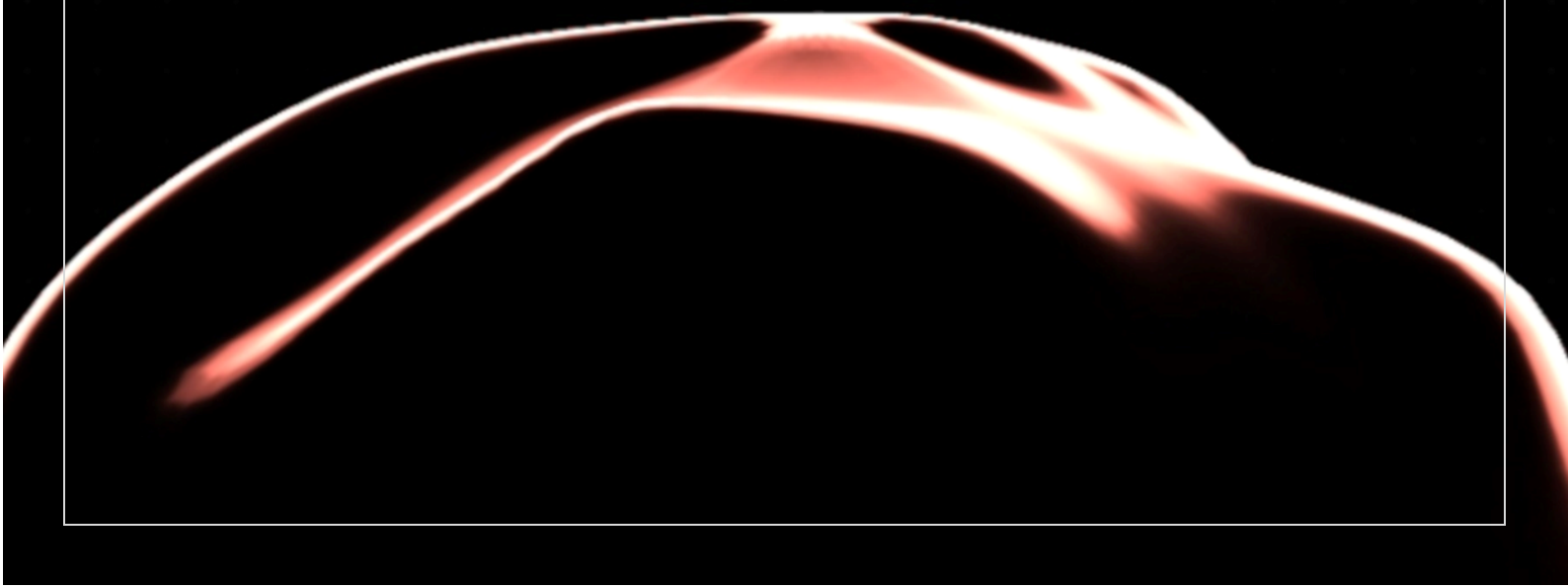
Cowsins™

FPS ENGINE DOCUMENTATION

V.0.9.6 BETA



INDEX

00.EARLY CONSIDERATIONS

00.a Cowsins Copyright and intellectual property.

Cowsins as a publisher on the Unity Asset Store, including Cowsin´s assets such as this specific one, COWSINS: FPS ENGINE, is protected under Unity Asset Store terms of service and EULA.

You can look at these on the official Unity Web Page.

The Unity Asset Store terms of service and EULA protect Cowsins™ as a publisher on the marketplace, including Cowsin's™ assets such as COWSINS: FPS ENGINE.

These are viewable on the official Unity website.

Last viewed: 27/03/22

UNITY EULA.9.1

The Assets are protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties.

UNITY EULA 6 REVERSE ENGINEERING, DECOMPILATON AND DIASSEMBLY

END USER may modify Assets. END USER shall not reverse engineer, decompile, or disassemble Services SDKs, except and only to the extent that such activity is expressly permitted under mandatory statutory applicable law.

CHANGES AND MODIFICATIONS

Following new updates, this document might be modified.

The asset may also experience multiple changes until the beta version is finished.

Both this document and COWSINS: FPS ENGINE are under development in a beta phase.

Current COWSINS´ FPS ENGINE version : 0.9.6

01.BASIC SET-UP

BEFORE READING: If you accessed this document through the Unity Project Window then you probably want to skip a few steps and move on to step number 4, since you already installed the package.

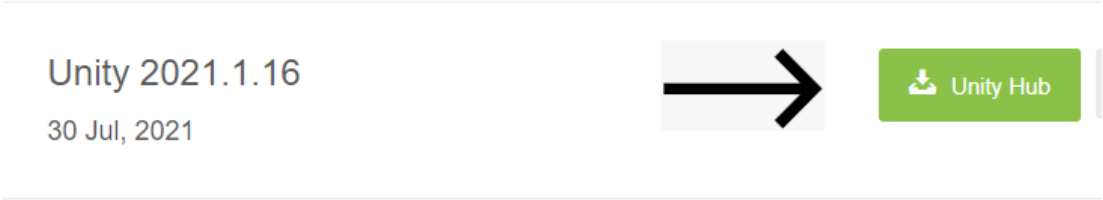
However, you can always check this instruction step by step if you run into any issues or haven't installed the asset yet.

STEP 01 – INSTALLING UNITY ----------

You can skip to the following step if you already have a version of Unity that supports COWSINS: FPS ENGINE. If not, please navigate to the Unity Download Archive (Last viewed on: 27/03/22). Choose a suitable version to download and install here. After that, choose "Unity hub" to download and install it, which is a convenient method to keep track of all your projects and versions in one location. You will be taken to the download page for the app if you already own Unity Hub.

Keep in mind that the asset was made using Unity 2021.1.16.

View the image below (PICTURE 1)



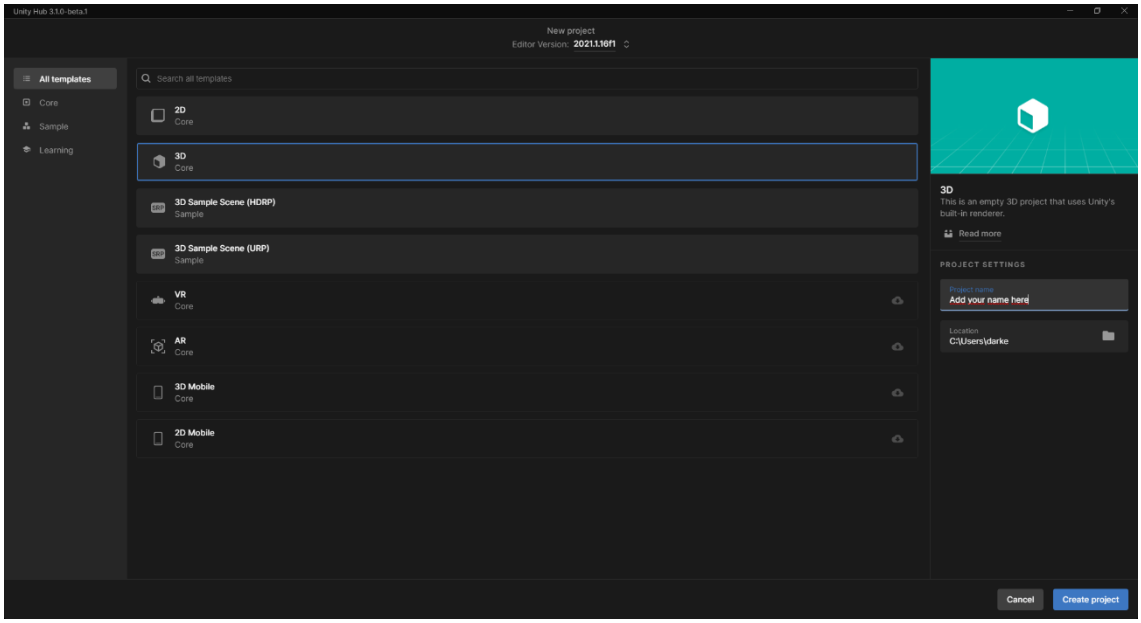
PICTURE 1. UNITY INSTALLATION THROUGH UNITY HUB

STEP 02 – PROJECT CREATION----------

After installing your desired version of unity, select "New Project" under "Projects" in the menu.

A new window will then open as a result. The render pipelines that Unity provides will be available for you to choose from here. The default standalone render pipeline was used to create the package. Choose 3D out of all the possibilities if you want to maintain all the materials and graphic settings for the project that the asset includes.

You can select a different render pipeline, but keep in mind that you will need to translate each material into the materials used by that render pipeline.



Remember to give your project an appropriate name as well.

STEP 03 – IMPORTING PACKAGE ----------

Once Unity is fully loaded, we´ll import the asset. To do that, go to the Unity Package Manager. On the new window, select “My Assets” which stands for the assets you own. Search “FPS Engine” (PICTURE 5) and click download. Once it is downloaded, click install.

Make sure to import everything so there are no errors..

STEP 04 – DEPENDENCIES ----------

Have in mind that the asset includes dependencies, such as TextMeshPro, InputManager or PostProcessing.

STEP 05 – URP OR HDRP USERS ----------

Note that even though FPS Engine is fully compatible with URP and HDRP, you may lose the world space shader as well as you will need to replace all the materials from the Built-in pipeline to the one you chose: Apart from that, FPS Engine should work as usual!

To upgrade built-in Shaders:

- Open your Project in Unity, and go to Edit > Render Pipeline > Universal Render Pipeline.
- According to your needs, select either Upgrade Project Materials to URP Materials or Upgrade Selected Materials to URP Materials.

Check Unity´s official documentation regarding this for more information.

Following these steps, your project ought to be operational. If you have any problems or inquiries, you can get in touch with Cowsins™.

SUPPORT :

E-mail : cowsinsgames@gmail.com

Discord: https://discord.gg/mqdnyYZ894

Twitter: https://twitter.com/cOWsins

02. ROADMAP

Welcome to COWSINS™: FPS ENGINE beta release version.

At Cowsins™ we have always been passionate about videogames, and specially about game development, and that´s why we built this asset with all our love, dedication, attention to detail and knowledge we could deliver.

We are so excited to see how the project evolves through the beta in a new thrilling journey for us.

So, what´s coming next? The upcoming update will feature major improvements and adjustments to many systems of the asset in order to make it even more solid!

You can suggest new features in our discord server!

https://discord.gg/9eYPeHg4fh

Check the official Trello board!

https://trello.com/b/Fl0xc5Er/fps-engine-roadmap

CHANGELOG

Version 0.9.6（current）

0.9.6

Weapon System Redesign

Custom Shot Methods

Better Weapon Bob Effect

Better Weapon Sway Effect

Better Input Manager

Device Detection

Device Detection UI

Improved Crosshair

Other Improvements

0.9.5

Wall Run

Wall Jump

Wall Bounce

Multiple Jumps (Double jump)

Directional Double Jumping

Coyote Jump

Auto Run

Simple procedural Weapon Shooting Effect

Added Main Menu

Added Simple Settings Menu

Speed Lines Effect

New Showroom

Input Manager new structure

Better FOV handling system

Improved fall damage system

Better Door Interaction system

Smooth Crouch Transition

Modify Weapon SFX Pitch

Enable or disable dropping

Shooting UI Effect

Numeric health and shield display

UI Sway Effect

Optimization improvements

Scale Fixed

0.9

0.8.5

Added a DEMO

Added a compass system.

Added customizable compass elements.

Added customizable Initial weapons.

New customization features:

- Jump Crosshair Spread
- Jump cooldown

Added new UI Elements: Low Ammo & Reload displays.

Improved pick-up system: Input detection and UI display

Improved presets system.

Improved melee attack.

Minor improvements and fixes.

0.8.1

Minor improvements and fixes.

New customization feature: Allow Crouch While Jumping Toggle.

0.8 - Launch version.

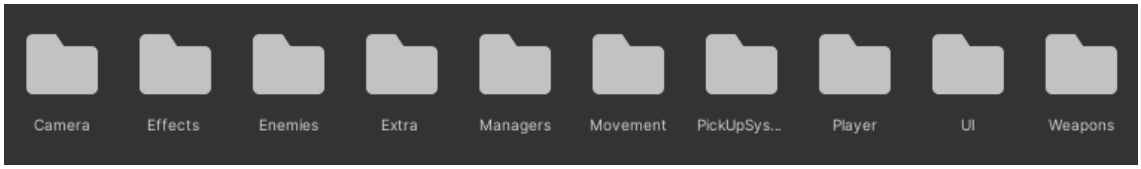
03. CONTENT

You may find all the information you want in this area, together with an explanation of each variable, to comprehend how the scripts included in the package operate.

Remember also that all the code is well documented with comments.

The scripts will be arranged similarly to how they are here, within the appropriate directories.

Examine the organization:



Information is shown using custom inspectors and is divided into tabs (not in every situation, but on the most important code included). Information is arranged in the documentation in the same way.

03.1 – CAMERA

MoveCamera.cs

This script places the camera at its correct position and makes sure it is properly situated.

> Head: Reference to our head = height of the camera. This is a transform inside the Player Prefab.

CameraFOVManager.cs

This script handles FOV for different movement states such as running, walking or wallrunning.

> Player: Reference to our player (Rigidbody).

03.2 – MOVEMENT

PlayerMovement.cs

Enables the player to move, carry out necessary activities, as well as some additional ones like slide. Here, we also deal with the stamina system and aim assistance.

Note that this script was built on top of Dani´s system, hence it is also based on physics and rigidbodies.

ACCESSORS

Vector3	crouchScale	Private set. Defines the size of the player when crouching.
Vector3	PlayerScale	Returns playerScale. This is the scale of the player.
bool	CanJump	Public set. Returns canJump, true if the player can jump, and negative if the player is not able. This is related to Stamina.
bool	ReadyToJump	Private set. Returns readyToJump.
bool	wallRunning	Private set. Returns true if the player is currently wall running.
bool	wallLeft	Private set. Returns true if the player detects a wall at the left side of the orientation.
bool	wallRight	Private set. Returns true if the player detects a wall at the right side of the orientation.
bool	wallOpposite	Private set. Returns true if the player detects a wall opposite of the orientation.

ASSIGNABLES

PlayerCam: Locate your camera parent, then add it here. The camera must be contained in an empty GameObject.

Orientation: Player orientation is provided via an empty object that has the same height as your camera.

UI: Player UI Canvas.

- DashUIContainer: Contains dashUIElements in-game.
- dashUIElement: Displays a dash slot in-game. This keeps stored at dashUIContainer during runtime.

UseSpeedLines: Displays speed lines effects at certain speed.

- SpeedLines: Speed lines particle system.
- SpeedLinesAmount: Being 1 = default, amount of speed lines displayed.

MOVEMENT

BASIC MOVEMENT INPUT SETTINGS

AutoRun: Enable this to instantly run without needing to press the sprint button down.

AlternateSprint: If false, hold to sprint, and release to stop sprinting.

AlternateCrouch: If false, hold to crouch, and release to uncrouch.

canRunBackwards: If true: runSpeed equals the speed of running backwards. If false, running backwards speed equals walking speed.

canRunWhileShooting: If true: Speed while shooting = runSpeed. If false: Speed while shooting = walkSpeed.

LoseSpeedDeceleration: Player deceleration from running speed to walking

BASIC MOVEMENT

Acceleration: Capacity to gain speed.

RunSpeed: Speed reached while running.

TIP: If you are making a tactical shooter, make this value lower than the walkSpeed so whenever you press the SprintButton, you will “shift”.

WalkSpeed: Speed reached while walking.

CrouchSpeed: Speed reached while walking crouched.

CrouchTransitionSpeed: Speed to crouch and uncrouch. (Speed of the motion)

MaxSpeedAllowed: Max speed the player can reach. Velocity is clamped by this value.

WhatIsGround: LayerMask. Assign all surfaces targeted as ground, so the player will be able to walk on these.

FrictionForceAmount: Force applied on movement. This goes against the movement itself. A higher value will stop the body easier.

MaxSlopeAngle: Max angle for a slope to be considered as walkable.

CAMERA

CAMERA LOOK

SensitivityX: Horizontal sensitivity (X Axis)

SensitivityY: Vertical sensitivity (Y Axis)

Controller Sensitivity X: Horizontal sensitivity (X Axis) using controllers

Controller Sensitivity Y: Vertical sensitivity (Y Axis) using controllers

Aiming Sensitivity Multiplier: Sensitivity will be multiplied by this value when aiming

CAMERA

NormalFOV: Default field of view of your main camera, MainCamera.

RunningFOV: Running field of view of your main camera, MainCamera.

WallrunningFOV: WallRunning field of view of your main camera, MainCamera.

FadeInFOVAmount: Fade Speed - Start Transition for the field of view

FadeOutFOVAmount: Fade Speed - Finish Transition for the field of view

CameraTiltTransitionSpeed: Speed of the tilt camera movement. This is essentially used for wall running

SLIDING

AllowSliding: Mark this as true if you want your players to be able to slide.

SlideForce: Force applied to the body in favour of movement on sliding.

SlideCounterMovement: Same force applied to the normal movement, but this time, applied on sliding.

JUMPING

MaxJumps: Amount of jumps you can do without touching the ground.

resetJumpsOnWallrun: Gains jump amounts when wallrunning.

resetJumpsOnWallBounce: Gains jump amounts when wallrunning.

doubleJumpResetsFallDamage: Double jump will reset fall damage

(only if your player controller is optable to take fall damage)

DirectionalJumpMethod: Method to apply on jumping when the player is not grounded, related to the directional jump.

NONE: No directional jump will be applied.

INPUT BASED: Movement will be based on the current input (WASD keys)

FORWARD MOVEMENT: Player will always go forward, independently of the inputs currently pressed.

DirectionalJumpForce: Force applied on an object in the direction of the directional jump.

JumpForce: The higher this value is, the higher you will get to jump.

ControlAirborne: How much control you own while you are not grounded. Being 0 = no control of it, 1 = Full control.

AllowCrouchWhileJumping: If enabled, players will be able to crouch mid-air.

canJumpWhileCrouching: If true, allows the player to jump while crouching.

JumpCooldown: Interval between jumping. When the player lands, time that must elapse in order to being able to jump again.ç

CoyoteJumpTime: Coyote jump allows users to perform more satisfactory and responsive jumps, especially when jumping off surfaces

AIM ASSIST

ApplyAimAssist: Mark this as true if you want your players to use an "auto-aim" or "aim assistance" system.

MaximumDistanceToAssistAim: Maximum distance for the player to detect a target

AimAssistSpeed: Speed for the crosshair to stick on the desired target.

AimAssistSensitivity: Range of the aim assist. The higher the value is, the more sensitive the system will be. The system uses a sphereCast to detect potential targets, so the higher the sensitivity, the higher the radius of the sphere.

STAMINA

usesStamina: Mark this as true if you want to use a stamina system. If you used a Player "DragAndDrop Prefab", note that the stamina slider UI is already included. So if you are using it, keep it as it comes. If not, make sure to remove that UI element from the scene. If you don´t do this, it will appear on your screen.

MinStaminaRequiredToRun: When you run out of stamina, this is the minimum stamina amount you have to regenerate before performing stamina consuming actions.

MaxStamina: Maximum amount of stamina you can store.

StaminaLossOnJump: Amount of stamina lost on jumping.

StaminaSlider: UI that shows your current and max stamina. Note that if you are not moving + you currently have the max stamina amount allowed, it will be removed from the screen until you perform any stamina consuming action.

ADVANCED MOVEMENT

WALLRUN

canWallRun: When enabled, it will allow the player to wallrun on walls.

whatIsWallRunWall: Define wallrunnable wall layers. By default, this is set to the same as whatIsGround.

useGravity: When enabled, gravity will be applied on the player while wallrunning. If disabled, the player won´t lose height while wallrunning.

> wallRunGravityCounterForce: Since we do not want to apply all the gravity force directly to the player, we shall define the force that will counter gravity. This force goes in the opposite direction from gravity.

maxWallRunSpeed: Maximum speed reachable while wall running.

upwardsWallJumpForce: When wall jumping, force applied on the Y axis.

normalWallRunForce: "When wall jumping, force applied on the X axis, relative to the normal of the wall.

stopWallRunImpulse: Impulse applied on the player when wall run is cancelled. This results in a more satisfactory movement. Note that this force goes in the direction of the normal of the wall the player is wall running.

wallRunMinimumHeight: Minimum height above ground (in units) required to being able to start wall run. Wall run motion will be cancelled for heights below this.

wallrunCameraTiltAmount: Rotation of the camera when wall running. The rotation direction gets automatically adjusted by FPS Engine.

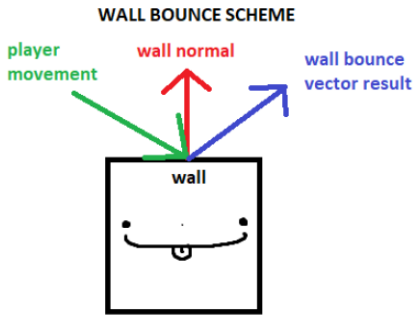
cancelWallRunMethod: Method to determine length of wallRun.

> NONE: Infinite wall running if requirements are met.
>
> TIMER: Cancel wall run after a certain time.
>
> wallRunDuration: Duration of wall run for cancelWallRunMethod = TIMER.

WALL BOUNCE

canWallBounce: When enabled, it will allow the player to wall bounce on walls.

wallBounceForce: Force applied to the player on wall bouncing. Note that this force is applied on the direction of the reflection of both the player movement and the wall normal.



wallBounceUpwardsForce: Force applied on the player on wall bouncing (Y axis – Vertical Force).

oppositeWallDetectionDistance: maximum Distance to detect a wall you can bounce on. This will use the same layer as wall run walls.

DASHING

canDash: When enabled, it will allow the player to perform dashes.

dashMethod: Method to determine how the dash will work

infiniteDashes: When enabled, You will be able to perform dashes with no limitation.

amountOfDashes: maximum (initial) amount of dashes. Dashes will be regenerated up to “amountOfDashes”, you won´t be able to gain more dashes than this value, check dash Cooldown.

dashCooldown: Time to regenerate a dash on performing a dash motion.

damageProtectionWhileDashing: When enabled, player will not receive damage.

dashForce: force applied when dashing. Note that this is a constant force, not an impulse, so it is being applied while the dash lasts.

dashDuration: Duration of the ability (dash).

canShootWhileDashing: When enabled, it will allow the player to shoot while dashing.

OTHERS

footstepVolume: Volume of the footsteps.

footstepSpeed: Play speed of the footsteps.

Footsteps: You will originally find the Default, Grass, Metal, Mud and Wood steps arrays. Add as many sounds you´d like for each of those surfaces. They will be randomly played assigning a random pitch to the sounds (within a limit) in order to approach a more realistic result.

Events: Custom events for you to customize your player even more.

You will also find the editor of the script in the same folder.

03.3 – PLAYER

PlayerStats.cs

Handles the statistics of the players, including health, shield, power-ups values and UI related to it. Also handles extra features such as fall damage.

PLAYER STATES

MaxHealth: Amount of health you will initially have. This is also the max health you can reach.

MaxShield: Amount of shield you will initially have. This is also the max shield you can reach.

FALL DAMAGE

TakesFallDamage: If true, the player will take damage when falling from certain heights.

- minimumHeightDifferenceToApplyDamage: Minimum height (in units) the player has to fall from in order to take damage.
- FallDamageMultiplier: How the damage will increase on landing if the damage on fall is going to be applied.

UI

Bar Health Display: Use image bars to display player statistics.

- HealthSlider: UI Element (Slider) that displays current and maximum health.
- ShieldSlider: UI Element (Slider) that displays current and maximum shield.

Numeric Health Display: Use text to display player statistics.

- HealthTextDisplay: UI Element (TMPro text) that displays current and maximum health.
- ShieldTextDisplay: UI Element (TMPro te¡xt) that displays current and maximum shield.

healthStatesEffects: This image shows damage and heal states visually on your screen, you can change the image to any you like, but note that color will be overriden by the script.

DamageColor: Color tint for health states effects image when hurt.

HealColor: Color tint for health states effects image when healing.

EVENTS.

You will also find the editor of the script in the same folder.

PlayerStates.cs

This script is fundamental. It must be attached to your Player object. It manages the player's state and determines the player's current condition as well as what to do in each circumstance. The Player States folder contains all of the states.

PlayerGraphics.cs

Handles the graphics object of the player, so it will always remain in place.

Player: Attach your player object (transform to copy position and rotation).

03.4 – WEAPONS

In this section, you can find many scripts. They all take part of the weapon system. However, one of them takes part of weapon creation system while the others, take care of handling the shooting aspect as well as identifying the weapon properly for it to work as expected.

This is potentially subject to change and be improved since we are still in beta phase.

If you want to learn how to access the weapon workshop and easily add new weapons, you may consult the documentation. The foundation of this system is the following script, which includes a huge variety of attributes and variables, allowing for a great degree of customisation. Please read them carefully:

Weapon_SO.cs

Take note that scriptable Objects frequently finish with the prefix _SO. In fact, scriptable objects are used across the whole weapons system, which is the right technique to keep all of your weapon data in separate files. If you do not know what a scriptable object means, you definitely want to check it out, since it is pretty convenient, however you do not really need to do that, since Cowsins: FPS Engine handles all of that for you, and teaches how to add weapons in an easy and fast way.

ACCESSORS

bool	Reloading	Public set. Returns reloading. True if the player is currently reloading.

BASIC

Weapon ID: it is ESSENTIAL to fill this properly. This is basically the main thing used to identify your weapon. This determines the order of the weapon inside the weapons array in the WeaponController.cs

FIRST weapon created: Weapon ID = 0.

SECOND weapon created: Weapon ID = 1.

THIRD weapon created: Weapon ID = 2.

etc...

Name: Give your weapon a name. This will be displayed in the pick up interface.

weaponObject: Attach your weapon prefab here. This weapon prefab will be instantiated on your screen when you own this weapon.

PickUpGraphics: Visuals that will appear on a dropped weapon which is pickeable.

Icon: Custom Sprite of your weapon.

SHOOTING SETTINGS

ShootStyle: Choose among Hitscan, Projectile and Melee weapon styles. Note that hitscan weapons shoot bullets that instantly reach their destination (they use raycasts and are usually used for rifles, pistols etc). Projectile weapons instantiate bullets that travel through the world with a certain speed (Usually used for rocket launchers, grenades etc...). Melee weapons are used for knives, swords etc...

> Projectile: Attach your projectile here. It must have a Bullet.cs script attached to it.
>
> ProjectileUsesGravity: When enabled, your projectiles will use gravity and describe a parabola. It is important to set a rigidbody to the projectile attached above. You can easily adjust the parabola with the rigidbody and projectile speed settings
>
> Speed: Speed of the instantiated projectile.
>
> HurtsPlayer: When enabled, projectiles will also hurt the player.
>
> ExplosionOnHit: When enabled, the projectile will cause an explosion on hit. Check new features unlocked under the “Visuals” tab.

FireRate: Time between each shot.

BulletRange: If shooting projectiles, it means the lifetime of the instantiated object, In case it is a hitscan / melee weapon, it means the maximum distance to detect a hit.

BulletsPerFire: Amount of bullets shot per tap.

> TimeBetweenShots: Time between each bullet. Keep this at 0 if you want to make a shotgun, or change the value in order to make a burst weapon.

AmmoCostPerFire: How much ammo you lose per each fire (& per each fire point)

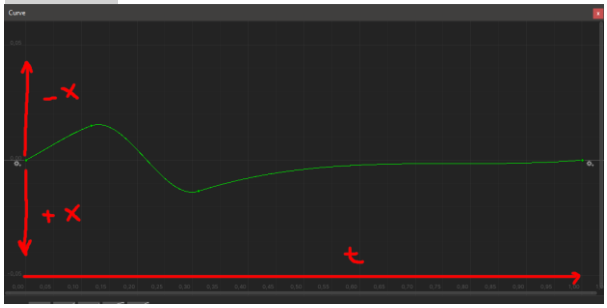
ApplyBulletSpread: If true, bullets will randomly spread.

SpreadAmount: Amount of randomness applied to the spread of the bullets.

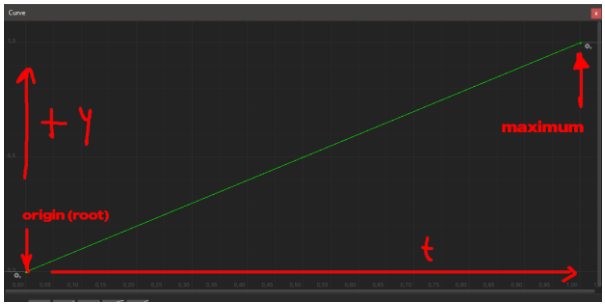
AimSpreadAmount: Amount of randomness applied to the spread of the bullets while aiming.

ApplyRecoil: Apply recoil on shooting

RecoilX: Recoil on the X axis.



RecoilY: Recoil on the Y axis.



These two movements are mixed to create the pattern.

XRecoilAmount: Distance of the recoil on the X axis.

YRecoilAmount: Distance of the recoil on the Y axis.

RecoilRelaxSpeed: When the user is not shooting, speed to go from the current recoil point to the origin point of the recoil.

ApplyDifferentRecoilOnAiming: Enable this to set different recoil amounts while aiming.

XRecoilAmountOnAiming: Distance of the recoil on the X axis while aiming.

YRecoilAmountOnAiming: Distance of the recoil on the Y axis while aiming.

ApplyWeight: If true, the weight of the weapon will affect the speed of the player.

WeightMultiplier: Being 1 the lightest, and 2 the heaviest.

PenetrationAmount: Amount of bullet penetration (wallbang)

PenetrationDamageReduction: Damage reduction multiplier. % / 100; 0.8f means 80% etc..

STATS (STATISTICS)

InfiniteBullets: If true, you will never run out of bullets. Bullets and Magazine UI won´t be displayed for infinite bullets weapons.

MagazineSize: Magazine capacity. Amount of bullets per magazine.

Limited Magazines: If true, your weapon will have a certain amount of bullets, so you can run out of ammo.

TotalMagazines: Amount of initial magazines. Amount of initial bullets = magazineSize * TotalMagazines.

DamagePerBullet: Amount of damage dealt per each body shot.

CriticalDamageMultiplier: If a headshot is landed, base damage (DamageperBullet) will be multiplied by this.

ApplyDamageReductionBasedOnDistance: if true, damage will decrease or increase depending on how far you are from the target

minimumDistanceToApplyDamageReduction: Damage reduction will be applied for distances larger than this

damageReductionMultiplier: Adjust the damage reduction amount

ReloadStyle: Place holder for future updates.

ReloadTime: Time required to fully reload your weapon.

AllowAim: Enable this if you want the players to be able to aim the weapon. Check new options unders spread options.

AimingPosition: Position of the weapon while aiming.

AimingRotation: Rotation of the weapon while aiming.

AimingSpeed: Interpolation speed between states.

AimingFOV: Camera field of view while aiming.

SetMovementSpeedWhileAiming: If true, you will be able to determine a velocity for your player while you are aiming.

MovementSpeedWhileAiming: Speed of the player while aiming.

VISUALS

CamShakeAmount: Amount of camera shake per shot. Recommended = .15f

ApplyFOVEffectOnShooting: If true it will apply a different FOV value when shooting to add an extra layer of detail. FOV will automatically lerp until it reaches its default value.

FOVValue: New FOV Value when shooting.

ShowBulletShells: If enabled, bullet shells will be instantiated on shooting.

BulletShellGraphics: Bullet shells 3D gameObject. This will be instantiated on shooting.

BulletHoleImpacts: Array of visual effects on bullets hitting objects. Attach each layer its own impact effect.

MuzzleFlash: Attach your muzzle flash visual effect here. This will be instantiated at each weapon muzzle position on shooting.

AUDIO

AudioSFX: Array of AudioClips for each specified event, such as firing or reloading. Add your clips individually for each situation.

UI

CrosshairResize: Crosshair size when shooting.

CrosshairPreset: Crosshair preset for the specific weapon

NOTE: You can save and load your own custom presets.

WeaponController.cs

The weapon controller handles everything related to shooting, even the inventory system for your guns.

INVENTORY

InventorySize: Max amount of weapons you can keep with you.

Weapons: Attach your weapon scriptable objects here.

InitialWeapons: An array that includes all your initial weapons.

REFERENCES

MainCamera: Attach your camera here. (The one with camera component attached on it)

WeaponHolder: Object that contains the weapon when they get instantiated.

VARIABLES

AutoReload: If true, players will have no need to press the reload button when they run out of ammo.

AlternateAiming: If false, hold to aim, and release to stop aiming.

HitLayer: Layers to be targeted as a hittable surface.

RemoveCrosshairOnAiming: If true, crosshair will be disabled while aiming.

CanMelee: If true, the player will be able to perform melee actions (see default inputs / keybindings).

MeleeObject: Melee graphics in the scene. Attach your knife or melee weapon here (must be in the scene).

HolsterMotionObject: Attach HolsterMotion. Path: CowsinsFPSController/Camera/CameraContainer/MainCamera/WeaponCamera/JumpMotion/GunsEffects¡/HolsterMotion

MeleeDuration: How long it takes to perform the whole melee action before being able to perform other actions (related to weapons).

MeleeDelay: Delay added to match the animation with the hit instant.

MeleeAttackDamage: Damage applied to targets. (per hit).

MeleeRange: Melee attack max distance. Any potential target placed between the maximum range and the player will be hit.

MeleeCamShakeAmount: Amount of camera shake applied on performing a melee action.

ReEnableMeleeAfterAction: Melee Interval after finishing the perform. This is added to avoid overlapping actions among melee attacks.

USER INTERFACE (UI)

BulletsUI: Text that shows the current bullets.

MagazineUI: Text that shows the current bullets left.

MagazineUI: Image that displays the current weapon.

InventoryContainer: Canvas Group containing the slots that are procedurally generated by the inventory system.

Crosshair: HUD Crosshair. Must have Crosshair.cs attached.

MagazineUI: Text that shows the current bullets left.

ReloadUI: Text that displays “Reload” when you have no ammo left.

LowAmmoUI: Text that displays “Low Ammo” when ammo is low.

ResizeCrosshair: If true, crosshair will automatically resize on shooting.

EFFECTS

Effects: Array of effects for each layer. These effects are spawned at the hit point when a shot is landed on a target.

EVENTS

Events

CustomShot: Used for weapons with custom shot method. Here, you can attach your scriptable objects and assign the method you want to call on shoot. Please only assign those scriptable objects that use custom shot methods, Otherwise it won´t work or you will run into issues.

You will also find the editor of the script in the same folder.

WeaponIdentification.cs

Makes sure to identify the weapon properly.

Weapon: Scriptable object. The weapon you want to identify your object with.

FirePoint: Points where the bullets come from.

In case you want to shoot bullets from two different points of a weapon (for example dual weapons), make two empty gameObjects and name them properly (such as FirePoint1 and FirePoint2). Afterwards, attach both to the array. Make sure there is no blank spaces in the array.

Bullet.cs

Handles projectiles. Every projectile bullet Prefab MUST have this component ATTACHED to it, as well as a collider and a rigidbody.

03.5 – PICK UP SYSTEM

InteractManager.cs

Allows the player to interact with objects around the environment.

REFERENCES

Mask: Layers to be targeted as interactable. Recommendation: Set it to interactable.

GenericPickeable: Generic pickeable prefab. Attach it here.

INTERACTION

ProgressRequiredToInteract: Minimum hold time to successfully interact.

Set this to 0 for instant interactions.

InteractInterval: Pace of interactions. The lower this value is, the faster the interactions will take place.

CanDrop: Allows player to drop weapons (holstered weapons).

DroppingDistance: Distance from the player where objects are dropped (always in front of the player).

UI

InteratUI: Interaction user interface. Interactions will be displayed here.

InteractText: Inside the Interaction UI, text where the actions are displayed.

Pickeable.cs

Every pickeable object must have this component attached in order to be consider as an object you can pick up. For weapons, WeaponPickeable.cs is used, and for bullets, BulletPickeable.cs.

Rotates: Applies rotation to the pickeable object.

Translates: Applies vertical translation to the pickeable object.

Image: Attach the image (child of the prefab) where the icons and the imagery will be displayed.

Graphics: Transform under which the graphics will be stored at when instantiated.

WeaponPickeable.cs

Weapon: Weapon to pick.

BulletsPickeable.cs

Amount Of Bullets: Amount of bullets that will be added to your total bullets when picked up.

Bullets Icon: Image to display at “Image”. See Pickeable.cs.

Bullets Graphics: Graphics to be displayed under “Graphics”. See Pickeable.cs.

03.6 – ENEMIES

IDamageable.cs

All damageable objects inherit from this interface.

Enemy.cs

This component defines enemies, plus, as said before, inherits from IDamageable. It is the base script for any enemy script. See TrainingTarget.cs.

IDENTITY

Name: Give your enemy a cool name! This will be displayed in the killfeed.

STATS

MaxHealth: Amount of health the enemy will initially have. This is also the max health the enemy can reach.

MaxShield: Amount of shield the enemy will initially have. This is also the max shield the enemy can reach.

UI

ShowUI: If true, it will display UI on the enemy´s head, such as statistics slider or damage pop ups.

healthSlider: Slider displaying current and maximum health.

shieldSlider: Slider displaying current and maximum shield.

DamagePopUp: Attach the provided prefab for simple pop ups.

X Variation: Amount of randomness on the X axis for the already defined damage pop up

HealthColor: Colour for the specific status to be displayed in the slider.

ShieldColor: Colour for the specific status to be displayed in the slider.

RECOMMENDATION: If you want to add more types of enemies, it is handy to make a new script which inherits from this one, since all the basic stuff is already handled, and you only have to worry about making the enemy logic.

TrainingTarget.cs

Inherits from Enemy.cs

TimeToRevive: When dead, time to resurrect again.

03.7 – EFFECTS

All these separate scripts are currently a placeholder. The plan is to mix them all into a single “effects” script for future updates.

Breathing.cs

Breathing effect. Examples of use: Camera, weapon to simulate the breathing motion.

Player: Attach your player with a rigidbody component.

Amplitude: Amount of breathing motion.

Frequency: Speed of the breathing motion. The higher the frequency is, the more it will waggle.

CamShake.cs

Grants an object the possibility of receiving camera shake orders. This can be called using one of the following methods:

```
Shake(float amount, float _power, float _movementAmount, float _rotationAmount);
```

```
ShootShake(float amount);
```

```
ExplosionShake(float distance);
```

CrouchTilt.cs

Tilting motion when crouching, generally used on weapons.

TIltRot: Rotation desired when crouching.

tiltPosOffset: Vector3 desired to add to the original position when crouching.

TiltSpeed: Speed to perform the crouching tilt motion.

HeadBob.cs

More realistic approach for the camera. When moving around, the camera will waggle recreating a realistic but stabilized move for the head.

Player: Attach your player with a rigidbody component.

headBobAmplitude: Amount of head bob motion.

HeadBobFrequency: Speed of the head bob motion.

WeaponBob.cs

More realistic approach for the weapon. When moving around, the weapon will waggle recreating a realistic but stabilized movement for the arm.

bobMethod: Select the bob method you want to use. Original method was brought in the first FPS Engine version, while Detailed is the last option added and it brings you more customization possibilities.

Player: Attach your player with a PlayerMovement component.

Speed: Speed of the weapon bob motion.

Distance: Amount of the weapon bob motion.

rotationMultiplier: Allows to modify each axis of the rotation (eulerAngles)

TranslationSpeed: Lerp velocity for the translation effect.

RotationSpeed: Lerp velocity for the rotation effect.

movementLimit: maximum amount of sway allowed.

bobLimit: Maximum amount of bob allowed.

AimingMultiplier: Multiplier when aiming.

WeaponSway.cs

More realistic approach for the weapon. When moving around, the weapon will tilt recreating a realistic movement for the arm. generally used on a weapon holder, bult also usable on elements like UI.

SwayMethod: Technique used for the sway effect. Simple method is the original method, while PivotBased was included in the 0.9.6 update and provides a more realistic effect, based on a pivot object.

POSITION

Amount: Amount of motion, dependant on players´inputs.

MaxAmount: Amount is clamped, being the maximum amount value = MaxAmount and the minimum amount value = - MaxAmount.

SmoothAmount: Smoothness of the motion.

ROTATION

TiltAmount: Amount of motion, dependant on players´inputs.

MaxTiltAmount: Amount is clamped, being the maximum amount value MaxAmount and the minimum amount value = - MaxAmount.

SmoothTiltAmount: Smoothness of the motion.

pivot: Transform that acts as the pivot. The weapon sway will be calculated around this object. It is usually placed approximately in the hand holding the weapon since it's the point where the weapon rotates towards where you are aiming.

swaySpeed: Lerp speed for the sway motion.

swayMovementAmount: Amount of translation allowed.

swayRotationAmount: Around the pivot, amount of rotation in the x and y axis.

swayTiltAmount: Around the pivot, amount of rotation in the z axis.

JumpMotion.cs

Motion on jumping and landing, generally on the Y axis for weapons but also for cameras and UI elements.

Player: Attach your player with a PlayerMovement component.

JumpMotion: This animation curves defines the motion on jumping. The object which has this component attached will evaluate the curve on each point and follow the path you created.

GroundedMotion: This animation curves defines the motion on landing. The object which has this component attached will evaluate the curve on each point and follow the path you created.

Distance: Amount of movement.

RotationAmount: Amount of rotation.

EvaluationSpeed: Speed of the motion.

CameraTilt.cs

Motion on moving around. Camera will tilt depending on your inputs.

Player: Attach your player with a PlayerMovement component.

TiltSpeed: Speed of the tilt motion.

TiltAmount: Amount of the tilt motion.

BulletUIEffect.cs

Apply a subtle effect on Bullet UI when shooting.

Size: Desired size for the text when shooting.

lerpSpeed: transition speed of the effect.

UI Sway.cs

Apply a subtle effect on any RectTransform object. Object reacts to mouse position.

Amount: Distance to travel (relative to the mouse position).

Speed: Speed for the object to reach the appropriate position. The lower this value is, the “lazier” the effect will be.

ProceduralShot.cs

Handles a simple procedural shot effect, essentially for basic shooting animations without having to animate or place holders.

Weapon: Attach your player with a WeaponController component.

ProceduralShot_SO.cs

PlaySpeed: How fast your animation will be played.

Translation

- X Translation: AnimationCurve that handles movement for the X Axis.
- Y Translation: AnimationCurve that handles movement for the Y Axis.
- Z Translation: AnimationCurve that handles movement for the Z Axis.
- Translation Distance: Vector3 that acts as a multiplier for each of the axis.
- Aiming Translation Multiplier: This value will be multiplied by Translation distance to get the appropriate distance when aiming or not aiming.

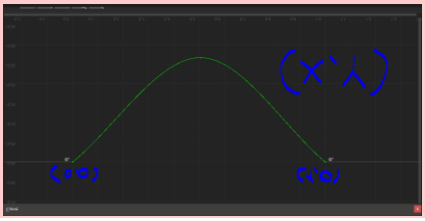
Rotation

- X Rotation: AnimationCurve that handles rotation for the X Axis.
- Y Rotation: AnimationCurve that handles rotation for the Y Axis.
- Z Rotation: AnimationCurve that handles rotation for the Z Axis.
- Rotation Distance: Vector3 that acts as a multiplier for each of the axis.
- Aiming Rotation Multiplier: This value will be multiplied by rotation distance to get the appropriate distance when aiming or not aiming.

IMPORTANT: ANIMATION CURVES USAGE.

In order for the system to work properly, each of your animation curves on each of your axis MUST meet the following requirements:

1. The beginning and end points Y values must be at 0.
2. The beginning point X value must be at 0.
3. The end point X value must be at 1.
4. Make sure BeginningPoint = (0,0) and EndPoint =(1,0)
5. You can edit and add points within beginning and end points as you like



03.8 – UI

UIController.cs

Manage UI actions such as displaying a killfeed panel

DisplayEvents: An object showing death events will be displayed on kill.

KillfeedContainer: Contains kill events and messages.

KilfeedObject: Object to spawn when killing an enemy. You can attach your own UI element or use the provided.

Crosshair.cs

Procedural crosshair creation system.

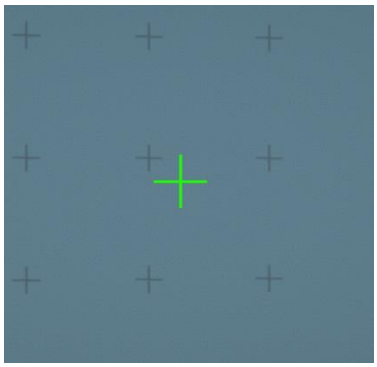
Player: Attach your player with a PlayerMovement component.

Size: Length of the crosshair.

Width: Thickness of the crosshair.

EnemySpottedWidth: Thickness of the crosshair when aiming towards an enemy.

DefaultSpread: Spread of the crosshair elements.



Spread = 0 Spread = 50

WalkSpread: Spread of the crosshair elements while walking.

RunSpread: Spread of the crosshair elements while running.

CrouchSpread: Spread of the crosshair elements while crouching.

JumpSpread: Spread of the crosshair elements while jumping.

DefaultColor: Normal color of the crosshair.

EnemySpottedColor: Color of the crosshair when spotted an enemy.

ResizeSpeed: Speed of the crosshair to change spreads.

Hitmarker: If true, show hitmarkers on hitting enemies.

HitmarkerObj: Attach the provided hitmarker object or your own.

CrosshairShape.cs

Parts: Enable or disable crosshair elements. Top, Bottom, Left & Right.

Hitmarker.cs

CrosshairSoundEffect: Attach your audio clip. This will be played when hitting an enemy.

UISlot.cs

NullWeapon: Sprite to display an empty slot.

03.9.1 – EXTRA(UTILITIES)

DestroyMe.cs

Can be added on objects to destroy them over time since they are enabled.

TimeToDestroy: Lifetime of your object.

IgnoreCollision.cs

Any object with this component will ignore collisions with the player.

LookAt.cs

Any object with this component will rotate towards the player.

03.9.2 – EXTRA(MAIN)

Checkpoint.cs

Text: Attach the text where you want the distance from the player to be displayed.

MeasureType: Select a measure unity among the provided.

Decimals: Amount of decimals to display.

UpdatePeriod: How fast you want the text to be refreshed / updated.

DamageMultiplier.cs

damageMultiplierAddition: Damage multiplier added. All the damage is multiplied by a damage multiplier, which is 1 at start. If you add 1.5f to the damage multiplier, then it will be 2.5f. All the damage will be multiplied by 2.5.

DoorInteraction.cs

DoorContainer: The parent of the object which has this script attached.

OpenedDoorRotation: Amount of rotation of the door when opened.

Speed: Rotation speed.

GetGameInformation.cs

showFps: when active, it will display fps information.

showMinimumAndMaximumFPS: When active, it will display minimum fps as well as maximum reached fps.

FPSRefreshRate: How fast FPS info will be refreshed / displayed.

FPS Object: Object that will display the fps counter.

min FPS Object: Object that will display the minimum fps counter.

max FPS Object: Object that will display the maximum fps counter.

AppropriateValueColor: Color to tint the text when the fps value is appropriate.

IntermediateValueColor: Color to tint the text when the fps value is not good neither bad.

badValueColor: Color to tint the text when the fps value is bad.

HealMultiplier.cs

healMultiplierAddition: Heal multiplier added. All the heal is multiplied by a heal multiplier, which is 1 at start. If you add 1.5f to the heal multiplier, then it will be 2.5f. All the heal will be multiplied by 2.5.

Healthpack.cs

HealAmount: Amount of health to be restored

Interactable.cs

InteractText: Text that will be displayed on the Interaction UIAmount of health to be restored

PointCapture.cs

CaptureSpeed: how fast the point will be captured

LoseProgressifNotCapturing: If true, progress will gradually be lost when player leaves the point

LosingProgressCaptureSpeed: Speed of progress loss

PowerUp.cs

Reappears: Wether the powerup should reappear after use or not.

reappearTime: Time to reappear when used.

Timer: Image that displays the reappear progress.

Destructible.cs

maxHealth: initial health of the object.

LootInside: Instantiate something cool when the object is destroyed, such as coins, a weapon, or any kind of loot, whatever you want! If this is empty, no reward will be instantiated

destroyedSFX: Audio clip to play when destroyed

ExplosiveBarrel.cs

explosionRadius: Max range of the explosion

explosionForce: Force to apply on rigidbodies on exploding.

damage: Damage dealt on explosion to any Damageable object within the radius."

"NOTE:Damage will be scaled depending on how far the object is from the center of the explosion

destroyedObject: Instantiate this object when the barrel explodes

explosionVFX: Instantiate this visual effect when the barrel explodes

Crate.cs

destroyedObject: Instantiate this object when the barrel explodes

Interactable.cs

InteractText: Text that will be displayed on the Interaction UI

DisplayKey.cs

Displays the appropriate key/button code depending on the current device detected.

ShowAndHide.cs

PauseMenu.cs

menu: Canvas group that contains all the pause menu UI elements. This will be fade in when paused and fade out when un-paused.

fadeSpeed: How fast the menu fades in and out.

03.10 MANAGERS

InputManager.cs

Handles user inputs. You may find the provided prefab “InputManager” inside the resources folder. This prefab gets automatically instantiated every time you play.

WARNING!!!

Sensitivity, ControllerSensitivityX, ControllerSensitivityY, AimingSensitivityMultiplier, AlternateCrouch and AlternateSprint settings have been moved to PlayerMovement.cs for 0.9.5 versions and above. Sensitivity has been split into SensitivityX and Sensitivity Y, and AlternateAiming has been moved to WeaponController. For users using FPS Engine v0.9 and below, you may find the following variables here.

FPS ENGINE v.0.9 and BELOW:

SENSITIVITY: Camera look sensitivity. The higher the value is, the faster your camera will move.

ControllerSensitivityX: Camera look sensitivity when using controller for the x axis (horizontal). The higher the value is, the faster your camera will move.

ControllerSensitivityY: Camera look sensitivity when using controller for the y axis (vertical). The higher the value is, the faster your camera will move.

AimingSensitivityMultiplier: Camera look sensitivity multiplier when aiming.

AlternateCrouch: If false, hold to crouch, and release to uncrouch.

AlternateSprint: If false, hold to sprint, and release to stop sprinting.

AlternateAiming: If false, hold to aim, and release to stop aiming.

DeviceDetection.cs

Handles different devices. It can detect whether the user is currently using Keyboard&Mouse or a gamepad (controller)

InputMode	mode	Returns the current InputMode used by the user. Input Mode is an enumerator that handles the possible devices. In this case, it handles Keyboard, and Controller (InputMode.Keyboard, InputMode.Controller)

CowsinsUtilitiesManager.cs

Handles various utilities.

SoundManager.cs

Handles sound management.

04.ASSET USAGE & HOW TO USE

04.1 HOW TO CREATE A PLAYER CONTROLLER

The asset provides a bunch of already made and set up player controller template. You can find them under the following path:

Assets/Cowsins/Prefabs/PlayerControllers

If you would like to make a new controller, just drag and drop the “BlackPlayerControllerTemplate” into your scene and start modifying it.

If you prefer though, you can always select one of the premade templates.

04.2 HOW TO ADD NEW WEAPONS

YOU WILL NEED:

- Custom arm and weapon models if desired.

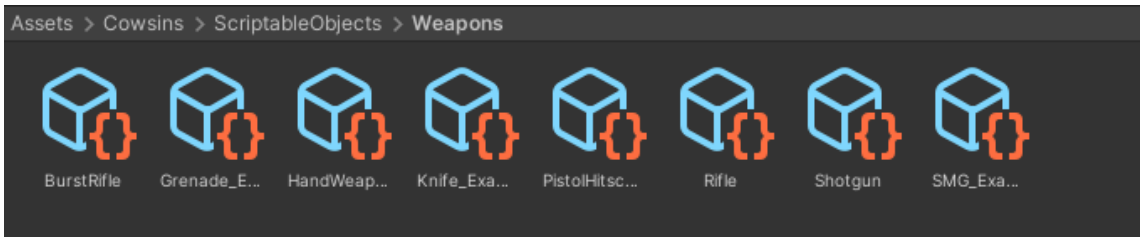
- Custom VFX if desired.

- Custom SFX if desired.

Note that you can use the assets provided by FPS Engine if you want!

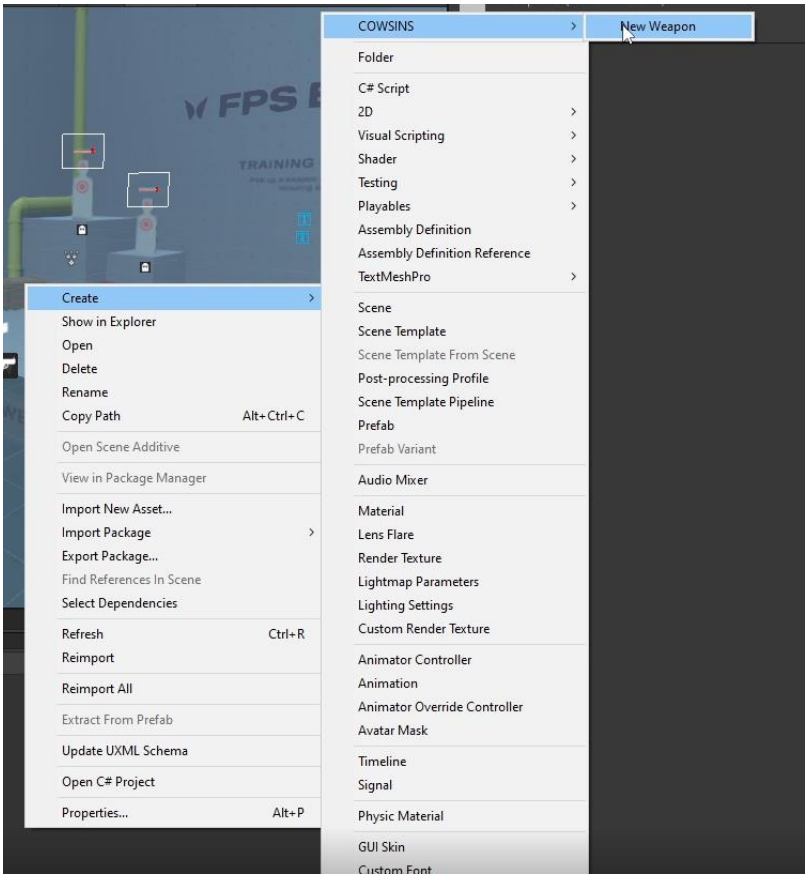
HOW TO PROCEED:

First things first, we must create a new weapon settings asset, which is stored as scriptable objects in the following path. You can store them anywhere actually, but it´s good practice to keep them all in the same folder.



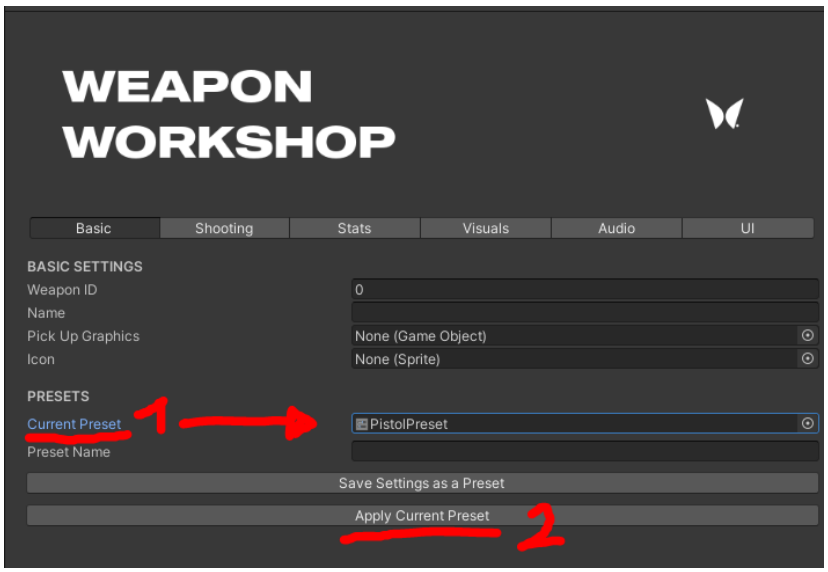
This is what is called the weapon workshop. We can create a new one and modify all the parameters from zero, but you can also use one of the provided presets as well as just duplicate any of the provided weapon settings, you choose how to do it.

For this guide, we will create a new one and apply a preset. To do that, right click on the assets tab, go on Create/COWSINS/New weapon

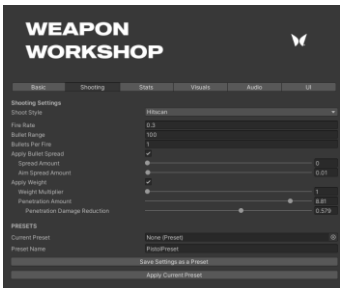


We can give this whatever name you like, for example: MyNewWeapon.

Now, it is time to apply the preset, for that select the desired preset in "CurrentPreset" and then click on "Apply Preset"

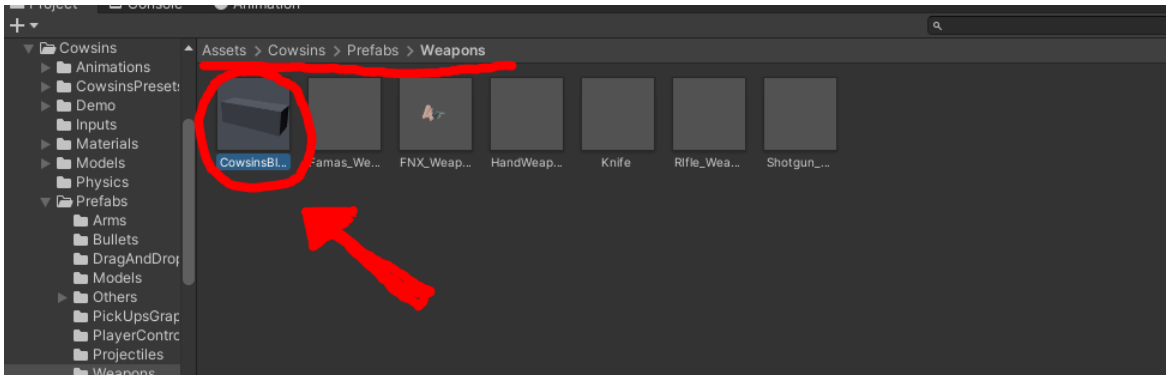


All the weapon settings will be overridden, you can now adjust all the parameters as you like!

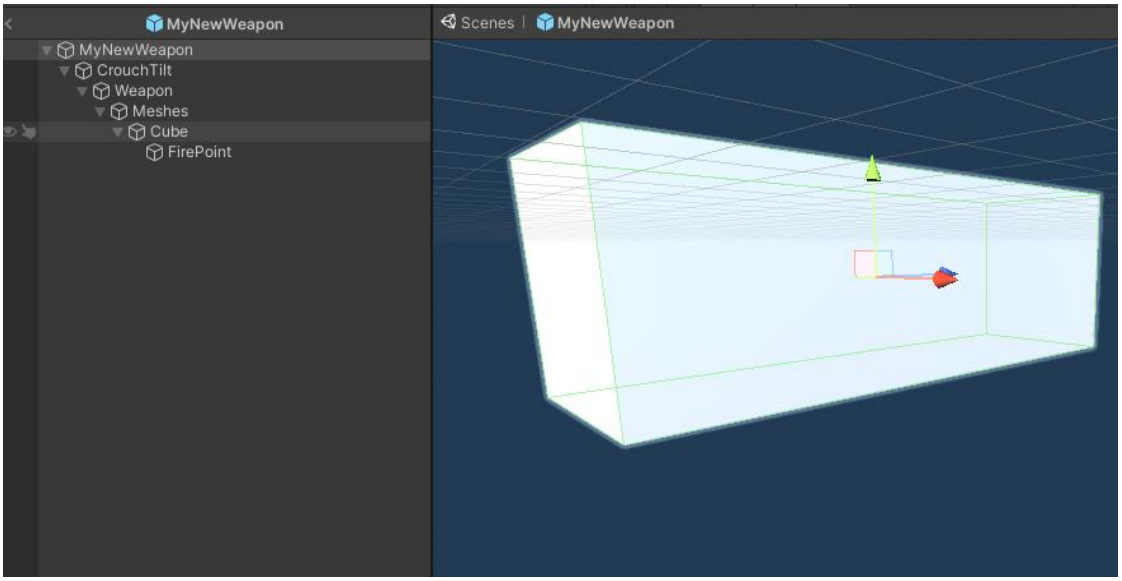


As mentioned earlier, you can just manually modify each value instead of applying a preset.

Once we have the weapon settings created, we need a weapon prefab. We will search the CowsinsBlankWeaponTemplate which is located under the following path

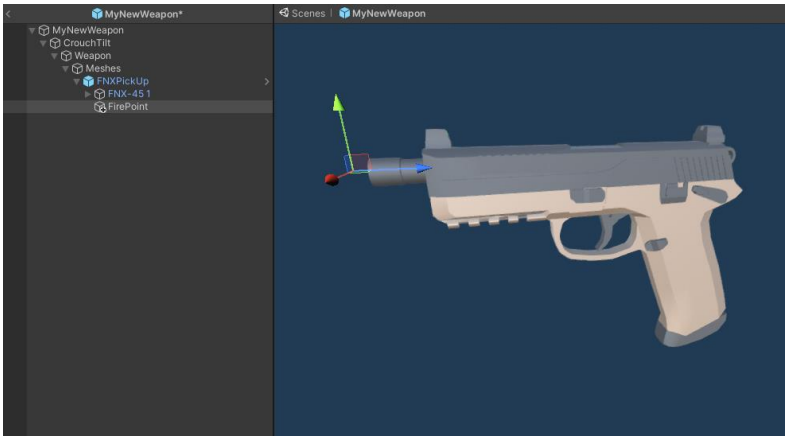


We want to duplicate this object (Ctrl+D) and give it a name, then open it.

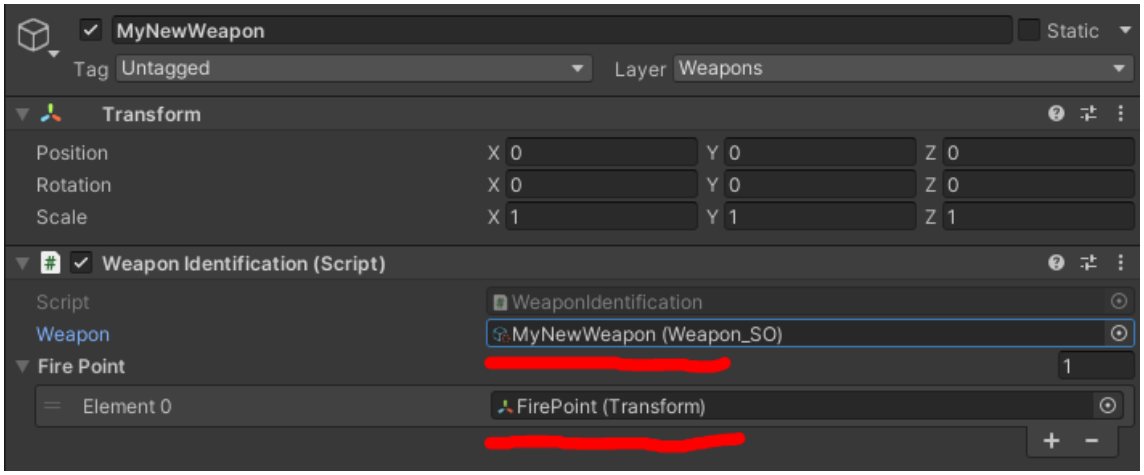


Here, we will import our model under “Meshes”. Note that the cube is approximately the size and location of the weapon we want to add. When you are happy with the location of the model, delete the cube.

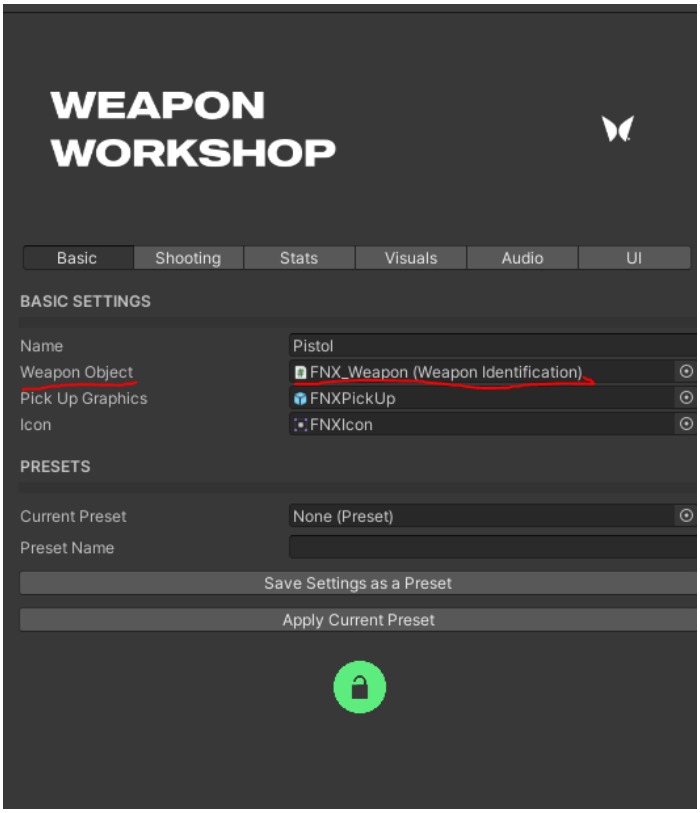
Now create a new empty object as a child of the imported model and call it FirePoint. Note that you can create as many fire points as you want, so you can make two handed weapons etc. You want to locate the fire point at the tip of the barrel.



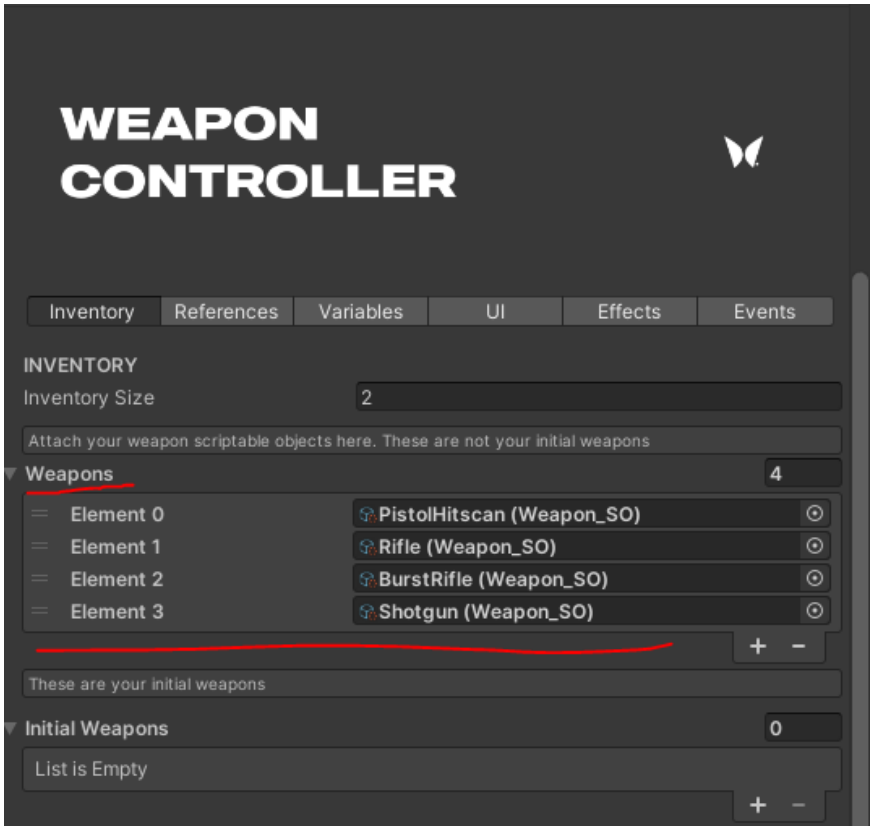
Now go to the root of the prefab and attach your fire points and the weapon settings we created before.



Now, go to your scriptable object and attach this prefab into weaponObject, under the Basic tab.



Once this is ready, go to your WeaponController in Player, and assign this scriptable object in the Weapons array.



Just like this, you will have a functional weapon. To learn how to assign animations, check 03.D

04.3 BEST PRACTICES WHEN ADDING WEAPONS

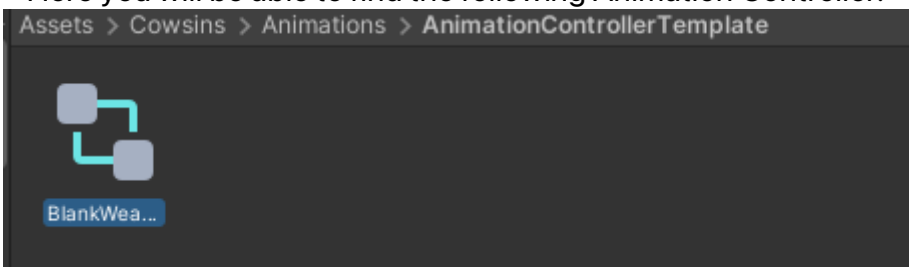
It is highly suggested to use the already provided template for making weapons.

It is recommendable to follow the same structure as the example weapons provided with the asset. Here is an example:

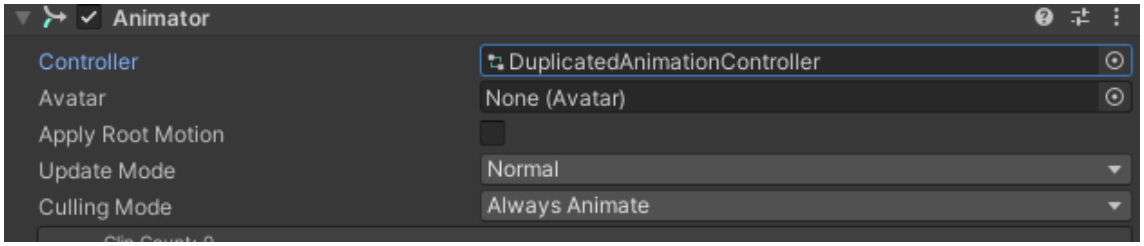
04.4 ANIMATING USING FPS ENGINE

First thing you should notice when animating a new weapon is that there is an animator attached to the root of the prefab. As you can see there is no Animation Controller Attached to it. There is an Animation Controller template stored in the following path: Assets/Cowsins/Animations/AnimationControllerTemplate.

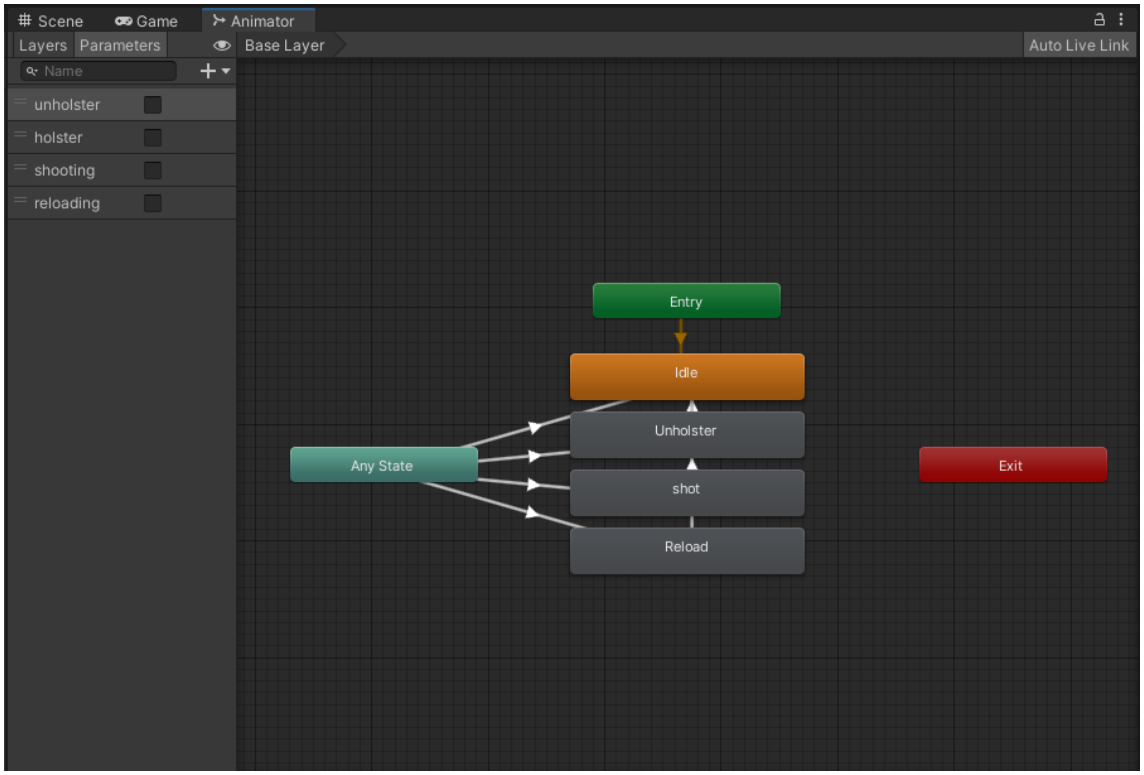
Here you will be able to find the following Animation Controller:



You can duplicate this and attach it to your “Controller” on the Animator component (attached to the root of the weapon you are currently adding).



You can open this animation controller and take a look. This will pop up:

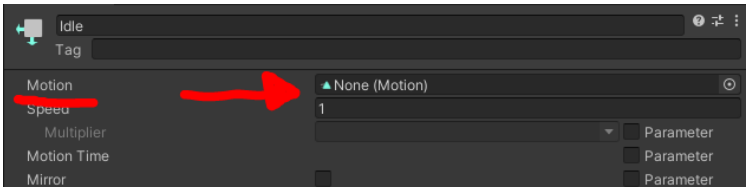


As you may notice, all these parameters and structure are already set-up.

All you have to do is to create an animation for each state (Idle, Unholster, Shoot, Reload) and then attach them to each animation State.

For example: Let´s say we have just made an idle animation. Click on the Idle Animation State.

Attach your animation in Motion. Repeat the process with the remaining animation States.



NOTE: Animations´ length like Reload depend on the statistics you set for your weapon (Reload time). When it comes to shoot, it does not matter as long as you use the blank Template, since the animation will be restarted after each shot. Regarding idle, it should be the only one marked as “loop”.

CUSTOM ANIMATION STATES

You can also make custom animation states if you need to, but take into account that these custom states are not provided by the asset so you will have to set them up. Next, you can find a guide to do this the best way possible using FPS Engine.

1. Open the animation controller for the weapon you desire to add a new Animation State.

2. Go to the parameters tab for this animation Controller and click on "+". Name a new parameter and remember its name, I called this one myCoolNewBoolean.

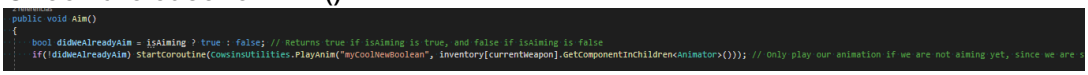
3. Make a new state and add a motion (animation clip)

4. Add the required animation transitions

5. Look for the suitable method where you want to introduce the state by code. Then write this:

```
StartCoroutine(CowsinsUtilities.PlayAnim("myCoolNewBoolean", inven-
tory[currentWeapon].GetComponentInChildren<Animator>()));
```

Note that you can ask for support in case you can´t find the appropriate function. Also note that some functions do run on Update, which means that you will likely want to check if we should run the animation or not. Check the case for Aim()

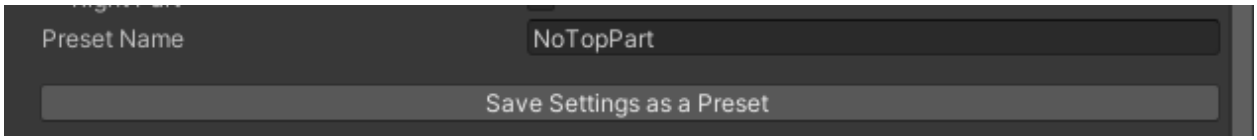


04.5 HOW TO SAVE PRESETS

Saving and loading presets is an important feature that FPS ENGINE handles for you, to make diverse processes easier and faster.

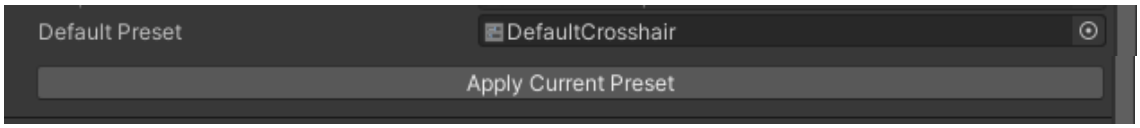
To save a new preset, you first have to customize the script as much as you want.

Once you are done, Give the preset a name in “PresetName” and click on “Save Settings as a Preset”. If you have never made a preset before or deleted the provided ones, a new folder called “CowsinsPresets” will appear in your assets folder. If this path already exists, your preset will be saved there already.



04.6 HOW TO LOAD PRESETS

To load a preset, the process is extremely easy and fast. Just select your desired preset in “Default Preset” (should be stored inside CowsinsPresets) and just click on “Apply Current Preset”.



04.7 HOW TO ADD A NEW CHARACTER

To add a new character, it is suggested to use an already provided template, which you will be able to find under the following path:

Assets/Cowsins/Prefabs/Players

Select your preferred option among the provided, drag it and drop it to your scene.

NOTE: ALL THE WALKABLE AND HITTABLE LAYERS MUST HAVE AN APPROPRIATE LAYER ATTACHED TO IT, OTHERWISE THE GAME WILL NOT WORK

04.8 CAN I IMPORT THE ASSET INTO A HDRP OR URP PROJECT?

Yes, but you must take a few things into account. The asset is built using the standalone render pipeline, so importing it to a basic 3D scene won´t cause any issues. In case your project requires to use HDRP or URP, you might need to update all your materials.

CONVERT ALL YOUR MATERIALS INTO HDRP MATERIALS:

Edit > Render Pipeline >Upgrade Project Materials to High-Definition Materials: Converts all compatible Materials in your Project to HDRP Materials.

Particle materials might need to be assigned manually.

Everything else that the asset offers works perfectly with HDRP and URP (Systems and coding, Inspectors, models, animations etc...)

04.9 DEFAULT KEYBINDINGS

You can check the default bindings for the keys here. Keep in mind that these can be modified in the Input Manager that Unity provides. Note that it does not use the new Input System, but the old one.

NOTE: KEYBINDINGS ON CONTROLLERS WHICH ARE NOT XBOX DEVICES MAY BE DIFFERENT.

KEYBINDING: KEYBOARD & MOUSE / CONTROLLER

MOVE AROUND: WASD / L

LOOK AROUND: MOUSE / R

SPRINT: LEFT SHIFT / L3

JUMP: SPACE / A

CROUCH / SLIDE: LEFT CTRL / B

RELOAD: R / X

INTERACT: E / Y

CHANGE WEAPON: MOUSE WHEEL / LB / RB

AIM: RIGHT CLICK / LT

SHOOT: LEFT CLICK / RT

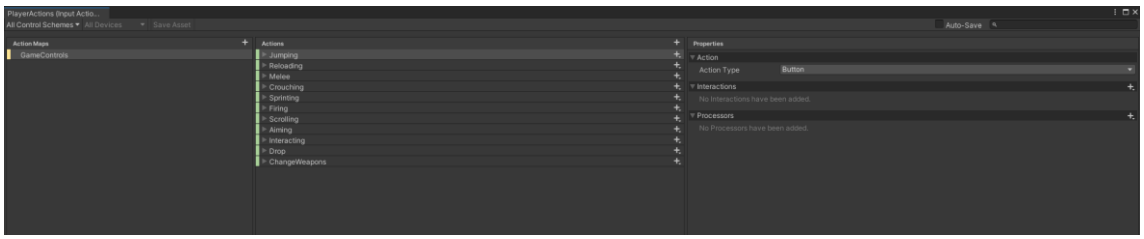
MELEE: F / R3

04.10 CHANGING KEY BINDINGS

NEW INPUT SYSTEM

In case you are using the new input system, changing your key bindings is quite straightforward. Search “PlayerActions” and double click it to open it up.

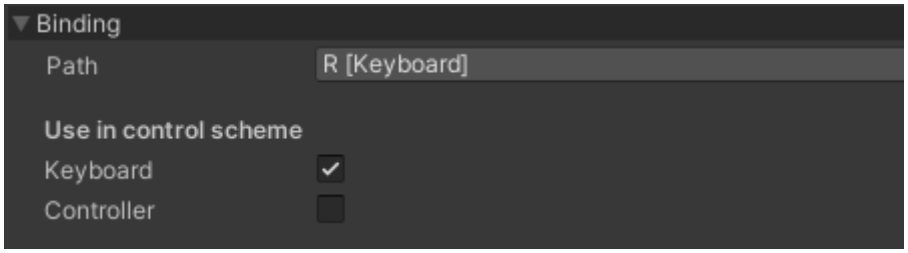
This window will pop up:



Please note that All Control Schemes is selected. Because of that, we will be able to see and edit both inputs for keyboard & mouse and controller.

Select any of the following Actions, take Reloading, for instance. Access its bindings.

As you can see, we can find inputs for Keyboard and Gamepad. In order to edit them select them, and under Binding you will find the option “Path”. Here, select your desired input.

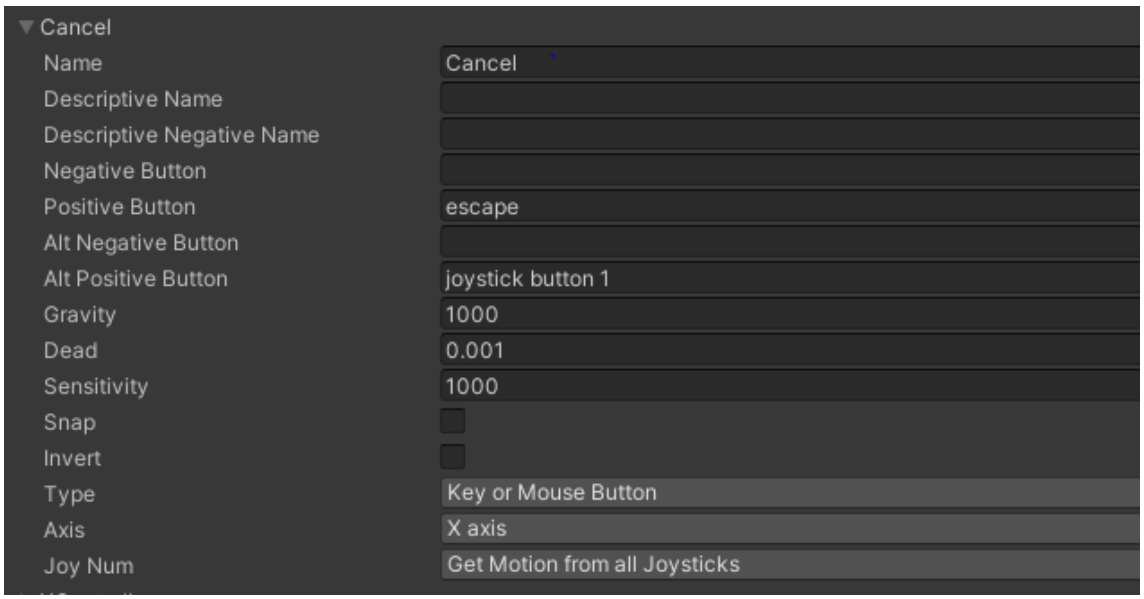


OLD INPUT SYSTEM

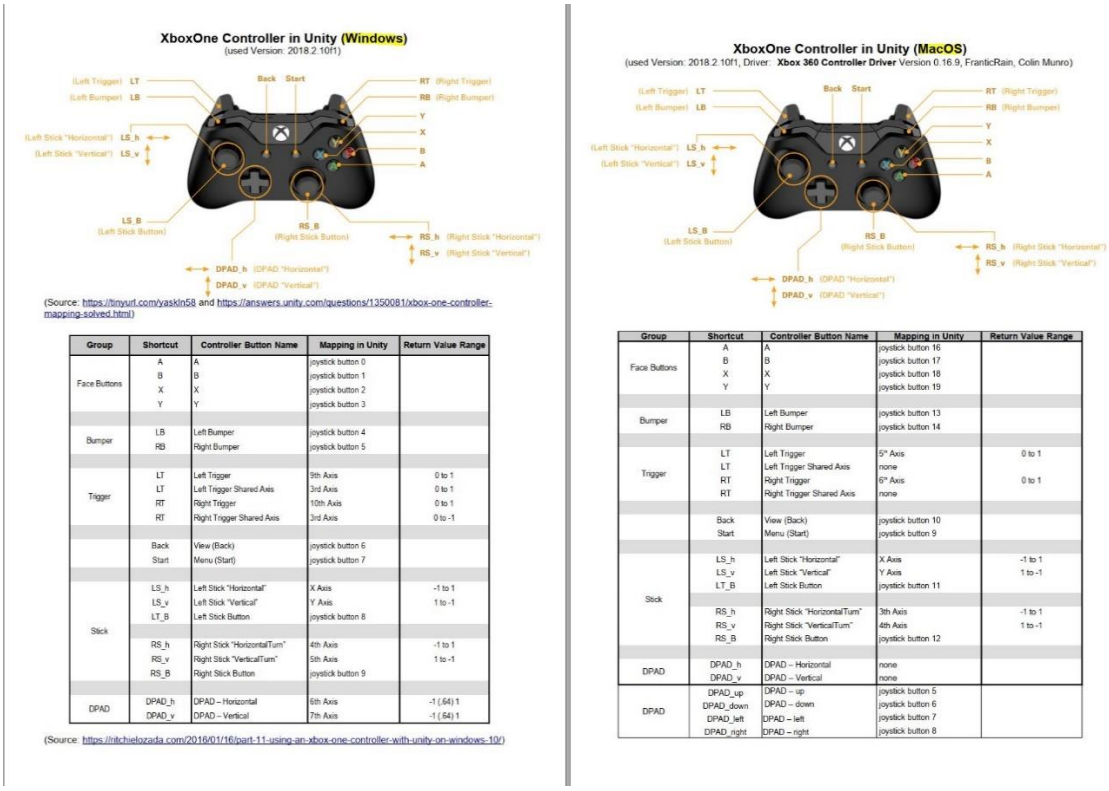
To change the key bindings using the old input system, follow the next path:

Edit/Project Settings/InputManager .... Select the input to modify and change it as you please.

Here, positive button stands for the keyboard input. Alt positive button stands for the controller inputs.



The following image illustrates names for XBOX controller inputs on both MacOS and Windows.



XboxOne Controller in Unity (Windows)
(used Version: 2018.2.10f1)

(Source: https://tinyurl.com/yaskln58 and https://answers.unity.com/questions/1350081/xbox-one-controller-mapping-solved.html)

Group	Shortcut	Controller Button Name	Mapping in Unity	Return Value Range
Face Buttons	A	A	joystick button 0	
	B	B	joystick button 1	
	X	X	joystick button 2	
	Y	Y	joystick button 3	
Bumper	LB	Left Bumper	joystick button 4	
	RB	Right Bumper	joystick button 5	
Trigger	LT	Left Trigger	9th Axis	0 to 1
	LT	Left Trigger Shared Axis	3rd Axis	0 to 1
	RT	Right Trigger	10th Axis	0 to 1
	RT	Right Trigger Shared Axis	3rd Axis	0 to -1
	Back	View (Back)	joystick button 6	
	Start	Menu (Start)	joystick button 7	
Stick	LS_h	Left Stick "Horizontal"	X Axis	-1 to 1
	LS_v	Left Stick "Vertical"	Y Axis	1 to -1
	LT_B	Left Stick Button	joystick button 8	
	RS_h	Right Stick "HorizontalTurn"	4th Axis	-1 to 1
	RS_v	Right Stick "VerticalTurn"	5th Axis	1 to -1
	RS_B	Right Stick Button	joystick button 9	
DPAD	DPAD_h	DPAD – Horizontal	6th Axis	-1 (.64) 1
	DPAD_v	DPAD – Vertical	7th Axis	-1 (.64) 1

(Source: https://ritchielozada.com/2016/01/16/part-11-using-an-xbox-one-controller-with-unity-on-windows-10/)

XboxOne Controller in Unity (MacOS)
(used Version: 2018.2.10f1, Driver: **Xbox 360 Controller Driver** Version 0.16.9, FranticRain, Colin Munro)

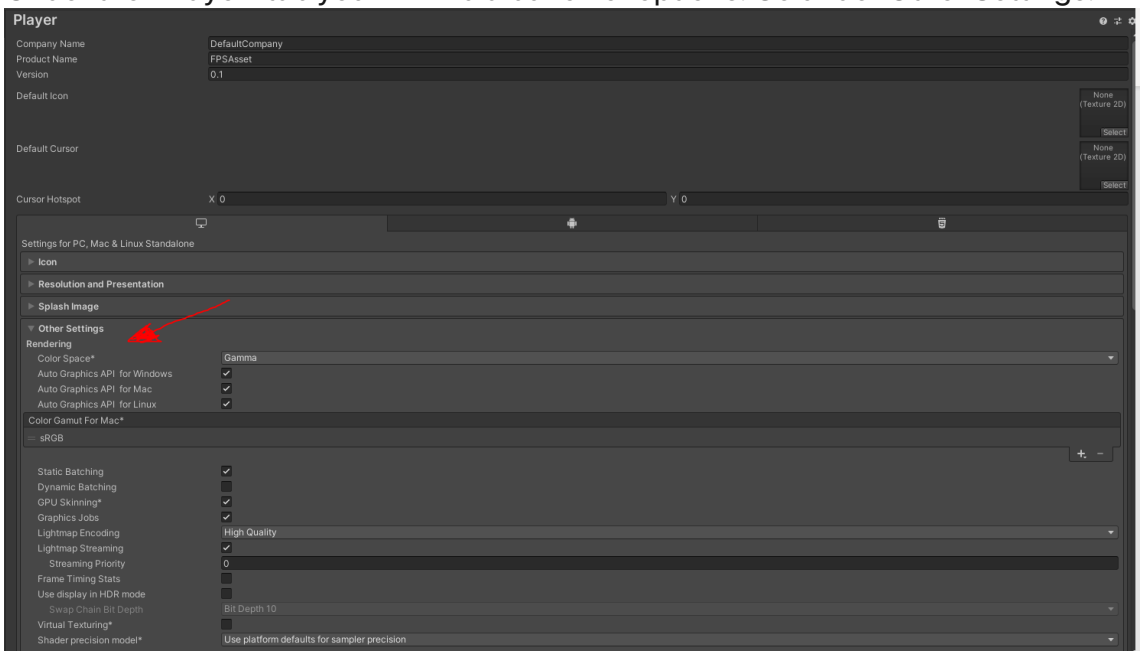
Group	Shortcut	Controller Button Name	Mapping in Unity	Return Value Range
Face Buttons	A	A	joystick button 16	
	B	B	joystick button 17	
	X	X	joystick button 18	
	Y	Y	joystick button 19	
Bumper	LB	Left Bumper	joystick button 13	
	RB	Right Bumper	joystick button 14	
Trigger	LT	Left Trigger	5th Axis	0 to 1
	LT	Left Trigger Shared Axis	none	
	RT	Right Trigger	6th Axis	0 to 1
	RT	Right Trigger Shared Axis	none	
	Back	View (Back)	joystick button 10	
	Start	Menu (Start)	joystick button 9	
Stick	LS_h	Left Stick "Horizontal"	X Axis	-1 to 1
	LS_v	Left Stick "Vertical"	Y Axis	1 to -1
	LT_B	Left Stick Button	joystick button 11	
	RS_h	Right Stick "HorizontalTurn"	3th Axis	-1 to 1
	RS_v	Right Stick "VerticalTurn"	4th Axis	1 to -1
	RS_B	Right Stick Button	joystick button 12	
DPAD	DPAD_h	DPAD – Horizontal	none	
	DPAD_v	DPAD – Vertical	none	
DPAD	DPAD_up	DPAD – up	joystick button 5	
	DPAD_down	DPAD – down	joystick button 6	
	DPAD_left	DPAD – left	joystick button 7	
	DPAD_right	DPAD – right	joystick button 8	

04.11 CHANGING SUPPORTED INPUT SYSTEM

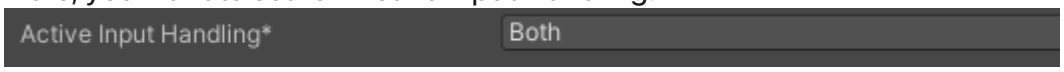
After installing the asset, you might want to change the current input system.

To do that, click on Edit (top bar)– Project Settings – Player

Under the "Player" tab you will find a bunch of options. Go under Other Settings.



Here, you want to search Active Input Handling.



You can select the following three options:

- Input Manager (Old): Uses the old input system.
- Input System Package (New): Uses the new input system. (NOT RECOMMENDED)
- Both – RECOMMENDED BY DEFAULT: Unity prioritizes new input system.

04.12 ADD NEW COMPASS ICONS / ELEMENTS / QUESTS

To add a new compass icon like shown in the image below, you must have a

Compass in your UI. You can use the provided in the templates.

By default, the asset provides examples for the checkpoint and enemies. However, you can add your own icons. To achieve that, attach CompassElement.cs on a gameObject and attach then your desired icon.

You can enable AddOnStart in case you want it to be visible as soon as you play, or call it from your custom code using the public functions provided by the asset:

Add(); `> Adds a new compass icon. It makes it visible.`

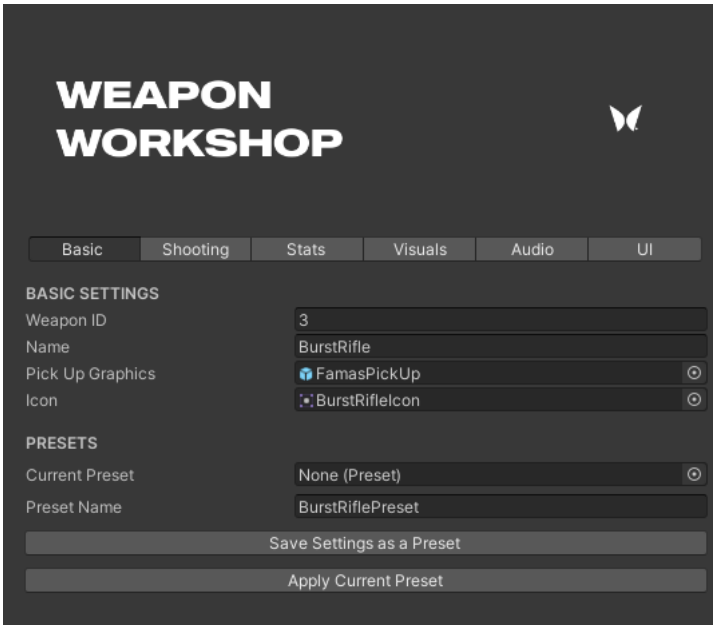
Remove(); `> Removes the current compass icon.`

04.13 CUSTOMIZE MY WEAPONS STATISTCS

To edit the weapon statistics, you must access the “Weapon Workshop”. All the weapon variables are stored inside Scriptable Objects. Follow the path:

Cowsins/ScriptableObjects/Weapons

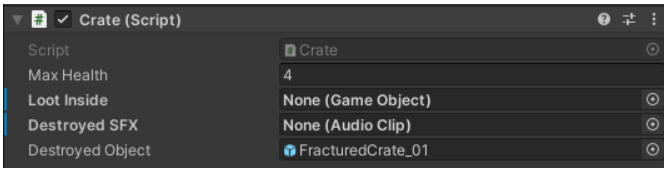
Select the weapon you would like to customize and you will see this new editor:



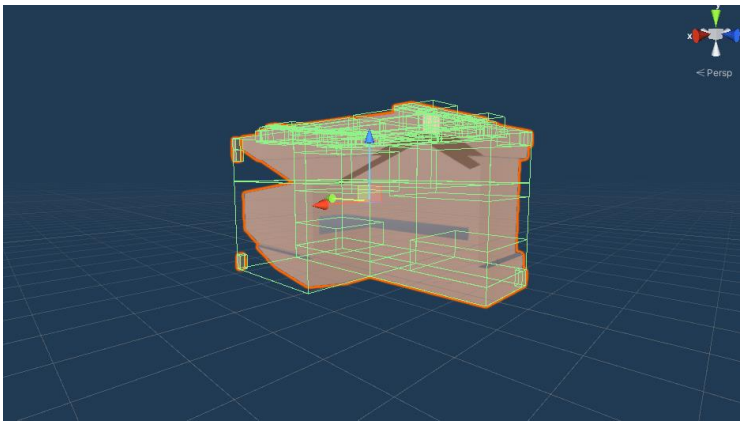
Here you have a few tabs you can access to such as “Basic, Shooting, Stats, Visuals etc.” In case you any help understanding what these variables mean, visit this.

04.14 ADD BREAKABLE (DESTRUCTIBLE) OBJECTS

FPS ENGINE provides an example on how to use destructible objects. Attaching Destructible.cs to an object will allow you to destroy that object, play an audio and spawn loot inside in case you want to do that. There is an example using Crate.cs, with this, a destroyed object will also be spawned, so that is recommended to be used.



Now, for the destroyed object, you would like to attach an object that has broken pieces in it. See the following image:



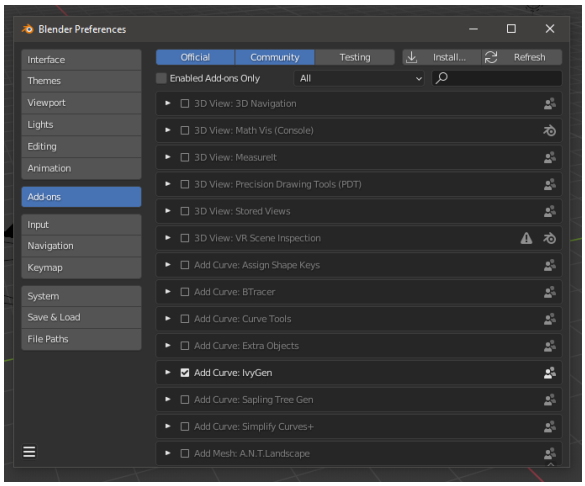
That is how the fractured crate looks like.

Do not forget to attach a collider (Simple box collider will work fine) and a rigidbody for the physics to work properly.

But... How can I destroy my model into different pieces?

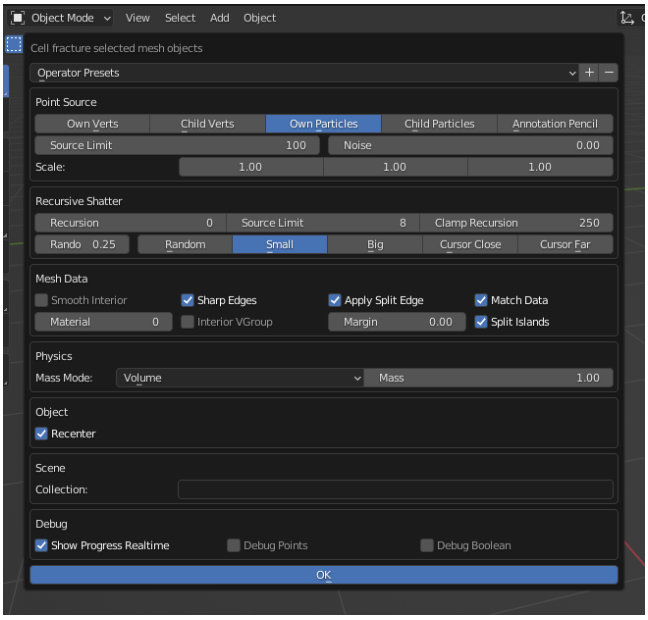
This example was made using Blender. Blender offers an add-on called Cell Fracture for this. Of course, you can use any other add-on you like.

To enable Cell Fracture go to Blender´s preferences (edit/ preferences), and then click on Add-ons.

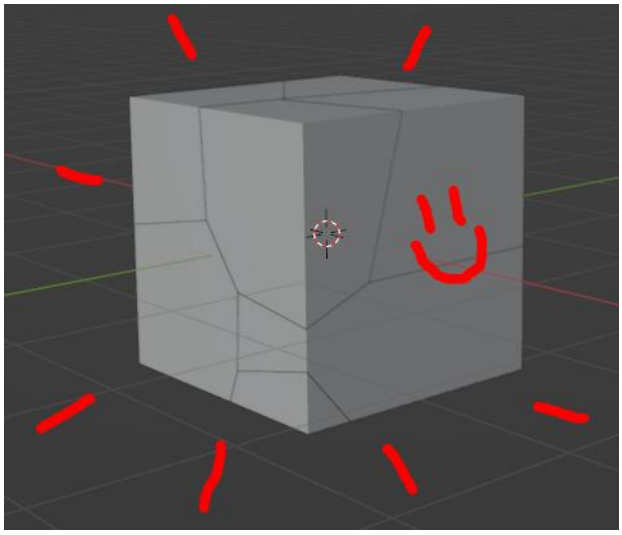


Search Cell Fracture and enable it.

In object mode, select your model and follow the path: Object/Quick-Effects/CellFracture. The next window will pop up

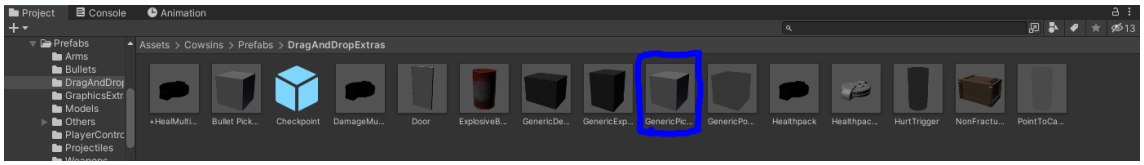


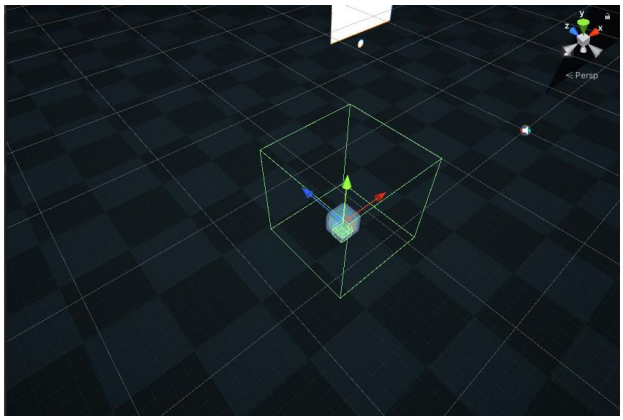
Here, you can customize the fracture as much as you want. Once you are happy, click Ok. Here is the result!



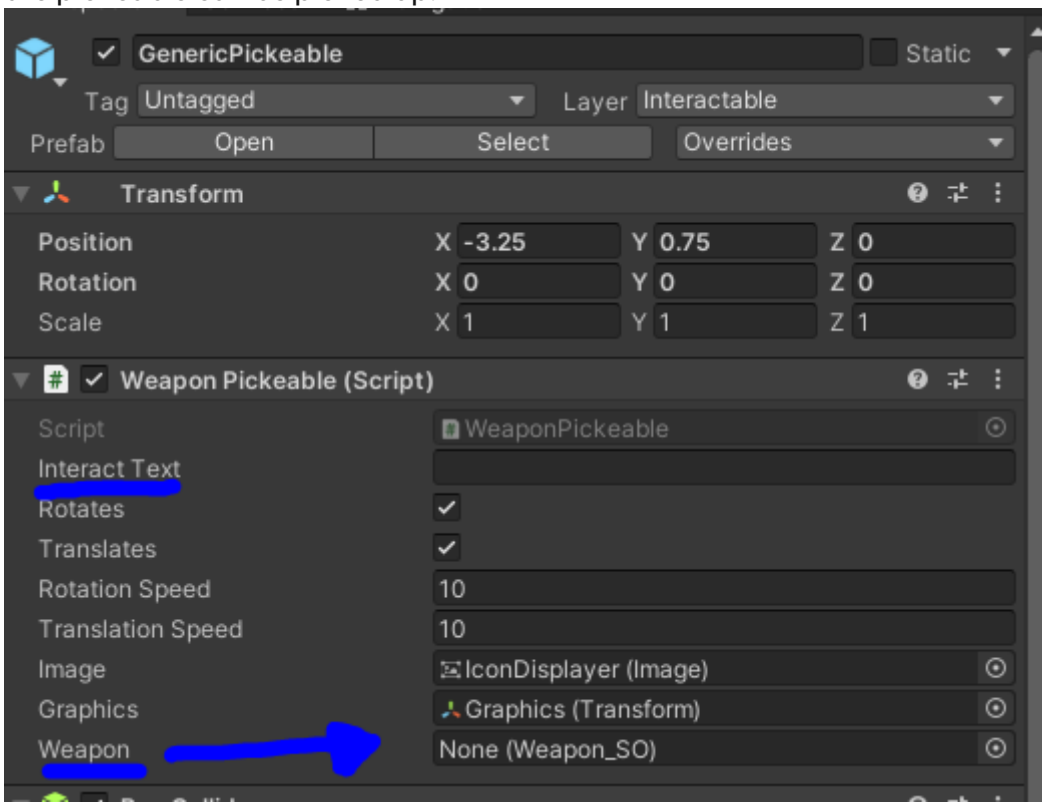
04.15 HOW TO CREATE NEW PICKEABLE OBJECTS

First, go to: Assets/Cowsins/Prefabs/DragAndDropExtras as shown, and drag and drop GenericPickeable.prefab into your scene.





Then, select it and go to its inspector and select the Weapon_SO you would like. Note that this Weapon_SO should belong to a fully set-up weapon. You can customize the interact text if you want. This text will be displayed on your screen when the pickeable can be picked up.



04.16 CONTROLLABLE & NOT CONTROLLABLE PLAYER

The player (Player Controller) is controllable when *controllable* in PlayerStats.cs is true. If it is false, you no longer will be able to perform any player action. Note that you can access the variable by using its static accessor *PlayerStats.Controllable.*

HOW TO MANAGE CONTROL

PlayerStats.cs offers a bunch of public functions you can call in order to grant or lose control.

GranControl() grants control to the player, overriding states like death and pause.

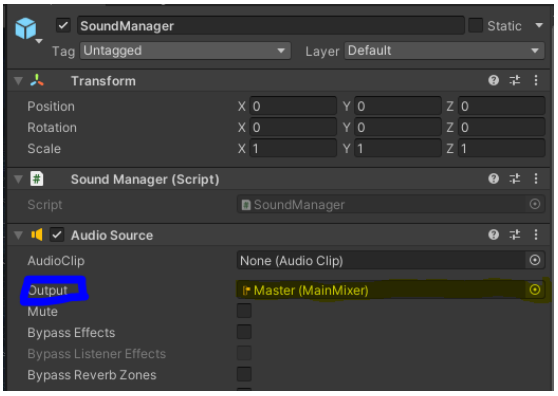
LoseControl() denies control to the Player, overriding states like death and pause.

ToggleControl() Toggles the control of the Player, overriding states like death and pause.

CheckIfCanGrantControl() Checks if the player is chooseable for being granted access. This is the recommended function to use in order to give control access to the player, since death and pause states will be taken into account.

04.17 SOUND MANAGEMENT

FPS Engine brings a simple SoundManager, that requires an Audio Mixer attached. The mixer provided is called MainMixer, and it currently only has a channel called “Master” in it. This way, you can adjust the master sound volume from the settings menu for all the Audio Sources that have MainMixer´s “Master” mixer attached as Output.

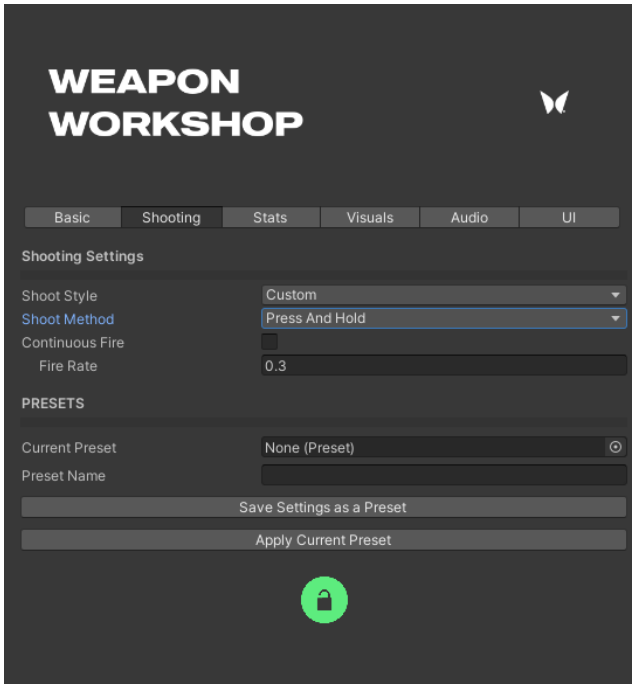


[Audio Mixer Official Documentation by Unity]

04.18 HOW TO USE CUSTOM SHOT METHODS FOR YOUR WEAPONS

Since 0.9.6, FPS Engine allows you to use custom methods on shoot, so you can create anything you´d like with that!

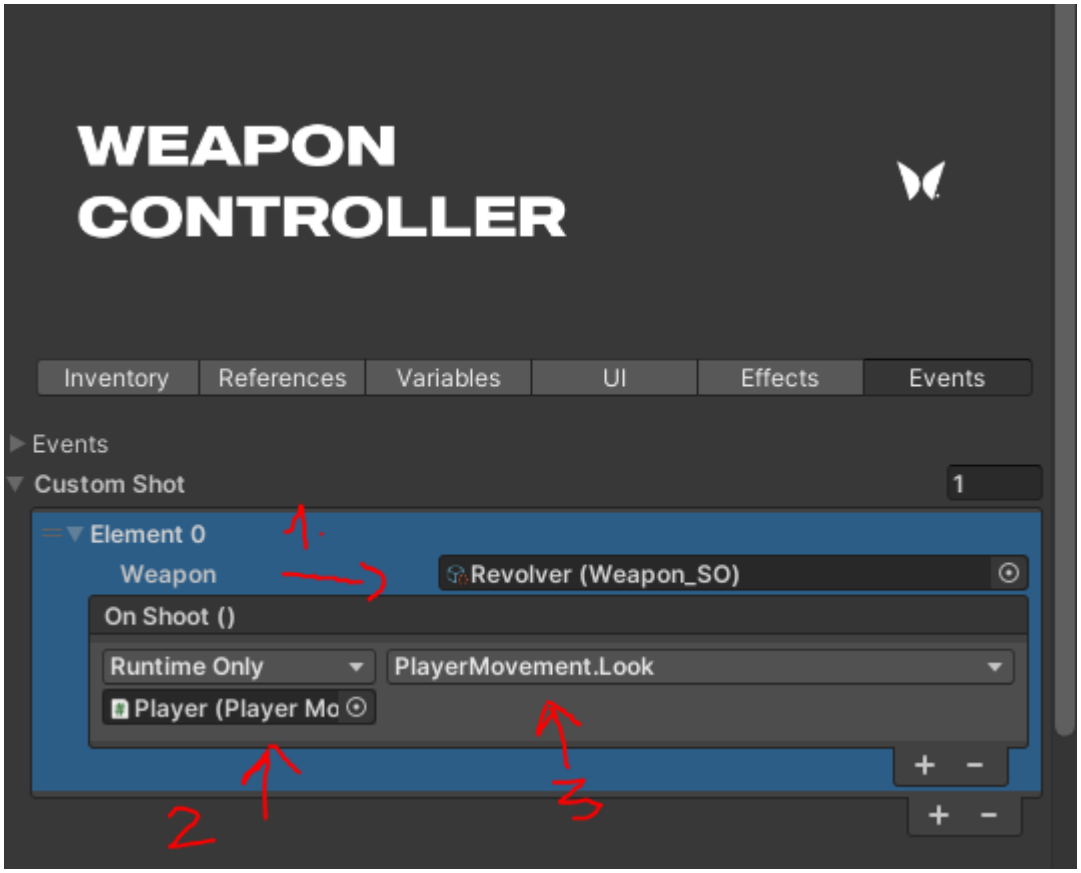
First, go to the scriptable object of the weapon you would like to modify, go under Shooting tab and change the Shoot Style to Custom



If continuous fire is true, your method will be called once per frame, otherwise you can just use the fire rate system that comes with FPS Engine. It is recommended to use Press And Hold Shoot method, as it's the standard for any weapon.

Now, go to the Weapon Controller in Player, move to the Events tab and:

1. Assign the scriptable object we just modified.
2. Attach the object that has the component with the method we want to call. You can of course write custom scripts and attach them to the player and call them from here.
3. Select the method we want to call On Shoot.


WEAPON
CONTROLLER
Inventory
References
Variables
UI
Effects
Events
Events
Custom Shot
1
Element 0
Weapon
Revolver (Weapon_SO)
On Shoot ()
Runtime Only
PlayerMovement.Look
Player (Player Mo
1
2
3

(FAQ) I have added a provided Player Controller Template, but I cannot move. ADDING NEW SURFACES

For the player Controller to work properly it needs to have a walkable surface

below. To have one, simply add a Ground layer to your ground. This way the player will recognize it as a surface that can be walkable.

As you can see there are other layers: Grass, Metal, Mud, Wood etc... You can also add these. These are used to determine the footsteps sounds as well as shooting impacts and VFX.

05.THIRD-PARTY RECOGNITION

Third-party assets must be acknowledged and appreciated since they played a significant role in the creation of the asset.

The project folders include their appropriate licenses as well.

freesound.org

Singularity Pack

Mixkit

DaniPixabay

Outfit

TastyTony

Hozq

BaungBoyz

Austincford

These third-party assets are protected under the following terms:
CC Attribution - Attribution 4.0 International

06. FIND SUPPORT

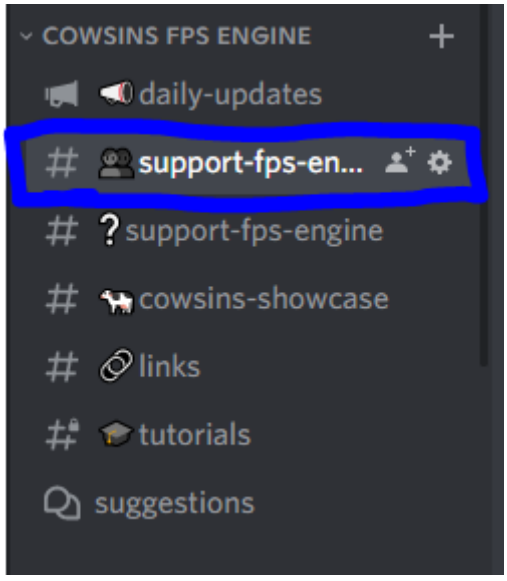
Looking for help? Search no more. Join the discord server and ask anything!

https://discord.gg/9eYPeHg4fh

After joining, there are two ways of contacting support.

1ST OPTION: Contact the owner of the server.

2ND OPTION: Go under “COWSINS FPS ENGINE” and ask for help in the chat channel.



Remember that for any of both options, you will have to verify yourself first by sending “DUOW” a DM with the invoice number / order ID / Purchase ID. These can be found in an e-mail sent by Unity after your purchase is complete.

We´ll love to help you! Moreover, we are nice and love games & game development, so you can hang out with us for sure!

OFFICIAL CONTACT EMAIL:

cowsinsgames@gmail.com

TWITTER:

https://twitter.com/cowsins