important settings in the weapon configuration**

There are a few settings which is very important for the working for your weapons:

- Weapon animator: change clip names to `Weapon Name + _Animation Name`, for example if you duplicate the SMG animator, you have to change SMG_Fire to AK47_Fire. Set the animation names to the weapon configuration base.

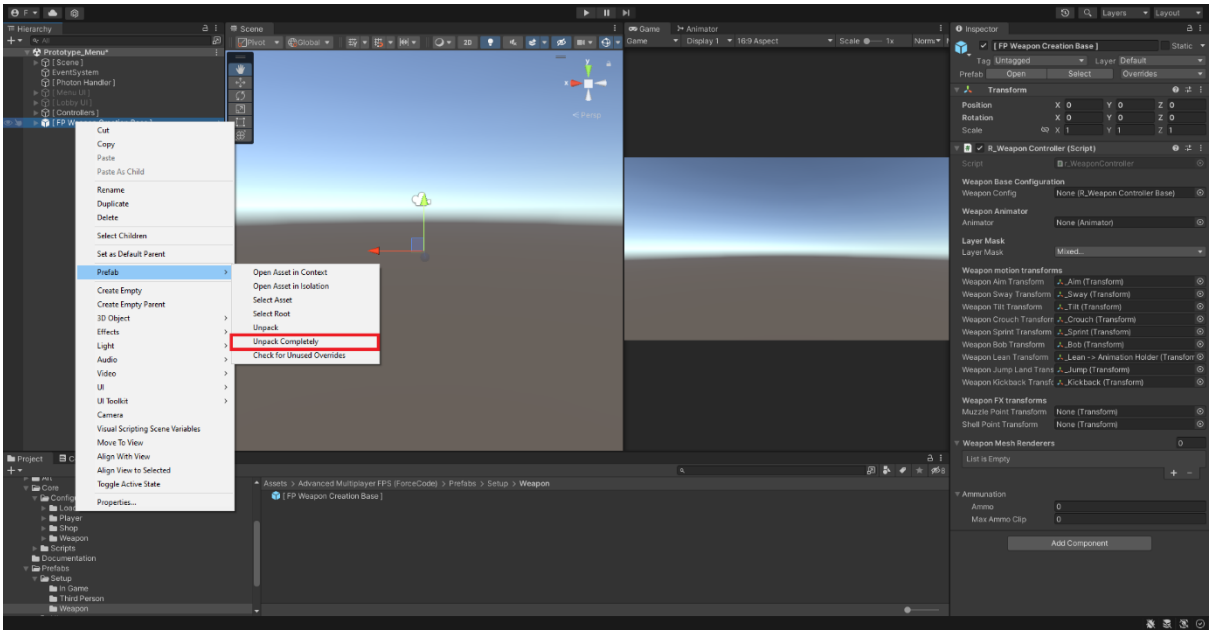
  The animation names you set in the first person weapon configuration will be used in the third person animator as well.

- Weapon name in configuration: Make sure you set your weapon name to the configuration and it is not the same as in other weapon configurations. The name must be the same in the weapon pickup base configuration.

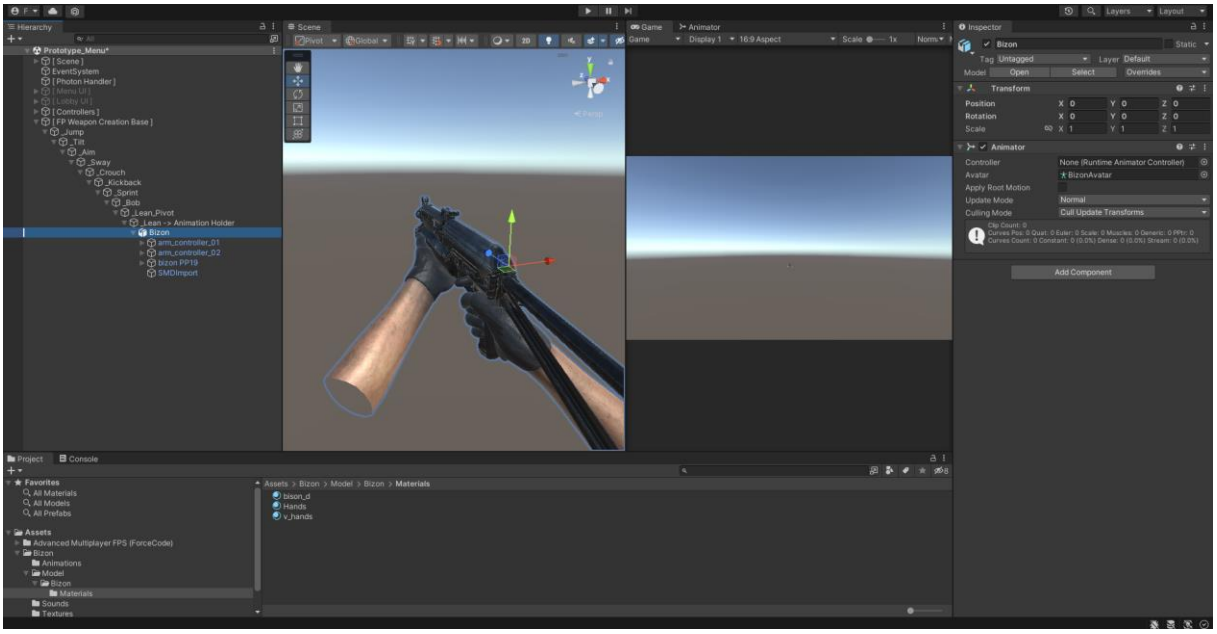


### 8.3.1 First Person Weapon

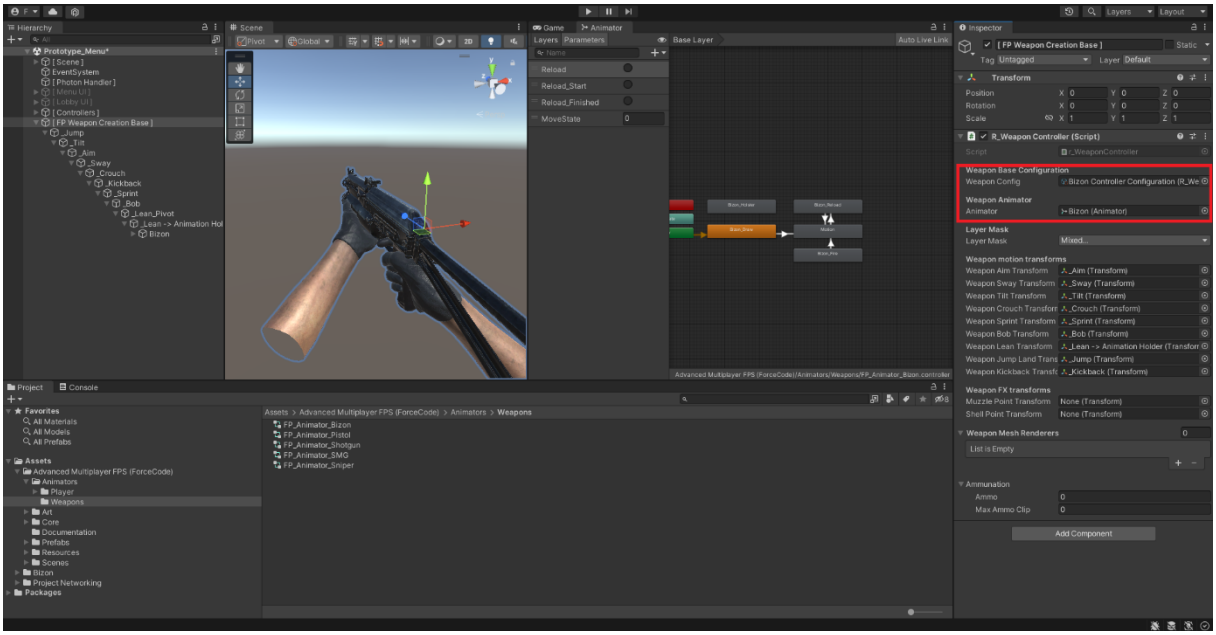
1. Put the weapon creation base prefab into the scene and unpack it completely. The prefab can be found in Project/Prefabs/Setup/Weapon.



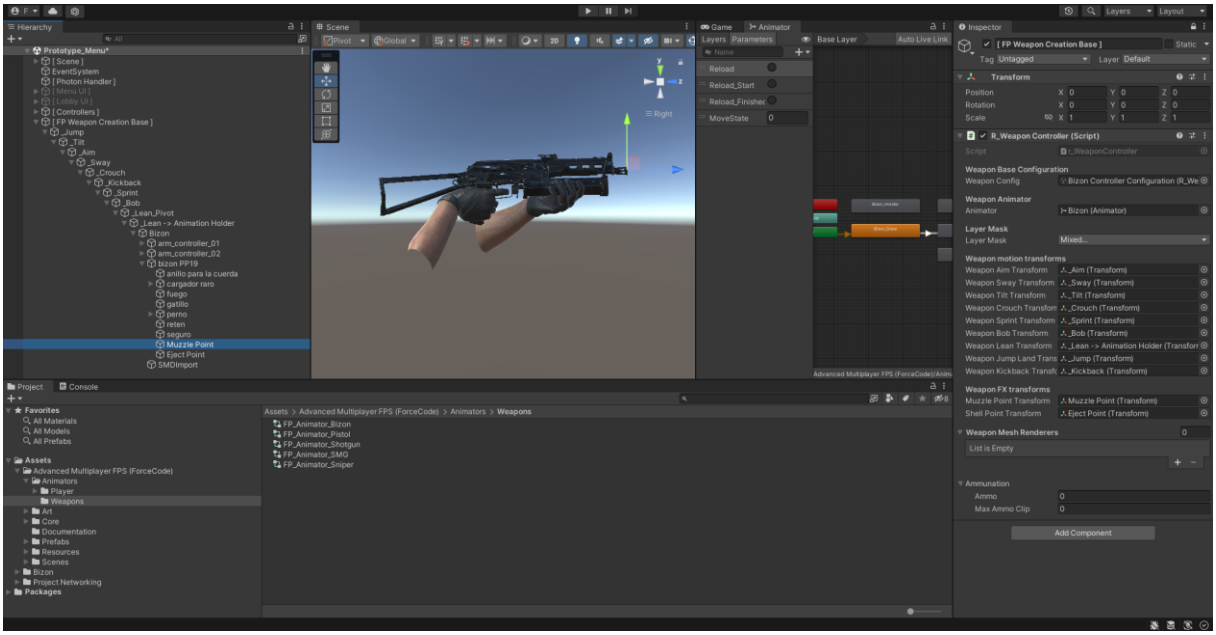

2. Put the animated weapon model under the "**Animation Holder**" transform parent and attach your duplicated animator.



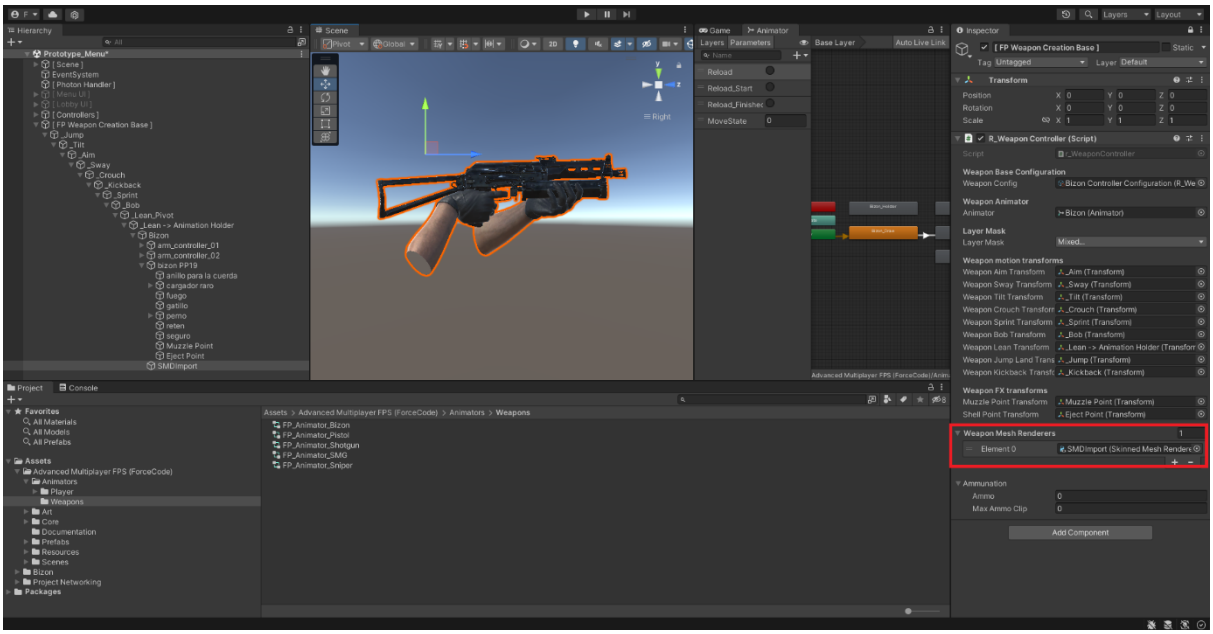
3. Attach your animator and weapon configuration to the r_WeaponController script in the inspector.



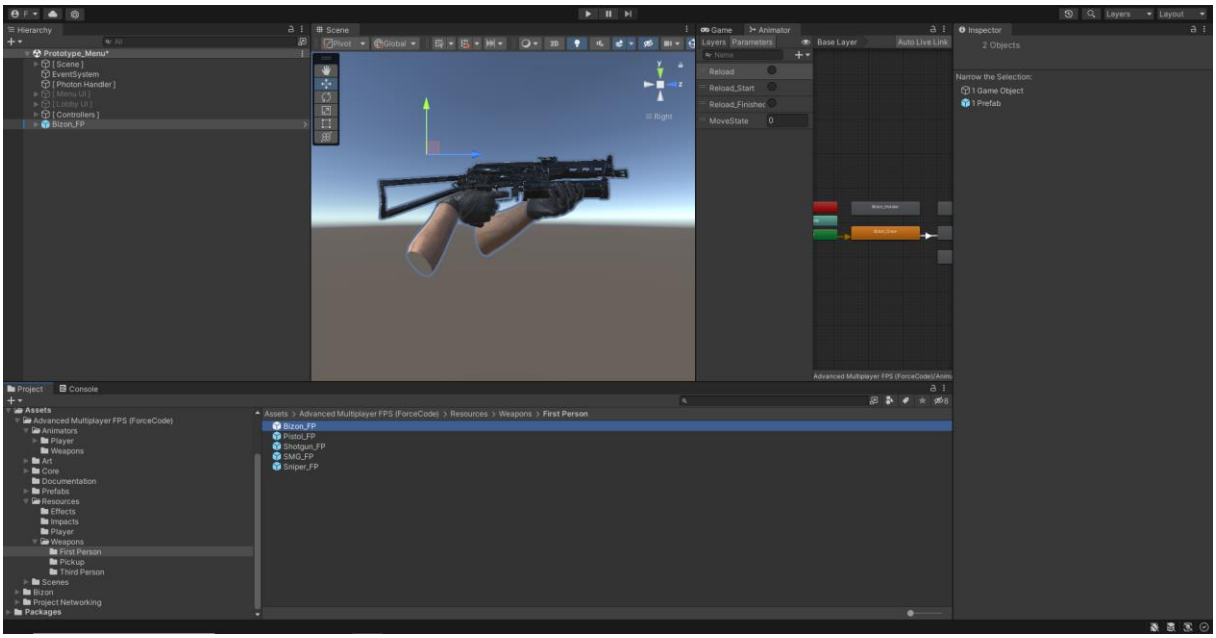
4. Create new empty transforms for the shell eject and muzzle point and attach it.



5. Lock your inspector and attach the Skinned Mesh Renderers of the weapon model and hands. Make sure you attach all other required components to the `r_WeaponController` in the inspector.

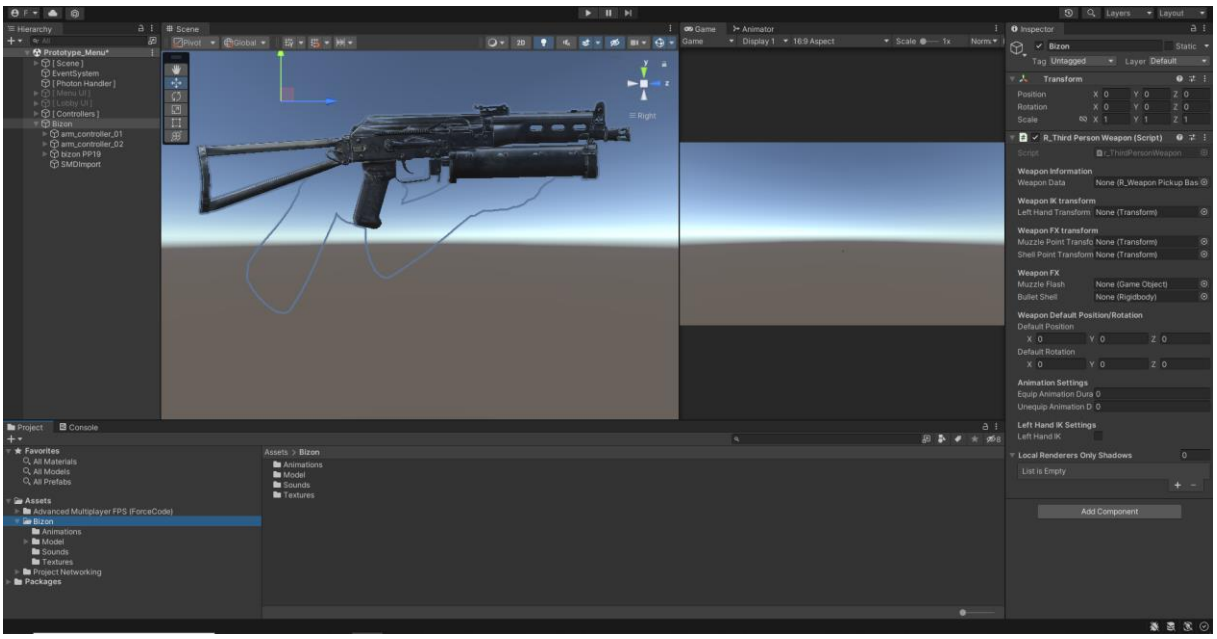


6. Change the Game Object name and save it as prefab. Move it to the `Resources/Weapons/First Person` folder.

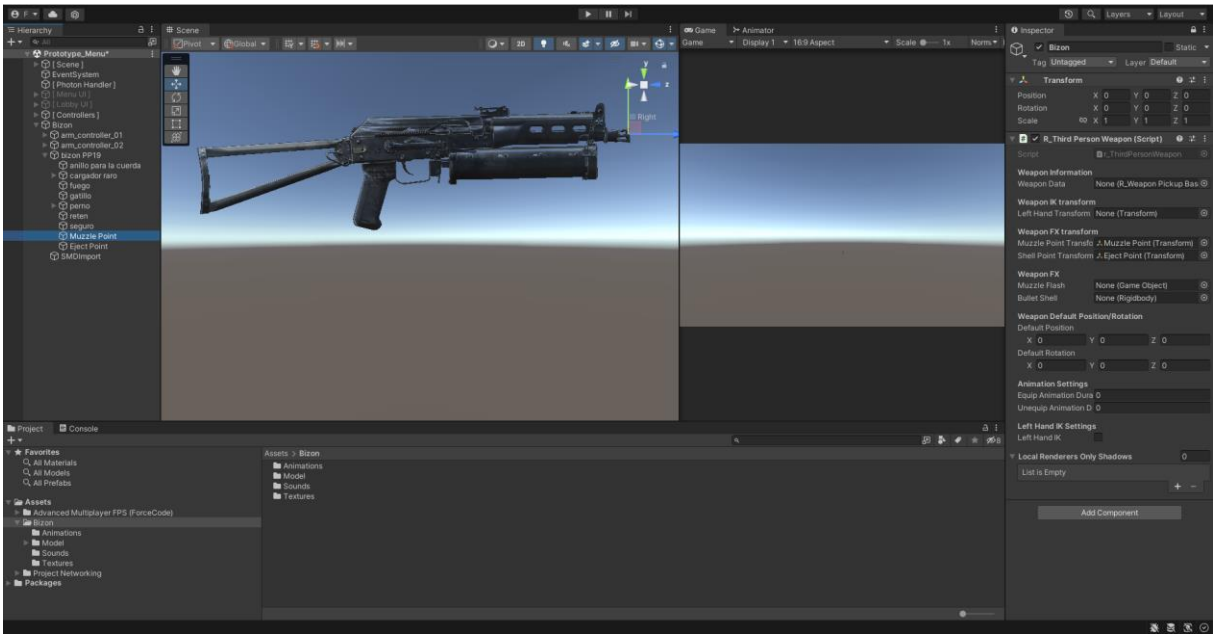


### 8.3.2 Third Person Weapon

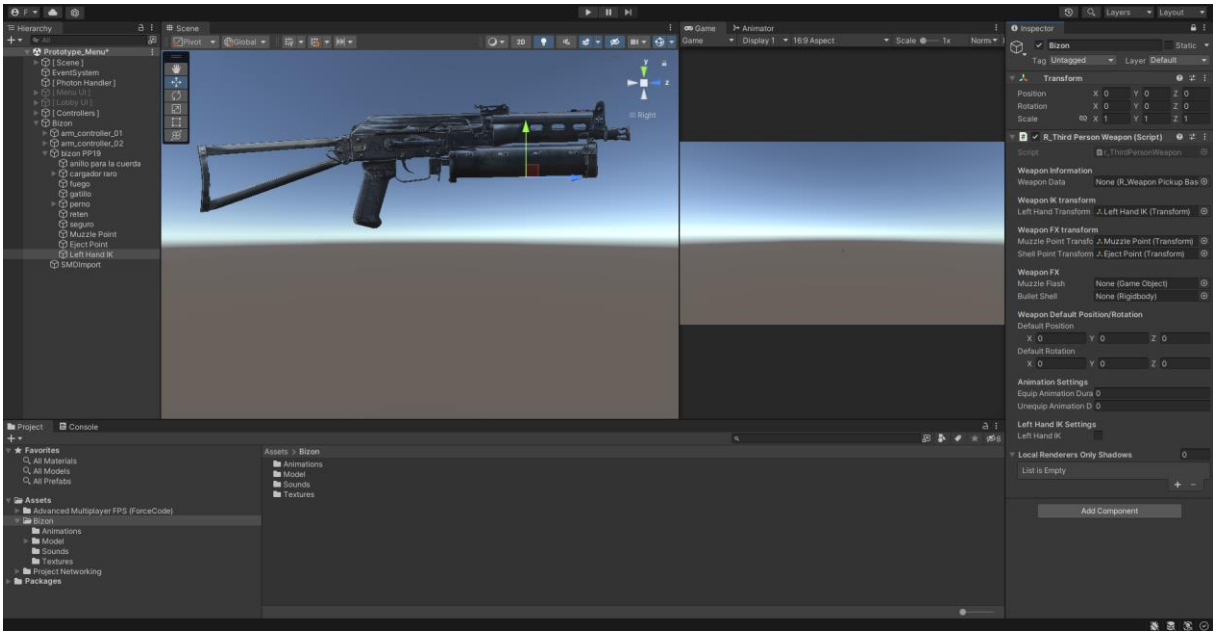
1. Put the weapon model without hands into the scene, unpack it and add `r_ThirdPersonWeapon` script to it.



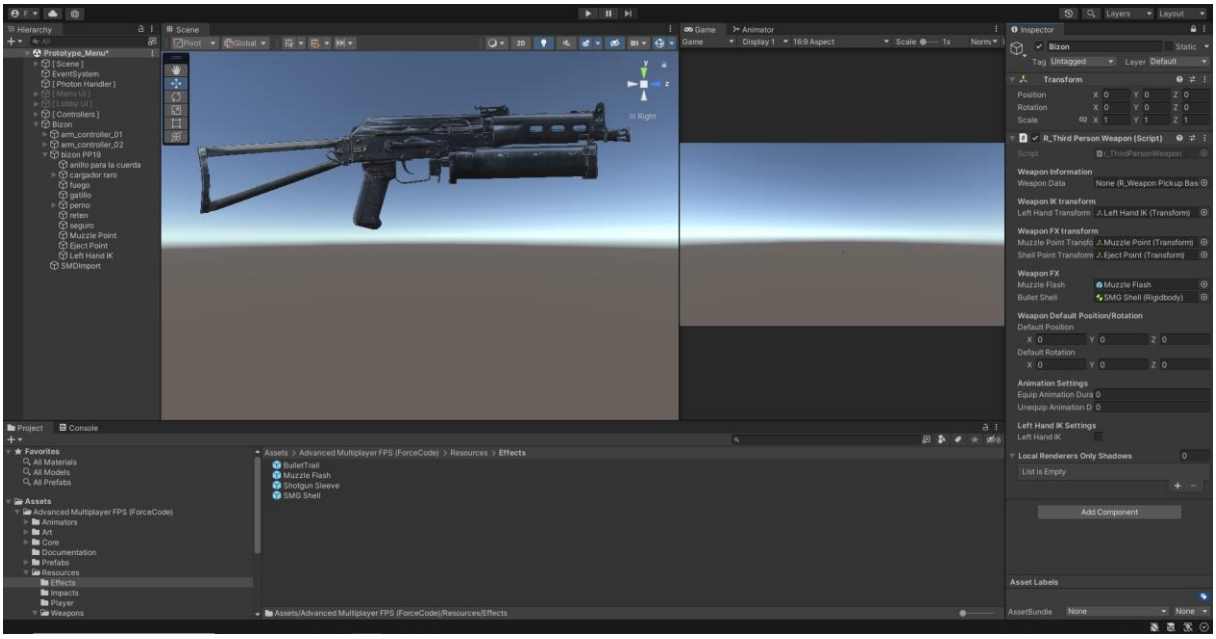
2. Create new empty transforms for the shell eject and muzzle point and attach it.



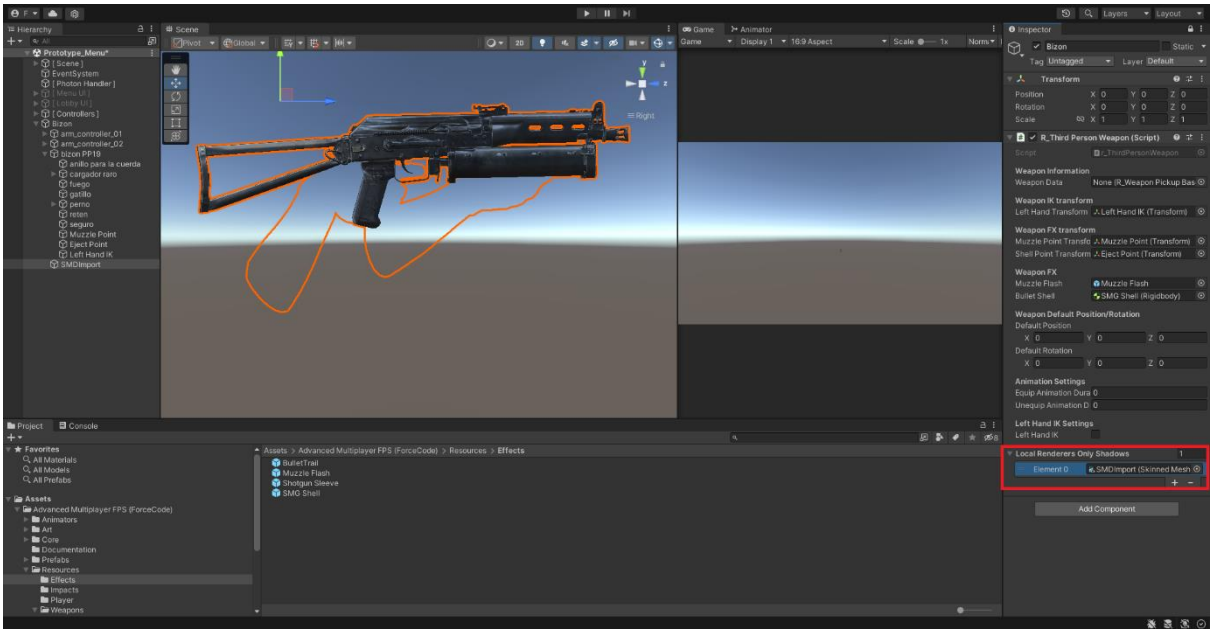
3. Create new empty transform and position it for the left hand IK.



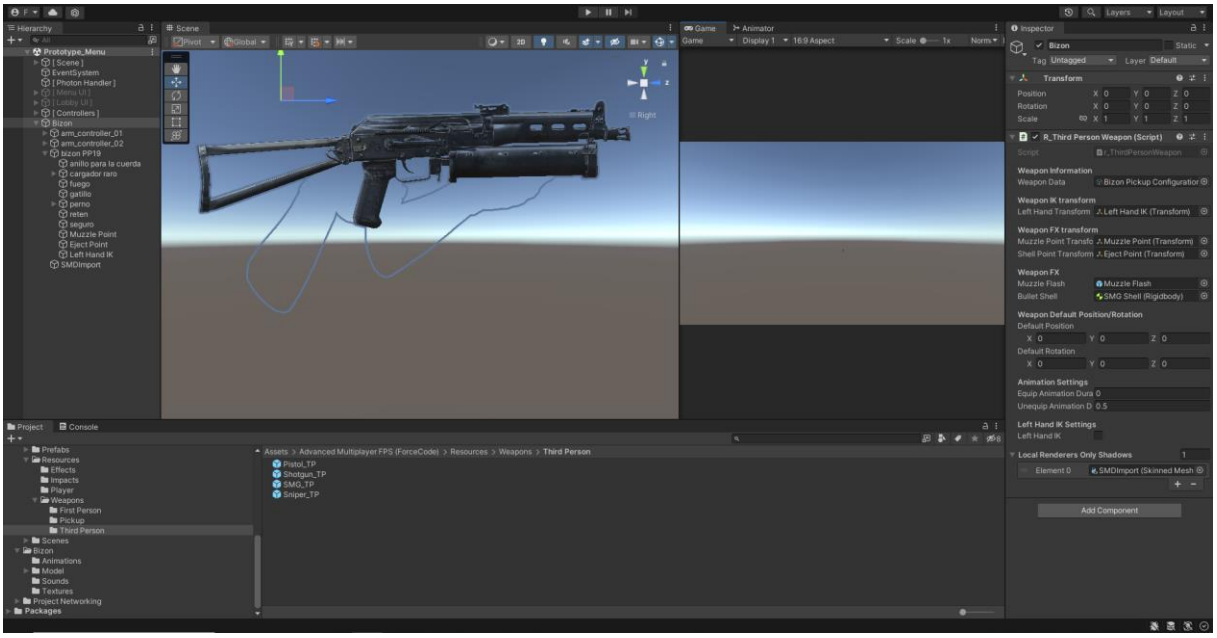
4. Attach the muzzle flash and bullet shell to the inspector. This can be the same as fist person. (The first person muzzle and shell is instantiated without networking But make sure to attach Photon View to the objects as it is used for third person). Already muzzle flash and shell can found in Resources/Effects.



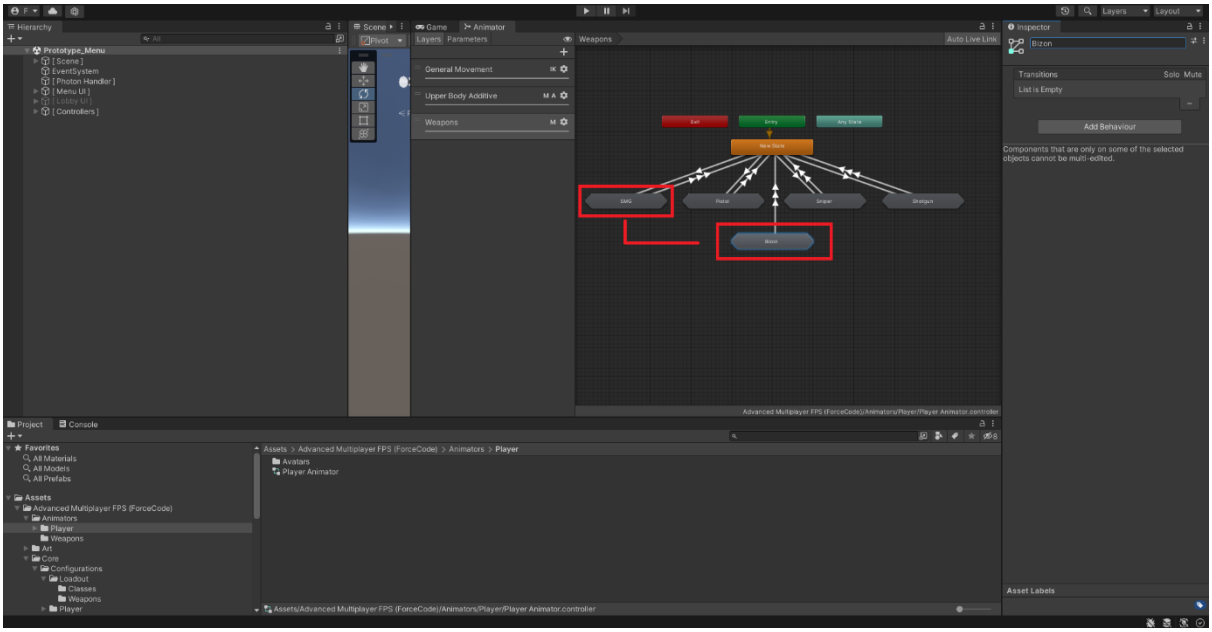
5. Lock your inspector and attach the mesh to r_ThirdPersonWeapon script in the inspector under Local renderers shadows only.



6. Attach all other requirements to the r_ThirdPersonWeapon script in the inspector. You can set the default position and rotation later after tested in runtime. Change the Game Object name and save it as prefab in Resources/Weapons/Third Person.

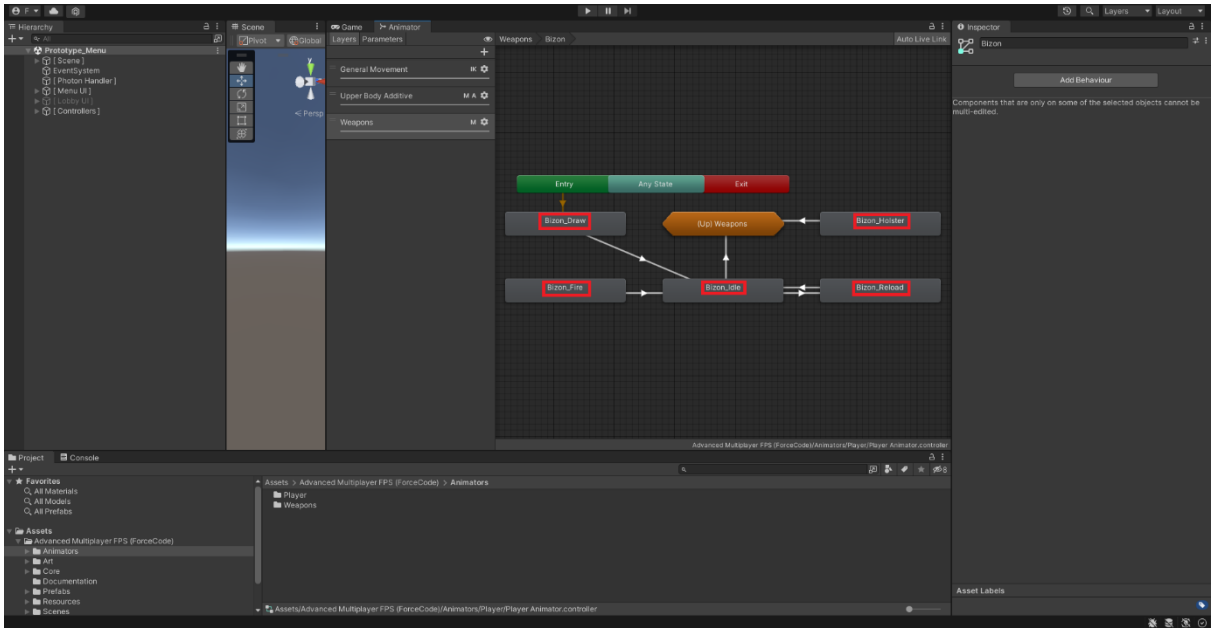


7. Open the main player animator and select the weapon animations layer. Duplicate one of the weapon states.

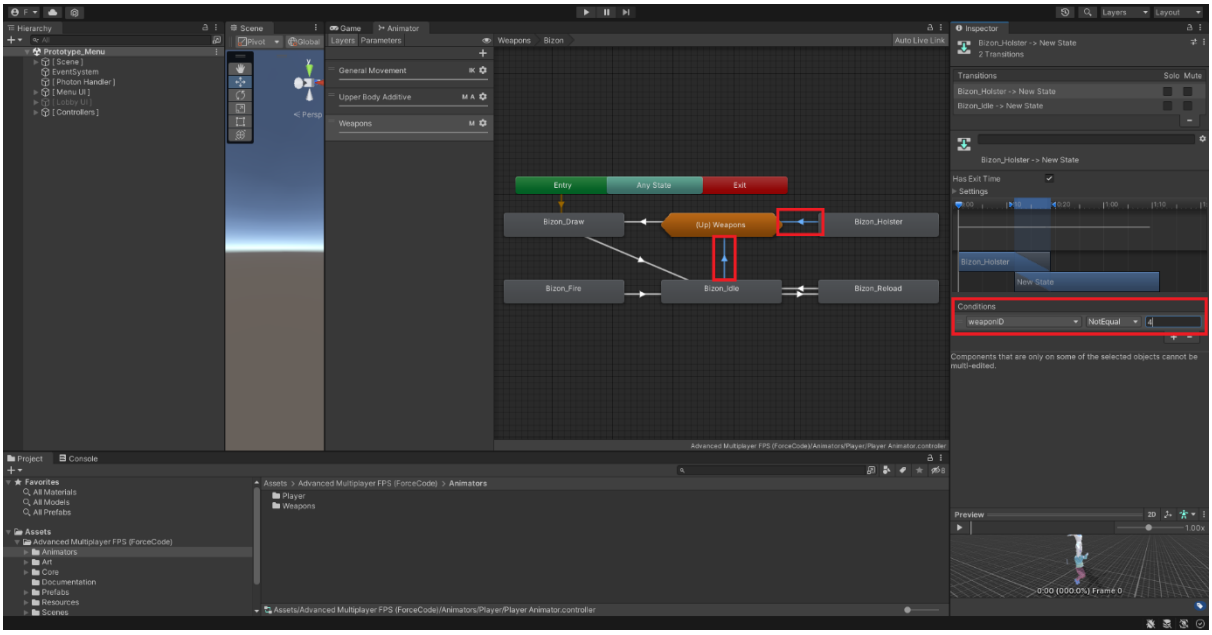


8. The weapon ID and animation names which we have set in the weapon configurations will be used here again. The animation state will play based on the current weapon ID which the player owns.

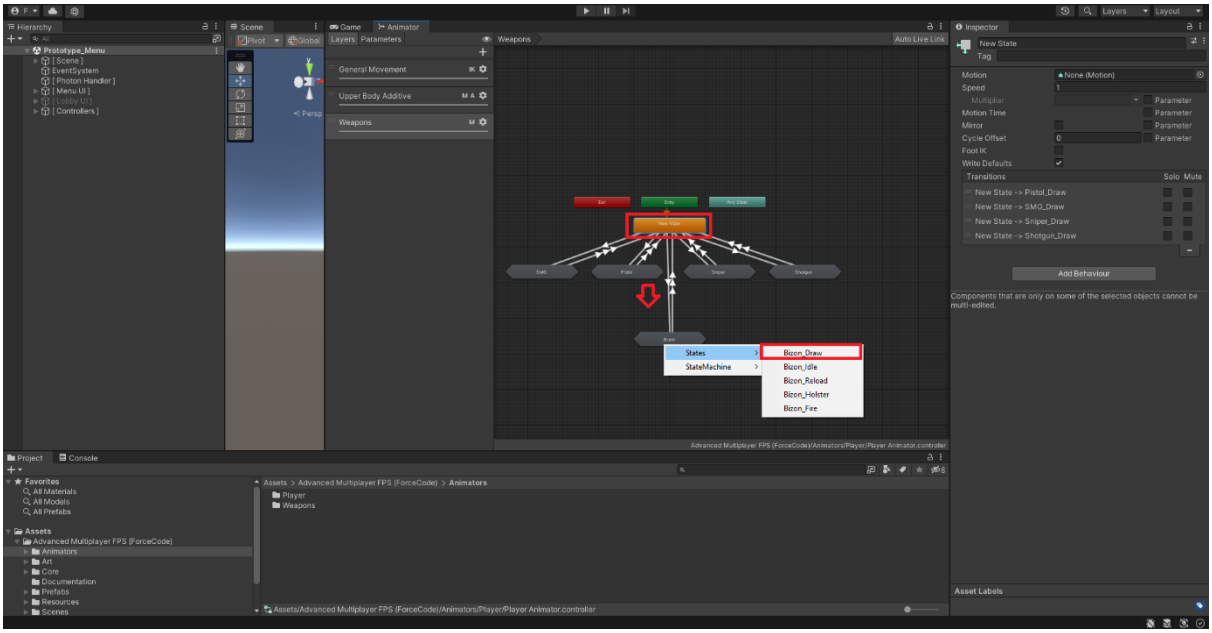
   Open the state and change the animation names, for example Bizon_Draw.



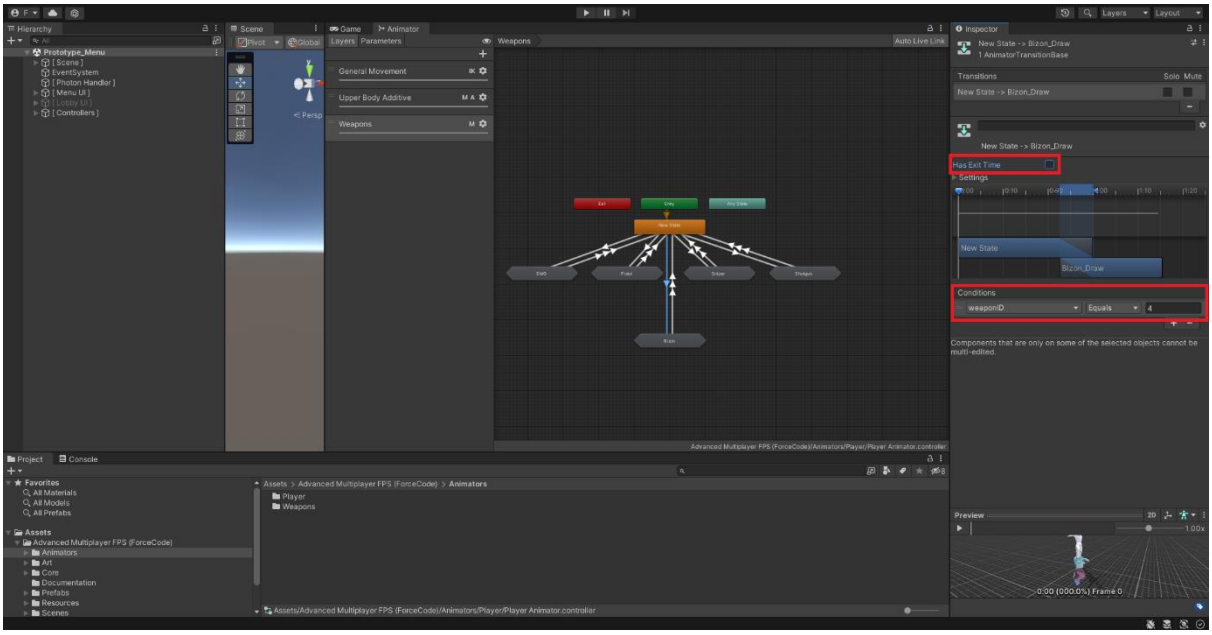
9. Select the transition from the animations clips to the orange “**(Up) Weapons**”. Change the weapon ID not equals to the weapon’s ID.




10. Go back to all weapon states and create a new transition from “**New state**” to your new weapon state “**draw animation**”.



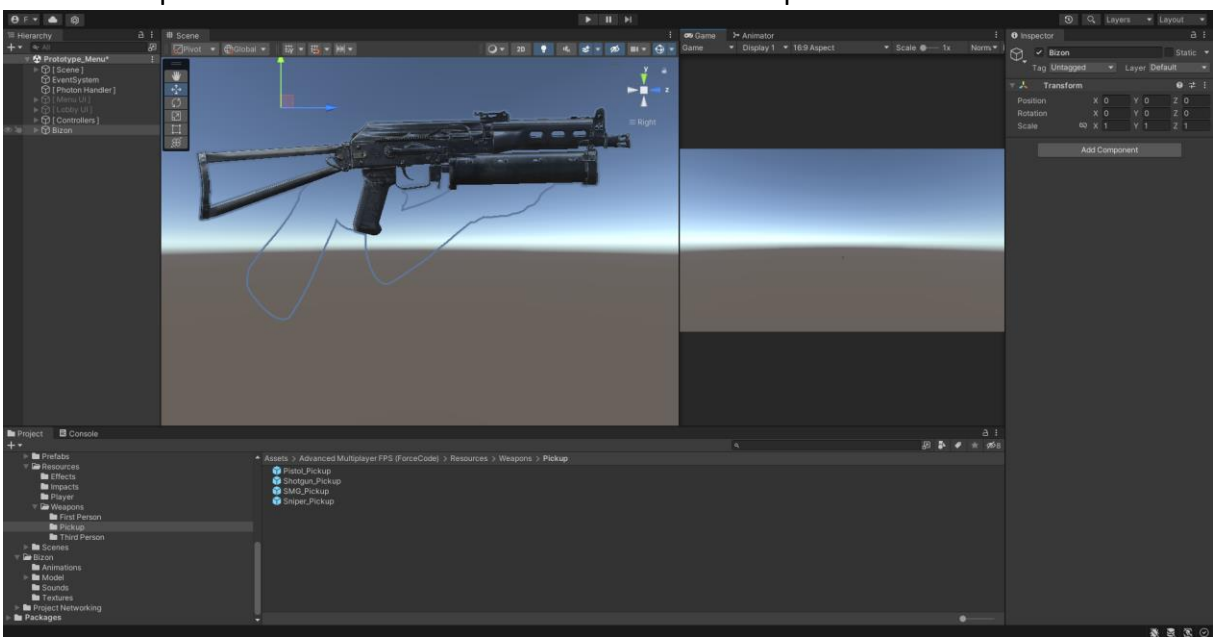
11. Select the new created transition, untick “**Has exit time**” and add new condition “**Equals Weapon ID**” and set your weapon ID.




Make sure you set your third person weapon animation clips. If you don’t have specific weapon animations, you can use general animations with left hand IK.

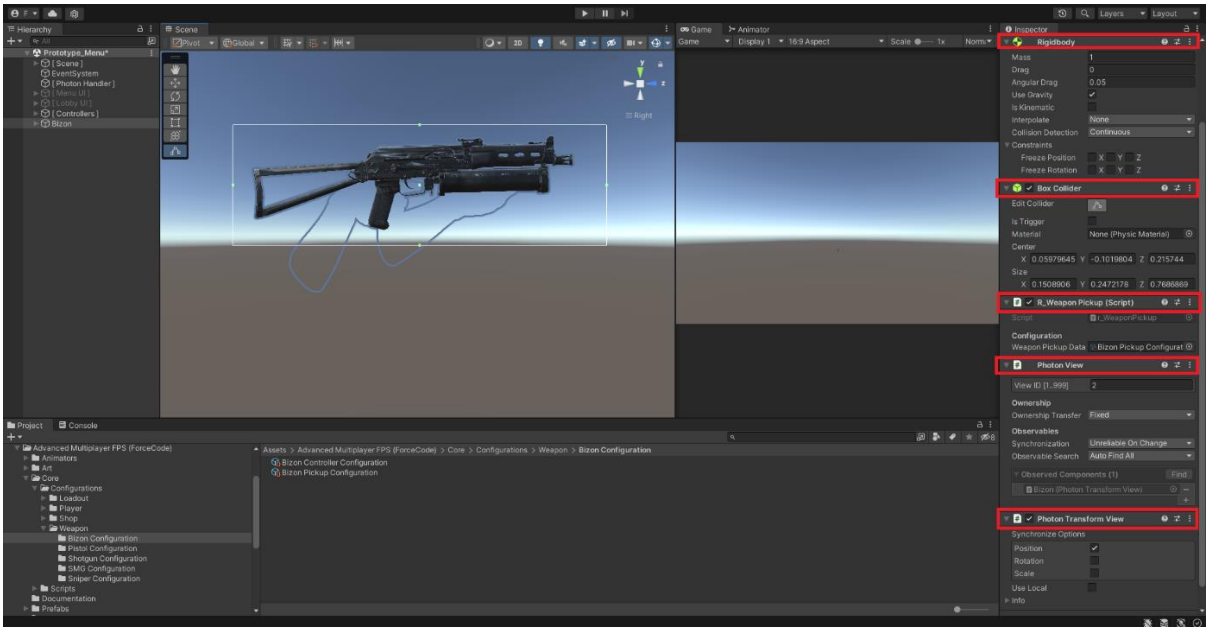
### 8.3.3 Pickup Weapon

1. Put the weapon model without hands into the scene and unpack it.

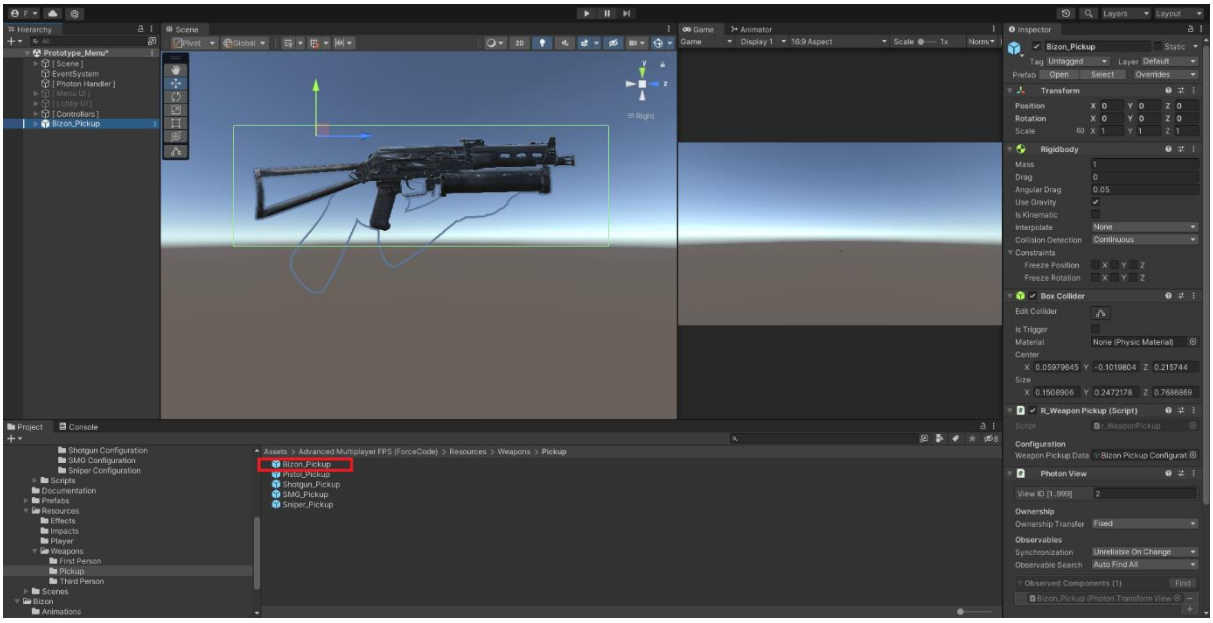


2. Make sure you add the following components to the weapon model transform with the settings:

- **Rigid body component** => change *collision detection* to continuous.
- **Box collider** => click on *Editor collider* to fit the model.
- **r_WeaponPickup** => attach the *r_WeaponPickupBase* configuration.
- **Photon View** => add component.
- **Photon Transform View** => select only position boolean.

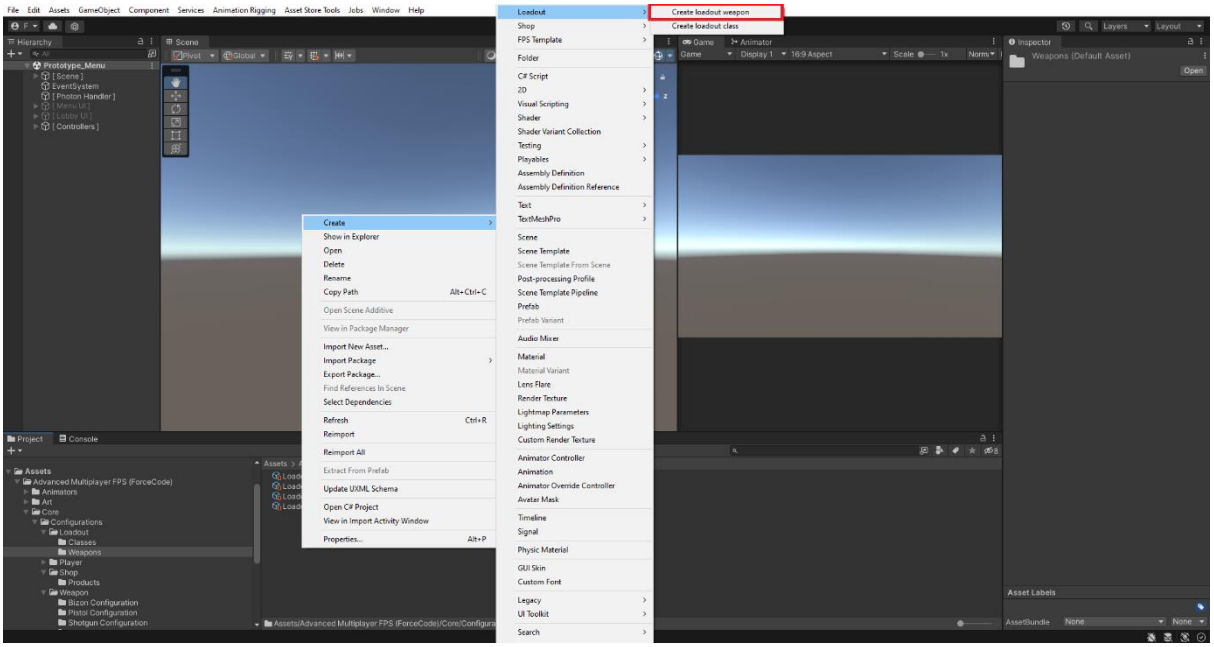


3. Change the Game Object name and save it as prefab in Resources/Weapons/Pickup.

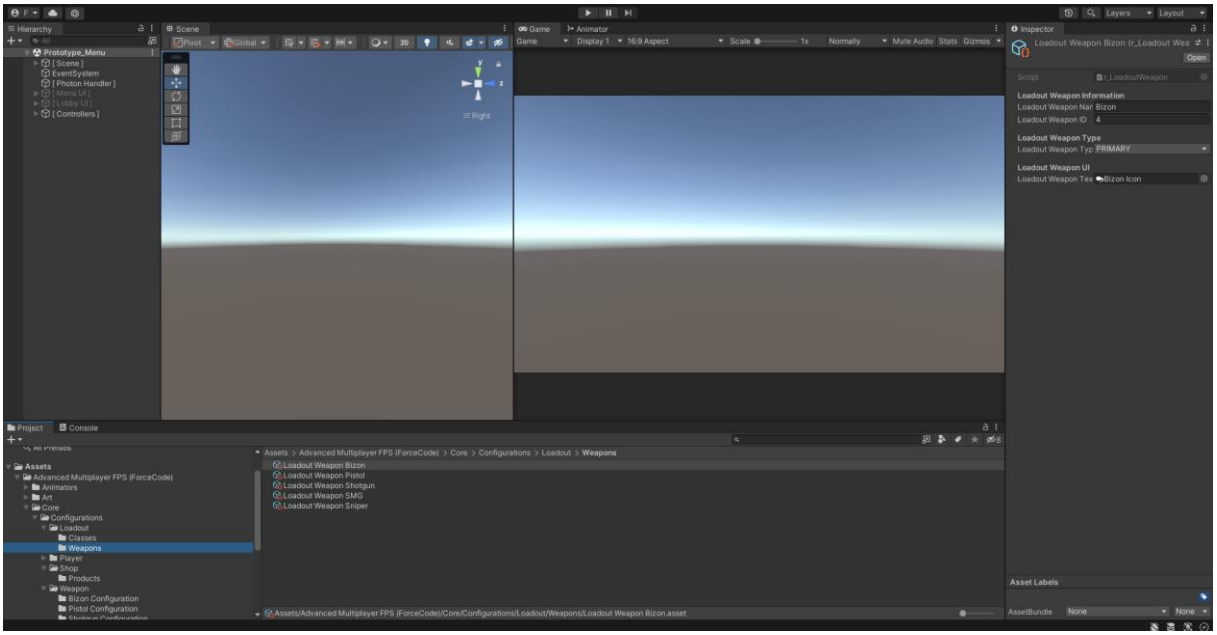


### 8.3.4 Add loadout weapon item

1. Create new loadout weapon configuration in Core/Configurations/Loadout/Weapons.

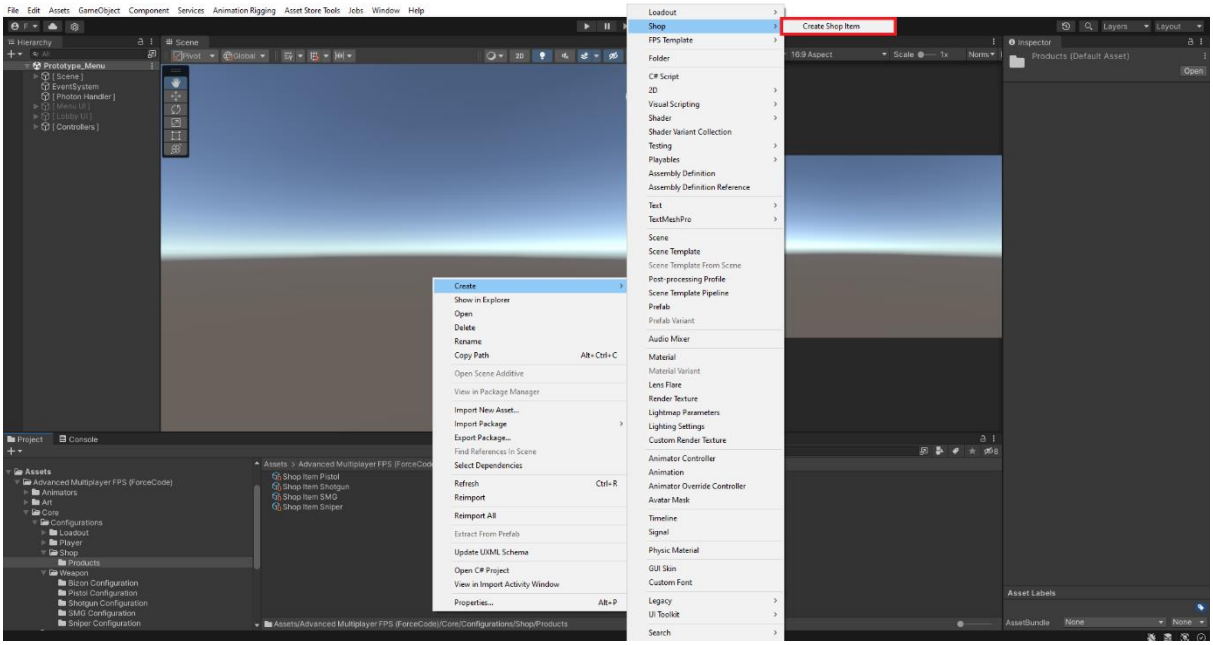



2. Set the same weapon name and ID you used for the weapon on other configurations. Attach a icon for the loadout preview.

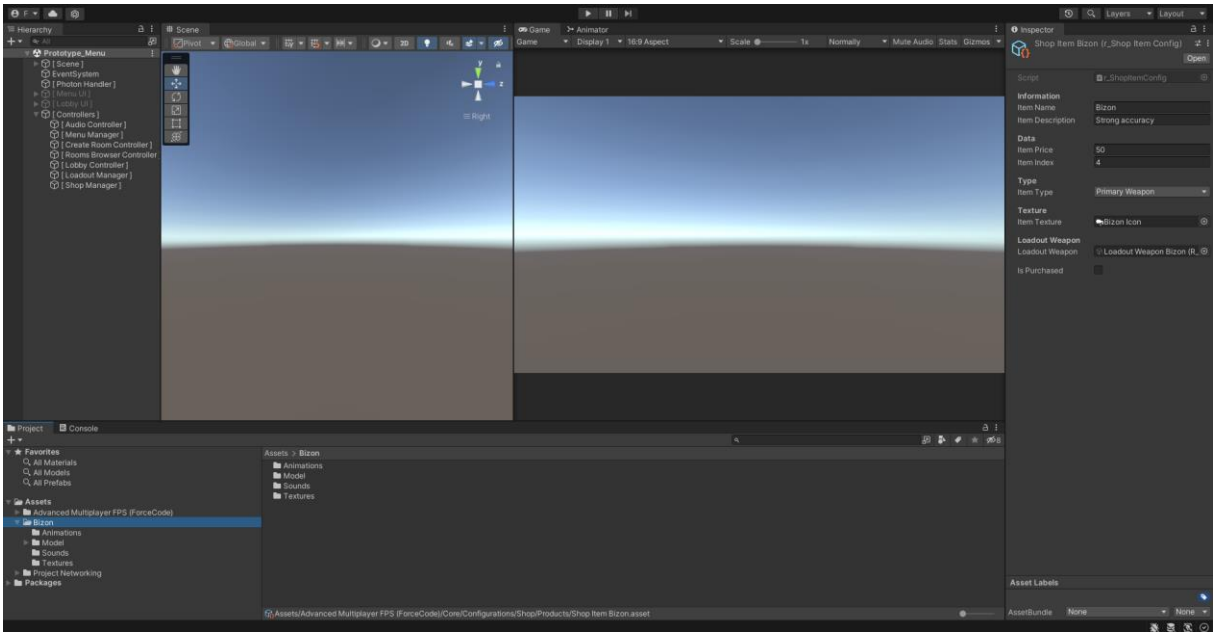


### 8.3.5 Add shop weapon item

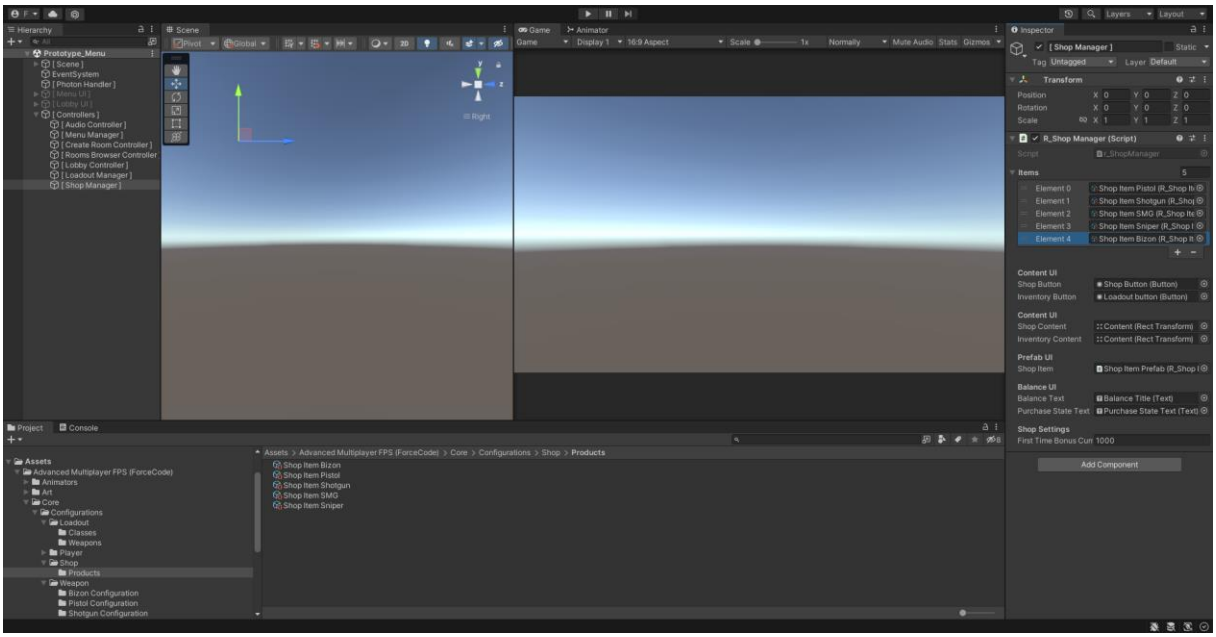
1. Create new shop item configuration `r_ShopItemConfig` in `Core/Configurations/Shop/Products`.



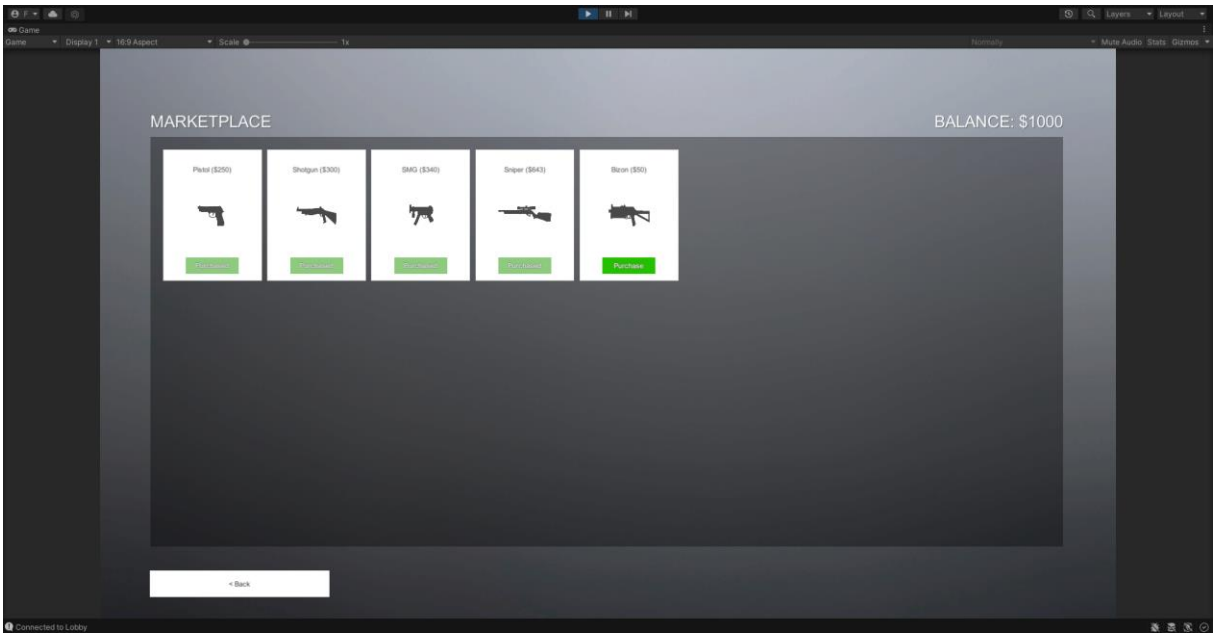

2. Set the all information of the weapon to the configuration and add the created loadout weapon configuration. Make sure **weapon name** and **ID** is the same as all other configurations for this weapon.



3. Go to the main menu scene and find **Shop Manager**. Add your weapon shop item configuration to the items list.

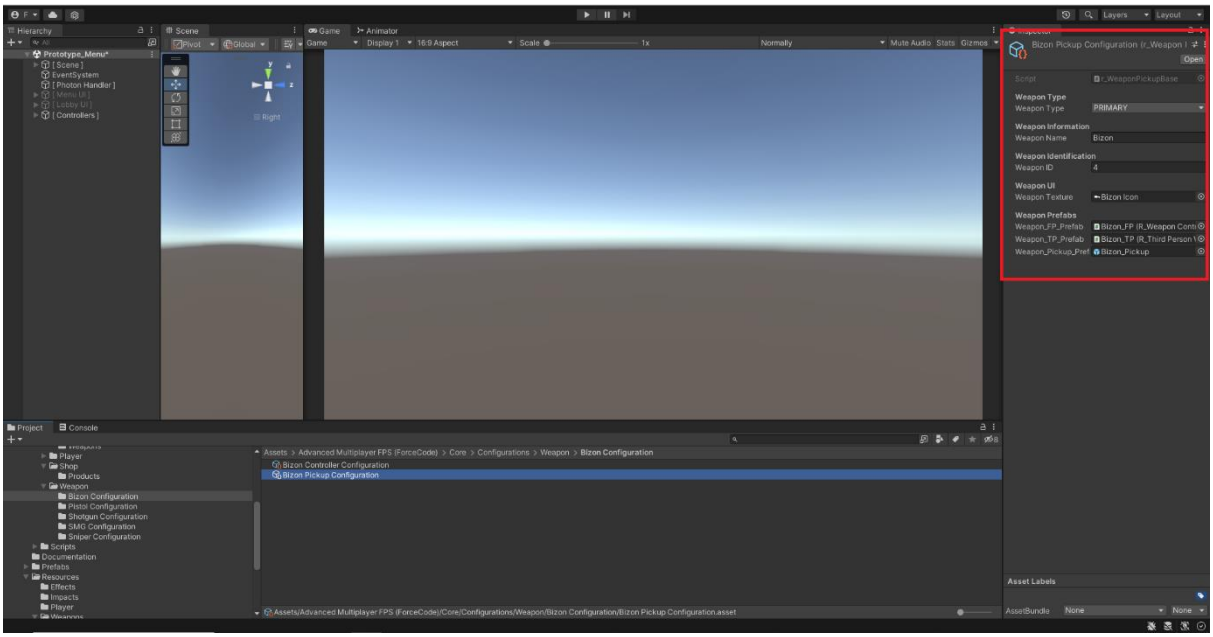


4. The weapon will be available in the shop now.

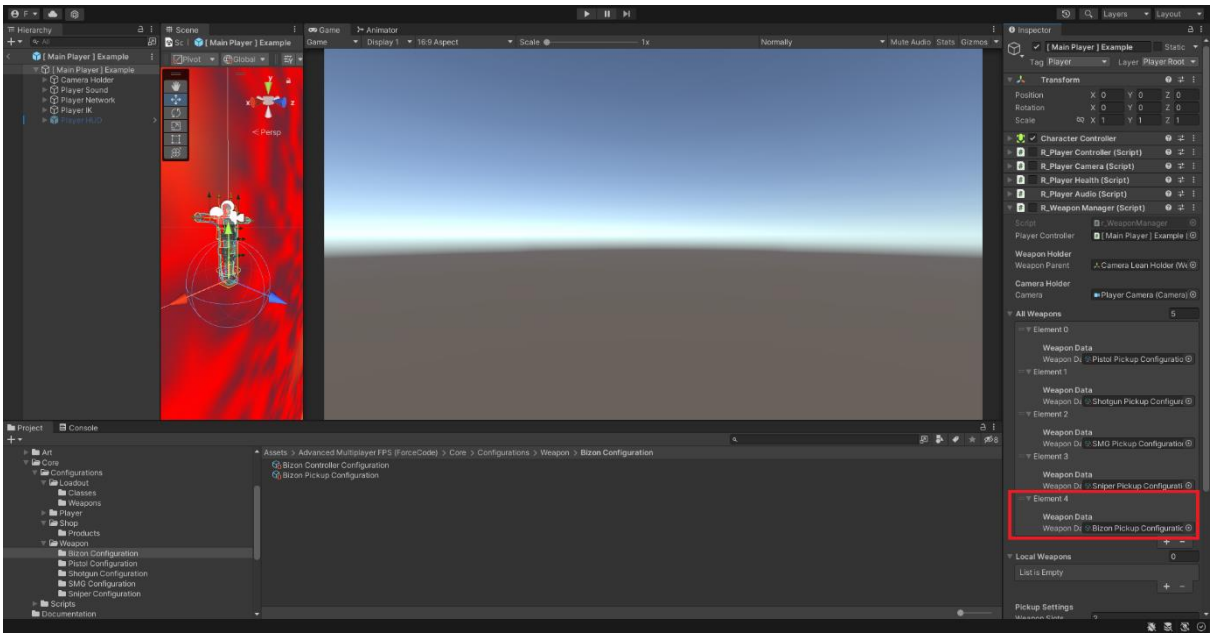


### 8.3.6 Finish weapon adding

1. Open the weapon pickup configuration, fill in the information and add all created prefabs to it.



2. Open your main player prefab and add the weapon pickup configuration to all weapon list in `r_WeaponManager` script.

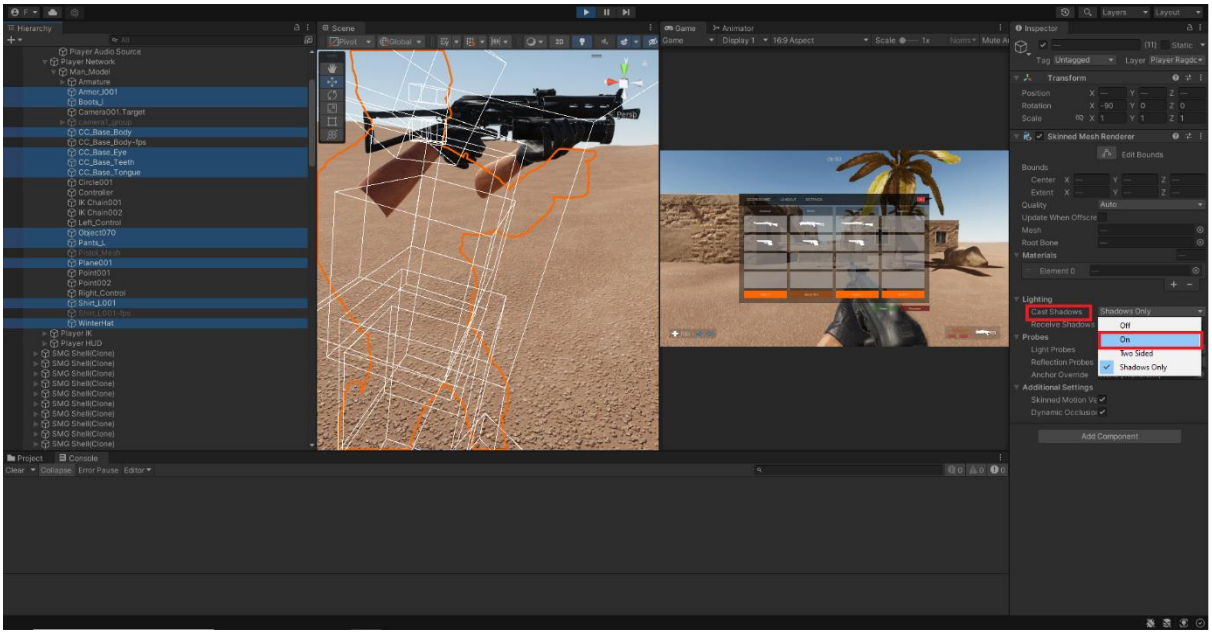


3. Get in to the game. Purchase the new weapon from Marketplace and select it to your loadout.

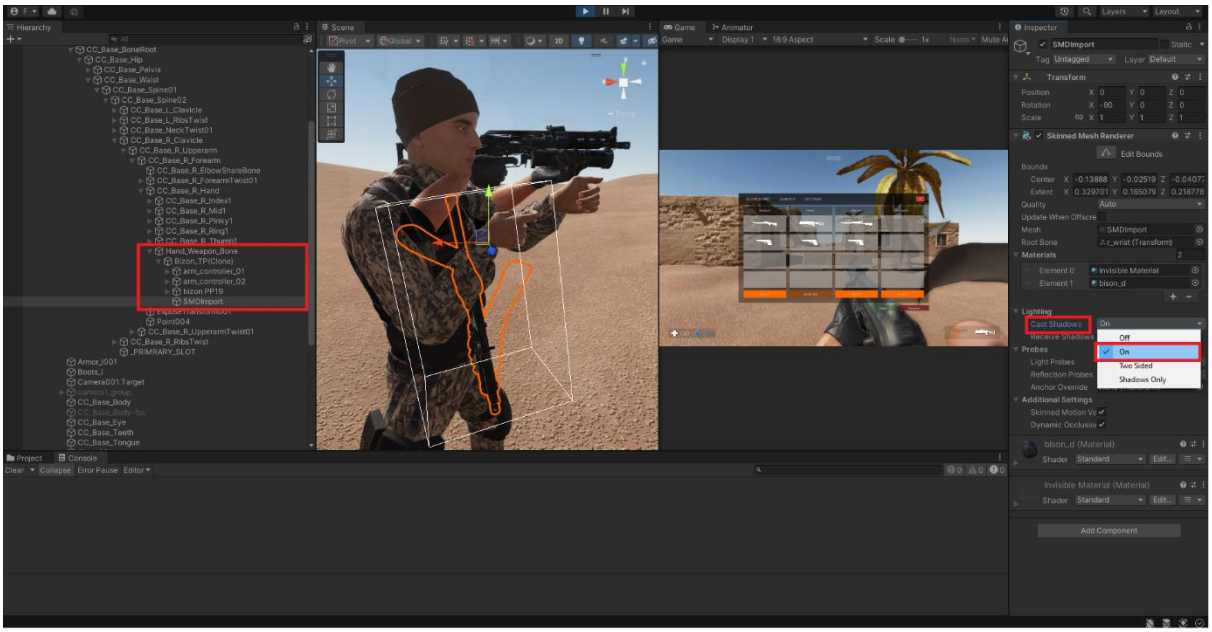
   Now it shows on the weapon loadout selection.


   As you can see everything is working. Open the weapon configuration and change everything to your own style like hip/aiming and other motions.

4. Enable the character models shadows to see the model. Select your models renderer shadows / clothing and set “**Cast shadows**” to “**On**”.



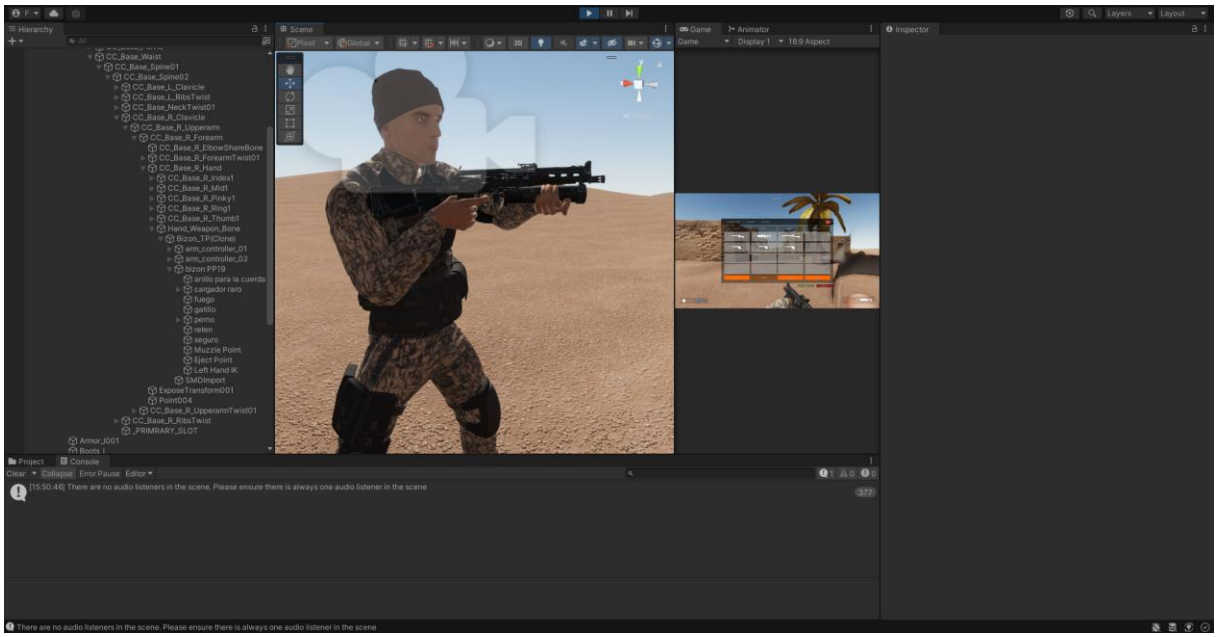
5. Select your character model with the script r_ThirdPersonManager. Click on the transform “**Weapon Parent**” in the inspector to open the armature which holds the weapon quickly. Select the third person weapon mesh and change “**Cast shadows**” to “**On**”.



6. Now you can see the character model and weapon is visible. The weapon position and rotation is not set yet. Change the position and rotation to your taste.

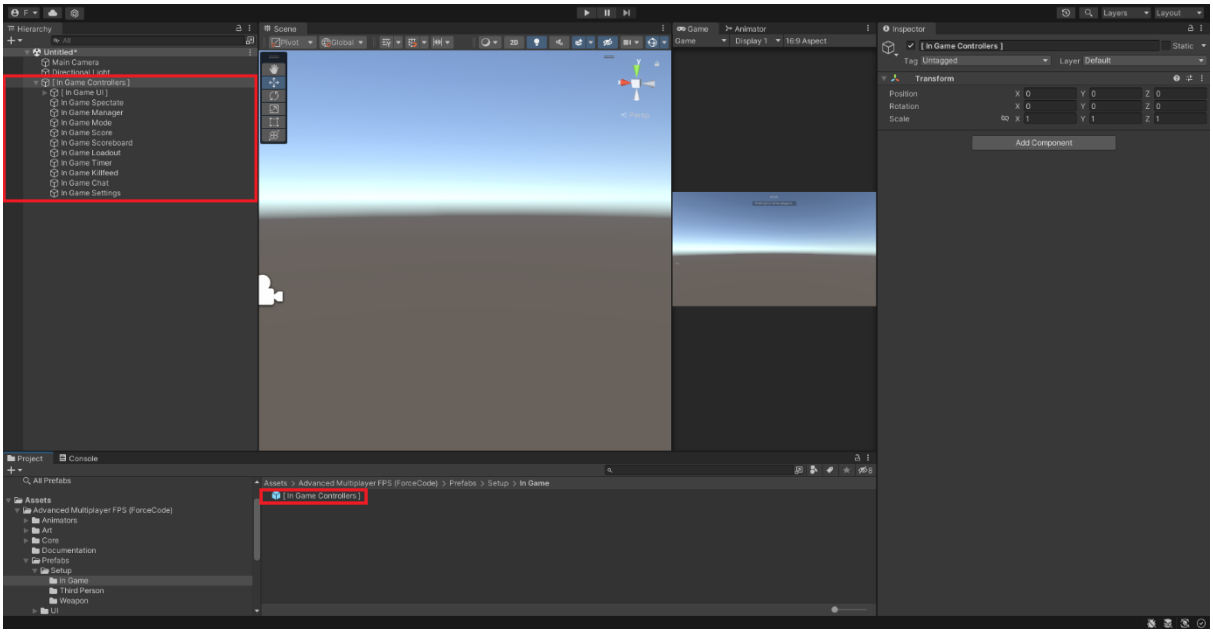
   For using the left hand IK, you can move the left hand IK transform on the third person weapon.

   Make sure you save the positions and rotation of the weapon and left hand IK, and save it to the third person weapon prefab.



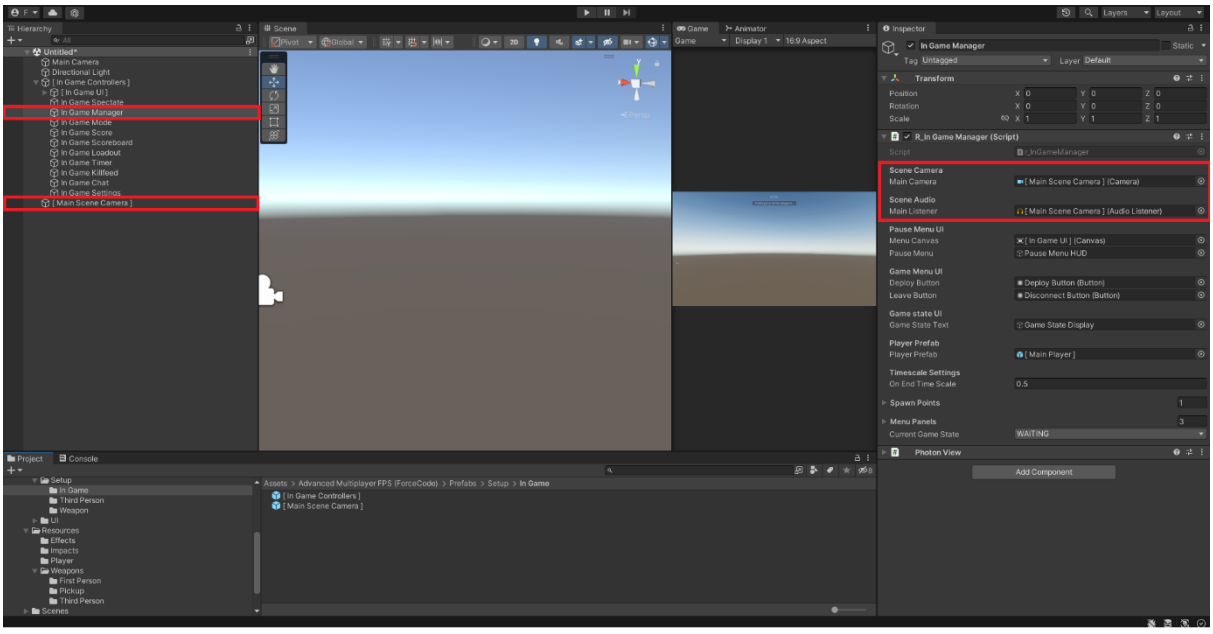
## 8.4 Adding new game scene

1. Go to the setup folder Prefabs/Setup/In Game. Create a new scene and drag the prefab "**In Game Controllers**" and "**Main Scene Camera**" to your scene and unpack it completely.



This prefab contains all the scripts and UI which is needed in the room.

2. Select the "**In Game Manager**" and drag the "**Main Scene Camera**" to the main camera and listener to r_InGameManager in the inspector.



Save the scene with name : GameMap + _ + GameMode (for example: Dust_FFA). Go to the main menu and add your map name to the create room settings.

# 9 Contact

For help or additional work, please contact us by mail or Discord (coming soon).

Mail: ForceCode1@gmail.com