Mars' Modular Multiplayer Photon FPS Kit

Table of contents

Design Principles of the kit

Abstract classes

As the name suggests, the kit is built on a modular design. To make this possible, there are a lot of base classes that are abstract, which means that they cannot function on their own and need to be overridden by a non abstract class, which can then function.

Scriptable objects

To make the kit cleaner, a lot of the so called modules are based on ScriptableObject instead of MonoBehaviour, which means that they do not exist in the game hierarchy but instead in the project. This means that runtime data, while it can, should not be saved inside them, because it is shared with all scripts that access that specific scriptable object. To solve this issue, a normal class, not a MonoBehaviour, can be saved in specific object variables in a MonoBehaviour (usually this would be Kit_PlayerBehaviour or Kit_IngameMain). Another great thing about this way is that fixed variables (such as e.g. BulletsPerMag) will not take up more memory than needed! If it was saved in the MonoBehaviour, it would take up memory each time it is instantiated. You can learn more about ScriptableObjects here and here.

Input System

Introduced for bots, the input for the whole player is stored in a class (“Kit_PlayerInput”). This allows the bots to use the same scripts as the players do. The downside is that these are only of type ‘hold’ and things such as GetKeyDown need to be coded manually using an additional variable and comparing if the state changed.

Project Setup

First Time Setup

The project comes for Unity 5.6.3p3 but can also work in other versions.
Before you can start using & modifying the kit, you will need to set the project up for the first time. To do so, you need to have a Photon AppId, which you can get here by signing up. When you have one, click on “Window” in the Unity toolbar, then on “Photon Unity Networking” and finally on “PUN Wizard”. After the wizard is open, click on “Setup Project” and follow the wizard. When it is done, you will also need to add your AppID to the “Voice AppId” in the “PhotonServerSettings” file (“Photon Unity Networking/Resources”). When that is done, you are ready to use the kit! Open the Main Menu in “MarsFPSKit/Scenes” and start to play around!

Layers

- User Layer 8: PlayerRoot
  - This layer contains only the root of the Player object. It is used so that raycasts would not hit the root collider (CharacterController) of the player.
- User Layer 9: PlayerColliders
  - This layer contains the active colliders of the player models (not of the ragdolls) so that they can ignore collisions with PlayerRoot.
- User Layer 10: IgnoreCollisionWithPlayer
  - Everything in this layer will not collide with the player.
- User Layer 11: PlayerRagdoll
  - This contains all objects of the Ragdolls so that they will not collide with the player, as they are not synced (Start position is, but Unity does not have deterministic physics, so all clients will produce different results)
- User Layer 12: LoadoutMenu
  - The default loadout menu uses this layer so that the objects may never be rendered by the scene camera

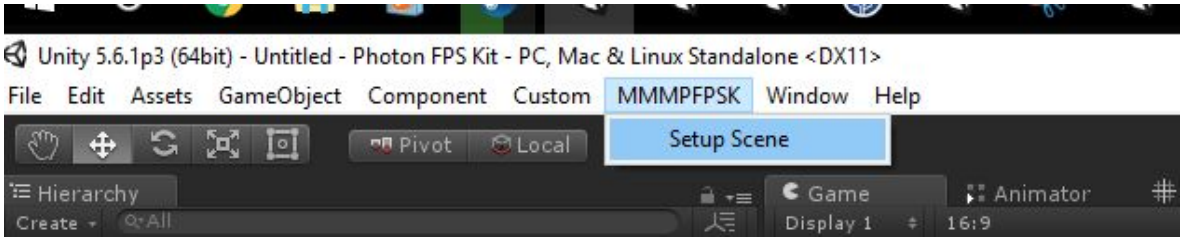
Tags

- Tag 0: PlayerCollider
  - Active player colliders (the same which are in User Layer 9) should have this tag.
- Tag 1: Concrete
  - A misc. Tag that will cause the concrete hit particles to be played (they will play by default)
- Tag 2: Dirt
  - A misc. Tag that will cause the dirt hit particles to be played.
- Tag 3: Metal
  - A misc. Tag that will cause metal the hit particles to be played.
- Tag 4: Wood
  - A misc. Tag that will cause the wood hit particles to be played.
- Tag 5: Blood
  - A misc. Tag that will cause the blood hit particles to be played.

Using the kit

Setting up a new scene

Setting up a new scene for use is very easy. Just click on “MMMPFPSK/Setup scene” and save the scene if asked.



This will save the scene, add the Ingame_Main prefab, add it to the build settings and add it to the Maps list of the game. Then you can edit the MapInformation and ingame settings as you like. You can also do these steps manually if you would rather do that.

Adding a new weapon

Prequisites:

- An animated weapon (Draw, Putaway, Reload and Fire are the animations needed at the very least; More animations like Fire Last, Fire Aim, Reload Empty, Shotgun reload can be used)
- Fitting sounds
- The weapon model separately (For third person / drop). If necessary you could

also use the one out of the animation files. I recommend to use a model with less detail than the first person one.

To add a new weapon, you first need to set up the prefabs. The variables are mostly self explaining. Put all non attachment renderers into the “All Weapon Renderers” array.
In the First Person Prefab, all renderers should be set to not cast shadows.
Regarding animations:

- Need to be on a generic rig (Mecanim)
- Duplicate the “FP_Generic” animator by clicking on it and pressing (Ctrl + D)
- Assign your new animations inside the animator. It is important that you do not change the names of the Mecanim States
- If you don’t have all needed animations (Like Fire Last / Fire Aim) you can also use other animations such as normal Fire or leave it out (If you don’t have a Dry Fire animation, you should leave that blank)
- Assign the animator to the instance of your model, and assign that Animator to the Kit_WeaponRenderer
- For reference, see the default weapons
- I recommend duplicating one of the default weapons so you can use the muzzle flash of that one and also keep the Run Pos / Run Rot settings

Do the same with the Third Person Prefab and the Drop prefab. If you already want to set up attachments, make sure that all three prefabs line up (The weapon information will tell you if that is not the case).
Once the prefabs are done, go into the Weapons folder (By default: “MarsFPSKit/GameInformation/Player/Weapons”) and select the appropriate category (You can add / remove / modify however you like, it is just for organizing it). Either create a new Kit_WeaponInformation and Kit_ModernWeaponScript or duplicate one. I strongly recommend to duplicate one because it is easier. Assign the Prefabs and assign your sounds on the **Behaviour**. Assign the Behaviour to the WeaponInformation. This is very important. Once you are done with that, you can add the WeaponInformation to the GameInformation (By default: “MarsFPSKit/GameInformation/Game.asset”). The array’s name is: “All Weapons”. Once you added that, your weapon is in game. Now you can start tweaking aim position, left hand IK position and so on to your liking. Note that some things (such as RPM) will not update while you are playing, to save some time on calculating.

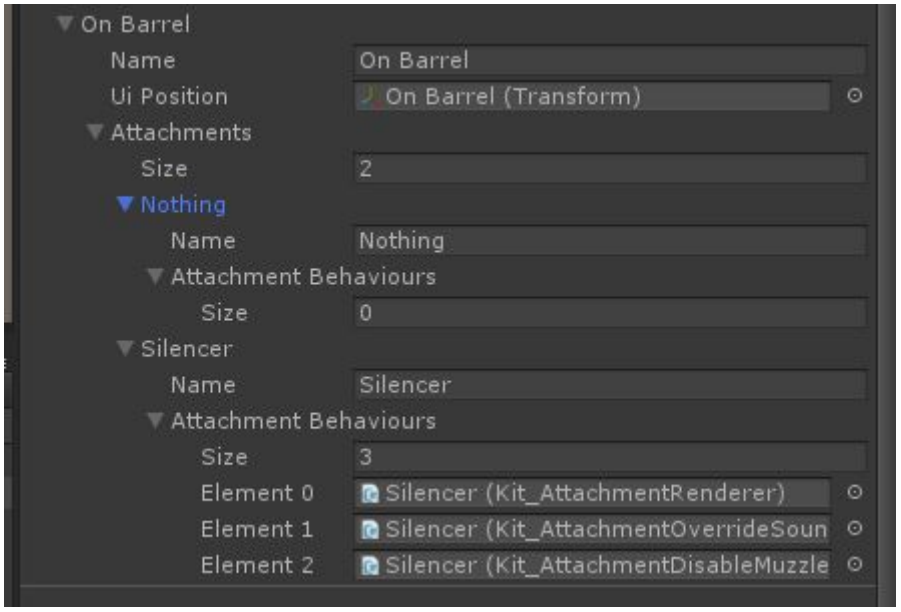
Using the attachments system

The attachments of a weapon are controlled purely by the assigned prefabs (First Person, Third Person and Drop).
Attachments are split up in two parts:

1. Attachment Slots
2. Attachments

A slot is something like sights, on barrel, etc..
An attachment is a child of a slot, it would be a Red Dot or a suppressor.
To add a new slot, simply extend the array “Attachment Slots” on all three prefabs and adjust the new slot name and UI position to your liking.

To add a new attachment, simply extend the “Attachment” array on the appropriate slot and add the behaviours you want for it.



Extended information on this subject can be found here.

Setting up spawns

In order to add a new spawn, you need to place a game object in the game world where you want the spawn to be. When that is done, add the “Kit_PlayerSpawn” script to it. In the “Game Modes” array, add all the Game Mode Behaviours, which you want the spawn to be enabled for. If the game mode supports multiple spawn groups (for example that could be in a domination type game mode, where 0 could be normal spawns, and 1 - 3 would be spawns for each flag). If it doesn’t, just leave it at 0. Don’t forget to save the scene and then you are done with setting up a new spawn. The game mode behaviours will automatically find the right spawns, so **LEAVE THE GAME OBJECT ACTIVE!**
A red gizmo box will be displayed where the spawn is. If you want to hide them, click on the script and collapse it in the inspector.

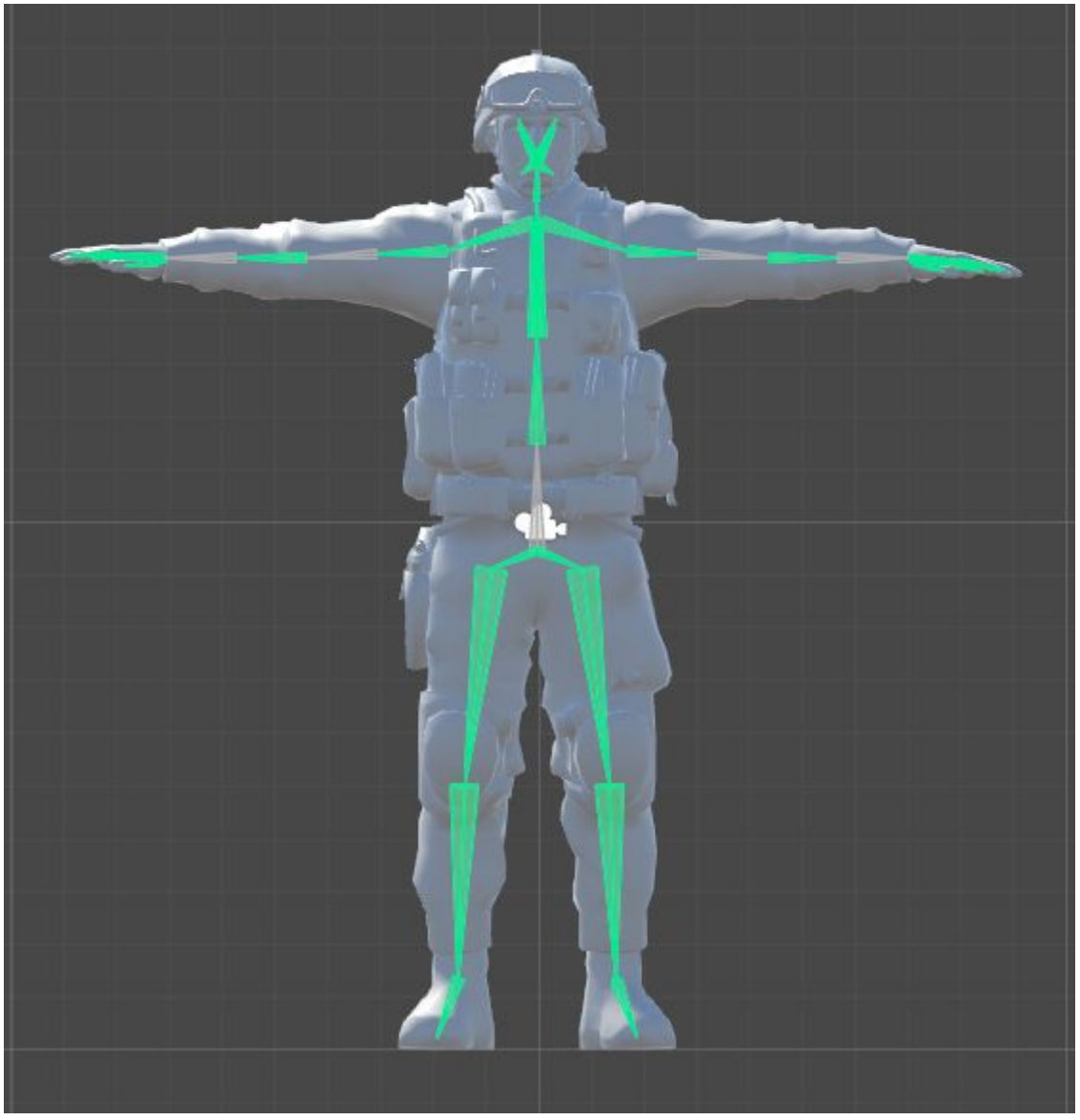
Setting up a new player model

Prequisites:

- A rigged, bipedal 3d model

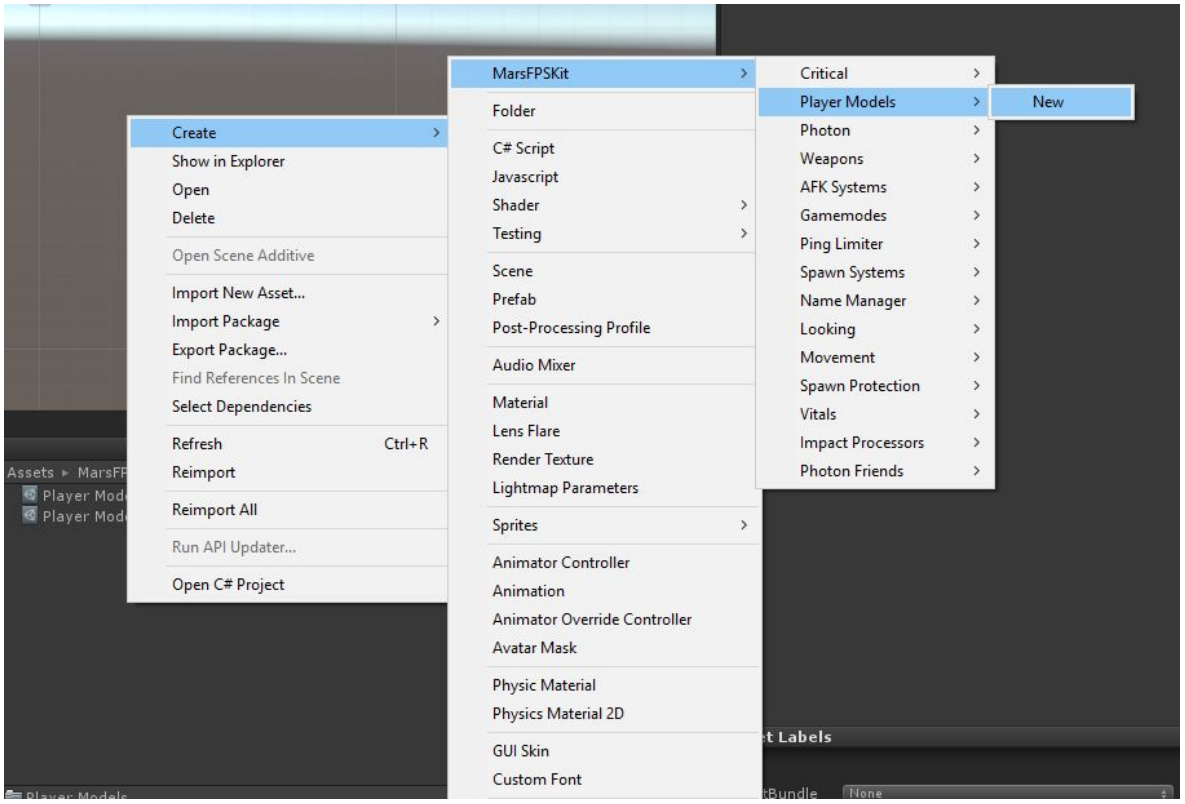
Preparing the model

1. Set its animation type to “Humanoid”, so Mecanim can retarget the animations
2. Configure the avatar to look like this:

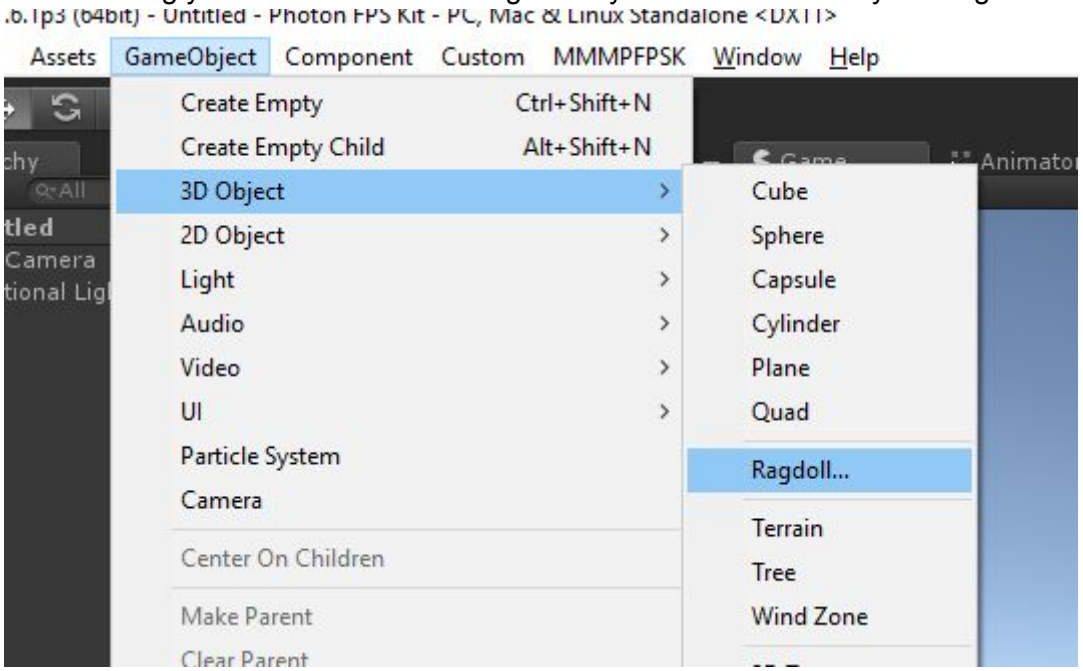


That was the preparation.
A player model consists of two prefabs and one GameInformation asset.

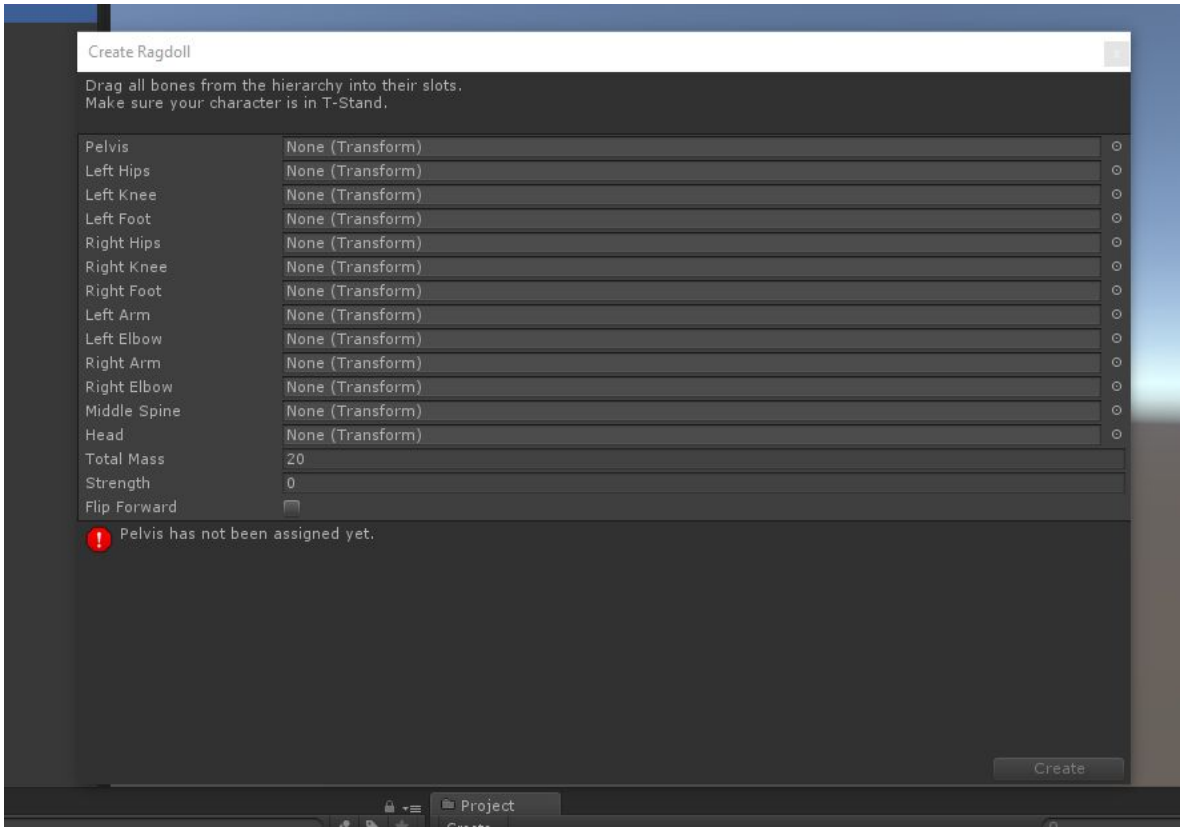
1. The actual third person prefab, which will be animated and which also has the hit colliders on it.
2. The ragdoll prefab, which will be initially synced, so it has to be in the “Resources” folder so that Photon can find it. Its name should also be unique.
3. The Game Information is a scriptable object of type “Kit_PlayerModelInformation” and can be created like this:



The first thing you should create is a ragdoll of your model. Do so by clicking on:



And follow the wizard:

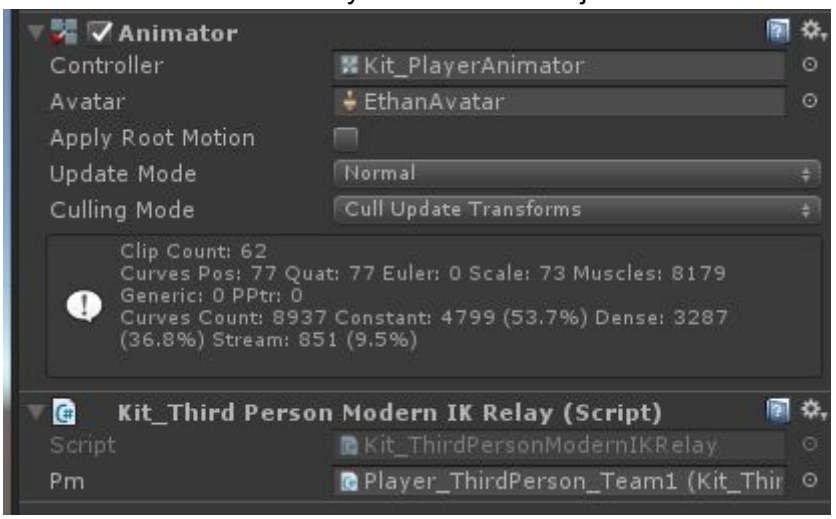


Once you are done creating the ragdoll. Check if it setup correctly (Collider size).

1. Now duplicate the ragdoll, so you have two instances of it in the scene
2. One of them will be the player model:
3. Remove all Joints and Rigidbodies from the ragdoll
4. Put it in the Layer “PlayereColliders”
5. All colliders should have the tag “PlayerCollider”
6. On all colliders, add the Kit_PlayerDamageMultiplier”
7. Now set up the ragdoll ID for each of them. It should follow the hierachy. The first one is 0, the second one is 1, third is 2 and so on, like this:

8. Now add a Kit_ThirdPersonModernPlayerModel to the root object.
9. Assign the Colliders to the “Raycast Colliders” array in the exact same order as you numbered them, like this:

10. Do the same with the “Ragdoll Collider Copy” array
11. Now assign all renderers to the “Fp Shadow Only Renderers” array. These will be set to “Only Shadows” when you are in first person.
12. Open one of the default player models and duplicate the “Weapons In Hands Go” object and put it on your new model.
13. Assign it to the “Weapons In Hands Go” property
14. Also assign all of the Sounds, which are under the “WeaponsInHandHolder” object
15. Assign the “Kit_PlayerAnimator” to the Animator of your humanoid model. It can be found in “MarsFPSKit/Animators”. Also make sure to turn off “Apply Root Motion”
16. Put the “Kit_ThirdPersonModernIKRelay” on the same object that the Animator is on, like this:



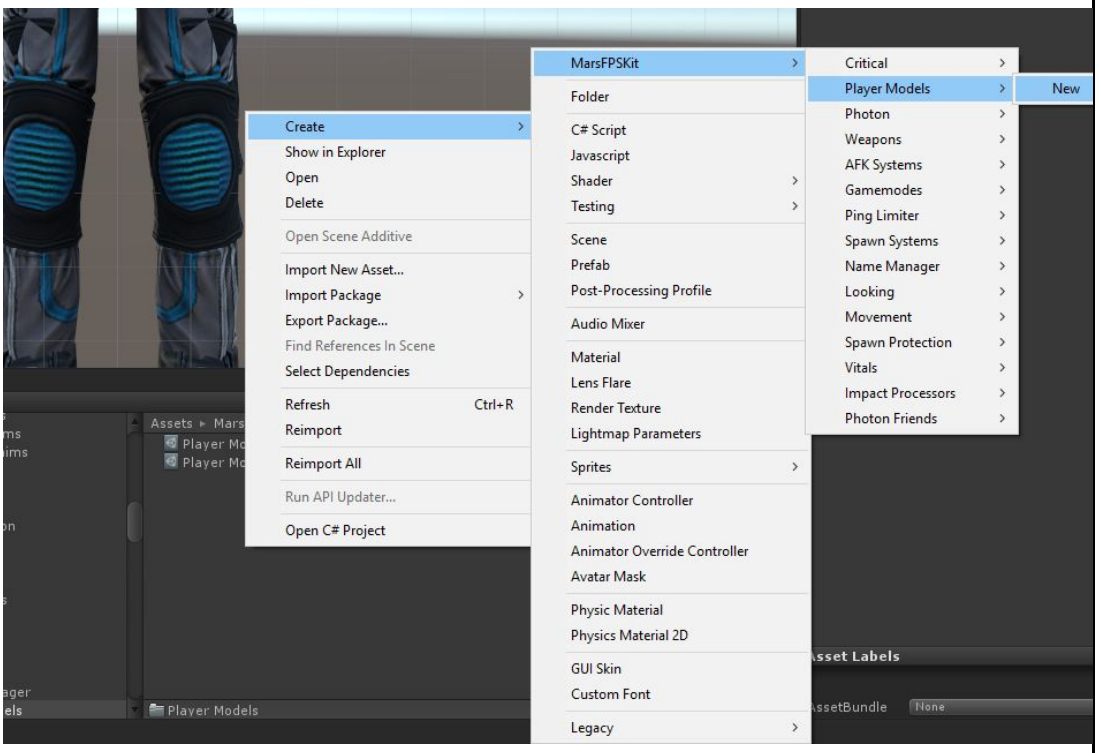
Make 100% sure that you have put “Apply Root Motion” to false.

17. Now assign the Animator of your model to the “Anim” property on the Kit_ThirdPersonModernPlayerModel sscript
18. Now create a NameTrigger and a NameAboveHeadPos object, you can look in the default model’s prefab to see how they are positioned and assign them to the “Name Above Head” properties.
19. Now assign the object, which has the Animator on it, to the “Ragdoll Go” property.
20. Save your Player Model as a prefab somewhere. I recommend following the default scheme, but it is not required.

21. Now it is time to set up the ragdoll:
22. Put all colliders in the “PlayerRagdoll” layer and assign the tag “Blood” to them
23. Add the “Kit_FirstPersonRagdoll” script to the root object and a Photon View
24. Set the Life Time as you like
25. Assign the colliders and rigidbodies in the EXACT same order as on the Player Model:

26. Create a “CameraPos” transform and make it a child of the head bone, like this:

27. Position and rotate is as you like. The camera will be parented to it when you are dead.
28. Now assign that transform to the “Camera Pos” property of the ragdoll script.
29. Make the ragdoll a prefab. It is VERY important that it is in the “Resources” folder. Otherwise it will NOT work. Also make sure its name is unique. Now assign that **PREFAB** to the “Ragdoll Prefab” property of the **Player Model Prefab**.
30. Now it is time to add it to the game. Create a new Kit_PlayerModelInformation asset, like this:



31. Now assign your MODEL prefab to the “Prefab” property of your newly created Information asset:

32. Last but not least, add it to the array of the team that you want it to use, in the “All Team X Player Models” array of the GameInformation (By default: “MarsFPSKit/GameInformation/Game.asset”. You can also replace the existing one. Both teams can use the same player model if you’d like to.
33. Now test your player model. You probably need to adjust the CamersPos and the “WeaponsInHandHolder”.
34. You’re done!

Modifying the UI

- Main Menu:
  The main menu is split in four parts:
  - Main menu

- Login menu
- Photon Friends menu
- Options menu

They all share the same canvas. Modifying them is very easy, just tinker around with the objects in the canvas and rearrange / modify as you like. Just make sure you reassign something new in the scripts if you deleted something. To avoid that, I recommend keeping things such as Texts in place and only changing their settings, but it is not required.
Prefabs used in the main menu (Can be found in “MarsFPSKit/Prefabs/UI)

- FriendEntryPrefab
  - Used for displaying a friend in the Photon Friends menu
- Options Menu
  - Is already present in the main menu as a disconnected instance
- RegionEntryPrefab
  - Used to connect to a different Photon Region in the Region menu
- ServerBrowserEntryPrefab
  - Used to display a room in the “Browse Games” part of the menu

- In Game UI

  Everything is a part of MarsFPSKit_IngamePrefab, which can be found in “MarsFPSKit/Prefabs”. This prefab is also the base of the kit and required in every map’s scene.
  Things that are a part of it (colored parts are a module):

  - Pause Menu
  - Options Menu (Is a completely separate MonoBehaviour)
  - HUD
  - Kill Feed
  - Chat
  - Kill UI (points that pop up)
  - Victory Screen
  - End game Map Voting
  - Ping Limit
  - Afk Limit
  - Voting menu
  - Loadout Menu (Is also a completely separate MonoBehaviour)

  Every UI can be modified as usual. Just make sure to reassign UI components if you deleted something.
  Connected prefabs (Can be found in “MarsFPSKit/Prefabs/UI)

  - Everything in the folder “Game Mode HUDs”
  - ChatEntryPrefab
    - Used to display a chat message inside the chat UI
  - KillFeedEntryPrefab
    - Used to display a “Who killed whom” message
  - MapVotingEntryPrefab
    - Used for the voting at the end of the round
  - PlayerMarkerRootPrefab
    - This is the arrow / name that floats over the head of a player
  - ScoreboardPrefab
    - This displays a player in the scoreboard in a non team based game mode
  - ScoreboardTeamPrefab

        - Same as the former except for team based game modes
    - VoiceChatEntryPrefab
        - This displays an actively talking player in the lower left (by default)
    - VotingSelectionEntryPrefab
        - This displays the voting if a player starts a voting process
    - WeaponCustomizationDropdownSlotPrefab
        - Used to change attachments in the loadout menu when you click on customize

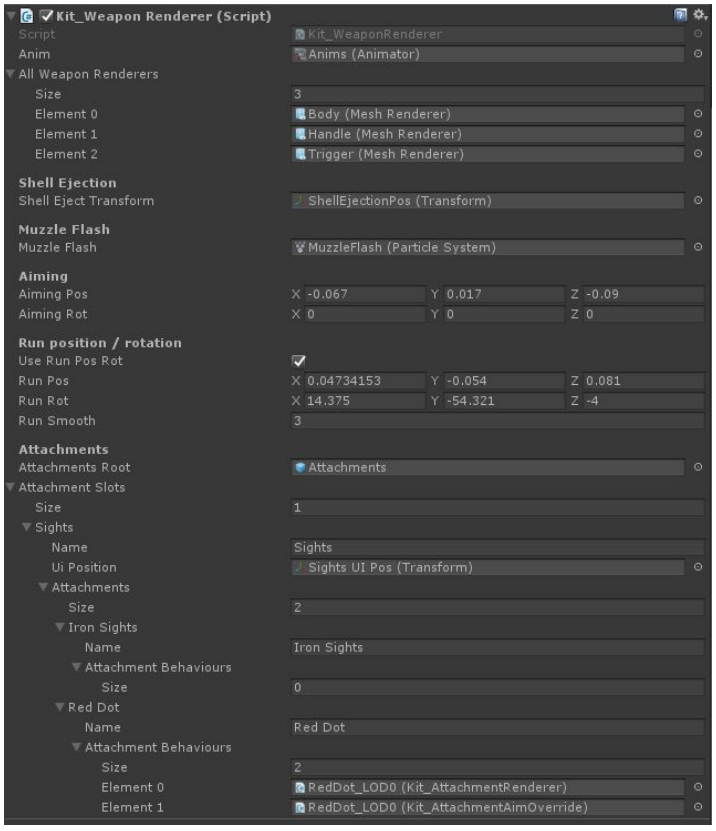
Extended Information

Weapon Setup

In the Mars FPS Kit, a weapon consists of:

- First Person Prefab
- Third Person Prefab
- Drop Prefab
- Kit_WeaponInformation
- Kit_WeaponBehaviour

The three prefabs are essentially the 'looks' of the weapon. These prefabs also carry the information for the attachments. It is VERY important that these line up. For example, it could look like this:
First Person Prefab:



- “Anim”

  This animator controls the animations of the weapon.

- “All Weapon Renderers”

  All renderers of this weapon should be assigned to this. Only default renderers, exclude attachments! These will be enabled when the weapon is selected and disabled when it is not.

- “Shell Ejection Transform”

  This is where the shell will be ejected, if it is enabled in the behaviour.

- “Muzzle Flash”

  This is a particle system that, if assigned and not disabled by attachments, will be played when the weapon is shot.

  NOTE:
  If you want to use multiple particle systems, use on system as the “master” and make all other particle systems children of this master in the hierarchy. Then assign the master to the slot.

- “Aiming Pos/Rot”

This is the default goto position and rotation in local space for ADS. It can be overridden using attachments. More about that here.

- “Use Run Pos Rot”

  If it is set to true, the weapon will be rotated to “Run Pos” and “Run Rot” with the speed multiplier of “Run Smooth”.

- “Attachments”:
  - “Attachments Root”

    This will be set to active / inactive depending on whether the weapon is currently in use (On screen). If you don't use attachments, you can leave this blank
  - “Attachment Slots”

    An attachment slot is a place where an attachment can be assigned. For example this could be “Sights”, “Under Barrel”, “Shaft”, “On Barrel”, etc..
  -
    - “Name”:

      This is the display name of this slot

    - “Ui Position”:

      This is where the Dropdown UI element for this slot will be displayed in the loadout menu. It is only required on the first person prefab but due to shared classes, will also be visible on the third person and the drop prefab. You can leave it blank there, it will do nothing.
  - “Attachments”

    A slot will be populated with the actual attachments. This could be various sights such as a Red Dot, a Micro T1, a sniper scope, a Kobra sight, etc or other stuff such as a suppressor or an extended barrel.
  -
    - “Attachment Behaviours”

      These define how an attachment acts. If it is empty, the attachment will do nothing, which can be useful if a slot is empty.

    - Currently available behaviours:
      - “Aim Override”

The “Aim Override” will override the “Aim Pos” and “Aim Rot” as described above. Useful for sights.

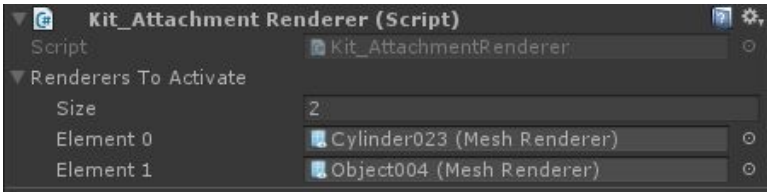
- “Disable Muzzle Flash”

The “Disable Muzzle Flash” will disable the muzzle flash. Useful for suppressors.

- “Override Sounds”

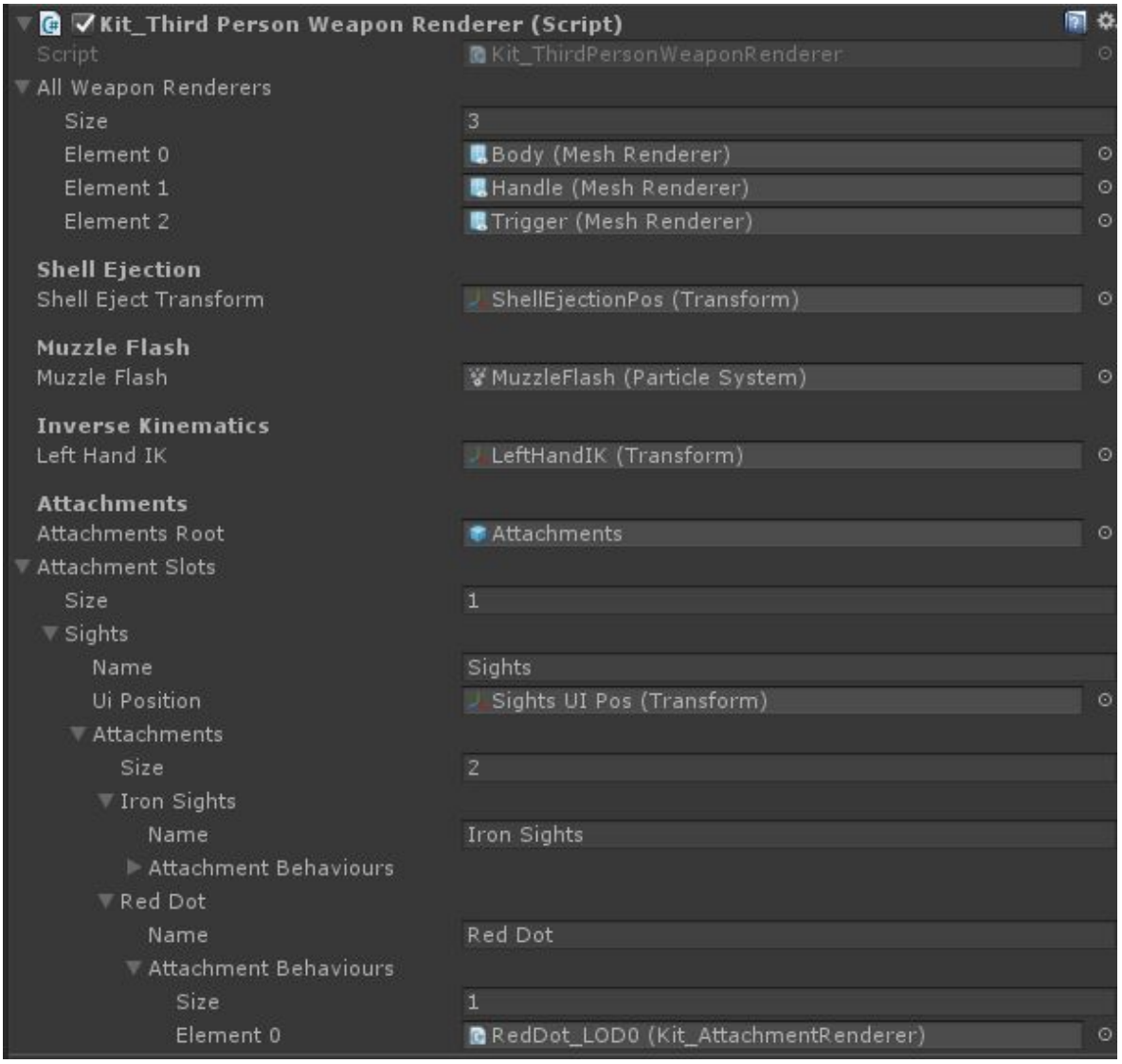
The “Override Sounds” will override the fire sounds and the third person falloff curve and range. Useful for suppressors.

- “Renderer”



The “Renderer” will enable the given renderers if the attachment is selected. Useful for any type of attachment.

Third Person Prefab:



As you can see, it is mostly the same as the first person prefab. It mostly behaves the same except:

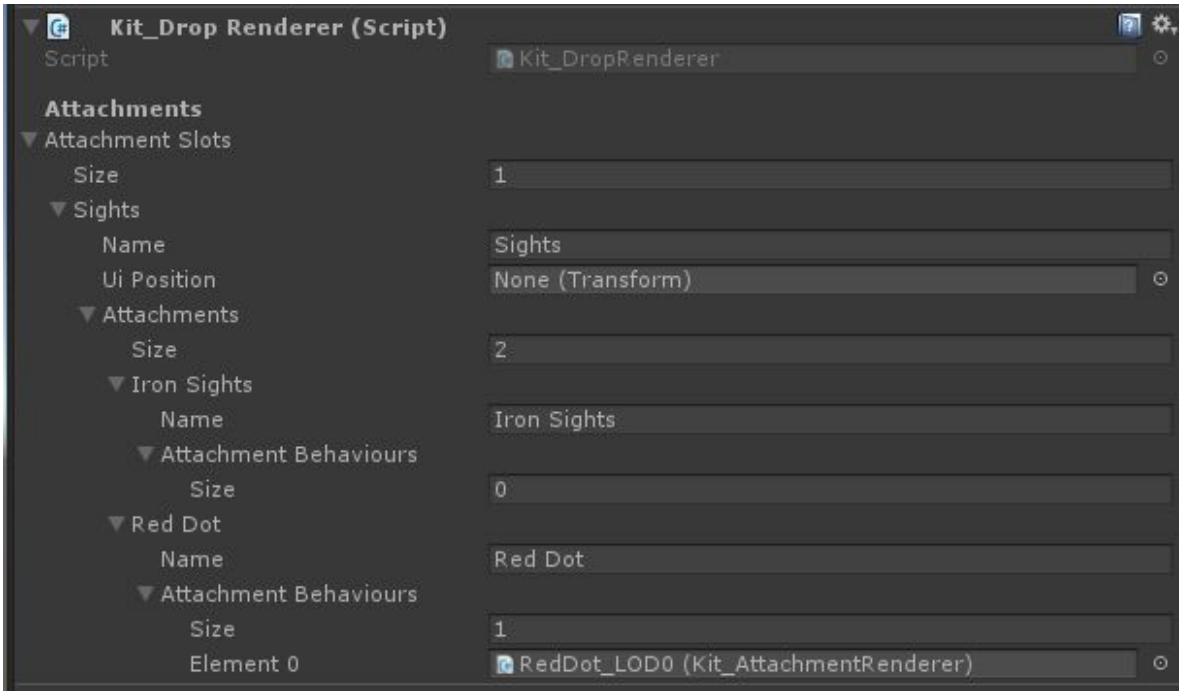
- “Inverse Kinematics”

  “Left Hand IK”: If this is assigned, the left hand will be positioned at the assigned transform using Mecanim. It is very useful to match the third person animations to the weapons.

- “Attachments”

  Certain attachment behaviours will not do anything in third person:
  - “Aim Override”: As there is no re positioning in third person, it will not do anything.

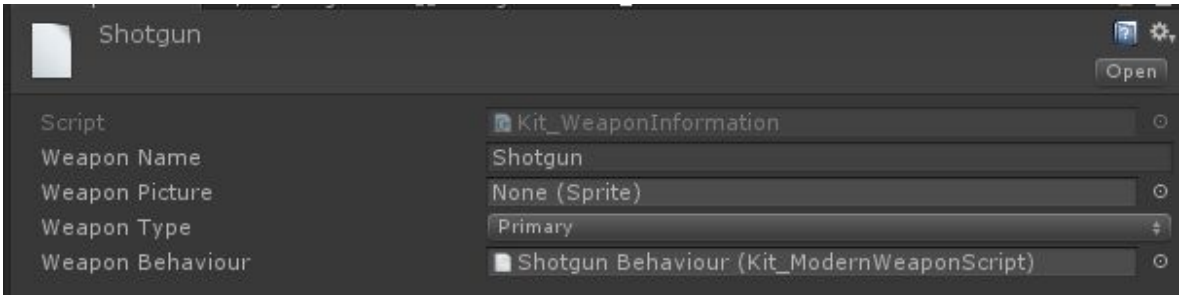
Drop Prefab:



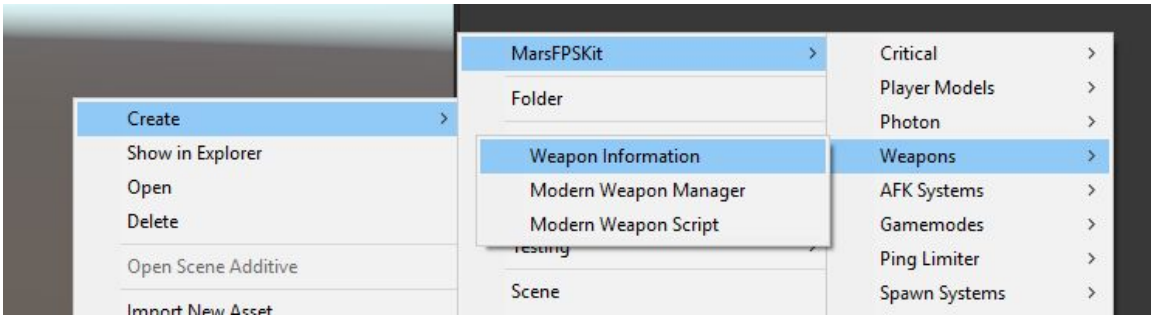
As you can see, it only has the attachments. Weapon drops don't do much except being there and looking correct.

NOTE REGARDING ATTACHMENTS:
Make 100% sure that First Person, Third Person and Drop prefab line up 100% in Slots and Attachments. Otherwise you will see errors and incorrect behaviour. The attachment behaviours can (and should be) different.

Kit_WeaponInformation:



This is a Scriptable object and can be created with a right click in the project and selecting:



- “Weapon Name”:

  How is this weapon called?

- “Weapon Picture”:

  Preview picture of this weapon for the loadout menu

- “Weapon Type”:

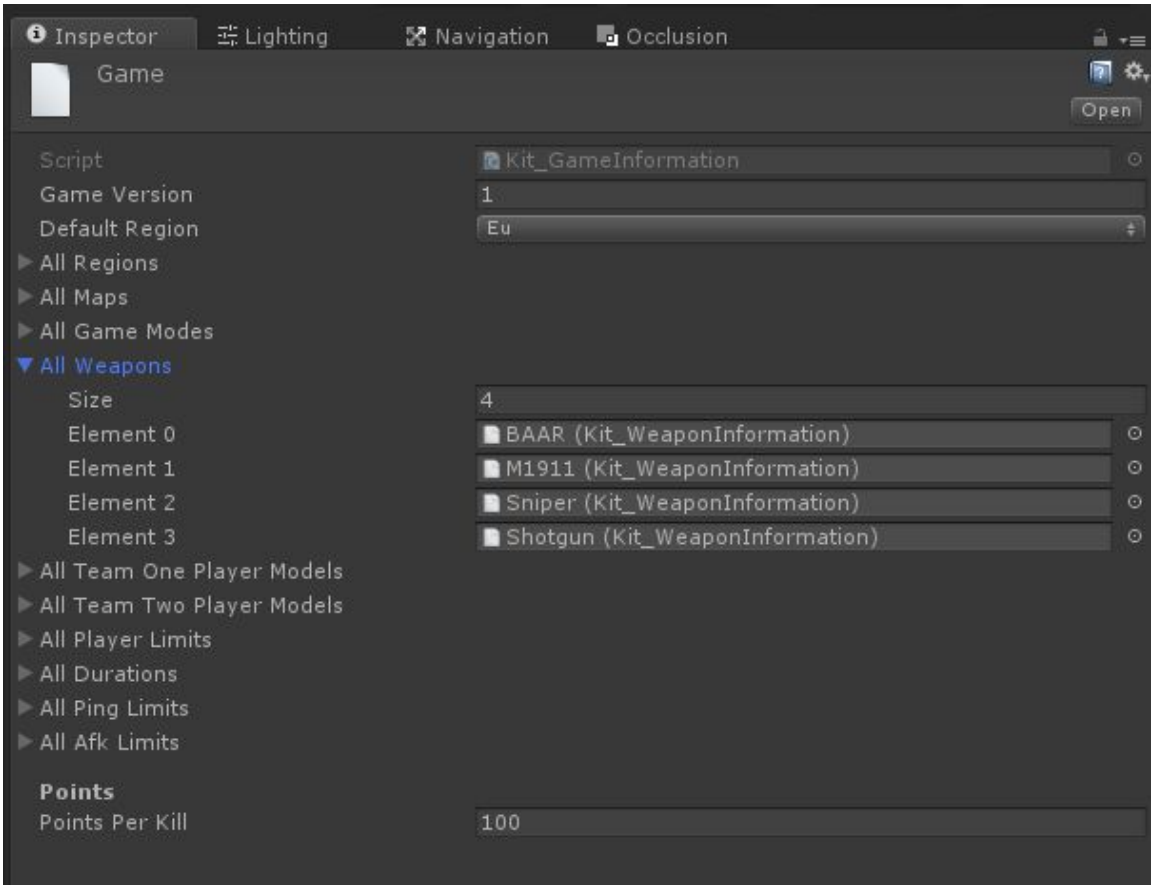
  Should this appear as a primary or secondary weapon?

- “Weapon Behaviour”:

  A scriptable object that defines how this weapon acts.

In order for a weapon to appear in game, it has to be in the “All Weapons” array of the “Kit_GameInformation” object, which can be found (by default) in:

“Assets/MarsFPSKit/GameInformation/Game”. It should look like this:

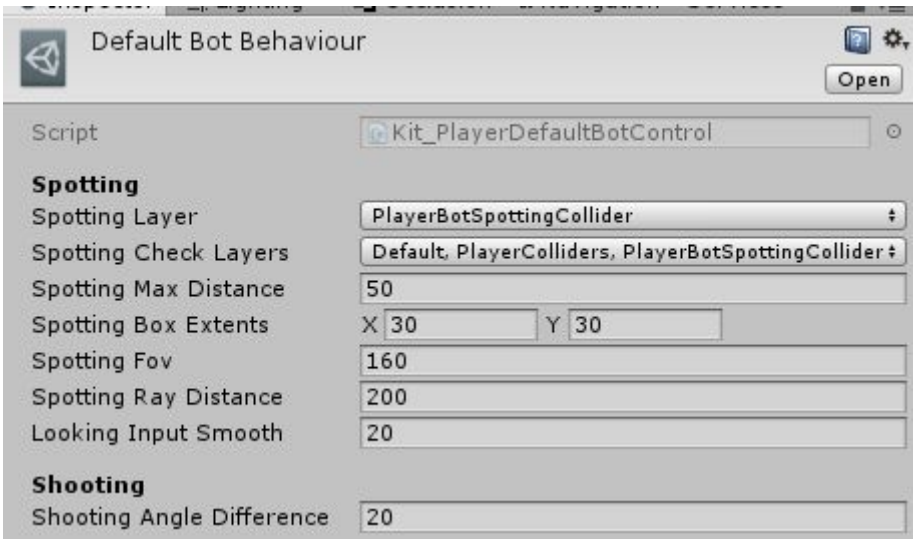


Bots

How they work

All input for the player is stored in a class and updated from the input component. This allows bots to “simulate” player input and use the same scripts as a normal player would. Always consider this when writing your own scripts!

Default AI Behaviour



Spotting:

Spotting Layer:
This is the layer that bots will “scan” for. The bot scan helper, which is a small box collider that is set to trigger, should be in this layer.

Spotting Check Layers:
These layers are checked with a raycast to check if a bot can see a player. This should include your game geometry aswell as the layer of the hit colliders of the player models.

Spotting Box Extents:
The box that the bot will use to check for players that he can see. Z is Spotting Max Distance.

Spotting Fov:
Essentially the camera FoV for bots. From the forward vector (of the camera), how big is the bot’s field of vision?

Spotting Ray Distance:
How far will the raycast mentioned in Check Layers go?

Looking Input Smooth:
Essentially how fast the bot’s simulated mouse input is. Increasing this will make bots a lot harder, decreasing this will make them easier.

Shooting:

Shooting Angle Difference:

At which angle will a bot start shooting (‘pressing left mouse button’)? To calculate this, the direct aiming vector (ideal shooting vector) is compared with the current forward vector of the bot’s ‘camera’.

Platforms

WebGL

Notes

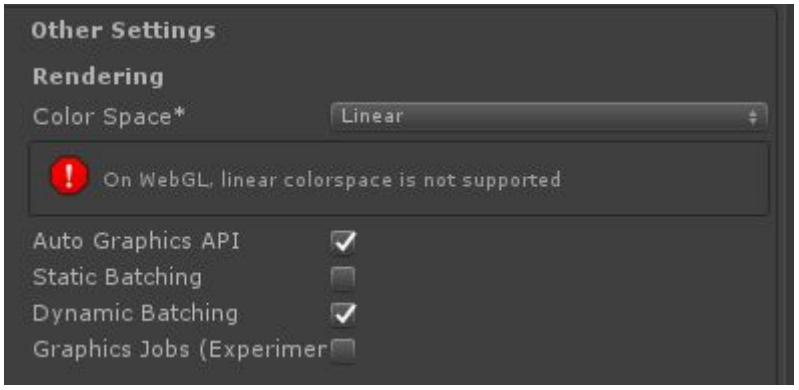
When you are targeting WebGL, you should note a few things:

- Change Color Space to “Gamma”
- Photon Voice is not supported on WebGL

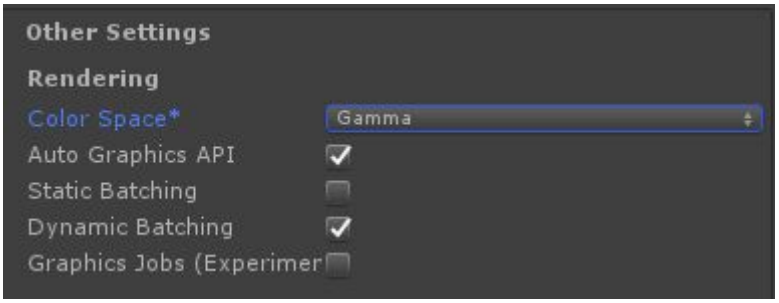
Guide

Color Space

After you have changed your build target to WebGL, you need to change the color space from Linear:



To Gamma:



Photon Voice

The version in the kit has been modified to automatically be disabled in case of WebGL. If you want to update Photon Voice manually, these changes will be lost. There are a lot of #if !UNITY_WEBGL
placed in the Photon Voice scripts. They are at the top of every Photon Voice script, so that it is completely disabled when the platform is set to WebGL.

Compiling

Since WebGL is an AOT platform, you will need to use IL2CPP. For that you need a C++ compiler installed. I recommend installing this one.

Running your build

To run your build, simply start the index.html file. In Chrome, this will not work (locally). Read more about it here and how to get it running on Chrome (locally).

WebGL FAQ

My build crashes with an invalid function pointer error

Most likely your Unity version has a bug with “Strip Engine Code”. Disable that option in the Player Settings to avoid this error. Upgrading to a newer Unity version may also solve this issue.

What is the memory requirement?

At least 512 MB are required for the kit to function in stock state for WebGL.

Special Controls

Leaning

- Lean Left: Q
- Lean Right: E

Voice Chat

- Push To Talk: Left Alt
- Global Voice: L
- Team Voice: K

Integrations

Photon Voice

The Photon Voice integration is already inside the Kit and requires no further work.
A note regarding the AppID: Using the same AppID you are using for the main game will result in doubled CCU. This is due to Photon Voice acting on its own, independently of the main game. E.g. when you join a PUN room, you will also join a Photon Voice room.

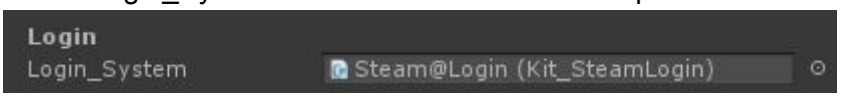
Steam

MMMPFPSK features an in house Steam integration. It will fetch information from Steam such as the username and friends. It will enable users to invite their friends to their respective Photon room.

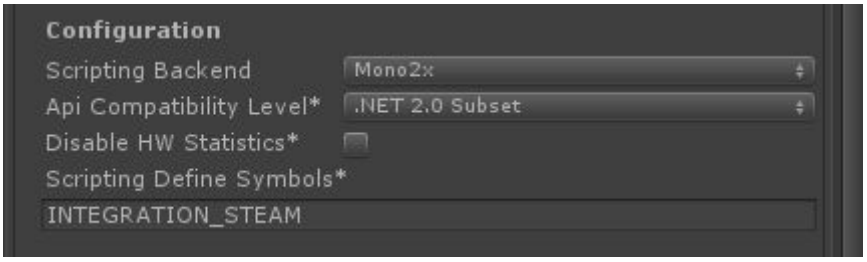
Getting Started

Main

1. Extract the “Steam Integration.zip” file in the “MarsFPSKit/Integrations” folder
2. Restart Unity
3. Drag the “Steam@Manager” in the “MarsFPSKit/Integrations/Steam Integration/Prefabs” (Referred to as “Prefabs folder” in the following steps) folder in the Main Menu scene
4. Do the same with the “Steam@Login” prefab and assign the Kit_SteamLogin script to the “Login_System” field in the Main Menu script



5. Disable the “Login” Game Object in the Main Menu under “Canvas/Main Menu” as it is no longer needed for Steam login
6. Define the scripting symbol “INTEGRATION_STEAM” in the Player Settings:



Friend Menu

The Steam Integration features an in game friends menu that allows you to join all your Steam friends that are currently playing your game. To set it up follow these steps:

1. Assign “Steam@PhotonFriends” to the “PhotonFriends” field in the MainMenu script
2. Disable the “Add Friend” object in “Canvas/Main Menu/Friends/Background Window/Add Friend
3. You are ready to go!

Using your own AppID

If you have acquired an AppID using Steam Direct or using a different way (e.g. Old Greenlight project), all you need to do is to change it in the “steam_appid.txt” file in the project root and in the Steam@Manager prefab in “MarsFPSKit/Integrations/Steam Integration/Prefabs”.

Features

- Use Steam account information (Username)
- Join other players with Steam’s “Join game” function (Does not function with the default “480”/”Spacewar” AppID. Needs a unique AppID to function
- Invite other players to your room
- See all your Steam friends in the kit’s friends menu

Recommended Assets

Animations

It is recommended to replace the third person player animations. These assets will do a great job:
[Rifle Animset Pro]

Rifle Crouch And Prone
Pistol Animset Pro

FAQ Help

My weapons are clipping through geometry?!

There are multiple solutions to this problem.

1. Use a dual camera setup. One camera renders the level geometry and the other one renders the first person weapons. I really do not recommend this for a couple of reasons.
    a. Many image effects will break with this setup
    b. It will decrease performance
    c. Level geometry will not cast shadows on weapons
2. Use First Person View 2. It is a really good asset for only $15 and will find application in all your next first person projects. Get it here.
3. Resize your weapons to make them fit inside the Character Collider. It will mean that you will have to adjust walking and running animations though.

The Loadout Dropdown Buttons are rotated.

This is most likely caused by rotating the “MarsFPSKit_IngamePrefab” object in the scene. Its rotation and position shall remain at 0,0,0. To move/rotate the scene camera use the “Spawn Camera Position” object.

How do I scale, rotate or move a weapon prefab (first, third-person or drop prefab)?

You should always create an empty game object and use that as the prefab itself. This should hold the scripts and have its position and rotation at 0,0,0 and scale at 1,1,1. The model itself (the one which has the animator on it in case of the first person prefab) should be directly under it in the hierarchy.

Need help?

Do you have a problem that you cannot solve using this document? You can seek help on Armedunity, we have a specific team for this kit and a big community that can help you with non kit related problems (in Unity!). If you need immediate help or you just want to join our community, join our Discord here!