PLATFORMER ENGINE

DOCUMENTATION

2

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: PLATFORMER 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: PLATFORMER 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: PLATFORMER ENGINE are under development in a beta phase.
Current COWSINS´ PLATFORMER ENGINE version : 2.0

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.

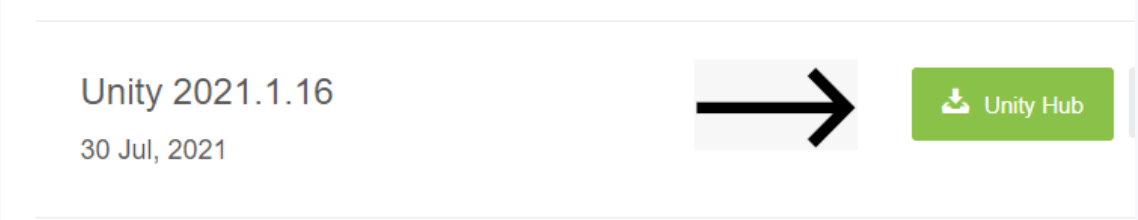
4

STEP 01 - INSTALLING UNITY ----------

You can skip to the following step if you already have a version of Unity that supports COWSINS: PLATFORMER 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)



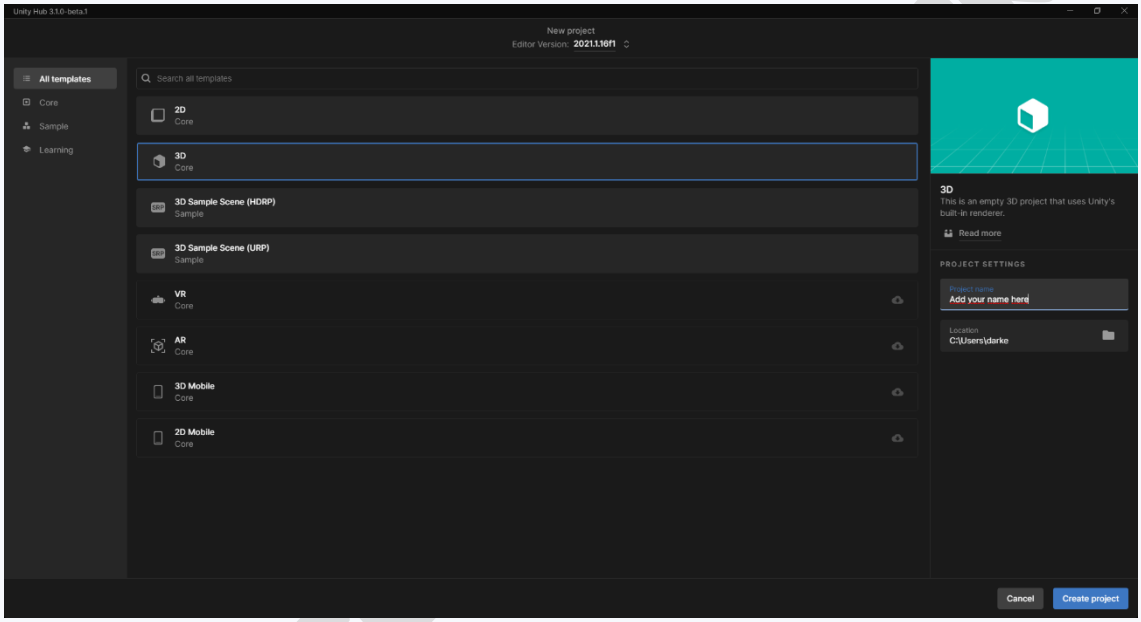
PICTURE 1. UNITY INSTALLATION THROUGH UNITY HUB

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 “PLATFORMER 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.

2. ROADMAP

Welcome to COWSINS™: PLATFORMER 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 to make it even more solid!

You can suggest new features in our discord server!

https://discord.gg/9eYPeHg4fh

Welcome to the documentation for the Unity Platformer Engine! If you want to create platformer games easily, you're in the right place. This guide will help you understand and use our package specifically designed for making platformer games.

Whether you're a beginner or an experienced game developer, we've got you covered. In this documentation, you'll find straightforward information, tutorials, and tips to help you make the most of this asset.

Inside these pages, we'll show you how to set up, configure, and enhance your platformer game, we've simplified it all. Dive in to discover how this asset can bring your platformer ideas to life in Unity. Let's get started on this exciting journey!

Please take note that the documentation's structure mirrors the organization of files you'll encounter within the "Cowsins Scripts" folder. This structure is as follows:

1. Camera
2. Editor
3. Effects
4. Enemy
5. Extras
6. Managers
7. Others
8. Pick Up
9. Player
10. PowerUp
11. UI
12. Utilities
13. Weapon

In the coming sections, we'll comprehensively delve into each of these categories one by one. This approach will ensure that you have a clear and structured understanding of how to work with each component and feature of this asset.

03.01 CAMERA

03.01.1 CAMERA CONTROLLER

The camera controller is responsible for managing camera movement, ensuring it accurately tracks the player's position as intended.

`cameraMethod`: Configures the style of the camera, the way it behaves.

`Target`: What object does this camera follow.

`cameraOffset`: Camera position variation.

`Boundary`: Size of the camera vision. Once the target is outside of these boundaries, it will start following the target.

`cameraLaziness`: The lower, the faster it will reach the destination.

`CameraToStatic`: Changes the camera method to static.

`CameraToSimple`: Changes the camera method to simple.

`CameraToBoundary`: Changes the camera method to boundary.

03.02. EDITOR

The purpose of the editor scripts is to enhance the presentation of inspector variables, providing a more visually pleasing and clearer representation.

HOW CAN I DISPLAY CUSTOM VARIABLES IN A CUSTOM EDITOR?

Make sure to add this inside the onInspectionGUI method:

```
EditorGUILayout.PropertyField(serializedObject.FindProperty("yourVariable"));
```

HOW CAN I ACCESS VARIABLES IN A CUSTOM EDITOR?

You can access any PUBLIC VARIABLE from the script you are making the editor for with the following line:

```
myScript.yourVariable
```

This works thanks to this line, inside the OnInspectorGUI method:

```
MyClass myScript = target as MyClass;
```

03.03. EFFECTS

03.03.1 CAMERA SHAKE

Handle Camera Shake Effect for the Camera. This uses Perlin Noise to ensure randomness.

Shake: Apply Camera Shake effect on the camera Automatically.

- Amount: Strength of the camera shake.
- Power: Strength of the Perlin Noise Effect.
- Movement Amount: Magnitude of Position Lerping allowed.
- Rotation Amount: Magnitude of Rotation Lerping allowed.

ExplosionShake: Apply Explosion Camera Shake effect on the camera Automatically, already configured to simulate explosions.

- Distance: Magnitude of distance from the explosion, the system automatically calculates the amount of camera shake.

03.03.2 HIT EFFECT

Name: Name of the hit effect. This should match the name of the layer.

hitVFX: Effect to display.

03.03.3 SURFACE EFFECT

Name: Type the name of your surface here.

stepVFX "Effect to display for the player steps.

stepSFX: Sound for the player steps

landVFX: Effect to display for the player landing.

landSFX: Sound for the player landing

speedModifier: "Speed multiplier on this certain surface.

03.04. ENEMY

03.04.1 ENEMY HEALTH

maxHealth: Initial & Maximum Health

maxShield: Initial & Maximum Shield

healthSlider: UI Element to display health.

shieldSlider: UI Elements to display shield.

deathVFX: Visual effect instantiated on death.

flashesOnDamage: "Set to true if you want the player to blink/flash on hit.

flashDuration: How long the effect is played.

flashSpeed: How fast the effect is played.

sprites: Array that contains all the spriteRenderers that will blink on damage. These must have the Blink material assigned.

03.04.2 IDAMAGEABLE

In Platformer Engine, any Damageable object inherits from the IDamageable class.

```
void Damage(float damage);

void Die(bool condition);
```

03.04.3 TURRET

pivot: Object to pivotate the muzzle from.

aimSpeed: Velocity to lerp the aim

damage: Damage to apply to the target on hit.

firingMode: Configure the way the firing works.

fireRate: Speed of the shooting

projectilePrefab: "Projectile object to instantiate on shoot [ONLY FOR PROJECTILE TURRETS]

projectileSpeed Speed of the projectile object [ONLY FOR PROJECTILE TURRETS]

projectileLifetime: Duration of the projectile before getting destroyed [ONLY FOR PROJECTILE TURRETS]

muzzle: Transform to shoot from

shootFromStart: Start shooting on awake.

whatIsPlayer: “Player” layer.

muzzleFlash: Effect to instantiate on the player on shoot.

shootSFX: Sound to play on shoot.

volume: Volume of the shoot SFX.

03.04.4 TURRET PROJECTILE

The projectiles shot by the turret do not use regular Projectile.cs attached, but Turret Projectile, which are specifically designed to target the Player.

03.05 EXTRAS

03.05.1 CAPTURE THE POINT

`progressRequiredToCapture`: How much time the player needs to stay inside the point to capture it.

`captureUI`: User interface that displays the progress on the capture. This can be a canvas that holds progressUI

`progressUI`: The image that specifically displays the progress.

03.05.2 CHECKPOINT

When the Player triggers the checkpoint, after the player dies it will automatically appear at the location of the checkpoint. You can subscribe any method you want to the onTriggerCheckpoint to perform custom actions, these actions won´t override the base behaviour of the checkpoint.

03.05.3 COIN

`minimumAmount`: Minimum number of coins to pick up.

`maximumAmount`: Maximum number of coins to pick up.

`pickUpSFX`: Sound on picking the coin.

IMPORTANT: Set minimum amount and maximum amount to be the same if you would like to collect a fixed number of coins.

03.05.4 COUNTDOWN

`InitialCountdownTime`: Measured in seconds.

`TimerText`: Displays the time remaining.

`StartCountdownAtStart`: Start counting from the start of the game

TIME FORMATTING:

`ShowHours`

`ShowMinutes`

showSeconds

03.05.5 CUSTOM TRIGGER

Subscribe to “action” to perform anything on triggering the object.

03.05.6 DESTRUCTIBLE

destroyedGraphics: Elements instantiated on destroying the object, more likely to be used to display broken elements of the object.

rewardsOnDestroy: Reward elements instantiated on destroying the object, such as coins, experience, or other items.

destroyedSFX: Sound played on destroying the object.

03.05.7 DESTRUCTIBLE PLATFORM

timeToDestroyAfterActivation: After a player triggers it, time to destroy in seconds.

03.05.8 DOOR

doorMethod: Configure the behaviour of the door. OpenByTrigger will automatically open the door when the Player approaches, no need to interact. OpenByInteract will force you to interact to open it.

autoCloseDoor: After a certain period, close the door?

autoCloseDoorTimer: Seconds to close the door.

doorCollider: Already configured in the prefab.

03.05.9 EXPERIENCE

minXp: Minimum amount to collect.

maxXp: Maximum amount to collect.

IMPORTANT: Set both minimum and maximum amounts to collect a fixed amount of experience.

03.05.10 FALLING OBSTACLE

Height

upTime: Time to move up.

idleTimeUp: Time to stay idle after moving up.

downtime: Time to move down.

idleTimeDown: Time to stay idle after moving down.

03.05.11 GRAVITY ZONE

gravityScale: Override the gravity scale value. Notice that the default gravity scale value is 9.8.

03.05.12 JUMP PAD

jumpForce: Force to eject the player in the vertical axis.

bounceSFX: Sound played on the player triggering the jump pad.

03.05.13 LOOT CHEST

lootUI: Object that contains the chest UI.

Rows: Rows to procedurally generate the chest UI Slots.

Columns: Columns to procedurally generate the chest UI Slots.

inventoryContainer: Object that contains the UI Slots.

inventorySlot: Reference to the UI Inventory Slot.

Loot: Loot inside the chest, notice that you cannot have more elements than Row*Columns.

03.05.14 MOVING PLATFORM

platformPositions: Array that stores the points where the moving platform will pass through.

loopType: Configure the behaviour of the platform. None: Stays idle. Reverse: Once it reaches the last platform position it lerps back, when it reaches the initial position, it starts again. Back: Once it reaches the last position, it lerps back to the first one.

Speed: Velocity Magnitude at which the platform moves.

startOnAwake: Start moving from the start.

StartMovement() - Start to move the platform.

03.05.15 NPC

interactInterval: Time in seconds between interactions.

Text: Text that displays the NPC texts.

NPCWelcomeMessages: Array containing all the welcome messages that this NPC has. Once you trigger an NPC any of these will randomly be assigned to the text UI.

03.05.16 ONE WAY PLATFORM

One-way platforms work the following way: The player can access them from below but then the player cannot go down again unless the crouch key is held.

03.05.17 SPIKE

`Damage`: Damage to be applied to the player per interval.

`damageInterval`: Seconds in-between damage applicable to the player.

03.05.18 TELEPORTER

`teleportMethod`: Configures the behaviour of the TP. You can either tp to specific locations or to another TP.

`toPosition`: Position to TP to.

`toTeleport`: Teleport to TP to.

03.06 MANAGERS

03.06.1 CHECKPOINT MANAGER

lastCheckpoint: Returns the position of the last checkpoint triggered.

03.06.2 COIN MANAGER

coins : Current coins amount.

AddCoins() : Adds coins to the current coin amount

Required int amount: Number of coins to add.

RemoveCoins(): Removes coins to the current coin amount.

Required int amount: Number of coins to add.

03.06.3 DEVICE DETECTION

mode: Current device. (Keyboard, Controller or Mobile Devices)

03.06.4 EXPERIENCE MANAGER

playerLevel: Starting from 0, current level of the Player.

experienceRequirements: XP required for each level to level up.

03.06.5 INPUT MANAGER
Handles Input processing.

03.06.6 SETTINGS MANAGER
Used for the main menu settings.

SetResolution(): Required int resolutionIndex: Index of selected resolution from the dropdown.

SetVolume(): Required float volume: Volume to set.

SetFramerate(): Required int framerateIndex: Target framerate to cap from the dropdown.

SetVSync(): Required bool vsync: If true, enable vsync, if false, disable.

LoadSettings(): Loads and applies saved settings.

03.06.7 SOUND MANAGER
Handles sounds and SFX in the scene.

03.06.8 UI CONTROLLER

03.07 OTHERS
03.07.1 AMMO TYPE SO
Allows to create new ammo types. It contains the necessary variables for them. Inherits from Item_SO.

03.07.2 FOOD SO
Allows to create new food types. It contains the necessary variables for them. Inherits from Item_SO.

03.07.3 IINTERACTABLE
Every Interactable object contains a script that inherits from Interactable.cs, which also inherits from this interface.

03.07.4 INTERACTABLE
Every interactable object inherits from this script, which at the same time inherits from the interface IInteractable, which makes it super simple to handle different time of interactions.

03.07.5 ITEM SO
All the items inherit from Item_SO, which is a Scriptable Object, as its name suggests (SO = Scriptable Object).

03.07.6 ITRIGGERABLE
Every Triggereable object inherits from this interface, that contains basic global functions, for enter, stay, and leave triggers.

03.08 PICK UP
03.08.1 ITEM PICK UP
Amount: Amount to pick up, these will be stored in the Inventory in case you use a full inventory system.

Item: Item_SO to pick up.

Graphics: Sprite Renderer to display the item icon.

03.08.2 AMMO PICK UP
ammoAmount: Amount of bullets to pick up.
ammoType: AmmoType_SO to pick up.

Graphics: Sprite Renderer to display the item icon.

03.08.3 WEAPON PICK UP
Weapon: Weapon_SO to pick up. If using hotbar only inventory, this will go to your hotbar. In case you are using the full inventory and your hotbar is full, it will be placed inside your inventory (if there is still space). In case the hotbar is not full it will be placed in a free slot.

Graphics: SpriteRenderer to display the item icon.

03.09 PLAYER
The player movement is managed by a state machine. The base for this state machine is handled with the following scripts: PlayerBaseState.cs, PlayerStateFactory.cs. The player has access to all the different possible states thanks to PlayerStates.cs. These are the player states:
PlayerDefaultState, PlayerCrouchState, PlayerDashState, PlayerDieState, PlayerGlideState, PlayerJumpState, PlayerLadderState, PlayerWallJumpState, PlayerWallSlideState.

03.09.1 PLAYER MOVEMENT

Graphics: Reference to the player graphics.

gravityScale: Gravity strength.

fallGravityMult: Gravity added. (As a multiplier).

maxFallSpeed: Limit the maximum fall speed.

maxUpwardsSpeed: Limit the maximum vertical speed (going upwards)

autorun: If enabled, the player will not be able to walk, just to run.

walkSpeed: Speed while walking.

runSpeed: Speed while running.

crouchSpeed: Speed while crouching.

horizontalLadderSpeed: Speed horizontally while in a ladder.

verticalLadderSpeed: Speed to climb a ladder.

runAcceleration: Capacity to gain speed and reach the maximum speed.

runDecceleration: Capacity to lose speed and go idle.

accelInAir: Capacity to gain speed and reach the maximum speed mid-air.

deccelInAir: Capacity to lose speed mid-air.

doConserveMomentum: remove the deceleration mid-air.

maxFloorAngle: Maximum floor angle to consider as a walkable area.

jumpMethod: Configures the way the jump works. Default: Press to jump. You are able to hold. Hold to Jump Higher: Press = Tiny jump. Hold = Higher jumps depending on the held time.
amountOfJumps: How many jumps you can do before landing. 1 is the default value assigned.

jumpForce: How tall you will jump.

apexReachSharpness: Adjust the stiffness of the jump. The higher this value is the sharper it will perform. Usually, sharp jumps are more responsive than smooth jumps, but it all depends on the style you are looking for.

jumpHangGravityMult: Jump Hang is the movement at the apex of the jump. This adjusts the gravity of the jump in that point.

jumpHangTimeThreshold: Enable jump hang depending on the current velocity, so it gets activated before reaching the apex.

jumpHangAccelerationMult: Acceleration multiplier on the apex of the jump.

jumpHangMaxSpeedMult: Speed multiplier on the apex of the jump.

fastFallMultiplier: When falling, you can press “S” to fall faster. Adjust the multiplier here.

wallJumpForce: impulse of the wall jump.

wallJumpRunLerp: Speed to adjust the velocity from walking to running when you wall jump.

wallJumpTime: Duration of the wall jump before going back to default state.

wallSlidingResetsJumps: Enable to reset the amount of jumps when wall sliding.

wallSlideSpeed: Velocity of wall sliding.

wallSlideVFXInterval: Interval in seconds to display Wall slide visual effects-

wallSlideVFX: GameObject that represents the visual effects for sliding.

dashMethod: Configure the behaviour of the dash. None: Disables dashing. Default: Based on Horizontal movement. AimBased: Dash wherever you are aiming. HorizontalAimBased: Dash Horizontally based on the aim.

amountOfDashes: How many dashes you have available.

dashCooldown: Time in seconds to being able to use dashes again.

dashGravityScale: Gravity when dashing.

dashDuration: Duration in seconds of the dash.

dashSpeed: Travel velocity when dashing.

dashInterval: When you stopped dashing, time to being able to perform a new dash (to avoid dash clipping)

invincibleWhileDashing: If enabled, the player won´t be able to receive damage while dashing.

glideSpeed: Speed of movement when gliding.

glideGravity: How fast the player goes downwards when gliding.

canDashWhileGliding: Allows dash if gliding.

glideDurationMethod: Adjusts the duration method for the gliding. None: Infinite gliding. TimeBased: Adjusts the duration based on a timer in seconds.

maximumGlideTime: Duration of the glide in seconds.

handleOrientationWhileGliding: Allow the player to orientate (based on the current orientation method) while gliding.

allowCrouch: Allow the player to crouch.

canCrouchSlideMidAir: Allow the player to crouch while airborne.

crouchSlideSpeed: if the player has enough speed momentum when crouching, it will perform a slide. Adjust the force of that slide.

crouchSlideDuration: Adjust the duration of the crouch slide.

usesStamina: "You will lose stamina on performing actions when true.

minStaminaRequiredToRun: Minimum stamina required to being able to run again.

maxStamina: Max amount of stamina.

staminaRegenMultiplier: Speed to regenerate the stamina.

staminaLossOnJump: Amount of stamina lost on jumping.

staminaLossOnWallJump: Amount of stamina lost on jumping.

staminaLossOnCrouchSlide: Amount of stamina lost on sliding.

stepHeight: Maximum height allowed for a surface to be allowed as a step.

playerOrientationMethod: Handles the way the player is oriented. None: The player cannot orientate itself. HorizontalInput: The player orientates based on the A & D inputs. Aim Based, the player looks at the crosshair. Mixed: Mix between Aim Based and HorizontalInput.

landOnEnemyDamage: Damage applied on an enemy when landing on it.

landOnEnemyImpulse: Upwards impulse applied to the player when landing on an enemy.

Step: Stores the SurfaceEffects for each surface.

footstepsInterval: Speed at which the footsteps are played.

footstepsVolume: Volume at which the footsteps are played.

landVolume: Volume at which the land SFX is played.

coyoteTime: Coyote Jump allows the player to jump responsively, even when leaving a ground or surface, there is still a timing that allows the player to jump. Adjust this number to configure the way that works.

jumpInputBufferTime: Allows for a more responsive jump.

03.9.2 PLAYER STATS

maxHealth: Initial health value.

maxShield: Initial shield value.

timeToRevive: Time to revive when dead (if there is any enabled checkpoint)

healthRegenerationMethod: Set the health regeneration method. None: Won´t regenerate. AlwaysRegenerates: Regenerate if the health is below the maximum value.

healthRegenerationAmount: Amount of regeneration.

healthRegenerationRate: How fast the regeneration occurs. The lower this number is, the faster the regeneration will be.

takeFallDamage: Enable if the player should be able to take damage.

dontTakeFallDamageIfWallSliding: Enable to disallow fall damage when wall sliding.

dontTakeFallDamageIfGliding: Enable to disallow fall damage when gliding.

minimumHeightDifferenceToApplyDamage: Minimum distance to apply fall damage when landing.

fallDamageMultiplier: Damage applied. This depends on the height of landing.

flashesOnDamage: Enable to apply flash or blink effect when damaged.

flashDuration: Amount of time in seconds that the flash takes to complete.

flashSpeed: Velocity of the flash. The higher this number is, the more blinks it will perform.

Sprites: Sprites to apply the flash effect on.

03.9.3 INTERACTION MANAGER

Inventory: Reference to the inventory object.

inventoryRowsAmount: Number of rows, the total number of items is rows*columns.

inventoryColumnsAmount: Number of columns, the total number of items is rows*columns.

timeToInteract: Time in seconds to hold the interaction key to successfully perform an interaction.

canDrop: If enabled, the player will be able to drop items by pressing the drop key.

dropDistance: Distance to drop the item from the player.

dropImpulse: Force to apply on the dropped item.

genericWeaponPickeable: Reference to the Generic Weapon Pickeable Prefab

genericAmmoPickeable: Reference to the Generic Ammo Pickeable Prefab

genericPickeable: Reference to the GenericPickeable Prefab

openInventorySFX: Sound played on opening the inventory.

closeInventorySFX: Sound played on closing the inventory.

03.9.4 PLAYER MULTIPLIERS

Handles multipliers for speed, damage received, damage dealt, among others.

03.9.5 PLAYER PROCEDURAL ANIMATOR

Graphics: Target object to apply the animations.

scaleLerpSpeed: Speed to adjust the scale.

jumpScale: Adjust the scale when jumping.

landScale: Adjust the scale when landing.

maxTilt: Maximum rotation allowed when moving.

tiltSpeed: Speed to adjust the rotation.

03.9.6 PLAYER ANIMATOR

Handles Player Animations. It is important not to confuse this with the Player procedural Animator. Player Animator handles anims from the Animator component.

Animator: Reference to this Animator.

03.9.7 GRAPPLE

grappleCamera: Reference to the player Camera.

grappleLineRenderer: Reference to the player Line Renderer component.

grappleMask: Sets the allowed grapple layers. You won´t be able to use the grapple on surfaces that are not included in this mask.

moveSpeed: Velocity from the initial grapple position to the grapple target.

grappleLength: Maximum distance allowed to grapple.

maxPoints: It ensures that the list only retains the most recent grapple points up to the specified maximum value, discarding the oldest points if necessary.

03.10 POWER UP

03.10.1 POWER UP

Regenerate: Set to true if the power up should appear again after being triggered.

regenerateTime: Time to regenerate in seconds.

03.10.2 DAMAGE DEALT POWER UP

Inherits from PowerUp.cs

valueAdded: 0 meaning 0%, and 1 meaning 100%, increment the damage dealt.

03.10.3 DAMAGE RECEIVED POWER UP

Inherits from PowerUp.cs

valueAdded: 0 meaning 0%, and 1 meaning 100%, increment the damage received.

03.10.4 HEALING RECEIVED POWER UP

Inherits from PowerUp.cs

valueAdded: 0 meaning 0%, and 1 meaning 100%, increment the heal received.

03.10.5 HEALTHPACK

Inherits from PowerUp.cs

healthAdded: Health to add.

03.10.6 INVINCIBILITY POWER UP

Inherits from PowerUp.cs

InvincibilityDuration: Time in seconds to be invincible.

03.10.7 JUMP HEIGHT POWER UP

Inherits from PowerUp.cs

valueAdded: Increment the jump height.

03.10.8 SPEED POWER UP

Inherits from PowerUp.cs

valueAdded: 0 meaning 0%, and 1 meaning 100%, increment the speed.

03.11 UI

03.11.1 COWSINS BUTTON

Cowsins Button works exactly as basic Unity Buttons, but the customization possibilities are way higher. You can subscribe to events such as onMouseEnter, onMouseStay, onMouseClick, onMouseRelease and onMouseLeave to create better UI experiences!

03.11.2 CROSSHAIR

`Player`: Needed reference to the Player weapon Controller. Used to access important information.

`ShowCrosshairIfWeaponIsNull`: If the player does not own a weapon, set to true if you want the crosshair to still be visible.

`defaultSpread`: Spread that the crosshair will lerp to (“idle spread”)

`resizeSpeed`: How fast the spread changes

`defaultWidth`: Width of each line of the crosshair

`defaultHeight`: How long each line is.

`defaultColor`: Basic color of the crosshair.

03.11.3 FPS DISPLAY

`frameRateText`: TMPro Text to display the frame rate:

`updateFrequency`: Seconds interval to display the information.

`showCurrentFrameRate`

showMinimumFrameRate

showMaximumFrameRate

03.11.4 INVENTORY SLOT

image: Image that displays the item icon.

background: Image that represents the background.

amountText: TMPro texts that displays the number of items the player currently has on this slot.

hoverSize: Scale of the slot when hovered.

hoverSFX: Sound to display on hover.

03.11.5 PARALLAX BACKGROUND

Cam: Reference to the player camera.

parallaxStrength: Amount of movement allowed for the background.

03.11.6 SEAMLESS SCROLL

Image: Image to apply the effect on.

Movement: Vector2 that represents the movement of the scroll.

03.11.7 TOOLTIP

itemIconDisplay: Image that displays the dragged icon.

textComponent: Text that displays the item name on hover.

imageComponent: Background.

horizontalMarginSize: Margin for the background to give space to the text. (Horizontal)

verticalMarginSize: Margin for the background to give space to the text. (Vertical)

03.11.8 TYPEWRITE EFFECT

typingInterval: Speed at which each character is revealed.

typeOnStart: Set to true if the effect should begin on start.

03.11.9 VIRTUAL CURSOR

cursorRectTransform: Reference to the RectTransform of this object.

moveSpeed: Sensitivity of the movement.

03.12 UTILITIES

03.12.1 DESTROY AFTER

timeToDestroy: Time to destroy this gameObject from start.

03.12.2 LOAD SCENE

sceneOrderID: Order ID of the scene you want to load. Notice that this scene should be included in the Build Settings. You will find the order ID of each scene in the Build Settings.

03.12.3 PLAY SOUND

Clip: SFX to play.

volume: Volume to play the Sound.

03.13 WEAPON

03.13.1 WEAPON CONTROLLER

playerCamera: Reference for the player camera.

controllerAimSensitivity: Sensitivty for the aim when using a controller.

weaponHolder: Transform that contains all the weapons during the game.

initialWeapons: Assign the Weapon_SO of your initial weapon. Do not assign more than the hotbar size.

hitLayer: Assign all the layers that can be hit by the player.

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.

canShootWhileLadder: Allow shooting while in a ladder.

canShootWhileGliding: Allow shooting when gliding.

03.13.2 WEAPON_SO

03.13.3 WEAPON IDENTIFICATION

`Weapon`: Assign this weapon.

`Muzzles`: Array that contains all the muzzles. Each muzzle will define a shooting point where the projectile will be instantiated (for Projectile weapons) and the muzzle flash will be spawned.

03.13.4 PROJECTILE

Every projectile weapon must be assigned a Projectile item, which is a game object with a Projectile component or one that inherits from it. Check Spear.cs for an example of this.

03.13.5 SPEAR

Spear inherits from Projectile. This is an example of how you can make custom weapons with custom projectiles.

04. GUIDES

These guides will provide you a better understanding of the most common use cases of Platformer Engine.

In case you need further assistance, make sure to join our Discord Server and verify your purchase, then you will be able to ask for support!

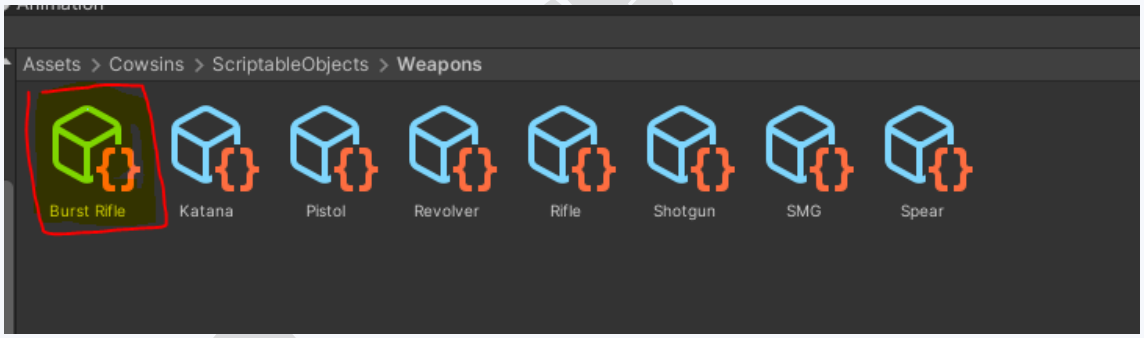
04.1 HOW TO ADD A CUSTOM WEAPON

Before adding a new weapon, make sure you have the following REQUIRED assets:

A weapon sprite, SFX for shooting, reloading, unholstering and others, and VFX for the Muzzle Flash and Hit Effects. Notice that you can of course use the provided assets.

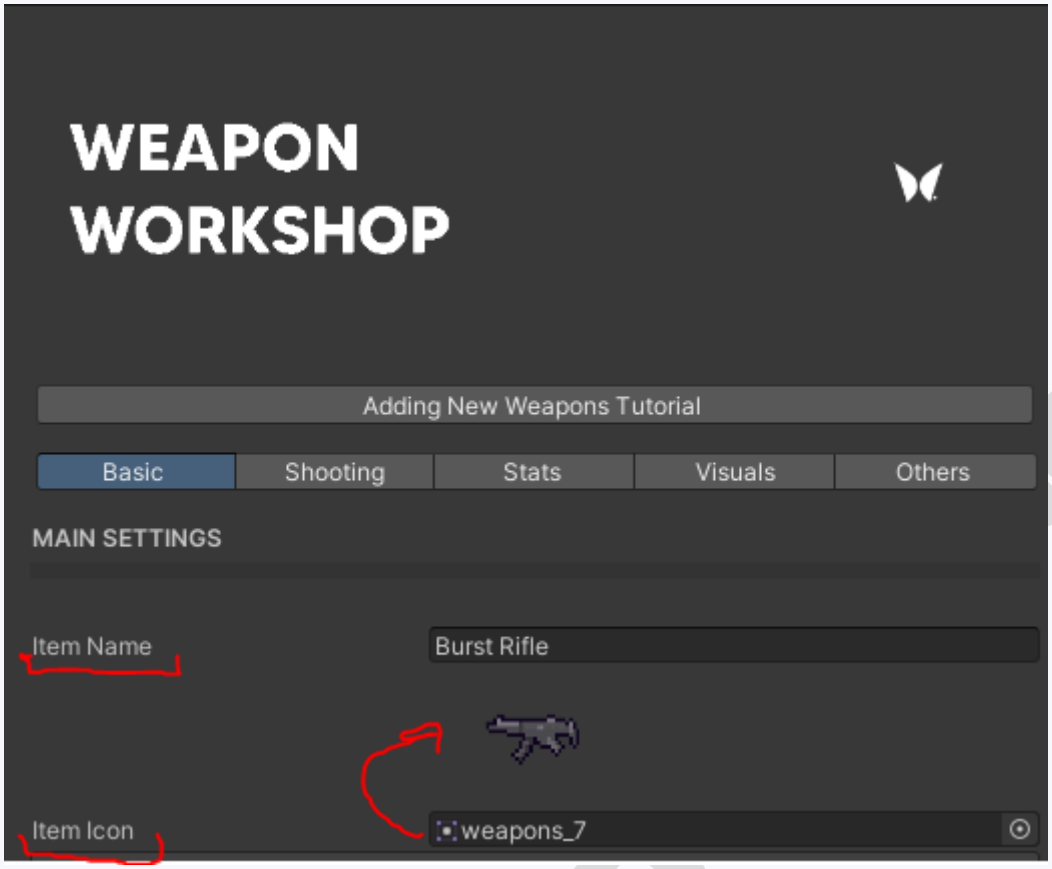
First, go to a proper folder to keep things clean and properly structured. It is highly recommended to move to Cowsins/Scriptable Objects/Weapons

Here, right click, move up to the Create section, and then click on “New Weapon”. You can name this file as you want, for this case, we named it BurstRifle:

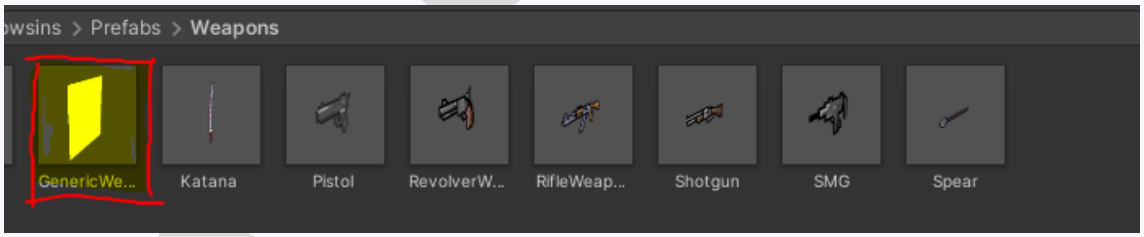


You can click on this file and start adjusting all the variables. You can also make use of the Presets for faster development.

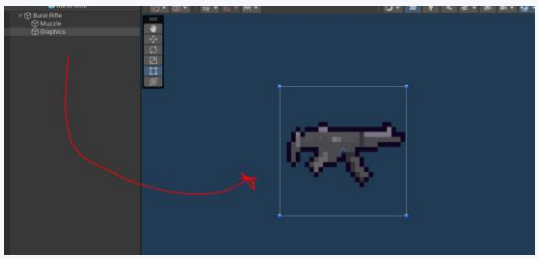
After this, make sure to fill up the next items:

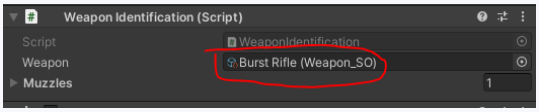


Below that, you will notice you need a weapon Object. A weapon Object is the weapon that you will see in game when your weapon is unholstered. To create it, go to Cowsins, Prefabs, Weapons and duplicate the GenericWeapon prefab.

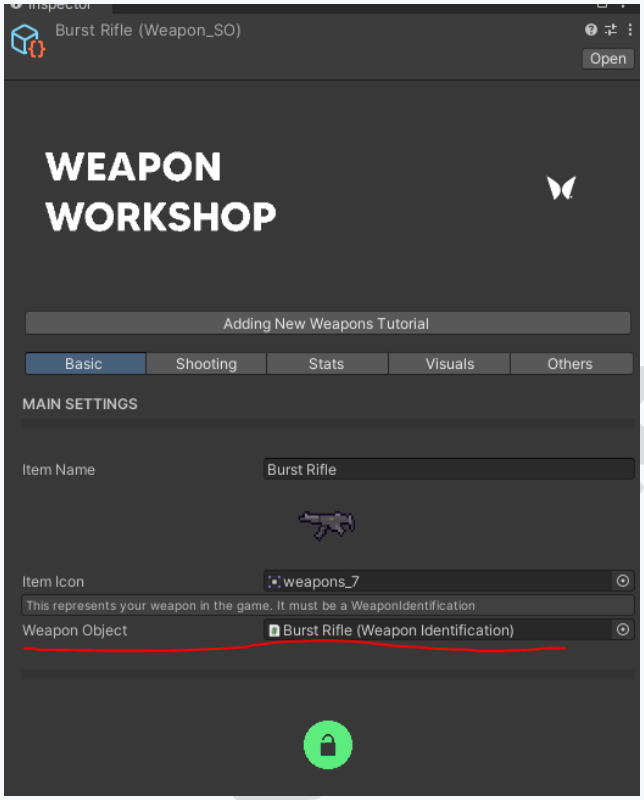


Inside it, change the sprite as well as the Weapon to the Scriptable Object we just created on the root of the prefab, in WeaponIdentification.



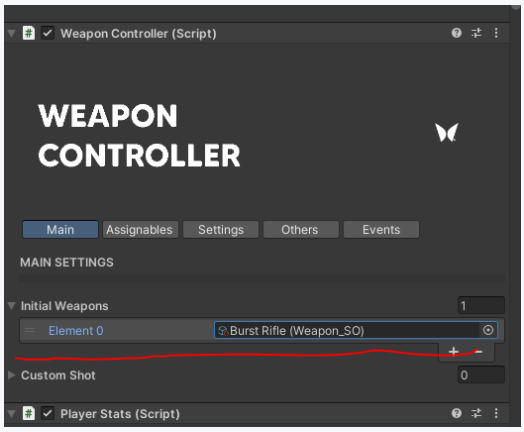


You will also notice that there is an array called muzzle. Here you can store all the muzzles points for your weapon. Each muzzle point defines a point where the projectile will be instantiated as well as VFX like the muzzle flash.

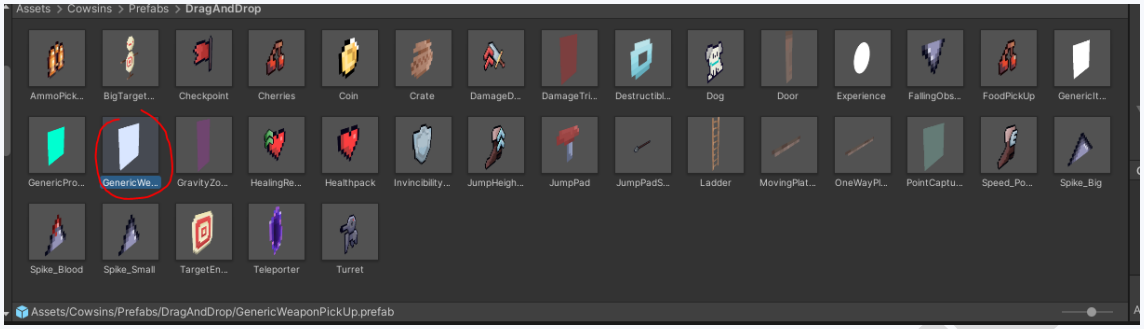


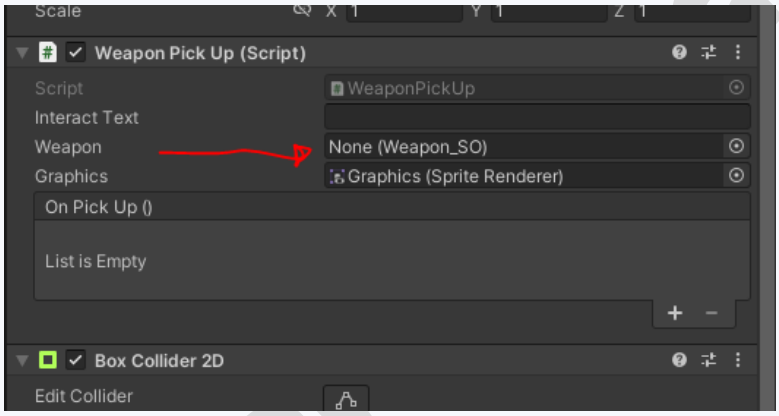
Once you are ready, assign this weapon object to the Weapon_SO. Just like this, we are ready to go.

You can assign your weapon on the Initials weapon to automatically receive it on your inventory,

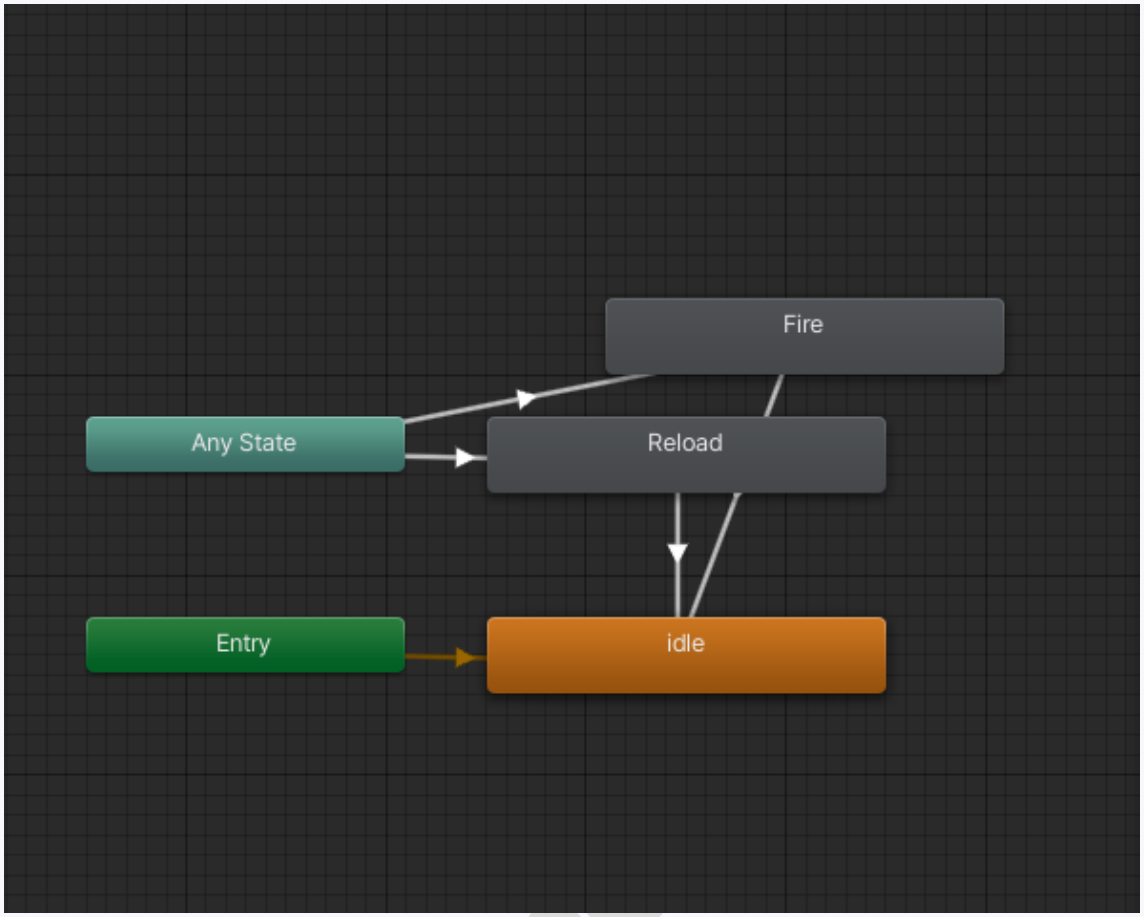


or you can drag and drop a generic weapon pickeable from Cowsins/Prefabs/DragAndDropExtras and assign the desired weapon_So. Like this, you will be able to interact with the pickeable and unholster the weapon.





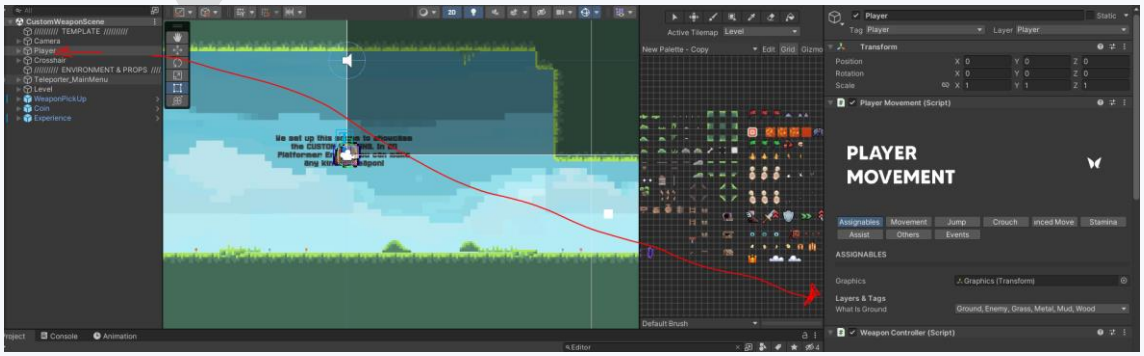
There is only one thing missing, the reload animation or Fire Animation for melee weapons. You can duplicate the Generic Weapon animator controller from Cowsins / Animations / Weapons, rename it and assign it to the Weapon Object Animator. There, create an animaton and assign it inside the according animation state.



Now, everything is fully set up, and you are ready to have fun with your new weapon in Platformer Engine!

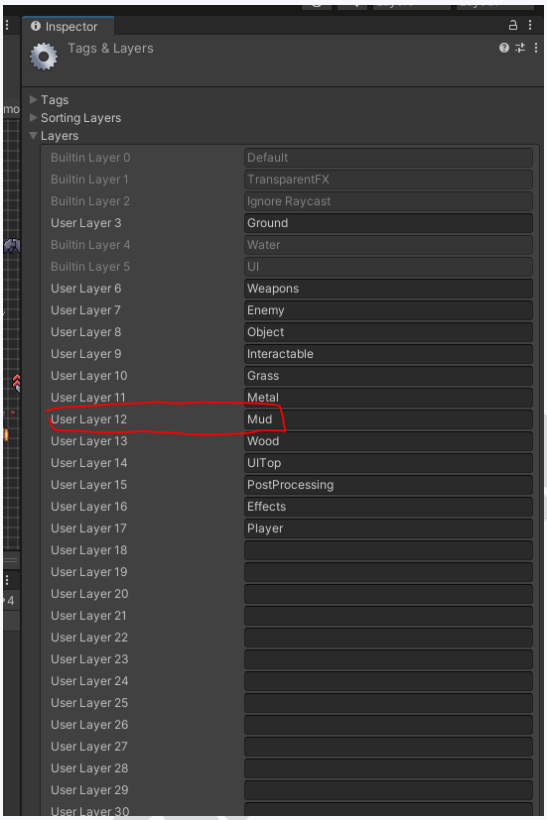
04.2 ADDING NEW WALKABLE SURFACES

In Platformer Engine, any surface must have a collider and a layer assigned. All the allowed surfaces are stored in the Player – PlayerMovement – Assignables – WhatIsGround



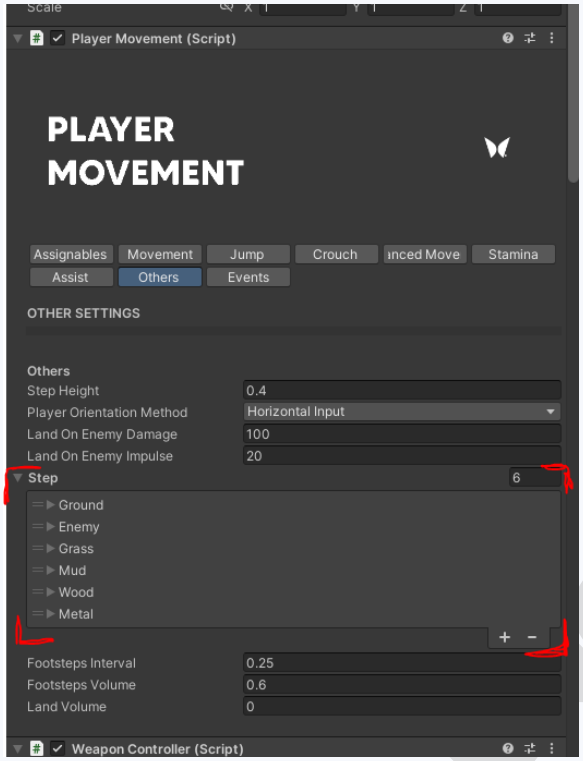
When you install the package, all of this is already set up for you, but what happens if you want to add a new layer that is not brought by the asset? The process is very simple!

First, you want to add a new layer. Go to the top right corner and click on Layer, then click on AddLayer. Here, add a layer, we added Mud.



After that make sure your layer is included in the whatIsGround layermask.

Just like that, Platformer Engine will detect the surface as a walkable area. However, what happens with footsteps? In this asset you can customize everything related footsteps, from steps sfx and vfx to landing sfx and vfx, as well as the volume to play the sounds. For that go to the Player – Player Movement – Others – Steps. Here you will see an array of classes.

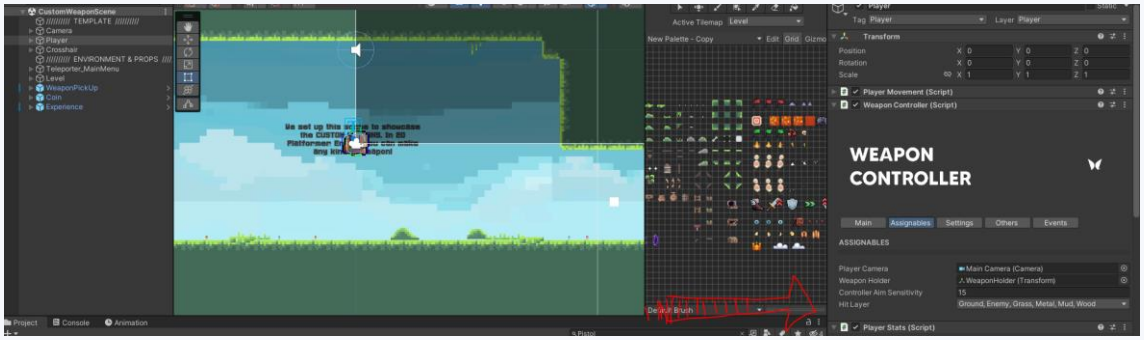


Each class represents a different layer.

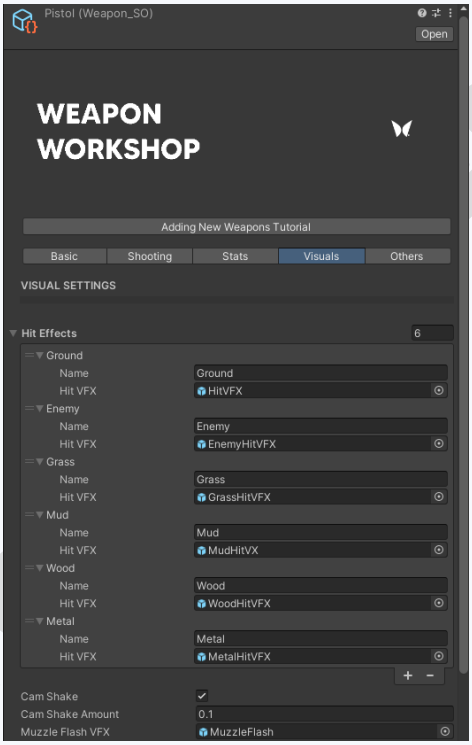
Make sure that "name" matches the exact same name as the Layer we just created. Once you make sure that is properly set up, you can adjust the other variables as you like!

04.3 ADDING NEW HIT SURFACES

In Platformer Engine, any surface that can be hit by a weapon has to be stored inside Player – WeaponController – Assignables – Hit Layer



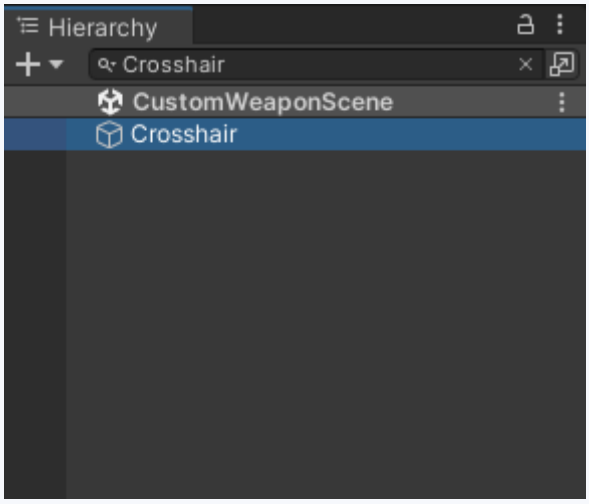
Then, for each weapon, you can configure graphics and sounds on hit for each of the hit layers. Go to the weapon_SO you want to modify, like Pistol. Then, go to Visuals.



You can assign hit VFX for each of the layers in HitLayer, just make sure that “Name” corresponds to the exact same name of the layer.

04.4 MODIFY THE CROSSHAIR

Search Crosshair in the hierarchy.



The crosshair is procedurally generated using GUI, so it is not based in the Unity´s UI.

When selecting the crosshair, you can configure all the required parameters.

05.THIRD-PARTY RECOGNITION

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

Special thanks to Dawnosaur for letting us use his controller as a base for Platformer Engine! Ctrl+Click on his name to know more about him!

Dawnosaur

Pixabay

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

06. FIND SUPPORT

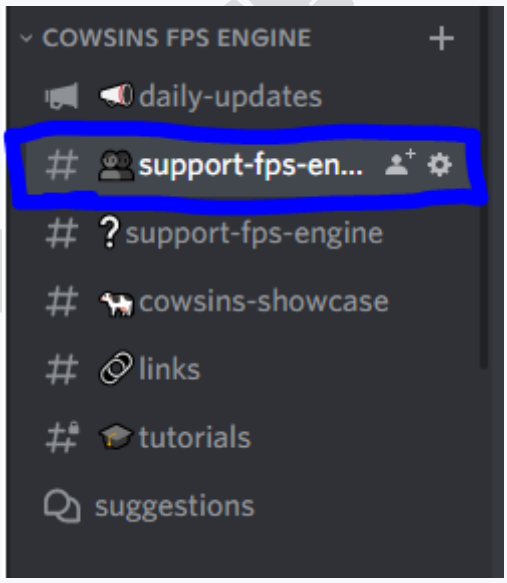
Looking for help? Search no more. Join the discord server and ask anything!

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 "PLATFORMER 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