Modular Multiplayer FPS Engine (Mirror)

Version 0.8.1.0 BETA

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 implement the functions.

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 and it is also persistent, that means that data that is changed at runtime is saved when you exit play mode.
To solve this issue, a normal C# 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 2022.3.57f1 but will also work in other (later) versions.
Before you can start using & modifying the kit, you will need to set the project up for the first time.
Firstly, import the Mirror package from the asset store.
Then, you need to connect the project with a Unity Project ID.
You can find this in **Edit/Project Settings/Services.** You can either create a new Project ID here, or link it to an existing project.
After you've done so, open the Unity Dashboard. Make sure you're logged in with the correct account.
In the dashboard, select your project and activate the Lobby, Player Authentication and Relay services. Once those are activated, your project should be ready and work online!
If you are having issues, don't hesitate to contact us.

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 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.

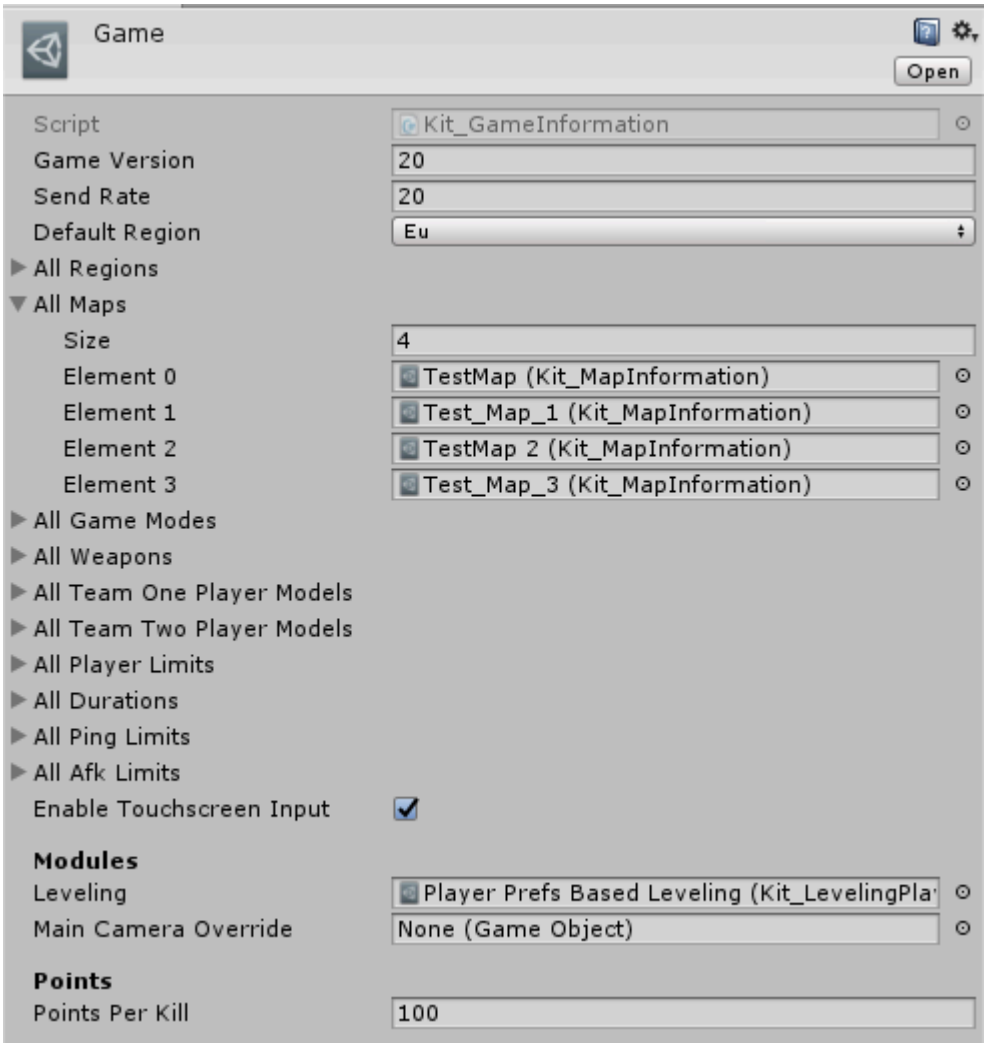
Mobile Controls

MMFPSE features built-in Mobile controls. In a few steps, you can have it converted to mobile.

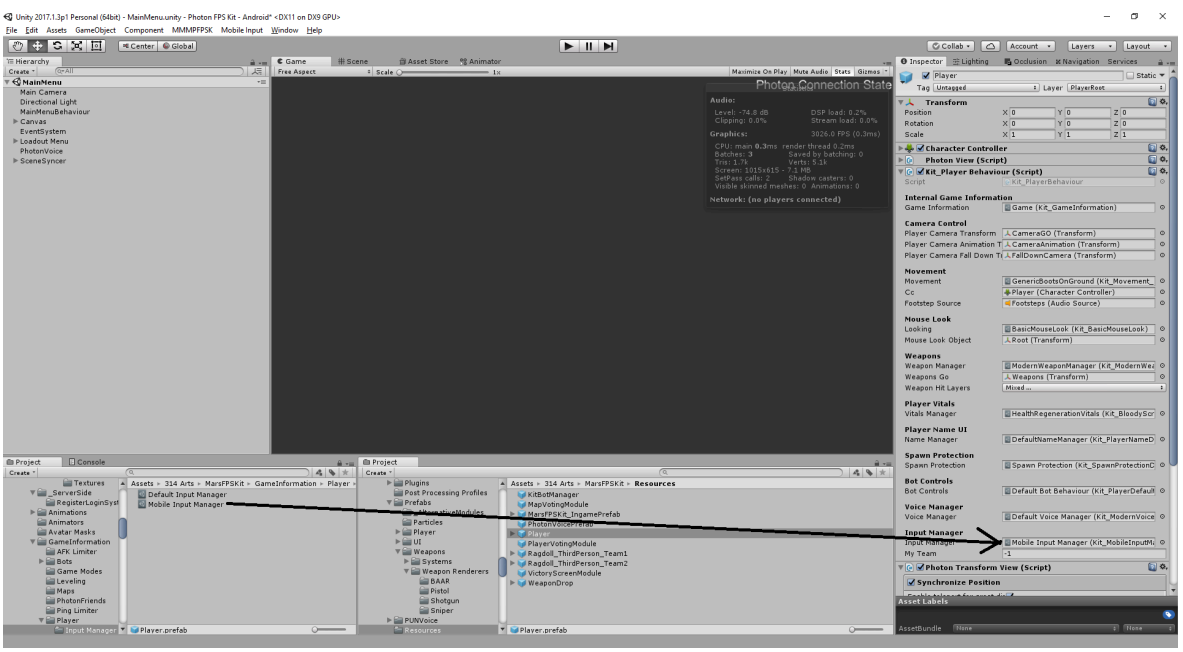
NOTE: Maps, Weapons, Character models are <u>not</u> optimized for mobile use.

First, change the Platform to Android / iOS (depending on what you want to build for).

Then, we need to enable touchscreen input, in the “Game” file, which can be found in “314 Arts\MarsFPSKit\GameData”, just check “Enable Touchscreen Input”:



Next, we need to set up the mobile controls.To do that, we need to assign the “Mobile Input Manager”, which can be found in “314 Arts\MarsFPSKit\GameData\Player\Input Manager” to the Player prefab, which can be found in “\314 Arts\MarsFPSKit\Resources”, like that:



The Kit will take care of the rest, once that is done.

Tips

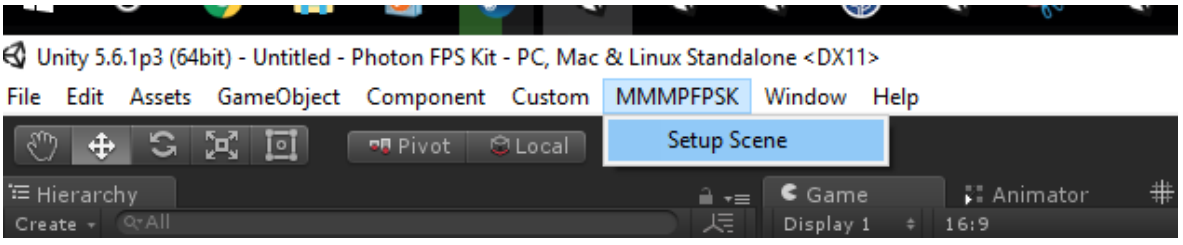
Performance

For performance reasons, you should lightmap (bake, not realtime) all your scenes. In addition, you should turn off any image effects and try to have your weapons in one part and keep batches in your level < 100.
Another important thing is, your renderer should be set to “Forward”. You can do that by modifying the camera in the MarsFPSKit_IngamePrefab prefab.

Using the kit

Setting up a new scene

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



This will save the scene, add the MarsFPSKit_IngamePrefab prefab and add it to the build settings. Afterwards, you will need to add the map to the Game Modes (default: Deathmatch, Team Deathmatch, Gun Game, Domination), to lobby and/or traditional (you do not need to add it to all game modes, that's what the system is in place for. 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 using a model with less detail than the first person one.
- Depending on how much detail you want, you can use LOD on the Third-Person model.

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

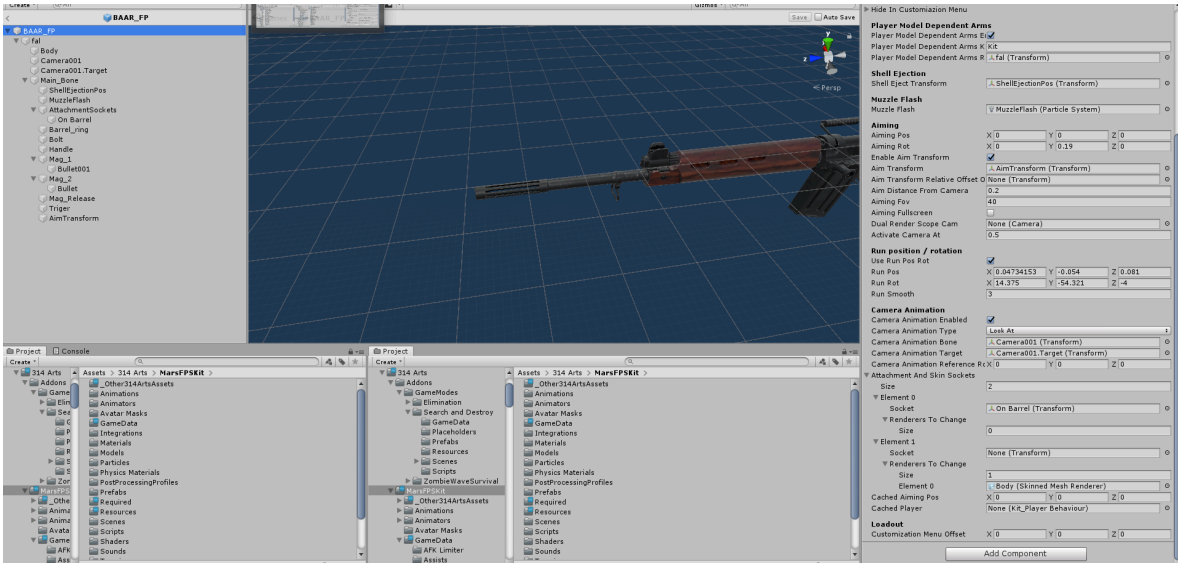
Do the same with the Third Person Prefab and the Drop prefab. If you already want to set up attachments, make sure that all three prefabs line up (The weapon information will tell you if that is not the case).
Once the prefabs are done, go into the Weapons folder (By default: "MarsFPSKit/GameData/Player/Weapons") and select the appropriate category (You can add / remove / modify however you like, it is just for organizing it). Either create a new Kit_ModernWeaponScript or duplicate one. I strongly recommend to duplicate one because it is easier. Assign the Prefabs and assign your sounds on the **Behaviour**. Once you are done with that, you can add the Behaviour to the GameData (By default: "MarsFPSKit/Resources/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 attachment system consists of two parts:

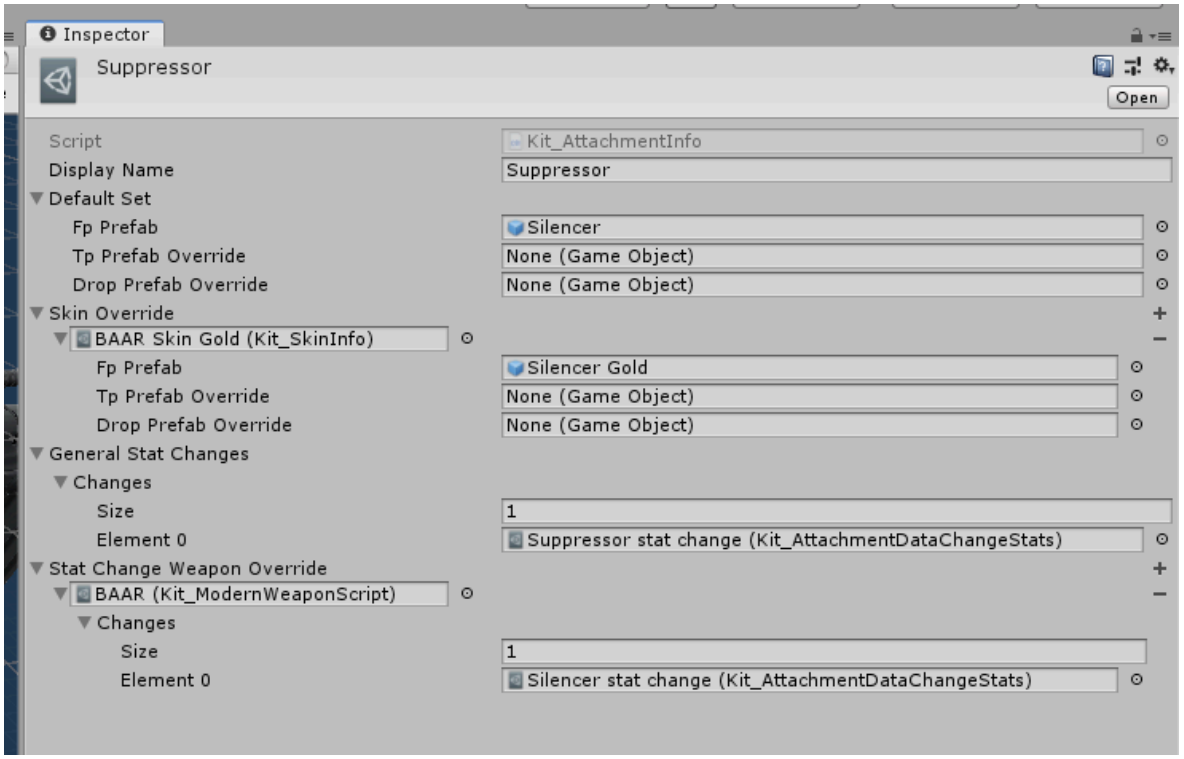
- Sockets / Renderers (for Skins) on the weapon FP, TP and DROP prefab
- Attachment / Skin info files assigned in the weapon's scriptable object

Sockets:
The socket defines where an attachment will be spawned in, for example on the BAAR prefabs you will find a socket called "On Barrel". This is where the Suppressor will be spawned in:



Renderers (for skins):
To apply skins to a gun, we have to change the materials on the weapon's (skinned) mesh renderers. Every renderer you assign here will have its materials changed to the ones set in the SkinInfo file.

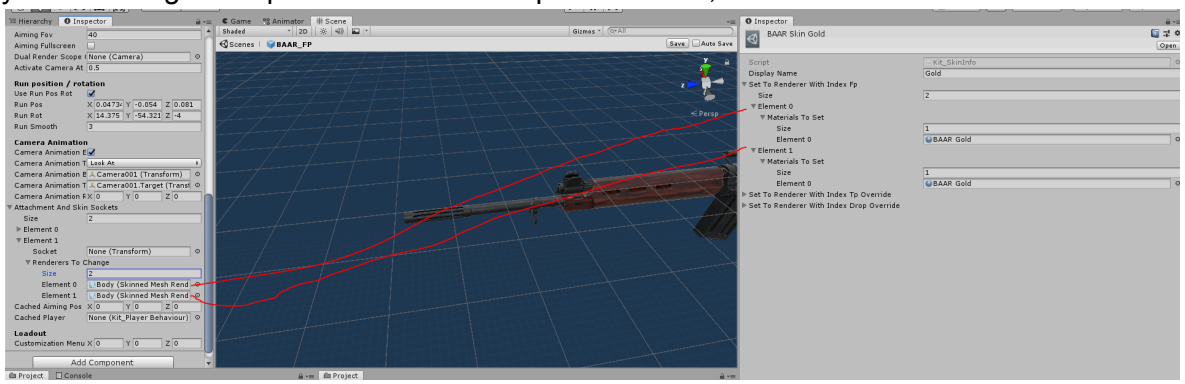
Attachment Info file:
This file contains all information on what prefabs an attachment should use and how it should change the weapon's statistics (if at all, attachments do not need to change them, so you can also leave this empty!)



The Fp Prefab is required, you can however use different prefabs for Third Person (Tp Prefab Override) and Drops (Drop Prefab override).

You can also have an applied skin change the look of the attachment too.
It is easy to set up: Add a new entry and assign your skin's info file on the left and assign the new prefabs you want to use if that skin is in use below it. You can add as many as you like. Do note that the complete set is used, so if by default you have a Tp Override assigned and the skin does not have one, the default Tp override will NOT be used.

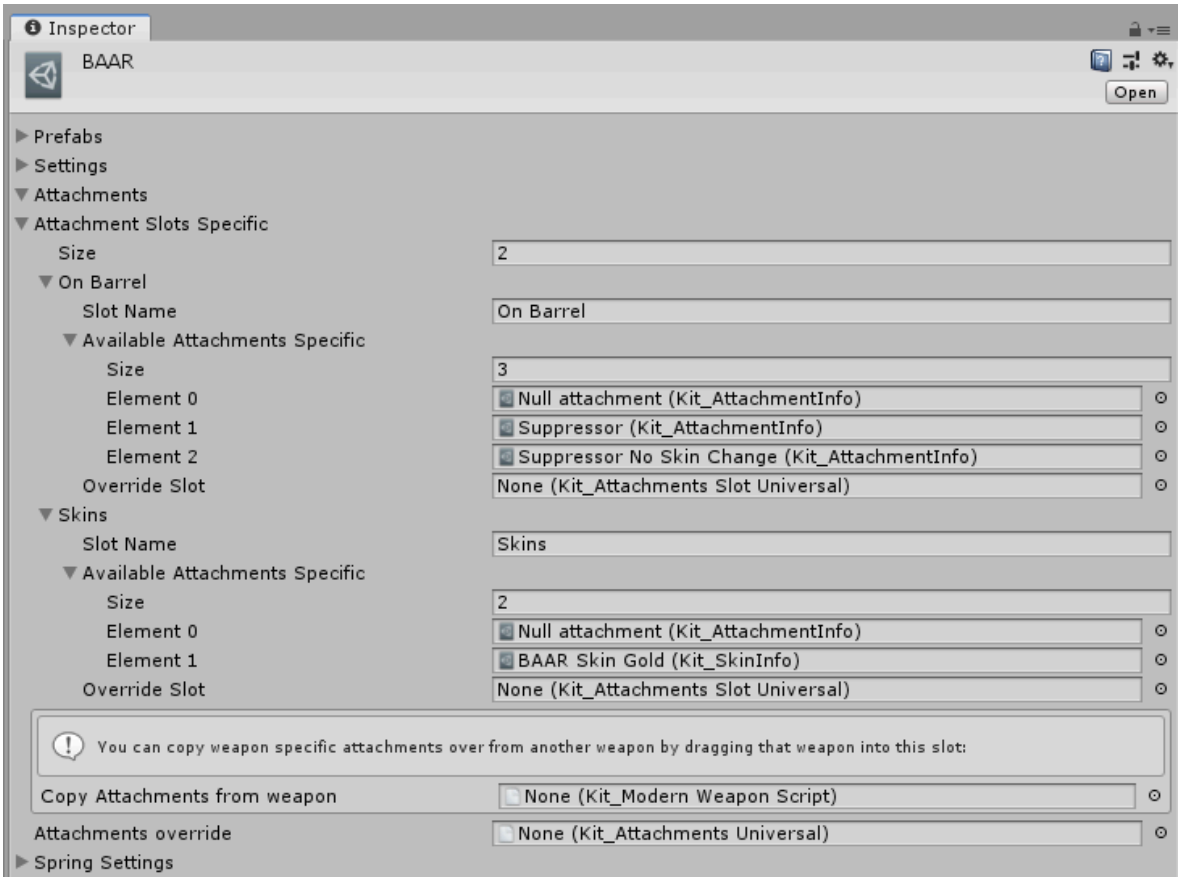
Skin Info file:
The skin info file basically just contains which materials to change. It is a nested array so you can assign multiple materials to multiple renderers, like this:



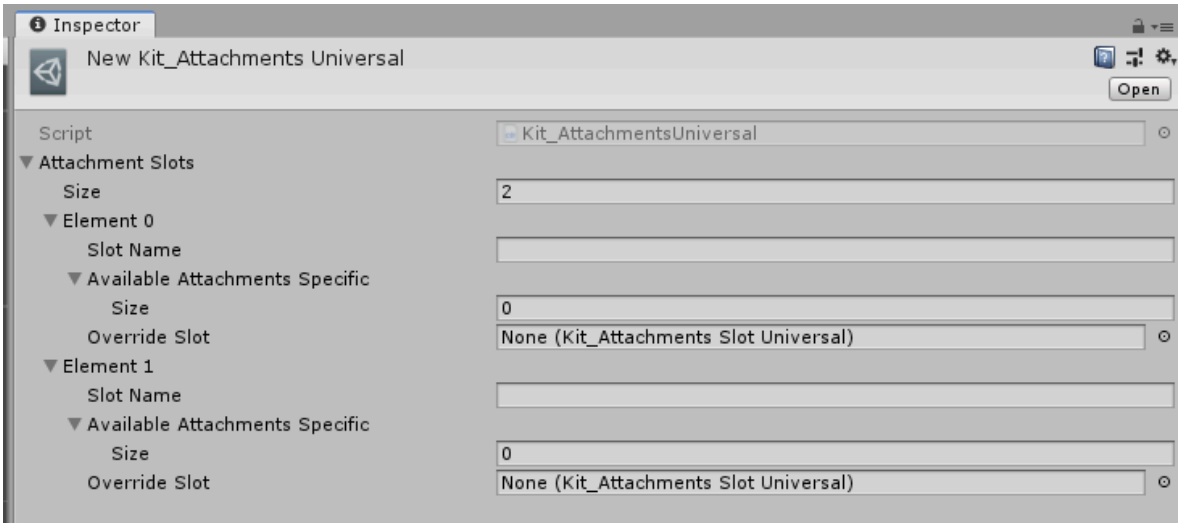
Do note that the Element in the left inspector is defined by the slot the skin sits in inside the weapon file.

Assigning Attachments:
Attachments live in a slot, so you would have a slot for e.g. On Barrel, Side Rail, Scopes,

Skins, etc. They get spawned into each socket / apply to the renderers as stated above.



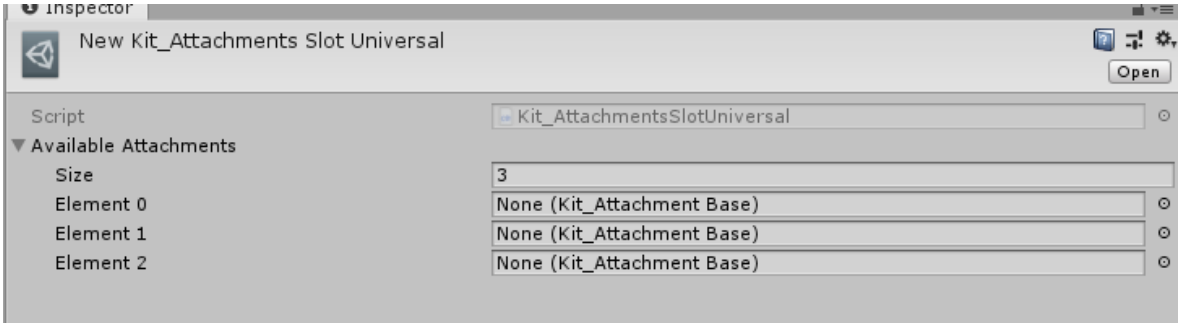
You can share the complete list between weapons by using an Kit_AttachmentsUniversal file (create one by right clicking in your project then selecting “MarsFPSKit/Weapons/Attachments/Universal Attachment List”) , which has the exact same properties as above:



You assign this file into the “Attachments override” property in the weapon file as shown in the picture above this one.

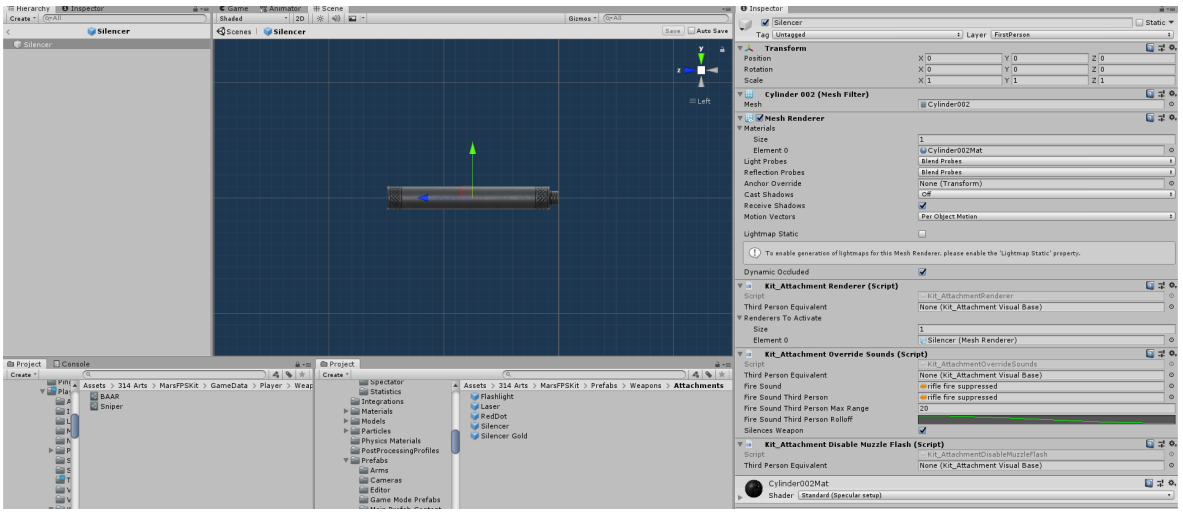
You can also just share individual slots by using an Kit_AttachmentsSlotUniversal file (create one by right clicking in your project then selecting

“MarsFPSKit/Weapons/Attachments/Universal Attachment One Slot List”):

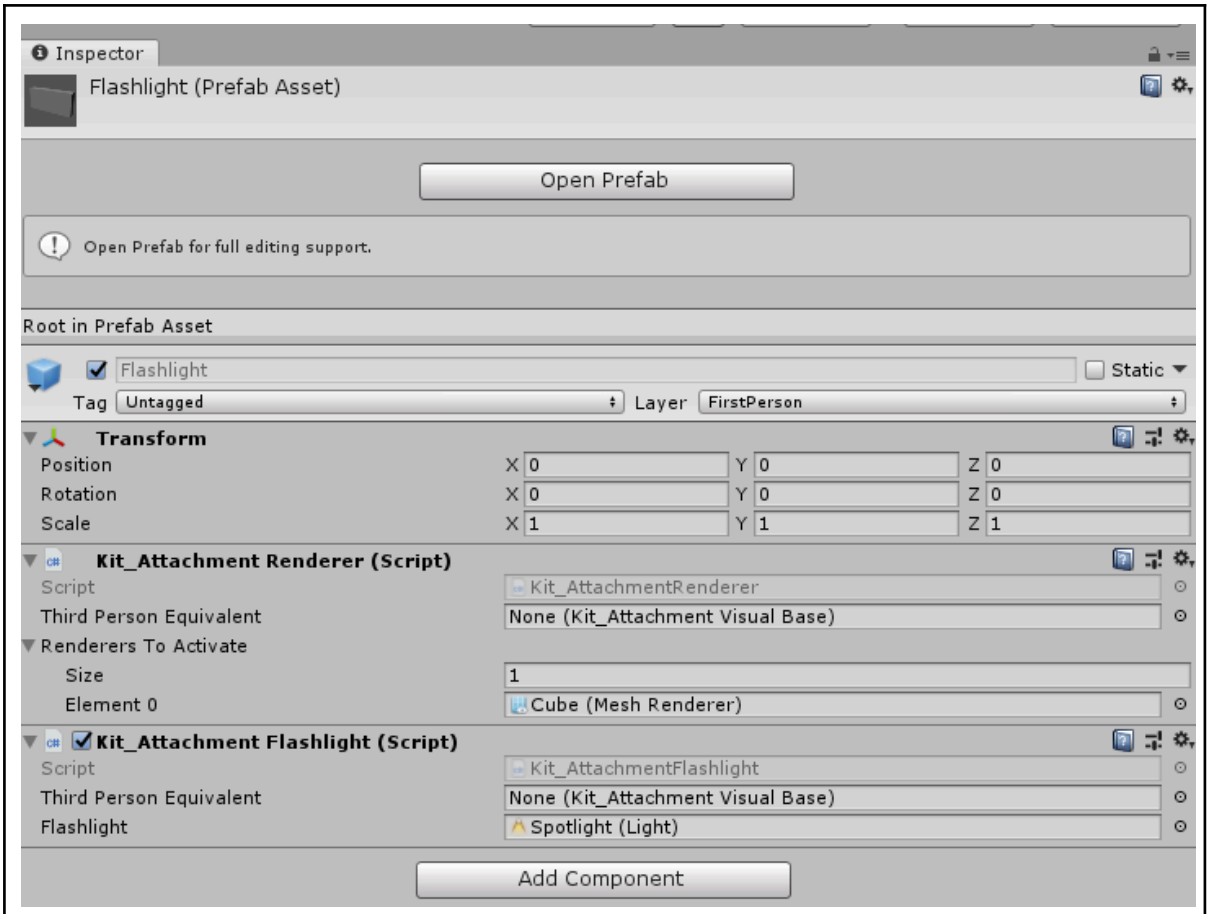


You assign this file to the “Override Slot” property in the weapon / Universal file.

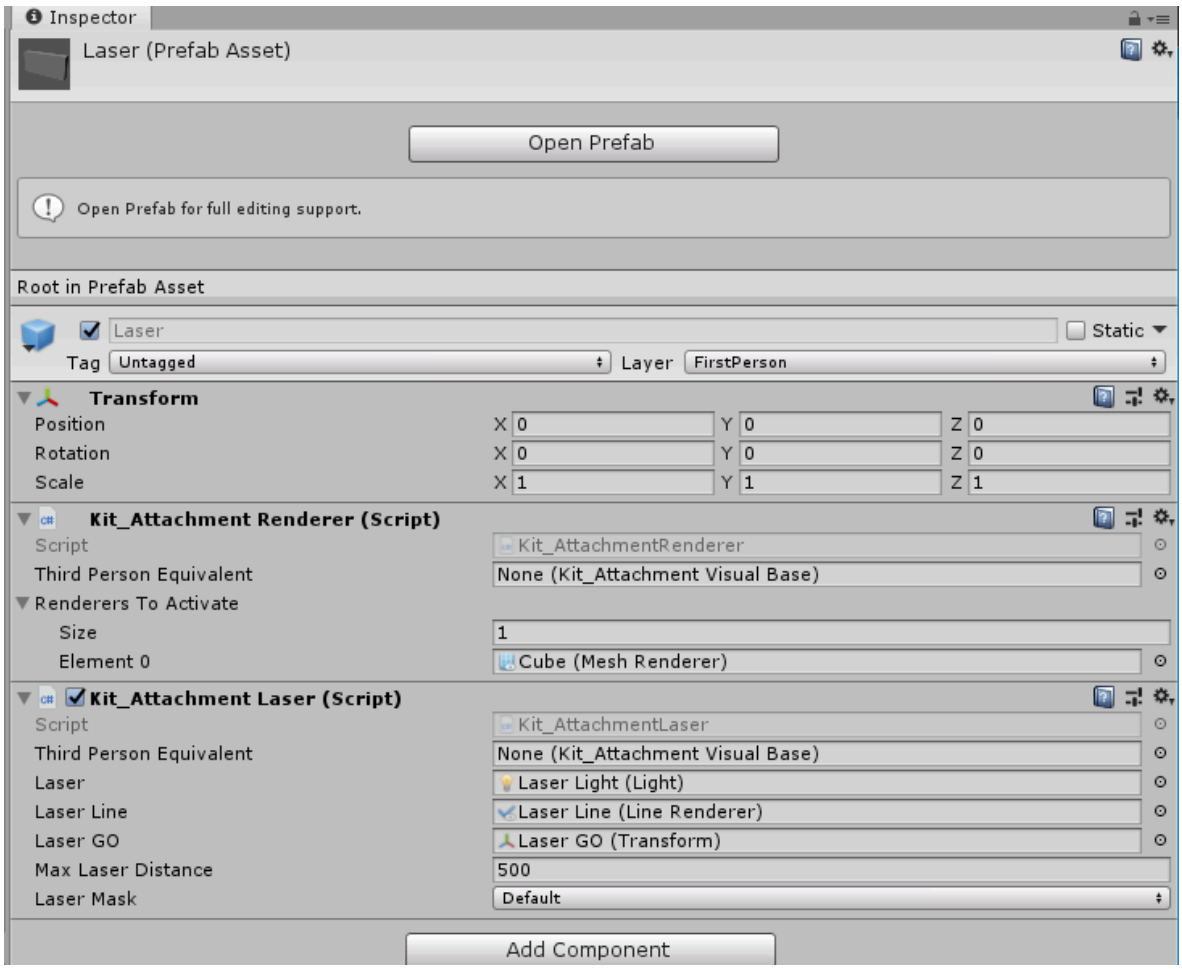
Attachment prefab file:
The prefabs get spawned into the sockets as described above.
You can put various scripts on it to describe the attachments behaviour, for example the suppressor:



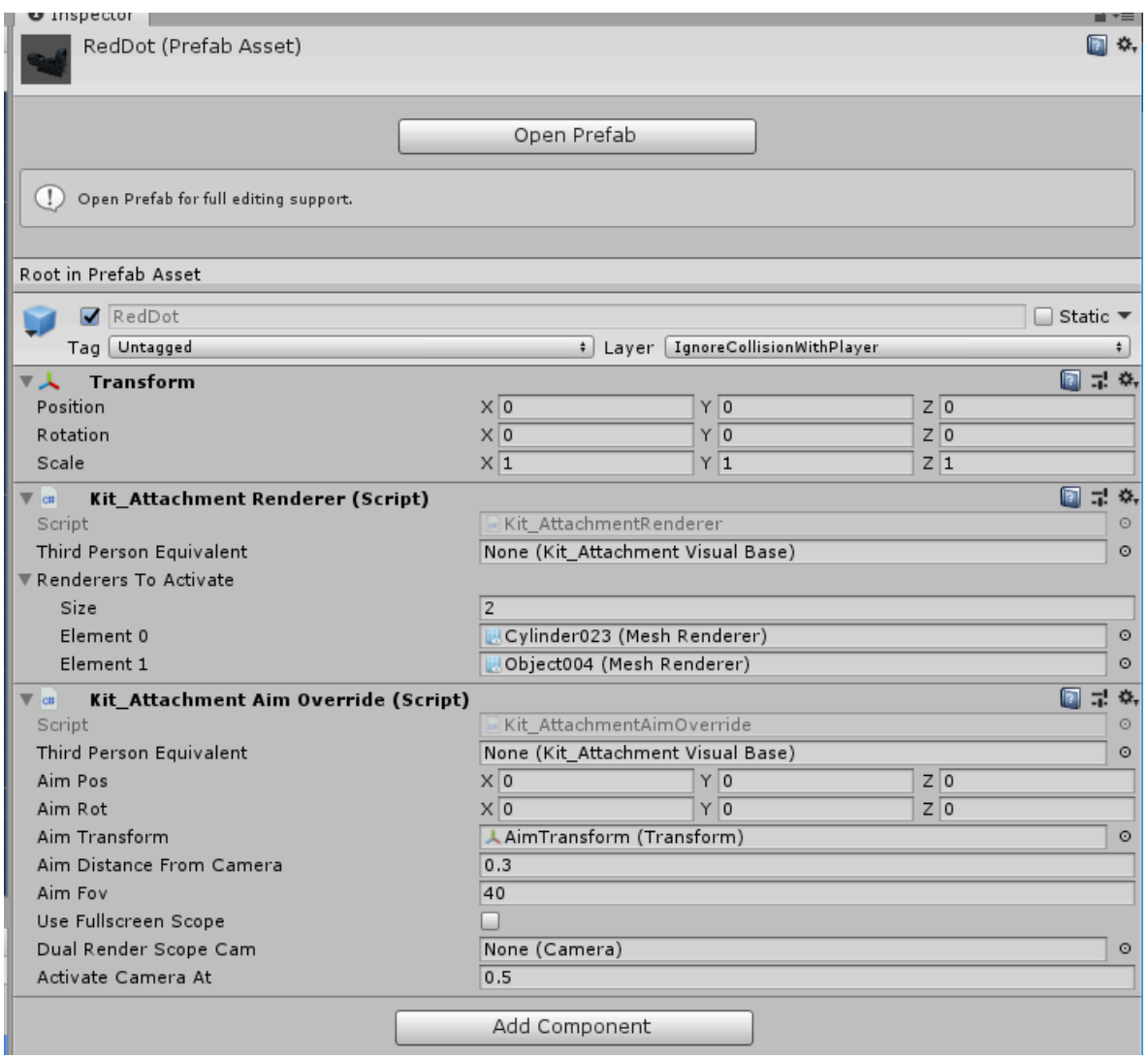
Flashlight example:



Laser example:



Scope example:



Note that the “Aim Transform” property only gets used if the weapon prefab has it enabled.

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. There should be at least one game mode assigned, otherwise the spawn will be useless. If you want to use the spawn for all game modes, you will need to assign all game modes to them (that is how the spawns in the example scenes are setup).
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 your game mode does not use Spawn Group IDs, just leave it set to 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.

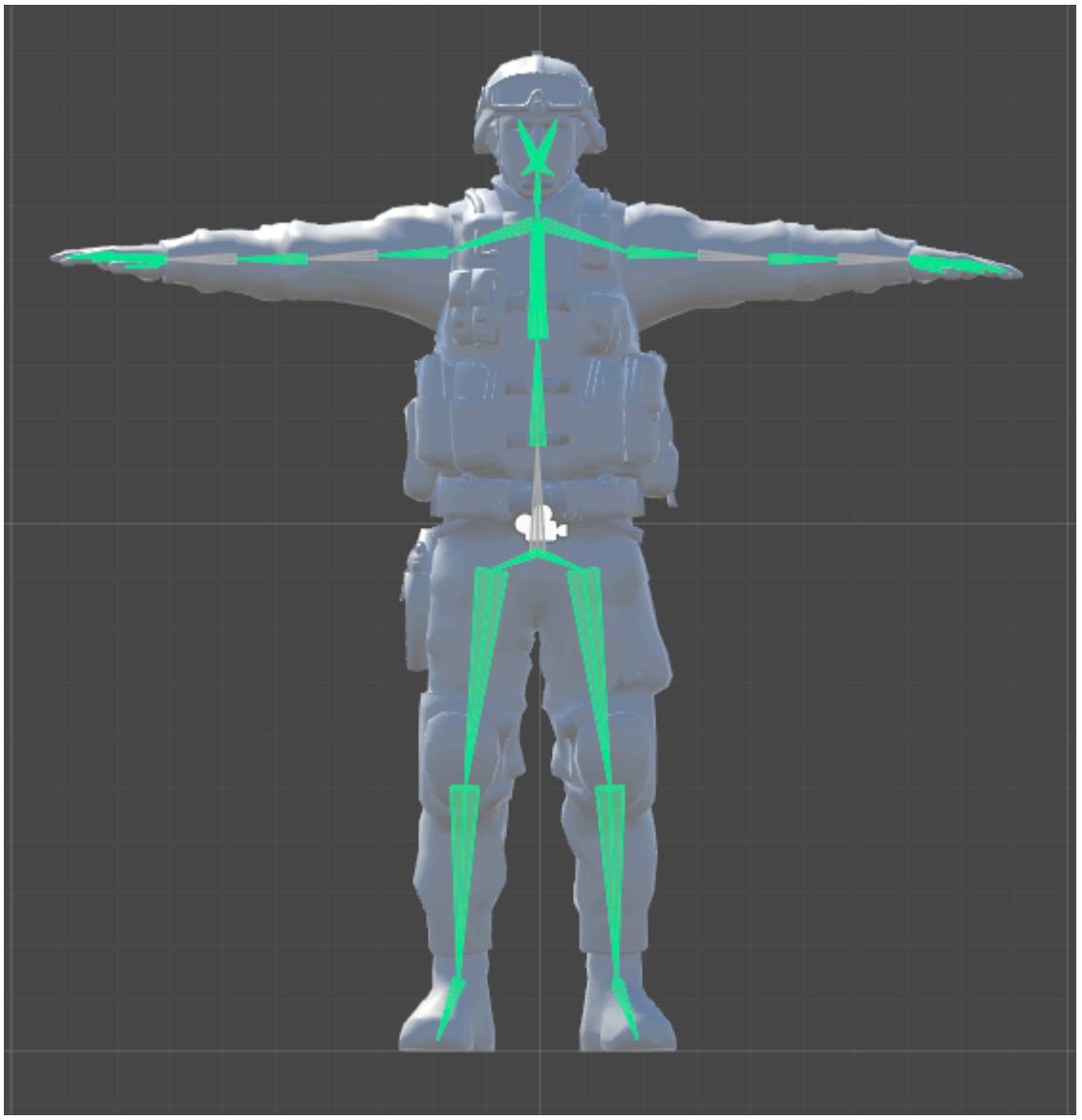
Setting up a new player model

Prequisites:

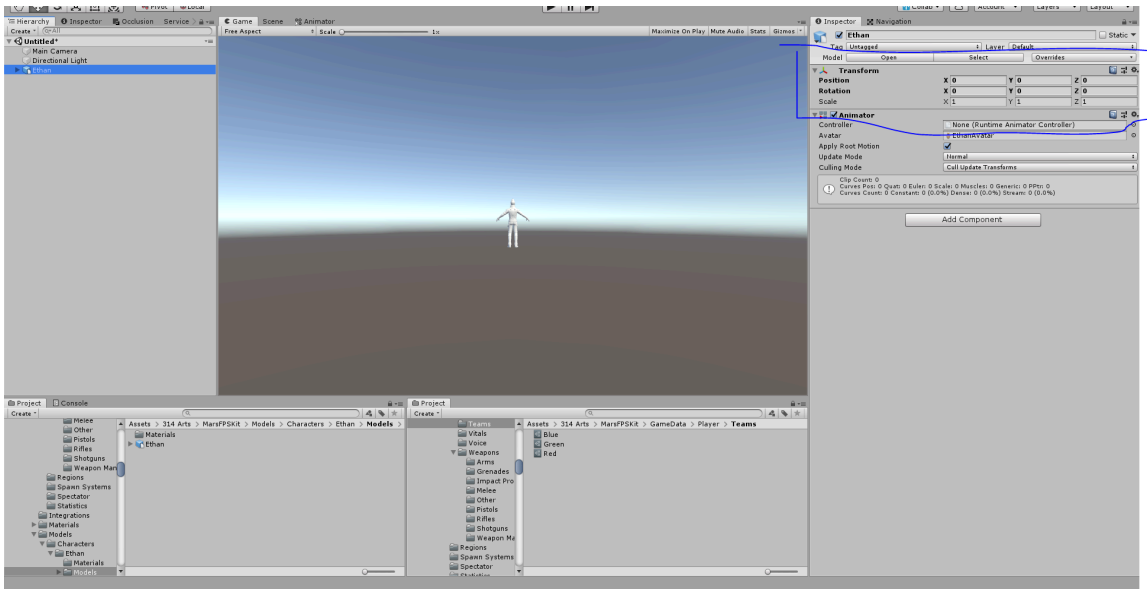
- A rigged, bipedal 3d model

Preparing the model

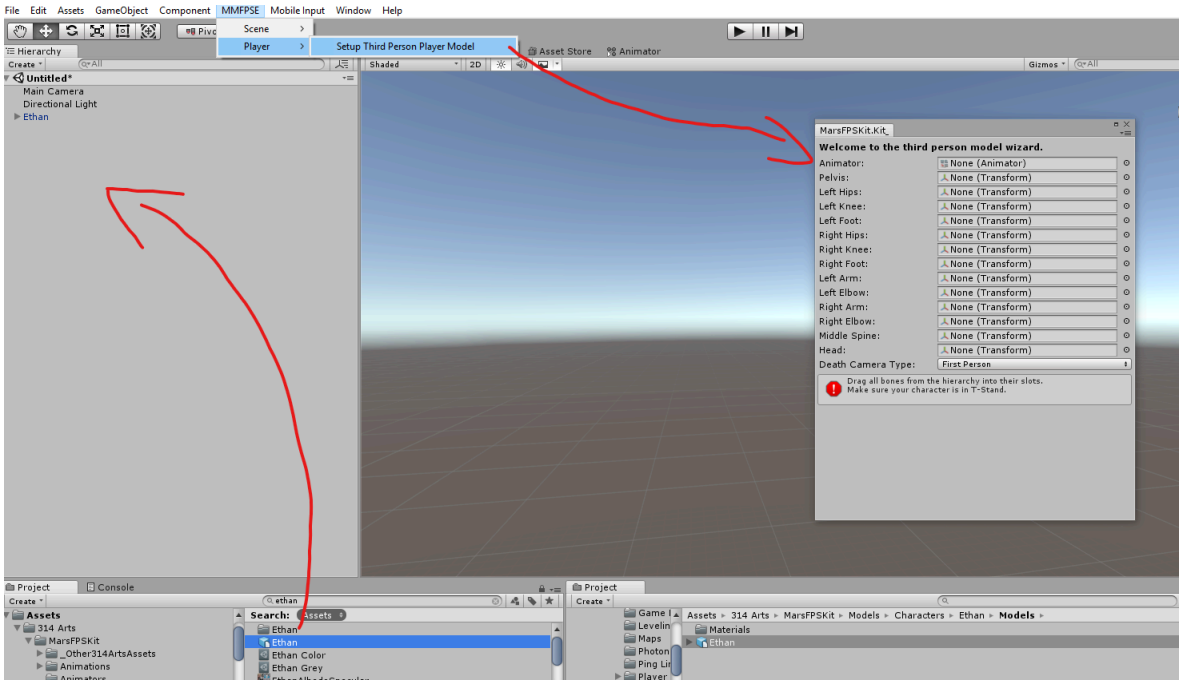
1. Set its animation type to “Humanoid”, so Mecanim can retarget the animations
2. Configure the avatar to look like this:



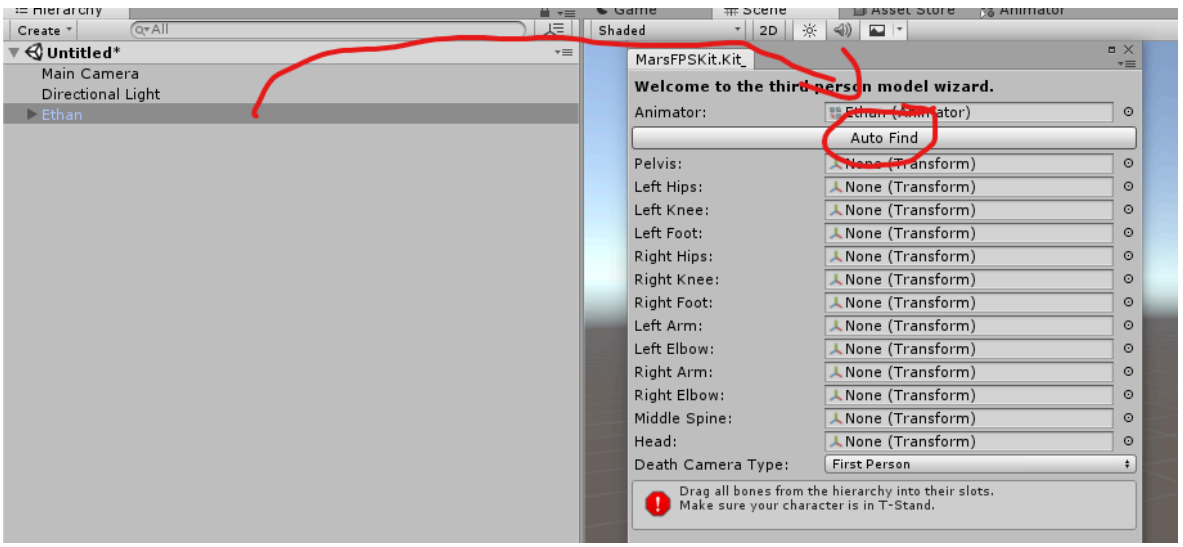
As of 0.5.6.0 the remaining part is extremely easy and fast!
First, drag your model into the scene and Position it at 0,0,0 and rotate it at 0,0,0 too, it should look like this:



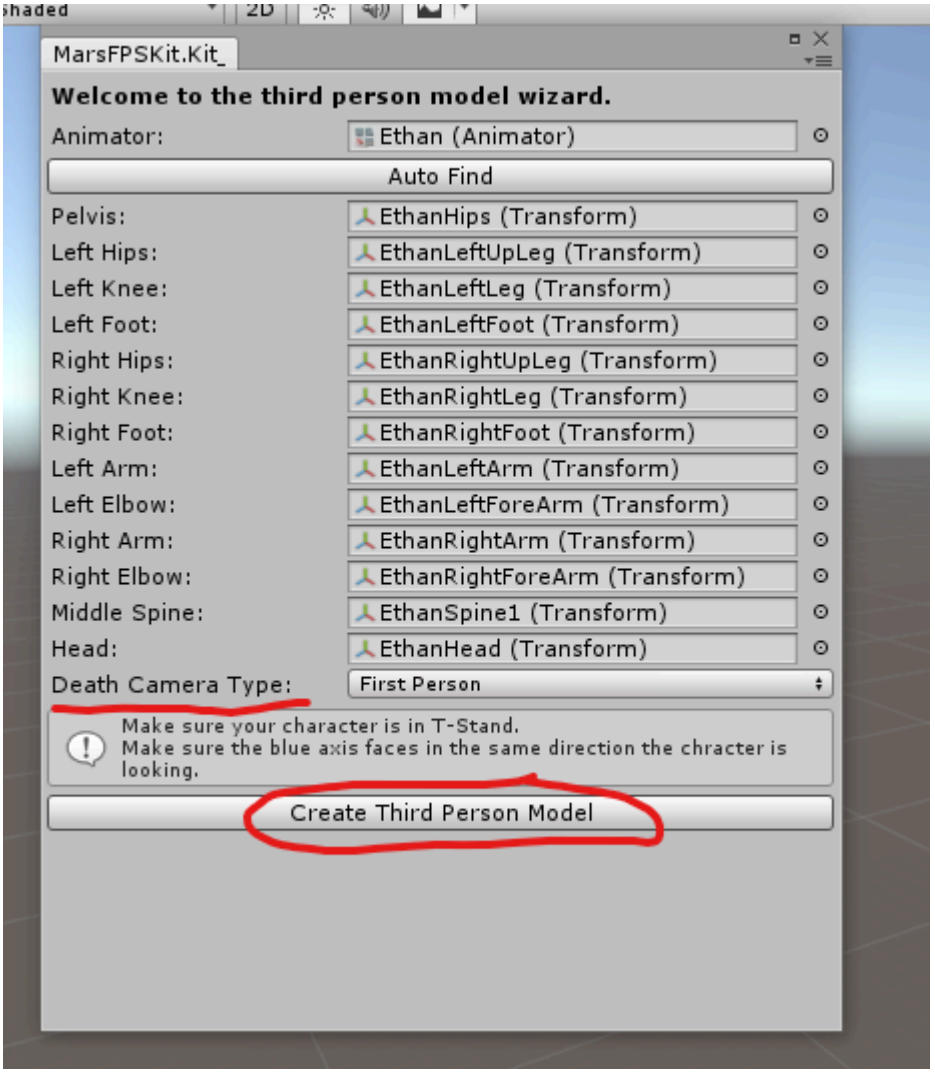
Now open the “Setup Third Person Player Model” tool which can be found under “MMFPSE/Player”:



Then, drag your model into the “Animator” slot (it should be the root object) and press “Auto Find”. Every transform should be automatically assigned. In case they are not found (maybe because your bones are oddly named), you will need to assign them manually to the appropriate slot.

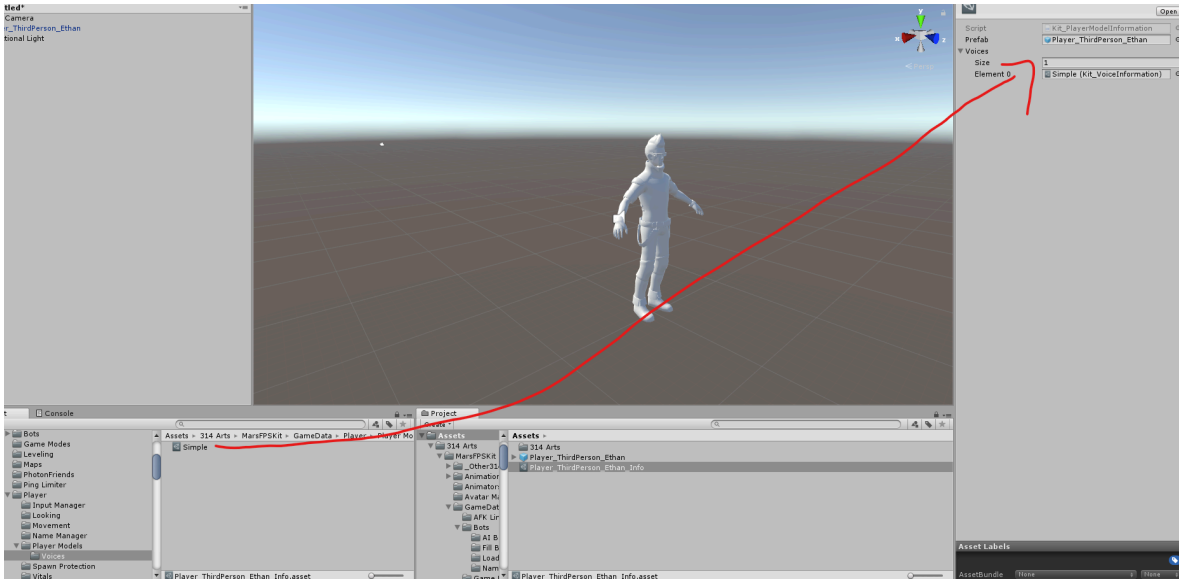


Then, select the Death Camera type you want to use and press the “Create Third Person Model” button:

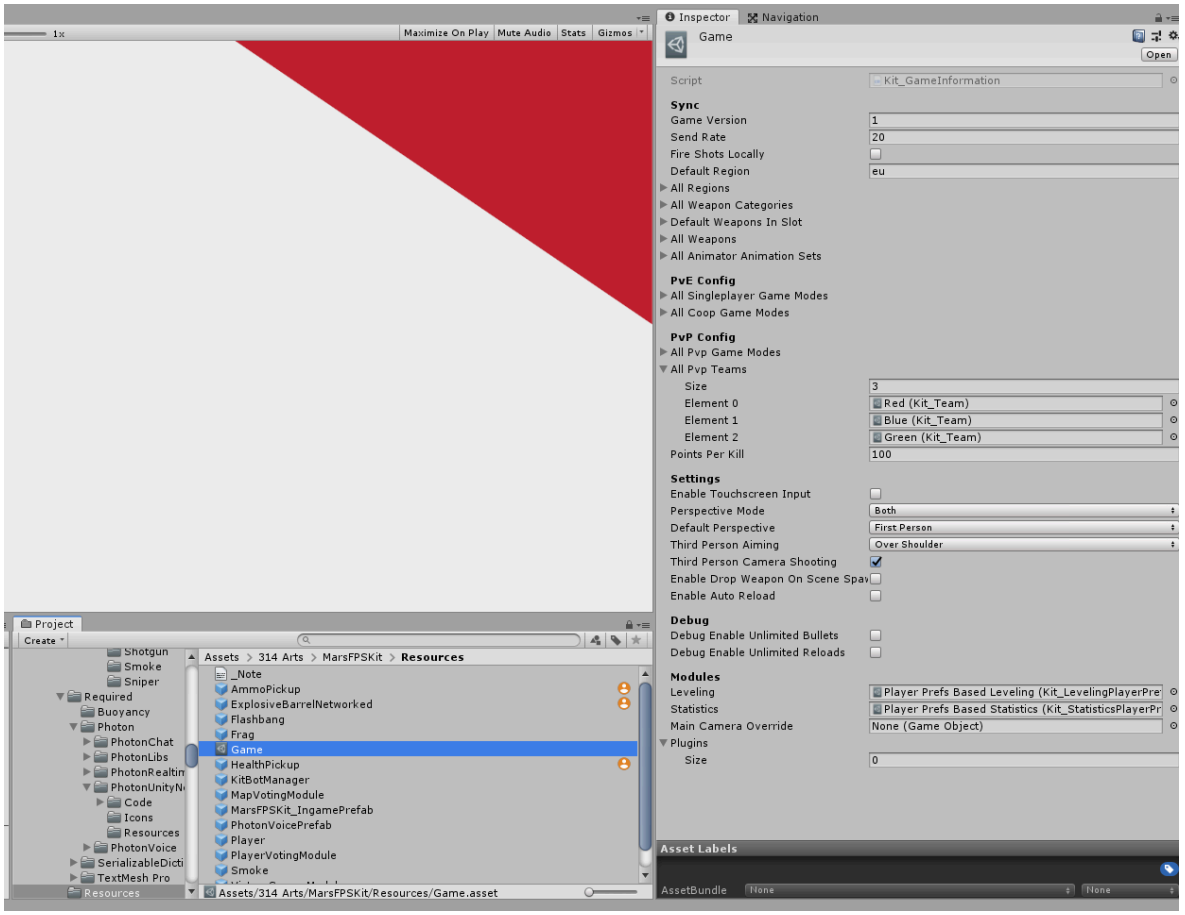


Then you will be asked where the model and the information file should be saved. Save

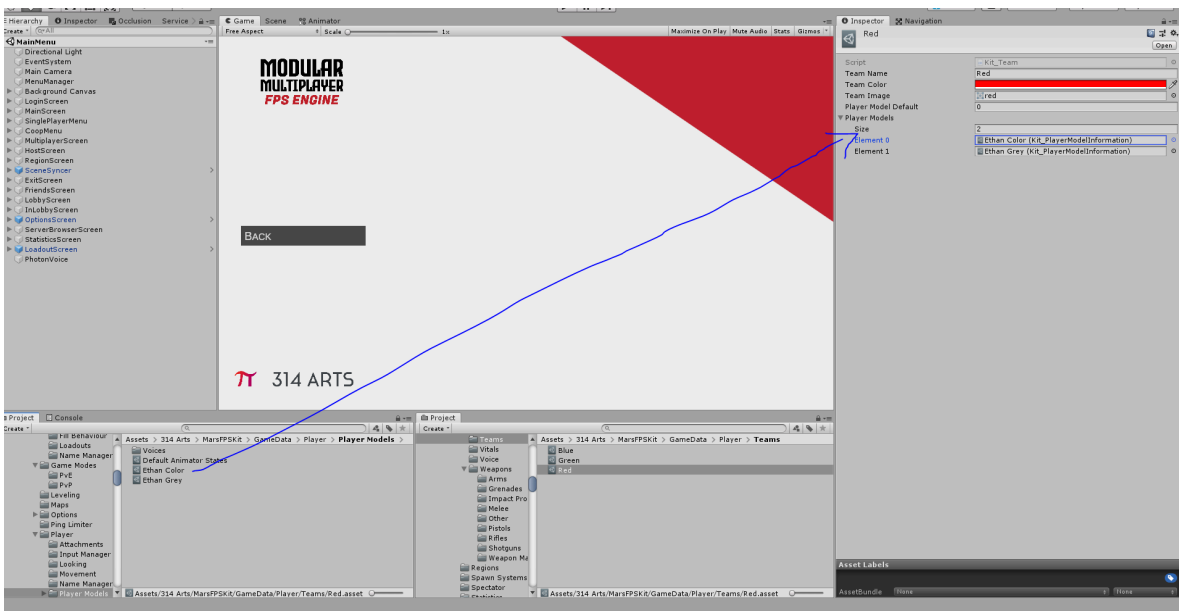
them anywhere you want (inside your project). Afterwards we need to assign at least one voice to our information file that was just created.



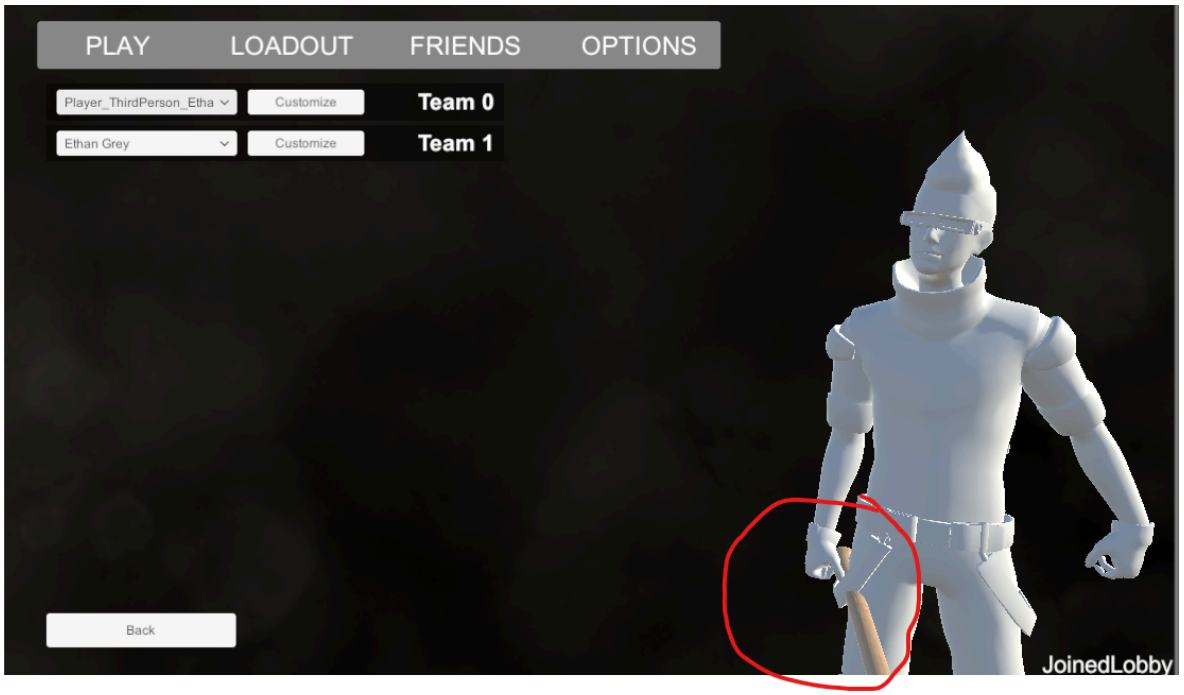
Almost done! Now we need to assign our information file to the “Game” file, which can be found under “314 Arts/MarsFPSKit/Resources”:



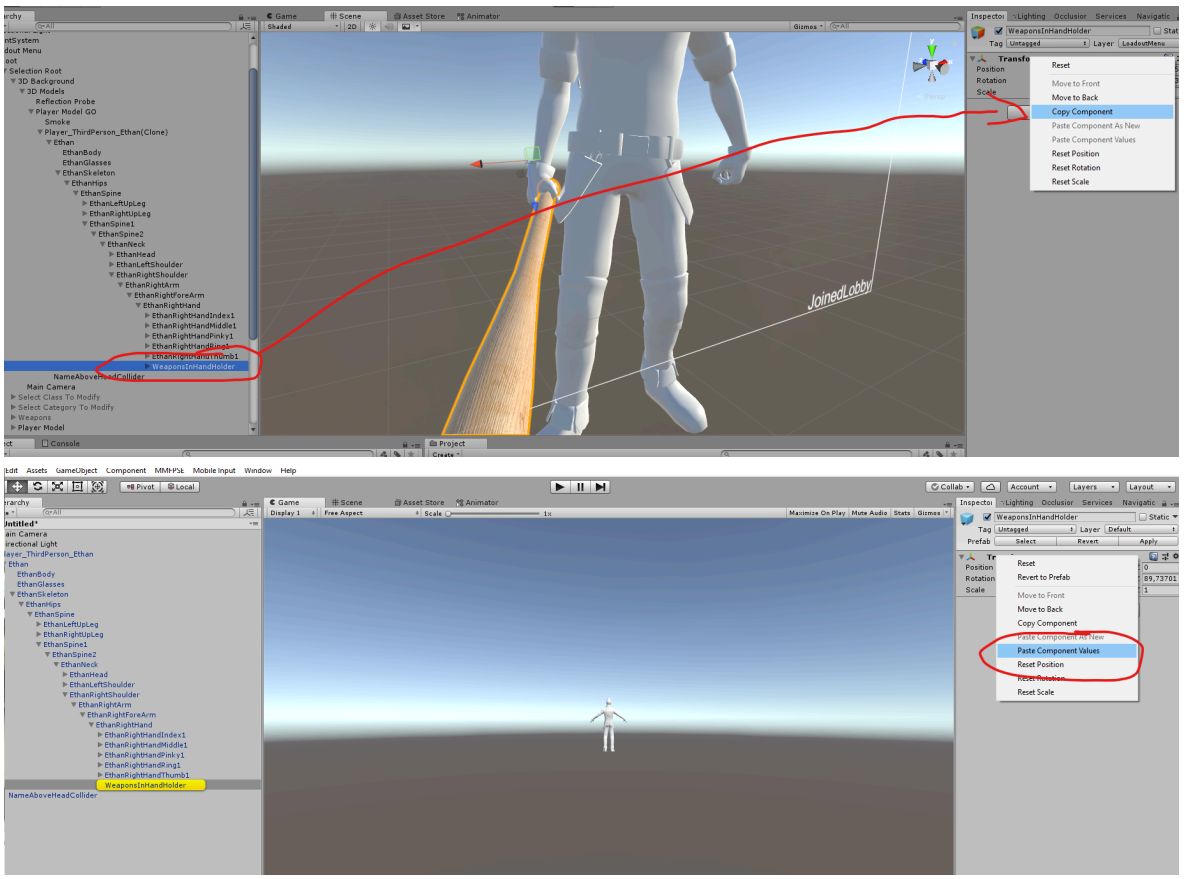
Here you can find the “All Pvp Teams” property. Inside those files (“Red”, “Blue”, “Green”), you can assign the player model:



Done! At least technically. Now your model can be selected in the loadout menu, however, as you can see, it's not configured 100% perfectly:



This can be easily fixed, by positioning and rotating the "WeaponsInHandHolder" like this and applying your changes to the prefab, by doing a right click on the "Transform" component, pressing copy component and then pasting these values inside the prefab:



Done!
Of course, everything can be adjusted manually to your liking.

Modifying the UI

- Main Menu:
  The main menu is split in four parts:
  - Main menu
  - Login 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)
  - Options Menu
    - Is already present in the main menu as a disconnected instance
  - 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):

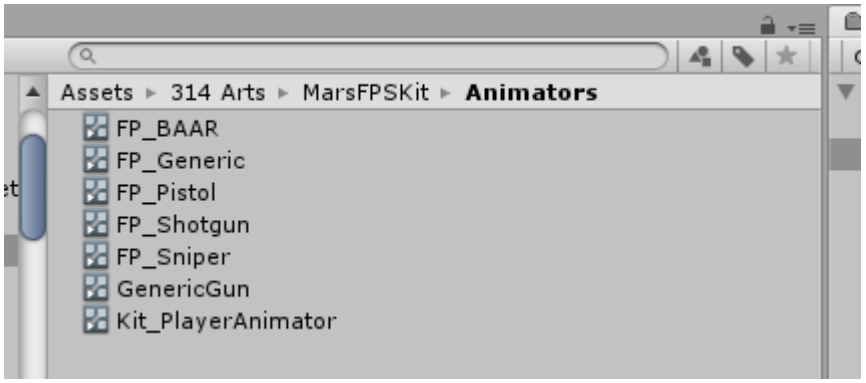
- Pause Menu
- Options Menu (Is a completely separate MonoBehaviour)
- HUD
- Kill Feed
- Chat
- Kill UI (points that pop up)
- Victory Screen
- End game Map Voting
- Ping Limit
- Afk Limit
- Voting menu
- Loadout Menu (Is also a completely separate MonoBehaviour)

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)

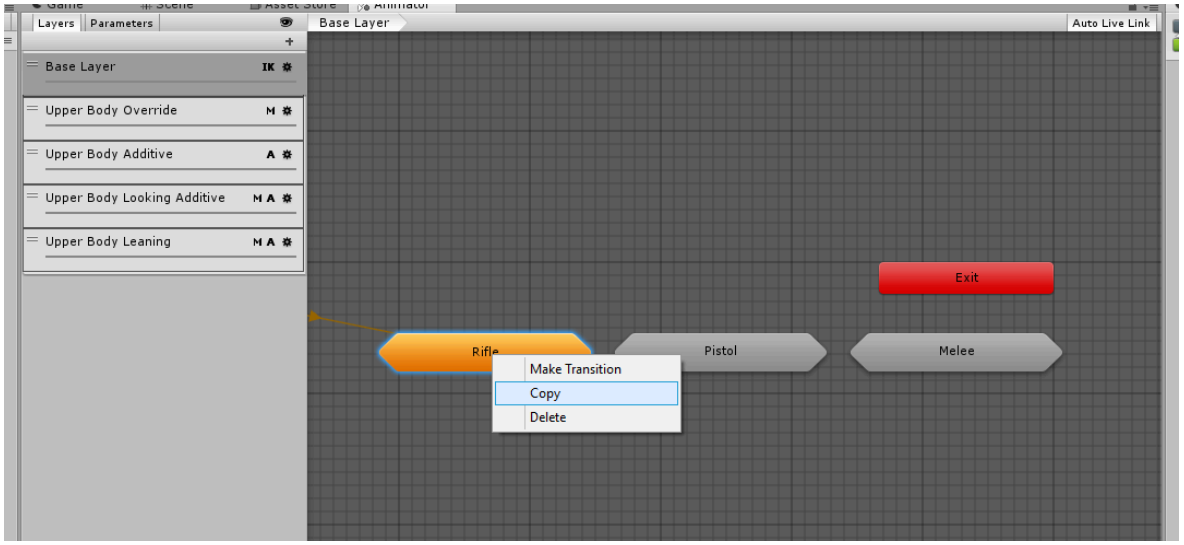
- Everything in the folder "Game Mode HUDs"
- ChatEntryPrefab
  - Used to display a chat message inside the chat UI
- KillFeedEntryPrefab
  - Used to display a "Who killed whom" message
- MapVotingEntryPrefab
  - Used for the voting at the end of the round
- PlayerMarkerRootPrefab
  - This is the arrow / 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

Adding a new third person animation set

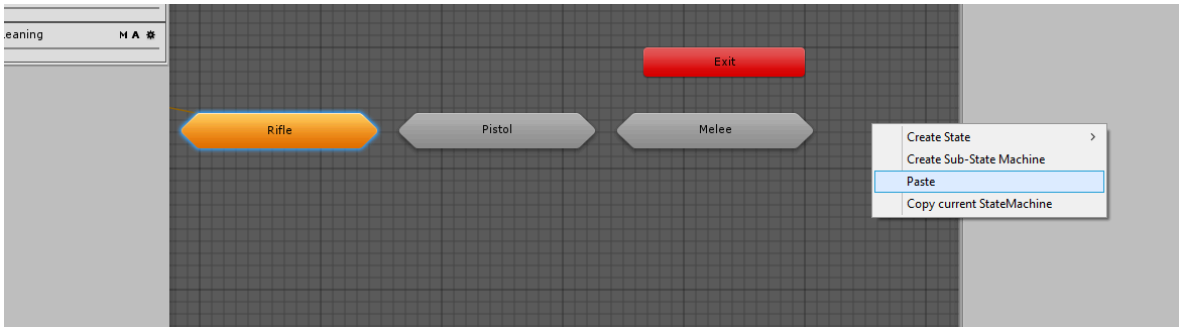
As of 0.5.2.0, adding a new third person animation set is EXTREMELY easy!
First, open the player animator "Kit_PlayerAnimator", which can be found in "314 Arts/MarsFPSKit/Animators".

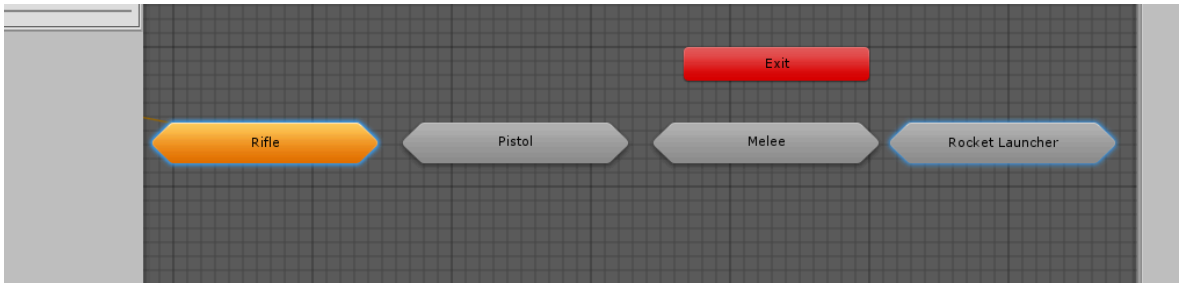


Secondly, right click on “Rifle” and hit copy.



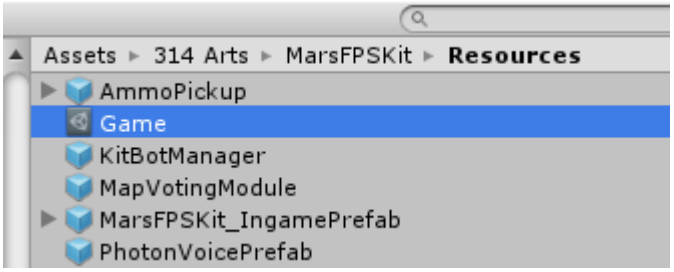
Thirdly, right click on an empty space and hit paste. Rename that pasted Statemachine to however you want to name it.



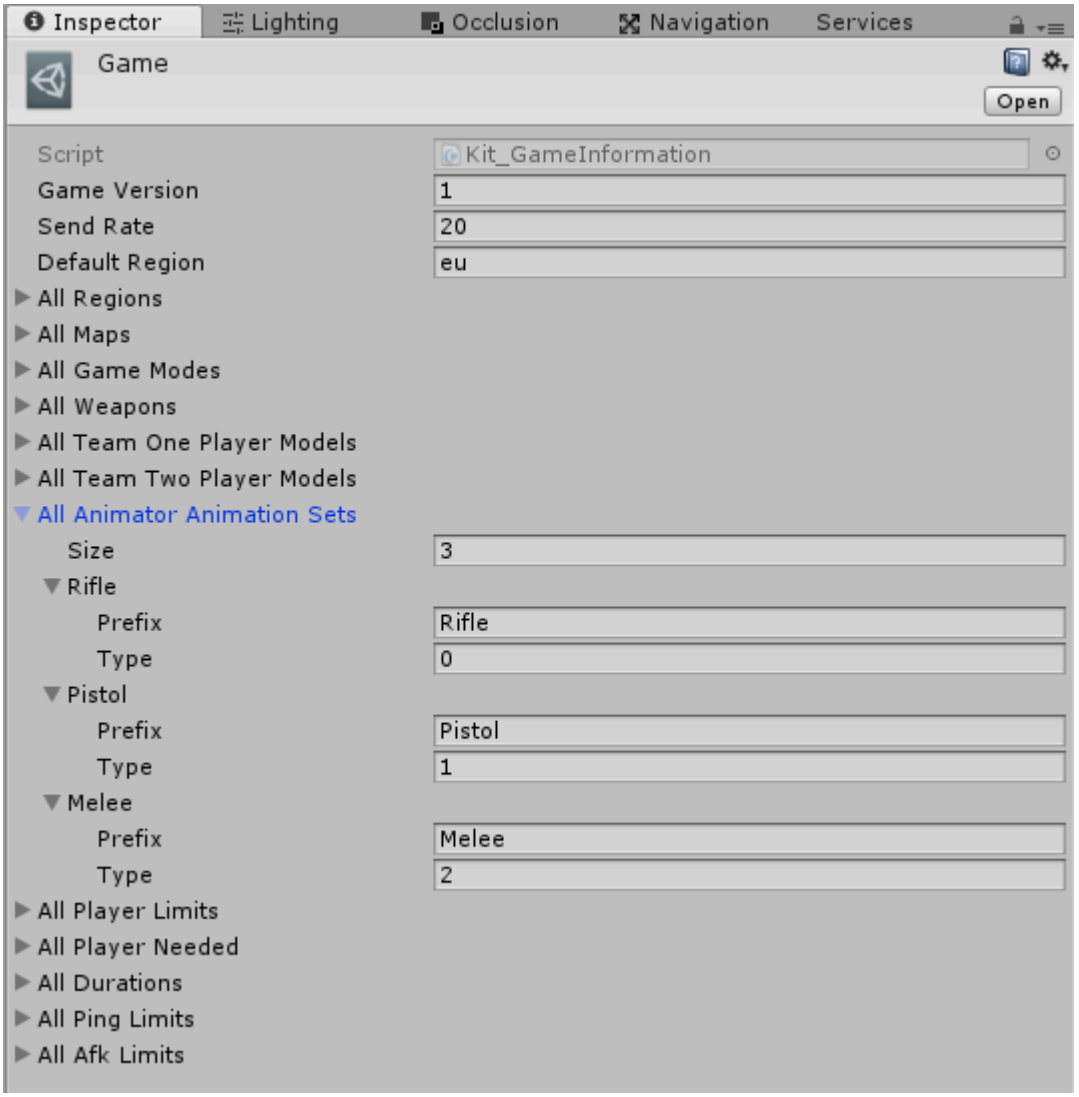


Then, select the “Game” asset, which can be found in “Assets\314

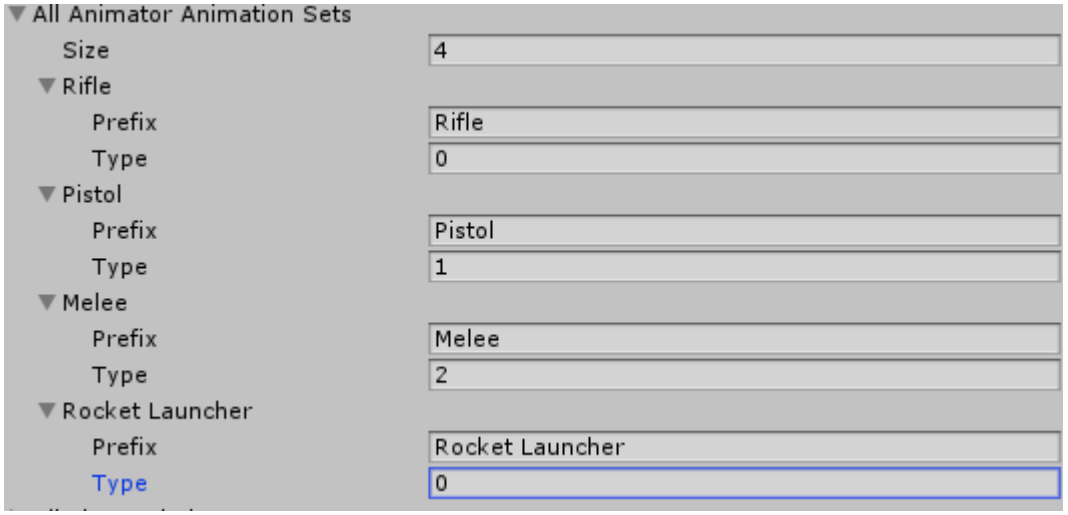
Arts\MarsFPSKit\Resources".



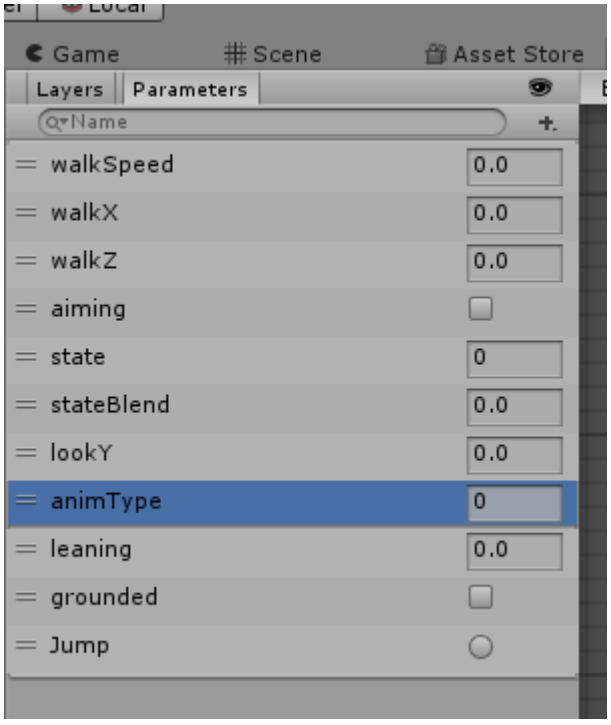
Under "All Animator Animation Sets", extend the length by one (example: 4 instead of 3) and in the new entry, set the Prefix to whatever you named the new Statemachine earlier.



->



The “Type” property is used for the looking layer “Upper Body Looking Additive” and will be set to the animator variable “animType” when the animation set is selected.

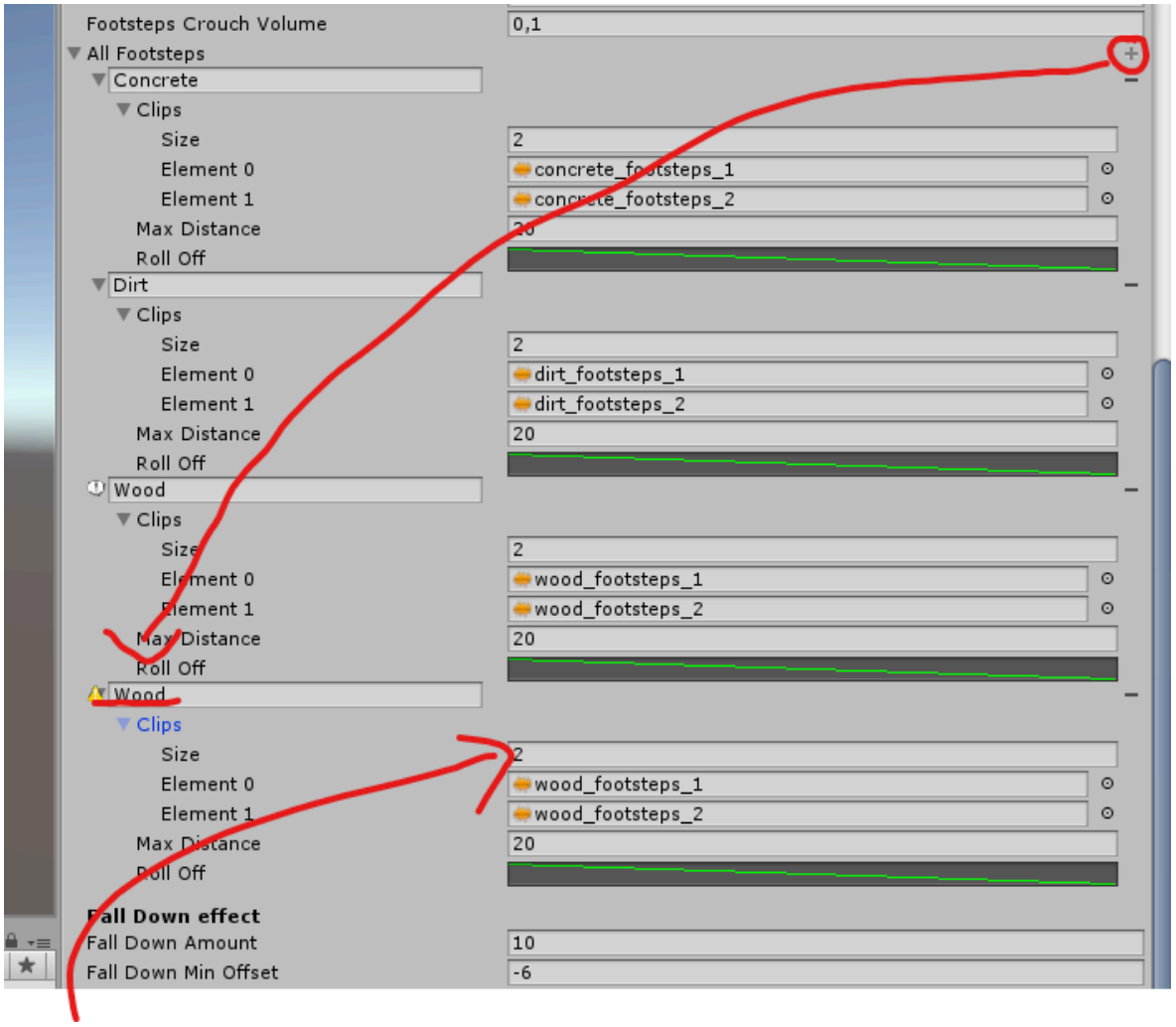


Adding new Footsteps

Footsteps are determined by tag of the object that you are walking on (except Terrains, for more on Terrains, see here).

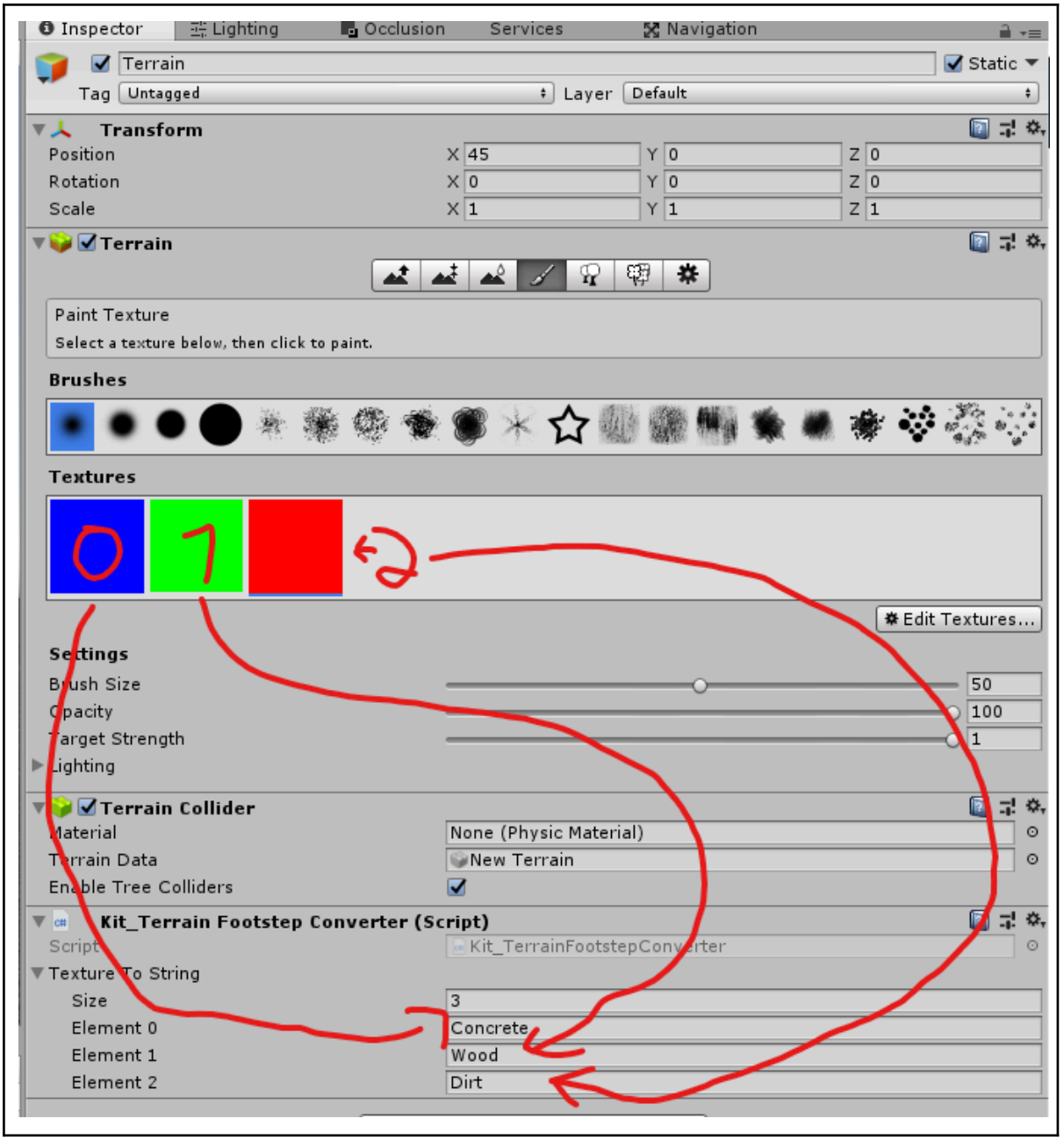
Adding a tag to the kit’s footstep system is really easy. It can be found on the “GenericBootsOnGround” object in “314 Arts\MarsFPSKit\GameData\Player\Movement”. To add a new supported tag, click the “+” button here and enter the tag into the marked

field, then you can drag in your new footstep sounds:



Terrain Footsteps

Footsteps on Terrain are determined via Texture to Tag script (called: “Kit_TerrainFootstepConverter”). Add it to your terrain and set the length of the array to the amount of textures you have on your terrain. The tag that the footstep system uses in then determined by the script, like this:



Extended Information

Dedicated Server

The best thing about using Mirror instead of PUN is that we can now have actual dedicated servers. You can just switch the platform to “Dedicated Server” in the build window to export a dedicated server. The kit will start a server automatically upon start. Settings can be configured via command-line arguments. All of the settings in the host screen are available: X stands for a number. It is the index in the array of the specified game mode and cannot be chosen completely freely.

- -name “name of server”

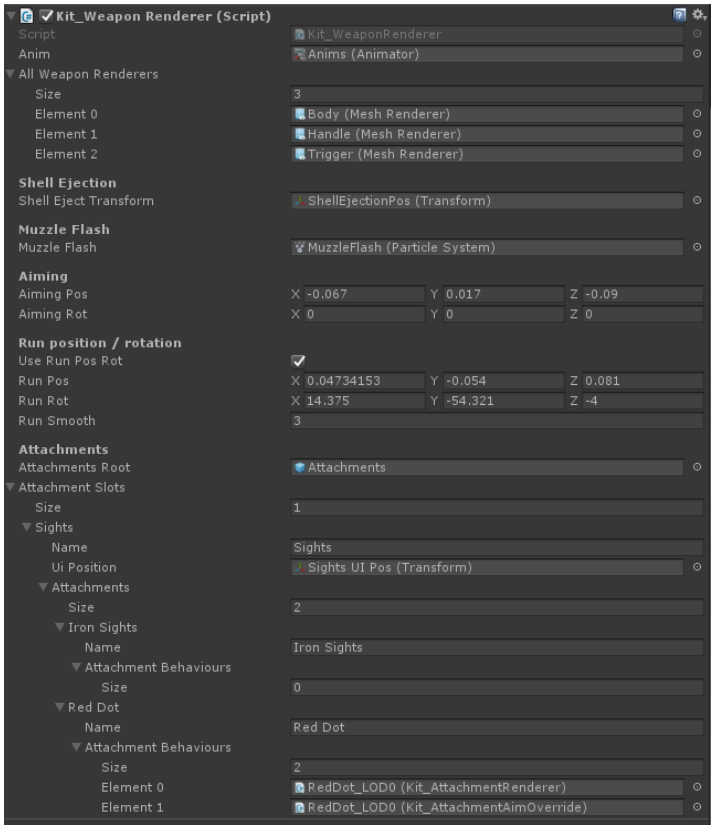
- -playerLimit X
- -playerNeeded X
- -map X
- -gameMode X
- -duration X
- -pingLimit X
- -afkLimit X
- -bots (just enables bots by supplying this argument)
- -password “the password you want to use”

Weapon Setup

In the Mars FPS Kit, a weapon consists of:

- First Person Prefab
- Third Person Prefab
- Drop Prefab
- Kit_WeaponBase (either a Kit_ModernWeaponScript, Kit_ModernMeleeScript or Kit_ModernGrenadeScript)

The three prefabs are essentially the 'visuals' of the weapon.
In case of Kit_ModernWeaponScript:
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. Please note that his only gets used if the aim transform is disabled.

- Enable Aim Transform

  Use the aim transform as described below

- Aim Transform

  The aim transform can be used to automatically calculate the position of the sight upon spawning. Place this at the center of your ironsights/scope.

- Aim Transform relative offset override

  If you want to use aim animation/position override hybrid (for scopes), you need to assign this to the point of the weapon where the aiming animation positions the weapon (center of screen)

- Aim Distance From Camera

  How far away (in meters) the weapon should be when aimed in

- “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.

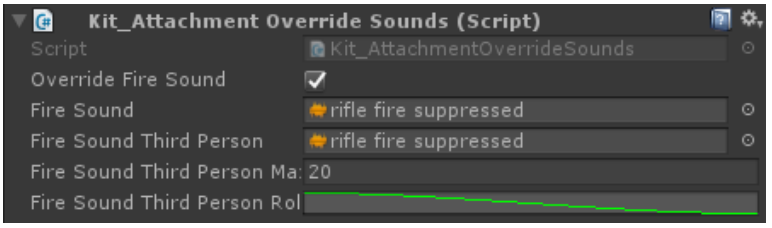
  - Currently available behaviours:
    - "Aim Override"

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

    - "Disable Muzzle Flash"

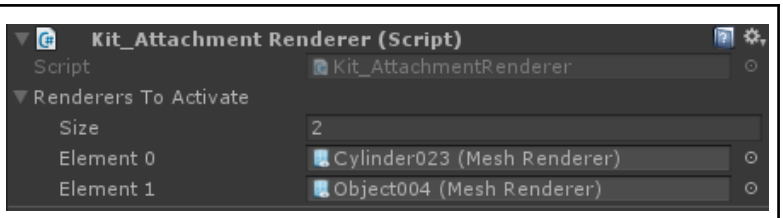
The "Disable Muzzle Flash" will disable the muzzle flash. Useful for suppressors.

    - "Override Sounds"



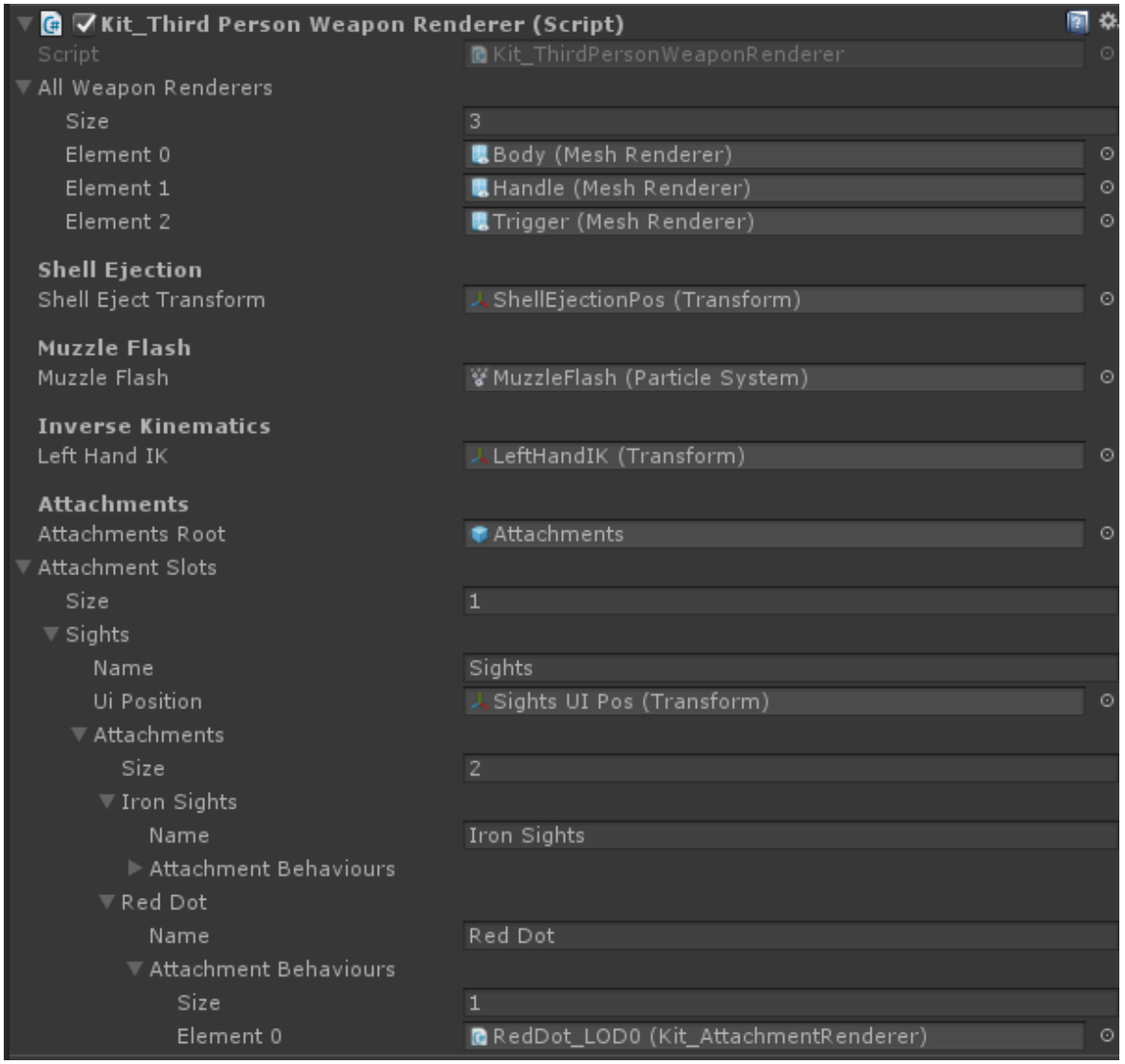
The "Override Sounds" will override the fire sounds and the third person falloff curve and range. Useful for suppressors.

    - "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”

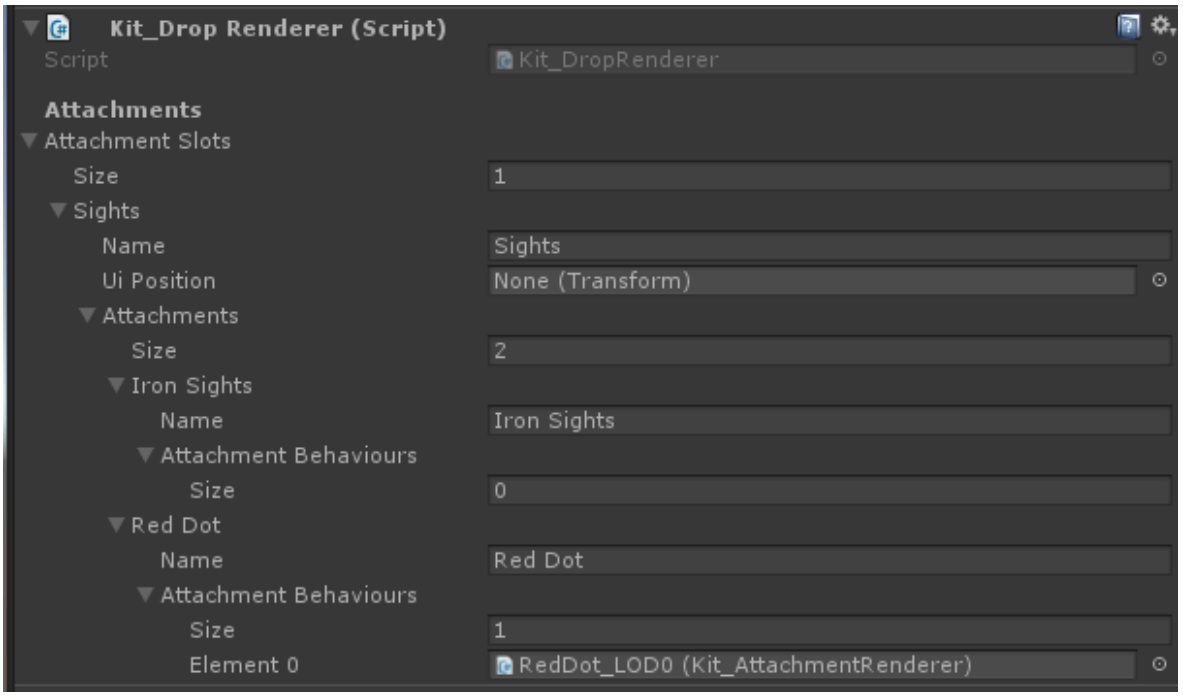
  “Left Hand IK”: If this is assigned, the left hand will be positioned at the assigned transform using Mecanim. It is very useful to match the third person animations to the weapons.

- “Attachments”

Certain attachment behaviours will not do anything in third person:

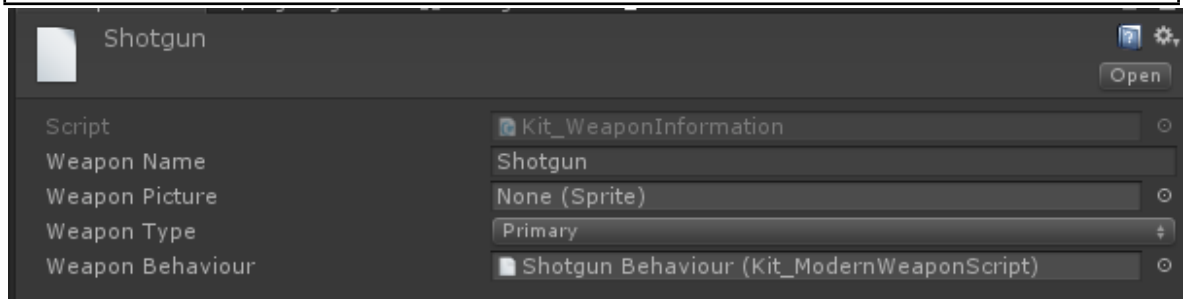
- “Aim Override”: As there is no re positioning in third person, it will not do anything.

Drop Prefab:

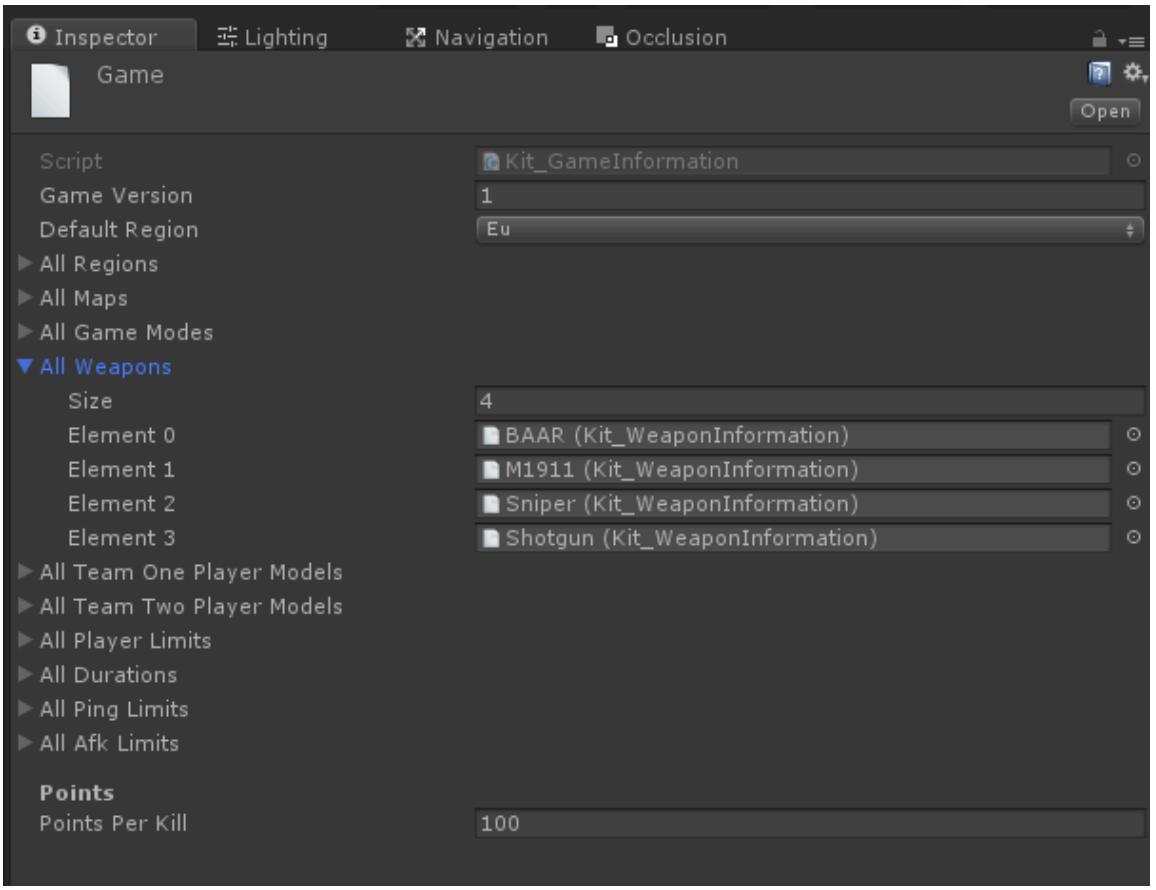


As you can see, it only has the attachments. Weapon drops don't do much except being there and looking correct.

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.

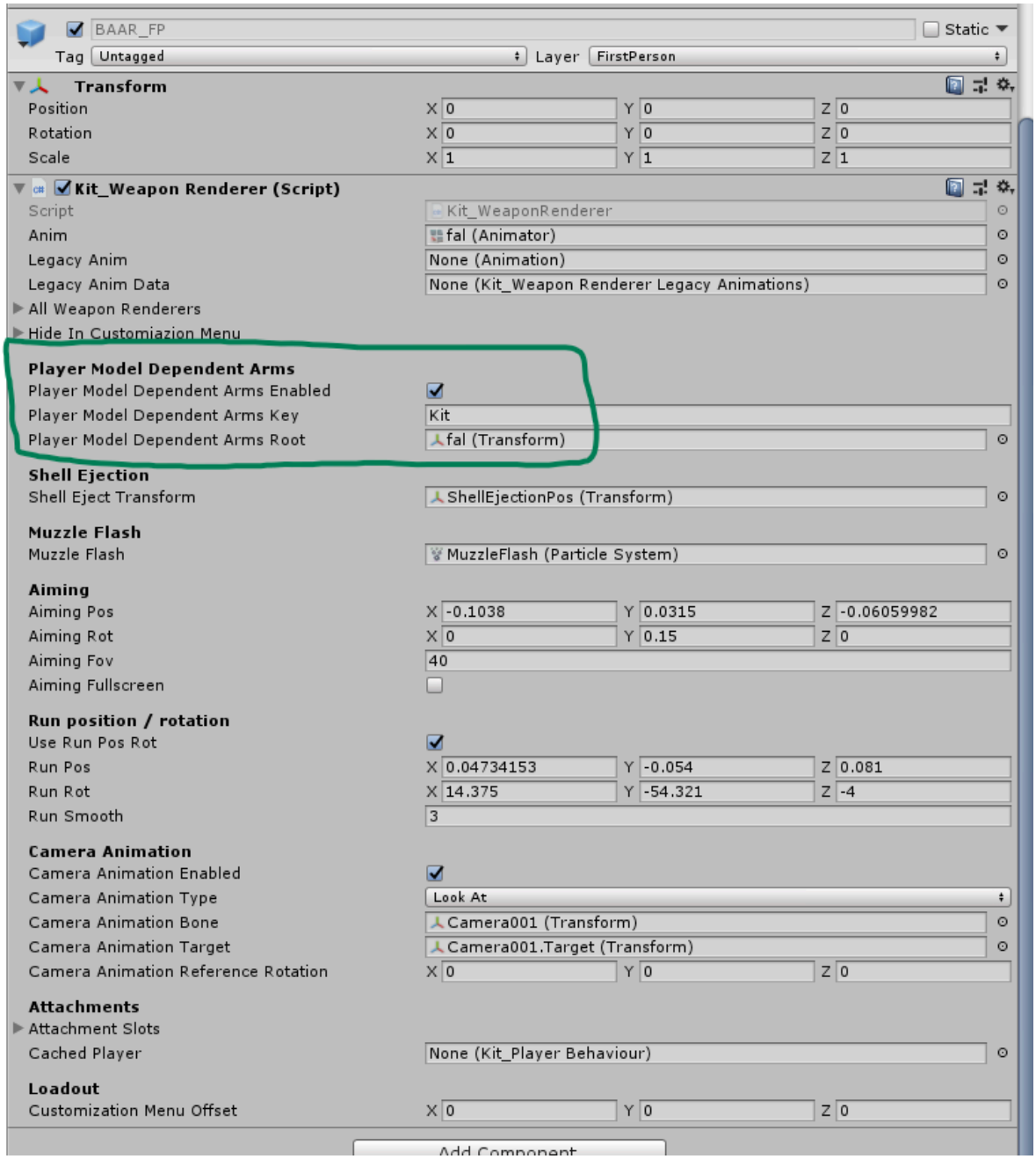


In order for a weapon to appear in game, it has to be in the “All Weapons” array of the “Kit_GameInformation” object, which can be found (by default) in: “Assets/MarsFPSKit/Resources/Game”. It should look like this:



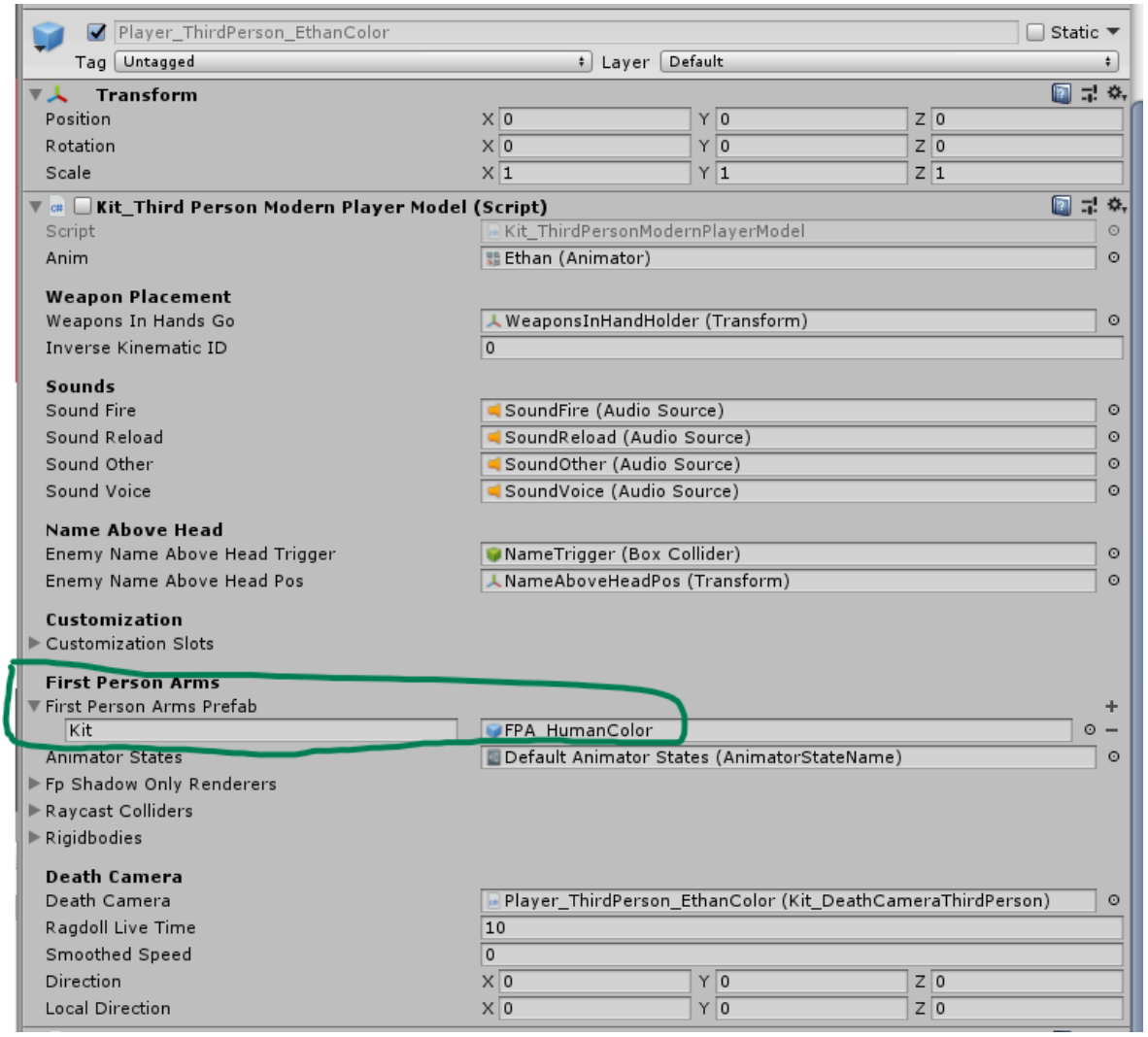
Player Model Dependent Arms System

The Arms System consists of three parts:
One part is on the **weapon prefabs** (the **_FP** prefabs) (Kit_WeaponRenderer, Kit_MeleeRenderer and Kit_GrenadeRenderer):



If you want to use it, check the "**Player Model Dependent Arms Enabled**" bool.
The "**Player Model Dependent Arms Key**" will be used in the second part of the setup (the player model) to get the correct set of arms. The default is "Kit".
The "**Player Model Dependent Arms Root**" is where the arms will be reparented to. Normally this would be the **Transform** of the **Animator**.

The second part of the system is on the **Player Model prefab**:



Here the key from the **_FP prefabs** is mapped to the correct arm prefab. Again: The default is “Kit”.

The third and final part is the **Arms prefab**:

It has just two properties:
Under “**Renderers**” you should assign all (Skinned) Mesh Renderers of your arms.

The Transforms that are under “**Reparents**” will be reparented to the (**Player Model Dependent Arms Root**) from the first part. Afterwards, the kit will call Animator.Rebind() to bind the arms’ bones to the animator.

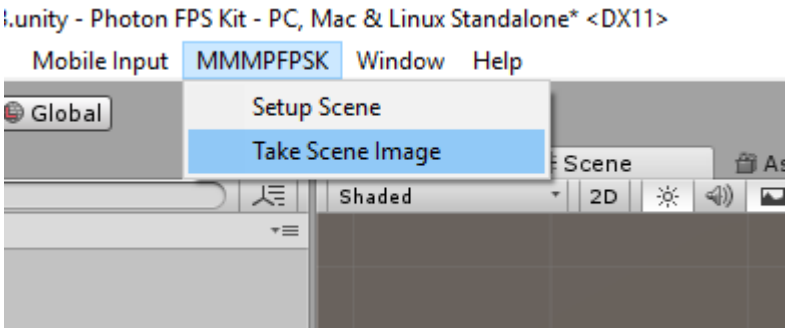
That’s all the magic behind it.
Please keep in mind when animating your weapons that you should **NEVER** parent a

weapon to the hands. This is bad practice in general and will also not work with the arms systems. The weapon's bones/meshes should always be independent of the arms' bones in first person animating (Of course this does not apply to some other setups such as a true Full Body setup, but in a split First Person / Third Person setup like this kit uses, this is true.

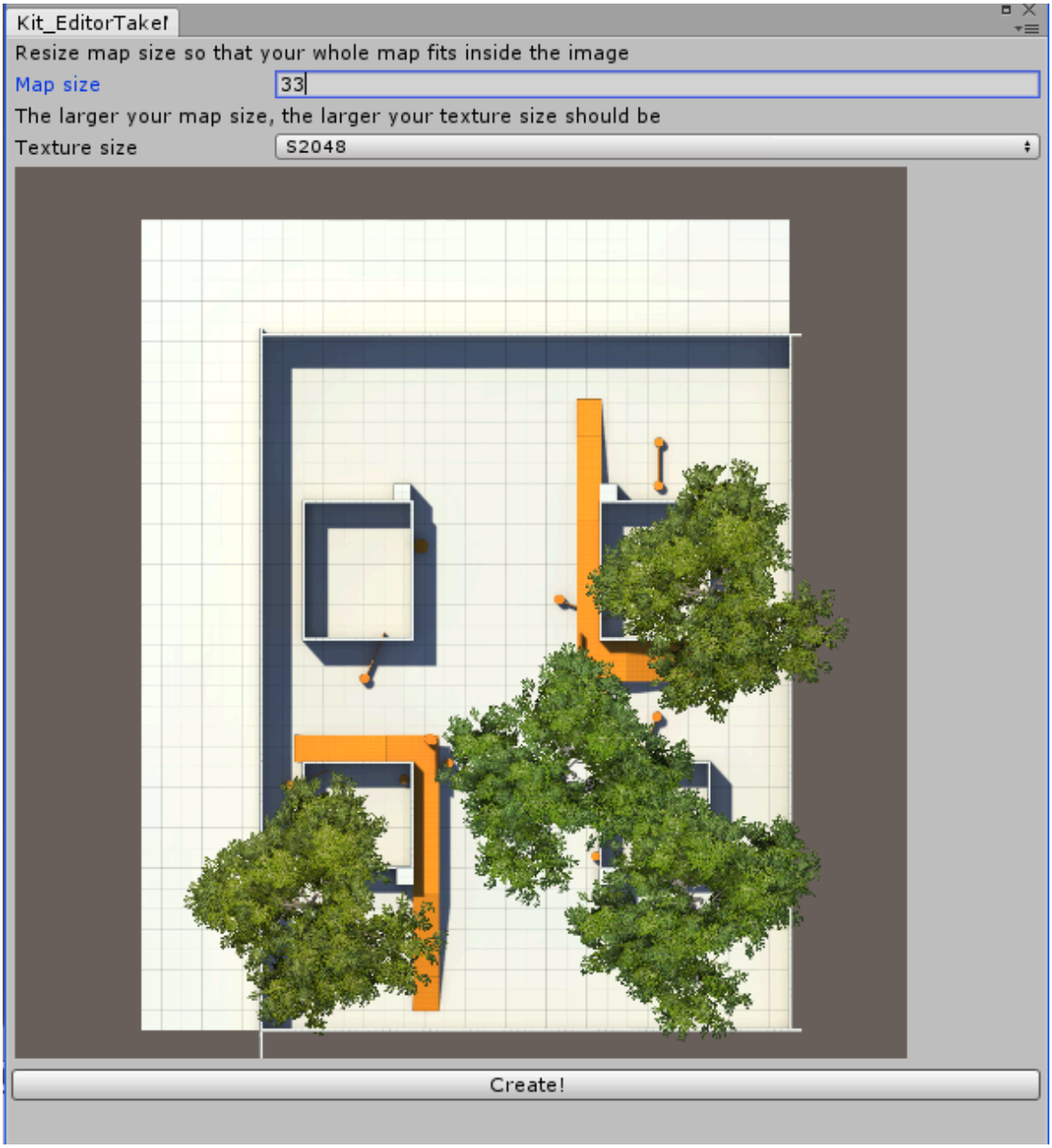
Minimap

Setting up a scene for the minimap

You will notice that by default, the minimap in a scene will be blue. This can be easily changed by generating the needed minimap scene picture. It will only take a few seconds to do:
First, open up the wizard:



Adjust "Map Size" to fit your scene, like this:



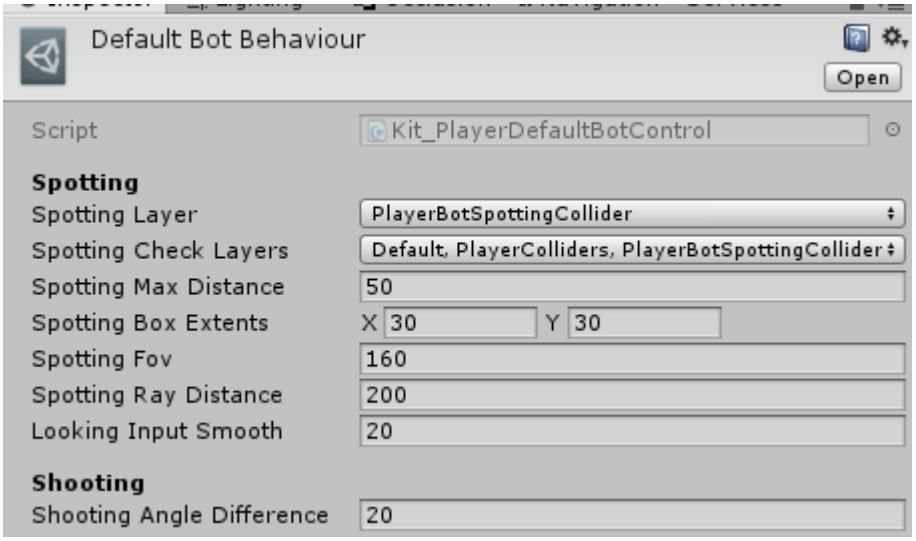
Select an appropriate texture size and press “Create!”. That’s it.

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’.

Game Modes

Domination

Domination game mode setup is a bit more complex. Here are some infos to help you:
Flags:
To add a flag, add the “Kit_Domination_Flag” script to an empty game object.
Make sure that you drag all your flags under **ONE, SINGLE,** game object that holds them.
Refer to the example scenes to see how it’s done.
That is very important because the **HIERARCHY order** determines the ID of the flag!

Flag Nav Points (Normal Nav Point script):
Flag Nav Points are used by the bots in order to capture them. Place them inside the capturing zone (see the “DominationFlagPrefab” prefab for that). The IDs start at 1 (!!!) because ID 0 is used for backup nav points. ID 1 belongs to Flag 0, ID 2 belongs to Flag 1 (see above) etc…

Flag spawn points (Normal spawn point script):
Flag Spawn Points are used to spawn when a team owns a flag. Like the nav points they don’t start at 0 for the flags either. In contrast to the nav points, these start at 3, because 0-2 are used for other purposes.

Spawn IDs 0-2
Spawn ID 0 is used as backup spawns.
Spawn ID 1 is used for team 1 during the spawn countdown
Spawn ID 2 is used for team 2 during the spawn countdown

You can use the scene checker to see if you have not forgotten anything. You can find it under “MMMPFPSK/Check Scene” in the toolbar.

Coding your own Game Mode

In the kit we differentiate between two types of game modes:

- PvP Game Mode (Multiplayer)

- PvE Game Mode (Singleplayer & COOP)

Each of these (Singleplayer, COOP and PvP) has its own list of game modes in the Kit_GameInformation file.

A game mode in MMFPSE is represented as a ScriptableObject.
For Singleplayer and COOP game modes the base class is called:

Kit_PvE_GameModeBase

For PvP (Multiplayer) game modes the base class is called:

Kit_PvP_GameModeBase

You create your own game mode by creating a new script and extending the base class of the game mode that you want to create. They sit directly in the “MarsFPSKit” namespace.

Both of these offer a series of callbacks that you can use to interact with the kit’s infrastructure. All of them have comments inside the script that explain what they do

A general timer (“timer”) and game mode stage (“gameModeStage”) is provided in Kit_IngameMain. You can use these but you don’t have to.

Both are automatically synced and controlled by the master client.

The timer offers a callback in the game mode scripts that fires once it has run out.

To display game mode specific things in the HUD, the scriptable objects contain a variable called “hudPrefab”.
This prefab, if assigned, is instantiated inside the kit’s HUD, so you do not have to add a Canvas yourself.
However, you have to add a script that derives from

Kit_GameModeHUDBase

Which offers an initialize and update callback. This class is universal for all game mode types.

For singleplayer and coop game modes you also have to supply a “menuPrefab” which contains the whole menu that you want to display in the main menu for this game mode.
Its base class is called

Kit_MenuPveGameModeBase

And provides a menu manager to navigate inside the menu, which is merged upon creation with the kit’s main menu manager (at runtime), as well as a Setup and Open callback which you can override.
An example menu is provided in the “Sandbox” game mode, everything is commented and explained inside the code to get you started.

Platforms

WebGL

Notes

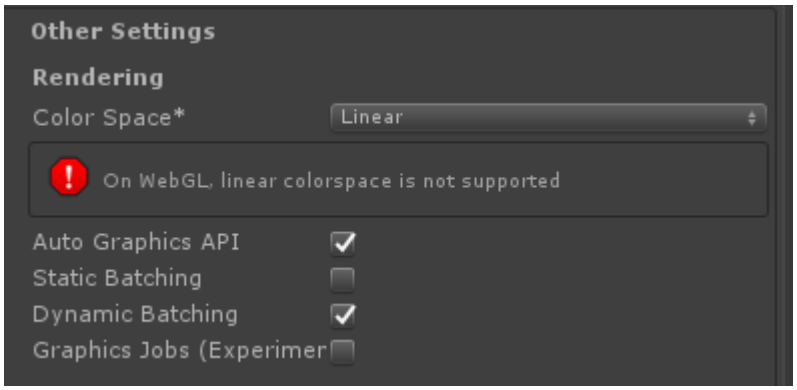
WebGL is currently untested and not officially supported.
When you are targeting WebGL, you should note a few things:

- You either have to change the Color Space to “Gamma” or disable “WebGL 1.0” graphics API

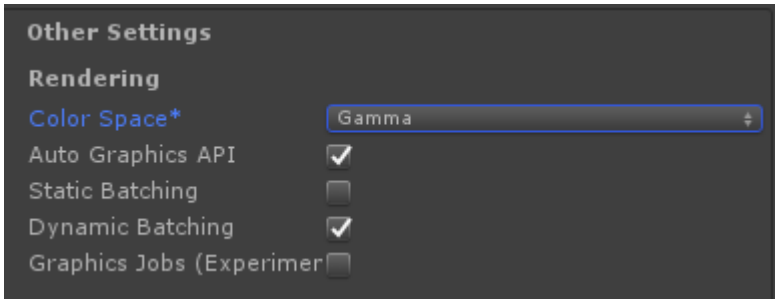
Guide

Color Space

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



To Gamma:



Compiling

Since WebGL is an AOT platform, you will need to use IL2CPP. For that you need a C++ compiler installed. If you do not have one installed already we recommend installing this one. Usually, the Unity Hub installs Visual Studio which has this included.

Running your build

To quickly prototype your build, use “Build and Run” under “File”.
If you want to run your existing WebGL 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). This only applies if you start the build from your local file system and not from a server

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

First Person View 2 (obsolete)

Unfortunately, FPV2 was abandoned by its author. If you are on 2018.1 it will still work out of the box, however later versions might not work and require additional shader work.

Drunken Lizard Games' First Person View 2 is the proper solution for avoiding first person weapons clipping through the environment.
Classic solutions like a dual camera setup have a lot of downsides, including, but not limited to:

- No shadows on first person weapons
- Performance downgrade
- Breaks some image effects (like TXAA)

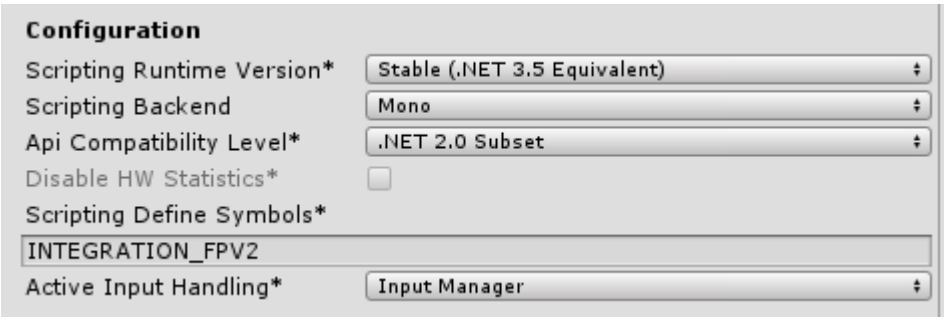
This is why the kit features a built-in integration for FPV2.
However, it requires some changes to be made.

Getting started

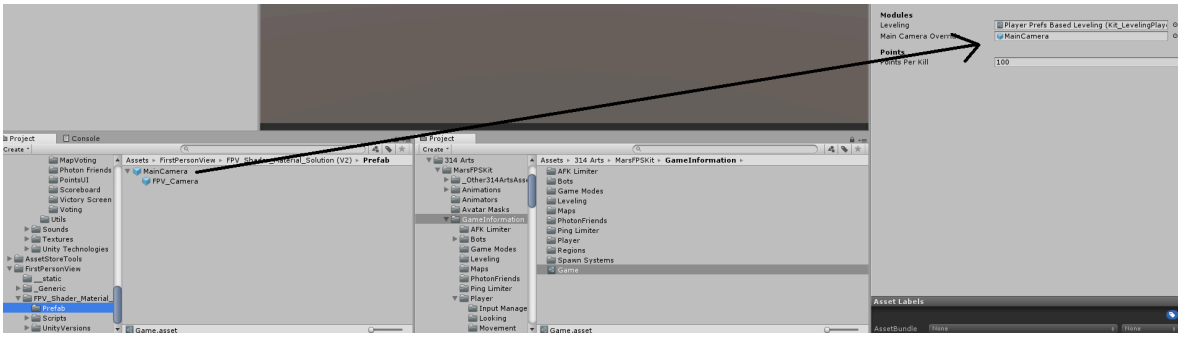
First, import FPV2 from the Asset Store.
Then, follow the setup instructions found in the FPV2 manual.

Enabling the integration

Second, enable the integration by defining the symbol "INTEGRATION_FPV2" in the Player Settings:



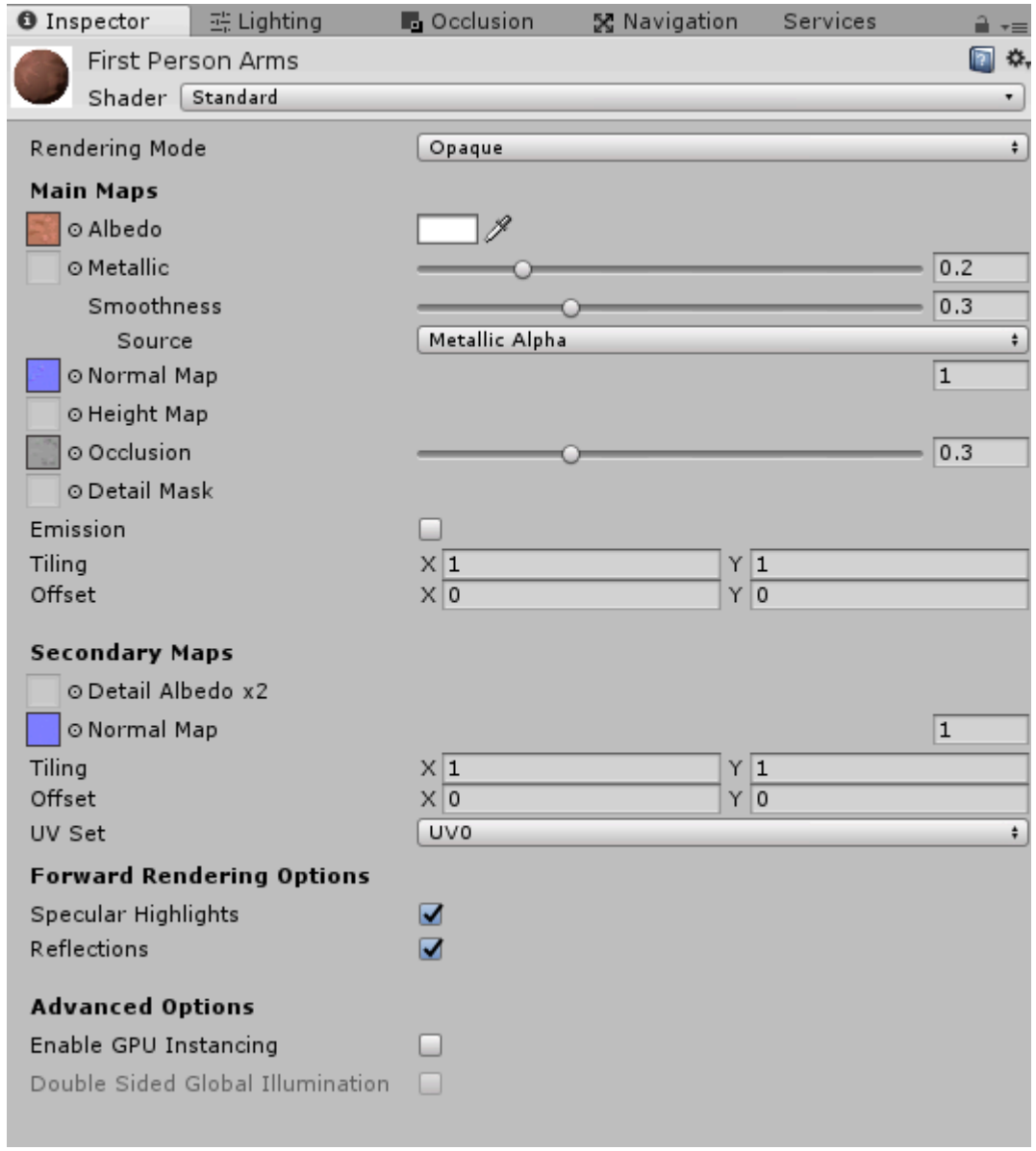
Then, assign the "MainCamera" prefab from the "Assets/FirstPersonView/FPV_Shader_Material_Solution (V2)/Prefab" folder to the "Main Camera Override" in the "Game" file:



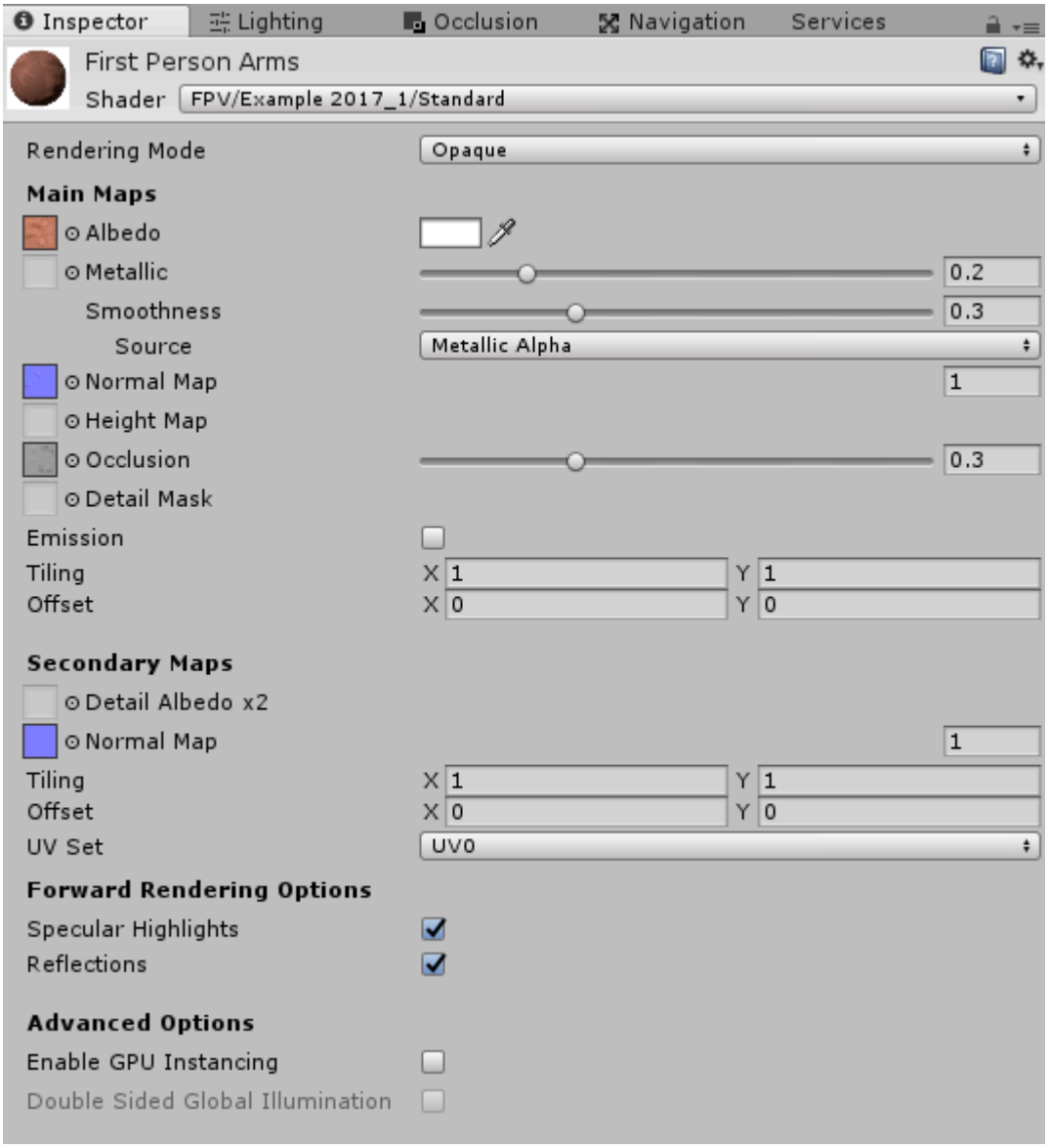
Finally, uncheck the “Minimap” and “LoadoutMenu” layer on the prefab’s camera in the “Culling Mask”.

Changing shaders

Thirdly, you need to change the shaders of the first person weapons and arms, to FPV variants. If you need to use custom shaders, refer to the FPV2 manual.
Arms example, from:


Inspector
Lighting
Occlusion
Navigation
Services
First Person Arms
Shader
Standard
Rendering Mode
Opaque
Main Maps
Albedo
Metallic
0.2
Smoothness
0.3
Source
Metallic Alpha
Normal Map
1
Height Map
Occlusion
0.3
Detail Mask
Emission
Tiling
X 1
Y 1
Offset
X 0
Y 0
Secondary Maps
Detail Albedo x2
Normal Map
1
Tiling
X 1
Y 1
Offset
X 0
Y 0
UV Set
UV0
Forward Rendering Options
Specular Highlights
Reflections
Advanced Options
Enable GPU Instancing
Double Sided Global Illumination

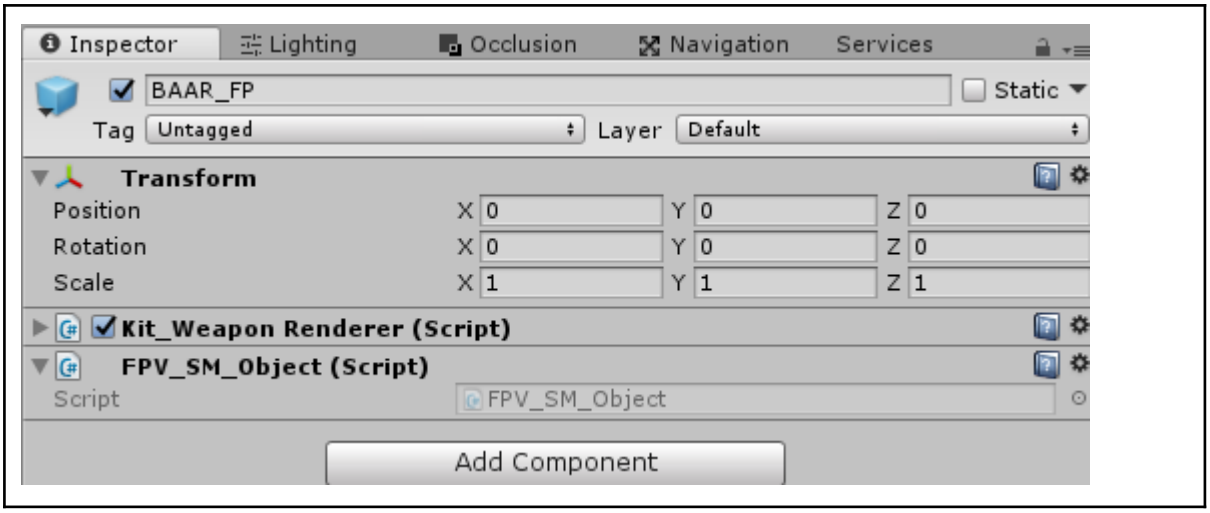
To:



Hint: Do not forget to use the correct Unity version shaders.
Second hint: You need to change the materials of all first person objects (By default: BAAR, Sniper, Pistol, Shotgun.
Third hint: Do not forget to change the materials of your attachments too.

Modifying the prefabs

At the very last, you need to add the “FPV_SM_Object” (It will show up as “FPV Object” when clicking Add Component) to all first person prefabs, meaning prefabs that (by default) end with “_FP”, like this:



Done

Now, play the project and check if all weapons work correctly. If that is the case, you're done!

Xsolla

Getting Started with the Xsolla Integration

1. This guide assumes you already have the kit setup in stock form.
2. Import Xsolla from the Unity Asset Store. Note: It has a higher Unity version requirement than MMFPSE does.
3. Under "314 Arts\MarsFPSKit\Integrations" extract the "Xsolla Integration.7z" file.
4. Make sure there are no compiler errors. If there are, contact us on Discord or via E-Mail. See the start page.
5. Go to the "314 Arts\MarsFPSKit\Integrations\Xsolla Integration\Prefabs\Master" folder.
6. Drag "XsollaLoadoutScreen" and "XsollaLoginScreen" prefabs in the "MainMenu" scene, which can be found in "314 Arts\MarsFPSKit\Scenes".
7. Drag "XsollaStore" and "XsollaLogin" prefabs from the "Xsolla" asset that you previously imported in step #1 into the main menu. They are in "Xsolla\Inventory\Resources" and "Xsolla\Login\Resources" folder.
8. On the "MenuManager" GameObject in the Main Menu, assign the XsollaLoginScreen and the XsollaLoadoutScreen to the respective slot under "Modules". Just drag and drop them in.
9. On the "StatisticsScreen", replace the script with the "Kit_MenuStatisticsForXsolla" and assign the same values as the original script had.
10. In the "Game" file in "314 Arts\MarsFPSKit\Resources\", assign "XsollaLeveling"

and "XsollaStatistics", which can be found in "314 Arts\MarsFPSKit\Integrations\Xsolla Integration\Game Data" to the "Leveling" / "Statistics" slot under "Modules" (at the bottom).

11.
12. Open the "MarsFPSKit_IngamePrefab" in "314 Arts\MarsFPSKit\Resources\".
13. Drag the "XsollaLoadoutScreen" prefab from Step #5 (the same prefab) under the "External Modules" Game Object in the "MarsFPSKit_IngamePrefab".
14. Assign the "XsollaLoadoutScreen" that you just dragged under "External Modules" to the "Loadout Menu" slot in the "Kit_IngameMain" script that is on the "MarsFPSKit_IngamePrefab".
15. The setup is now done. Now you need to setup Xsolla.

Setting up Xsolla

1. Go to https://xsolla.com/
2. Create an account there
3. In your publisher dashboard, create a game.
4. The modules you will need for the full integration is the Store, Login and Pay Station.
5. You need to setup virtual items to use the Xsolla Loadout (the Weapon store). This is the default setup, the external IDs are very important, they need to match exactly. Copy them for the default setup:
6. The important bit here are the "sku" settings. These will be mapped to in-kit weapons/player models via dictionaries. For the stock kit, copy the skus exactly.
7. In the Login module, create and setup a Login project. Copy the ID that is displayed to you.
8. In Unity, go to Window -> Xsolla -> Edit Settings
9. Paste the Login ID that you just copied to the Login ID at the top.
10. In the Xsolla Dashboard, the game that you have created has an ID displayed above it. Copy that ID and paste it in the same file as the Login ID under "Store SDK Settings" and then "Project ID".
11. If everything is done correctly, the Xsolla integration is now ready to be used. Test it out by starting the game. If it does not work, please contact us.

How Xsolla skus are mapped to in kit items

The mapping settings can be found in "XsollaMarsSettings" in "314 Arts\MarsFPSKit\Integrations\Xsolla Integration\Game Data". Weapon mapping is easy. It is the dictionary with the name "Store Weapon Id To Ingame Weapon Id". On the left (key) is the SKU from the Xsolla Dashboard, on the right you have the Weapon ID from the "Game" file (index in All Weapons array) ("314 Arts\MarsFPSKit\Resources\").

The player model mapping is similar, but it is per team and can be found under the "Team Settings" array. The "Team Name" property is just for navigation purposes and does not actually do anything. Teams are also identified via ID (index in "Game file/ "All Pvp Teams"). The Group name on Xsolla is defined as the following: "playermodels_" + teamName (teamName here is the "Team Name" property in the "Team" file!

Under the team, the "Store Player Model Id To Ingame Player Model Id" maps the SKUs from this team to the respective player model index in the "Team" file for this team.

For example, Team #0 ("Red" file in "All Pvp Teams" in the "Game" file) has "Ethan Color" as Index #0 in its "Player Models" array, therefore ethancolor is mapped to index #0.

Starting items

The player needs starting items. Its starting gear is defined on client side by SKU property. Weapons are mapped per category (primary, secondary, melee, lethal, nonlethal) in the "Starter Gear" array. Enter the SKU of the weapon that you want the player to have by default.

The same applies to the player models, under each team, one player model SKU MUST be entered as a starting gear. Entering nothing in any of these will result in errors and mayhem!

Item Types

All Xsolla item types (as of 22.05.2021) are supported. Non-Consumable, Consumable and Non-renewing subscription.

Video

This whole process is available as a video tutorial:
MMFPSE Xsolla Integration Tutorial Pt 1/2 Kit Setup
MMFPSE Xsolla Integration Tutorial Pt 2/2 Xsolla Setup

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

All the text is offset!

This most likely happens due to a Text Mesh Pro mismatch. Try updating TextMeshPro from the package manager.

If that doesn't help, try setting up a new project without importing package manager dependencies.

Ultimately if that does not fix it you can change the text back manually

My weapons are clipping through level 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. (OBSOLETE) 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. Integration instructions here.
3. Use First Person View 3
4. 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.

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.

I replaced the hit indicator. Now it is not centered anymore.

If you did not touch the UI itself (positions and pivot), the reason for this is most likely your replacement image. You need to make sure that it's completely ready to be rotated around its center. Check out the default image to see how it should look.

[...] is inaccessible due to its protection level.

This error is most likely caused by using an outdated Unity version. For proper function, please only use Unity 2018.1 and up.

My snipers are inaccurate.

To make your snipers fully accurate, go to its behaviour, open the tab "Bullet Spread" and make sure Aim base is close to 0 (0 for 100% accuracy) and Aim velocity base add is 0.

Need help?

Do you have a problem that you cannot solve using this document? You can seek help and/or join our community on our Discord here!
Be sure to have your Invoice ID ready to confirm your purchase of MMFPSE.

Changelog

0.8.0.0 BETA

- First release of Mirror version