Cowsins™

BULLET HELL
ENGINE

v.0.5
DOCUMENTATION

BULLET HELL ENGINE BY COWSINS.
UNITY ASSET STORE

COWSINS BULLET HELL ENGINE

INDEX

00. EARLY CONSIDERATIONS 3

01.BASIC SET-UP 5

02. ROADMAP 8

03. GUIDES & ASSET USAGE 9

03.1 – PATTERNS SYSTEM, HANDLING & CREATION 9

03.3 – IMAGE PATTERN 10

03.4 – HOW TO USE TOUCH INPUT FOR MOBILE DEVICES 12

03.6 – FLASH | BLINK | HIT EFFECT BASICS 13

03.7 – PLAYER CONTROL ACCESS 14

03.8– OPTIMIZATION RECOMMENDATIONS 16

03.9– MAKE A MOBILE GAME WITH BULLET HELL ENGINE 17

04. ASSET CONTENT 18

04.1 – CAMERA 18

04.2 – DAMAGEABLE 19

04.3 – EFFECTS 23

04.4 – EXTRA 25

04.5 – MANAGERS 28

04.6 – PATTERN 36

04.7 - PLAYER 43

04.8 – POWER UPS 47

05.THIRD-PARTY RECOGNITION 48

06. FIND SUPPORT 49

You can click (control key + left click) on the headers to directly visit the section.

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: BULLET HELL 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: BULLET HELL ENGINE.

These are viewable on the official Unity website.

Last time 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.

ASSET USAGE FAQ

Can I use this asset for my own games?

Yes! You can use Bullet Hell Engine for your own games.

Can I use this asset for my commercial games?

Yes! You can use Bullet Hell Engine to make your own game and sell the game on platforms like Steam, itch.io, Epic Games Store, Google Play, App Store… Cowsins won´t ask for any royalties from your earnings.

Can I use the assets (VFX, Models, Sprites, SFX, etc…) from Bullet Hell Engine for my own game?

The assets included in the package are mainly examples of projects created with Bullet Hell Engine. However, you can of course still use them for your own projects!

Is it mandatory to give credit to Cowsins for the asset?

Credit is not mandatory, but as always, it is highly recommended. Credit shows respect to the creator of the asset, and it helps a lot as well!

Both this document and COWSINS: BULLET HELL ENGINE are under development.

Current COWSINS´ BULLET HELL ENGINE version: 0.5

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.

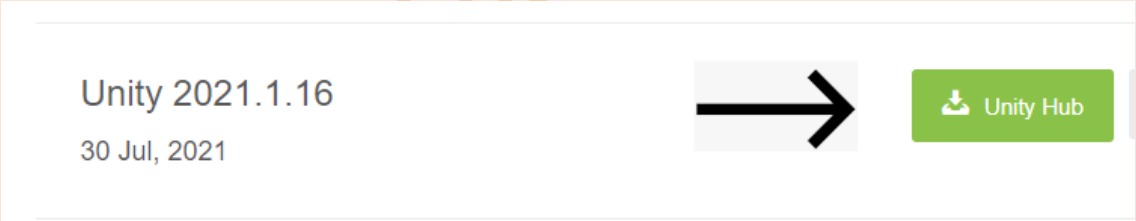
However, you can always check this instruction step by step if you run into any issues or haven't installed the asset yet.

STEP 01 – INSTALLING UNITY ----------

You can skip to the following step if you already have a version of Unity that supports COWSINS: BULLET HELL ENGINE. If not, please navigate to the Unity Download Archive (Last time 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 2021.3.0.f1

View the image below.



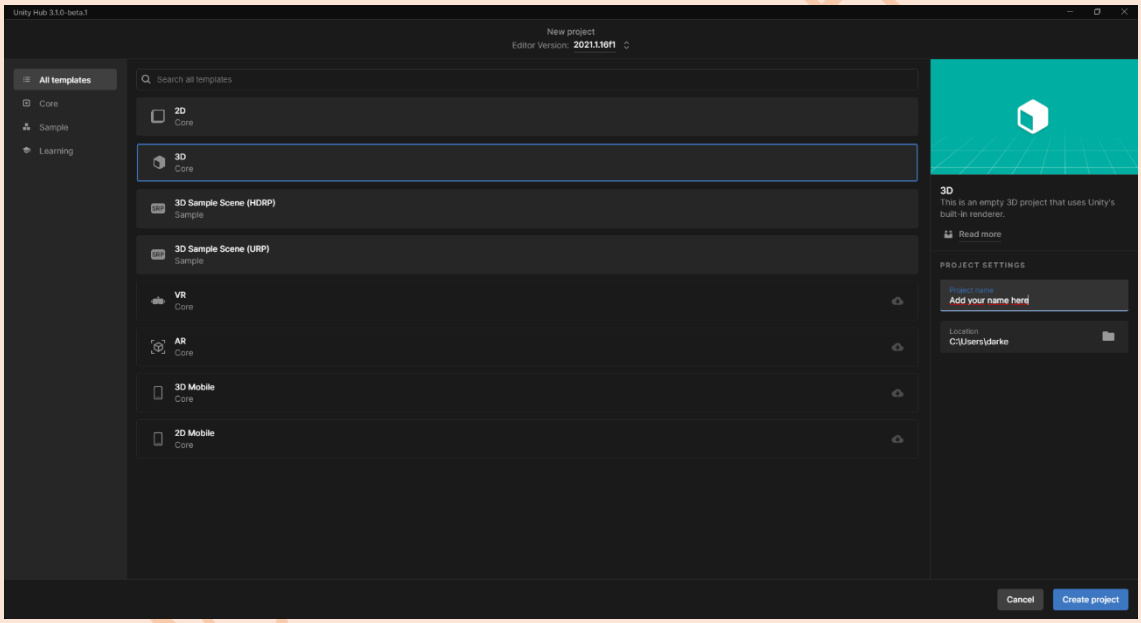
PICTURE 1. UNITY INSTALLATION THROUGH UNITY HUB

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 “Bullet Hell Engine” 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. The asset will also override your project settings, so make sure your project is a safe environment where you can install the asset.

All of these are required for the asset to work correctly.

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

Note that even though Bullet Hell Engine is fully compatible with URP and HDRP, you may need to replace all the materials from the Built-in pipeline to the one you chose, moreover, the blink shader for 2D sprites won´t work: Apart from that, Bullet Hell Engine should work as usual! (An URP/HDRP version of the shader will be added in the future!)

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™: BULLET HELL ENGINE 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, as well as with the other templates available on the Unity Asset Store.

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 and performant!

You can suggest new features in our discord server!

https://discord.gg/9eYPeHg4fh

Check the official Trello board!

https://trello.com/b/hgAdY3G3/bullet-hell-engine-roadmap

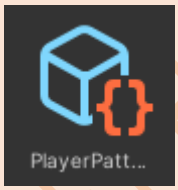
03. GUIDES & ASSET USAGE

03.1 – PATTERNS SYSTEM, HANDLING & CREATION

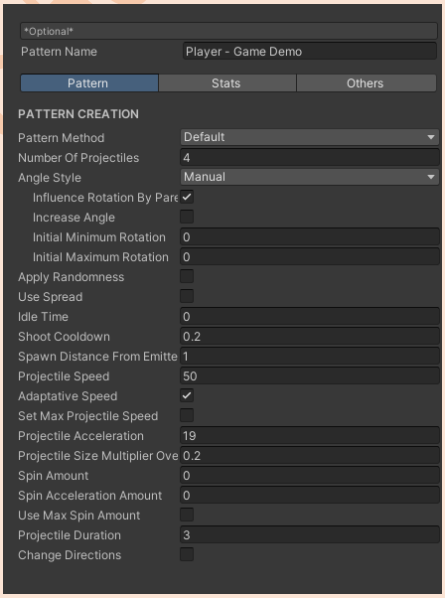
Bullet Hell Engine´s keys are the pattern creation and spawning / instantiating systems.

A pattern is defined by a Pattern_SO (pattern scriptable object).

You can easily create new patterns by right clicking on the project folder and following the route: COWSINS/ New Pattern.



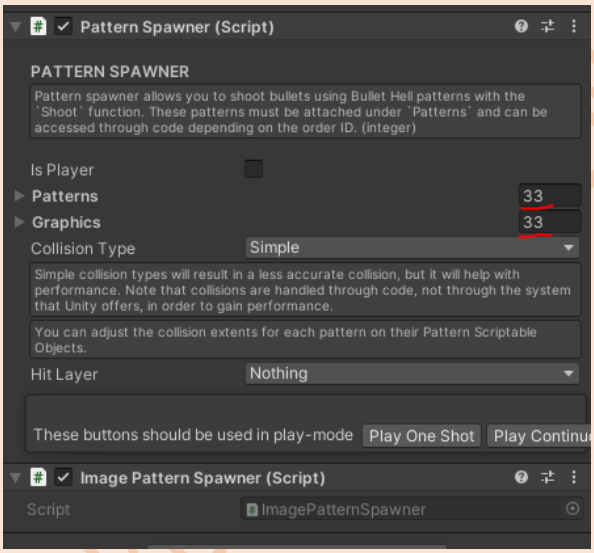
You can select these scriptable objects and modify its pattern customization features, statistics, etc…



It is very important to highlight that visuals cannot be modified here. A pattern_SO essentially describes and defines the way a

PatterSpawner will instantiate the bullets, and stores it into different files. These Pattern_SO files can be used for both 2D and 3D at the same time.

But, what is a PatternSpawner? PatternSpawner is a gameObject in the scene that handles a PatternSpawner.cs component in it. Pattern Spawner basically allows you to spawn these patterns through code (very easily!). You will be able to define the amount of different patterns you want to be able to shoot, and then define the visuals for each of these patterns. The length of the Patterns and Graphics arrays MUST be the same.

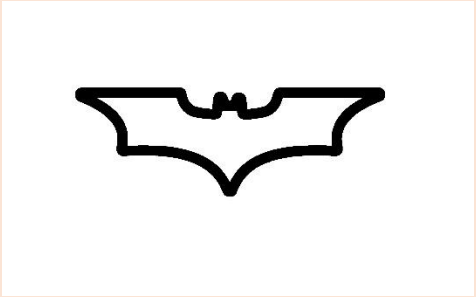


For more information, please visit PatternSpawner in the Asset Content section.

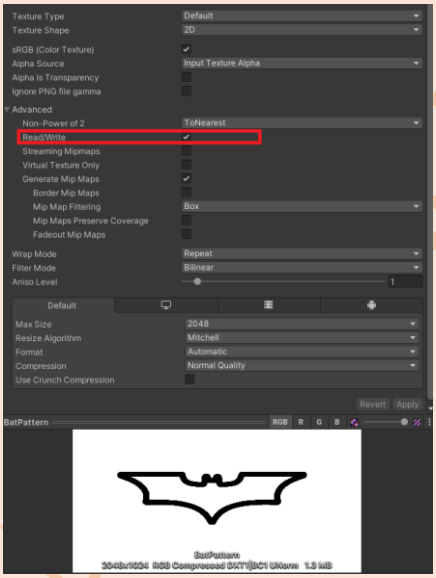
03.3 – IMAGE PATTERN

Bullet Hell Engine provides two ways of defining patterns, the default mode, and the image pattern mode. Before you keep reading, it is also recommendable that you visit Pattern_SO in the Asset Content section.

Here, the proper format of the images will be explained. Take the following image as our example.

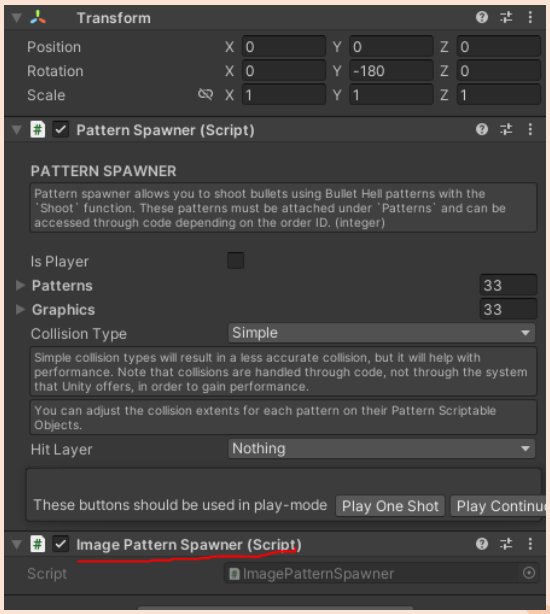


For the system to properly read the information of your image, the background must be white (or very close to white). On import, you also must set Read/Write to true, otherwise Unity won´t be able to read the information.



The system will store the black colour as information where a bullet can be spawned, so don´t fill your shapes if you only want to draw the contour.

For the Image Pattern to work, your PatternSpawner.cs MUST have the ImagePatternSpawner.cs attached.



Note that this pattern method is still experimental, so it might be unstable / less performant. It is highly recommendable to avoid high resolution images, but if you use high resolution images, make sure to read the next paragraph:

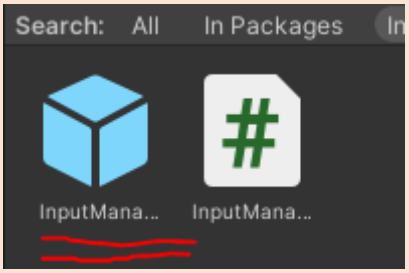
It is also highly recommendable to use bigger steps for both the x and y axis. The step (you can set that in the Pattern_SO, only for Image pattern method) is the amount pixels that the system will check. If your xStep is 30, for instance, the system will check every 30 pixels and determine whether to place or not a bullet. The higher this value is, the less pixels the system has to check, and so the more performant it is.

However, depending on the size of your image, you may encounter issues with the spawning of the bullets. If your image is very light and your step is high, the system will skip many important spawn points, so you have to look for a balance. Generally, with low res images, low step values work better, while for high res images, high step values are the best option.

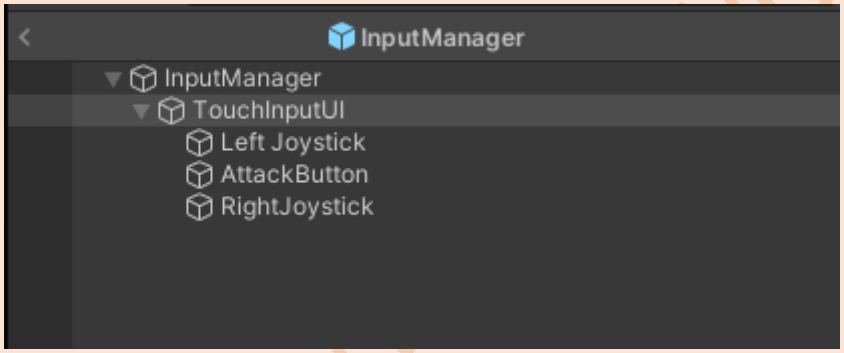
Note that the system stores these bullet positions in a concurrent bag (a concurrent bag is similar to a list in C#) at the start of the game, when you hit play, so it doesn´t have to generate the pattern again each time you shoot. This is done regarding performance and optimization.

03.4 – HOW TO USE TOUCH INPUT FOR MOBILE DEVICES

Bullet Hell Engine provides Touch Input Support. To enable it, search the InputManager in the Project Folder and open the prefab.



Here you can enable the TouchInputUI Canvas, as it is disabled by default.

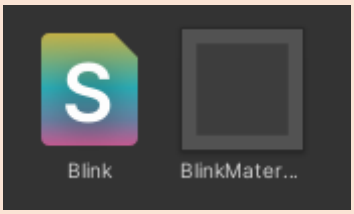


03.6 – FLASH | BLINK | HIT EFFECT BASICS

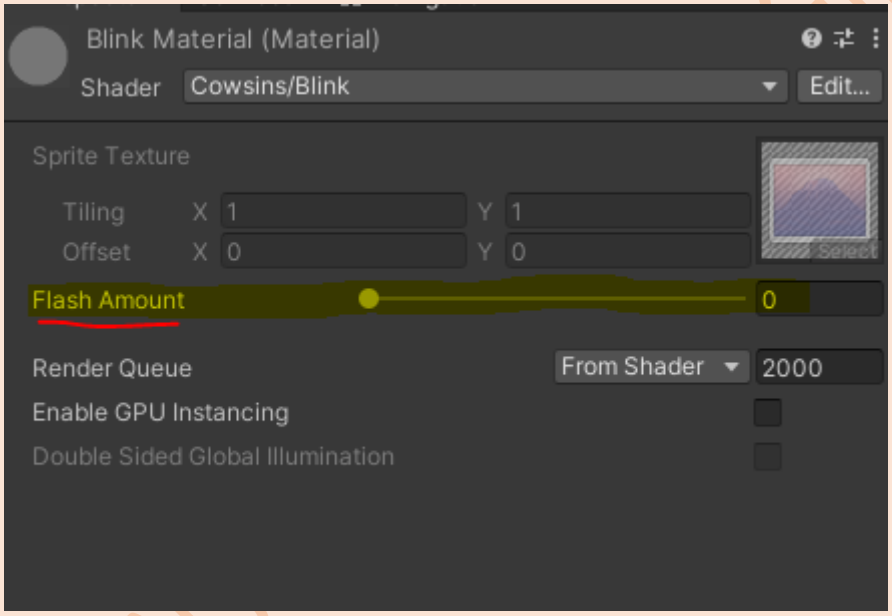
Before getting into details regarding the hit blink effect, it is very important to highlight the above, mentioned here. For version v.0.5, URP and HDRP users won´t have access to Blink.shader so this won´t work for them.

2D BLINK EFFECT

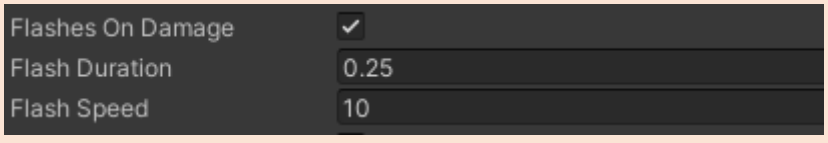
This one is handled through blink.shader and a material with this shader assigned.



This shader allows us to modify the blink amount, as seen in the following picture. However, you won´t need to modify this value, since its handled through code on your Enemy2D.cs script.



If you need your enemy to blink, please assign this material on your sprites! You can use the same material for all your sprites. Afterwards, feel free to modify the blink values.



03.7 – PLAYER CONTROL ACCESS

The player is handled by a controllable variable (Boolean). If controllable is true, you will be able to move around, shoot, etc... if controllable is false, you won´t be able to perform these activities.

controllable variable is stored inside PlayerHealth.cs, and it can be accessed from anywhere simply by writing PlayerHealth.controllable since it is a static varible. Notice that you cannot directly change its value.

In order to grant control to your player, you can use `PlayerHealth.GrantControl();`

In order to lose control of your player, use `LoseControl();`

```
public static bool controllable { get; private set; } = true;

public static void GrantControl() => controllable = true;

public static void LoseControl() => controllable = false;
```

You can read more about this in the PlayerHealth.cs section.

03.8- OPTIMIZATION RECOMMENDATIONS

When making a bullet hell game, you will probably end up having hundreds of bullets while playing. That is the reason why Bullet Hell Engine brings a pooling system. But what is a pooling system exactly?

A pooling system prevents the game from creating garbage when instantiating and destroying game objects. In most games, garbage is one of the most important issues regarding optimization if it is not treated. The pooling system, instead of destroying bullets, disables them and stores them “hidden” in an array. Whenever a new bullet should be shot, it checks for room in the pool. If there are enough bullets in there, the pool manager will enable a bullet instead of instantiating it. If there are no more available bullets, the system will be forced to instantiate them.

A pooling system increases performance considerably (it works great, actually!), but there are still considerations to take if you want your performance to be the greatest possible.

3D MODELS

This is mainly aimed at 2D games, but in case you need to use 3D models for your bullets, watch out, these are definitely not recommended. If you require these to be 3D, at least make sure they are low poly.

ANIMATIONS

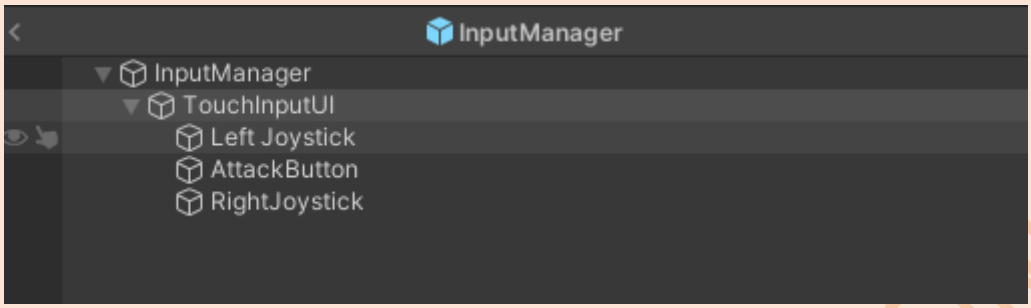
Your bullets can be animated. In fact, there are a bunch of examples of that in the asset. However, it is not recommended either. Try to replace animations for shaders or something similar every time you can!

BULLET LIFETIMES

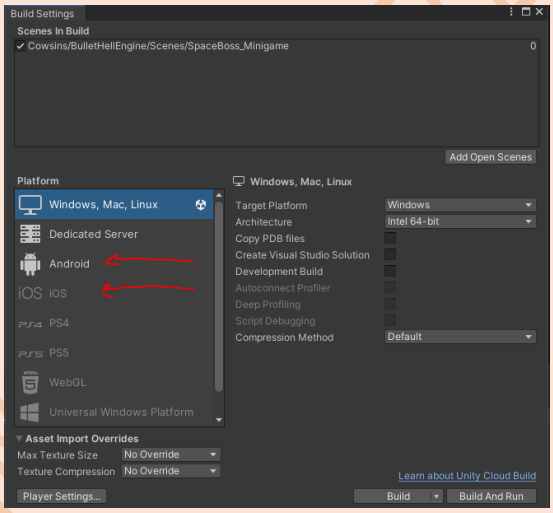
Adjust bullet durations / lifetimes so they will be sent back to the pool after a certain amount of time. Setting a high bullet duration time will result in having bullets travelling around the world with no purpose!

03.9- MAKE A MOBILE GAME WITH BULLET HELL ENGINE

To make a game for mobile devices, first we will need to enable touch input UI. For that, search InputManager.prefab and enable the touchInputUI object. You can modify this canvas as much as you want!



After that, go to the build settings (File/BuildSettings) and change the target platform in case you did not do it before. Notice that you must have this installed before.



After that, you can export the apk (for Android) and install it on your phone!

Check:

ANDROID

https://docs.unity3d.com/Manual/android-export-process.html

iOS

https://docs.unity3d.com/Manual/UnityCloudBuildiOS.html

04. ASSET CONTENT

Throughout this section, you will be able to find detailed explanations of the different scripts´ variables provided by Bullet Hell Engine.

It is organized the same way the scripts folder is organized; you will be able to find different sub-sections per each folder in the project tab inside Unity.

04.1 – CAMERA

CameraFOV.cs

Handles the field of view (FOV) of the camera.

fov	Float (private set)	Returns the initial Field of View of the camera.

FollowPlayer.cs

Attached on the parent of your camera, handles its movement.

Player: The camera will follow this transform object.

Speed: Lerp velocity. The higher the value, the faster it will reach Player´s destination.

LookAtPlayer: If true, the camera will point towards the player.

04.2 – DAMAGEABLE

IDamageable.cs

IDamageable.cs is an interface that handles damage systems of an object. Damageable.cs inherits from IDamageable.cs.

Damageable.cs

Handles health and damage systems of an object. Inherits from the interface IDamageable.cs.

MaxHealth: Initial and maximum health of your player.

MaxShield: Initial and maximum shield of your player.

health	Float (private set)	Current health of the damageable object.
shield	Float (private set)	Current shield of the damageable object.
isDead	Bool (private set)	Returns true if health < or = 0
TakeDamage()	Void PARAMETERS: damage – Damage to deal	Damages the damageable object. It affects both health and shield.
TakeHeals()	Void PARAMETERS: heal – Heal to apply	Heals the damageable object. It affects both health and shield.

Enemy.cs

Handles basic Enemy behaviour (related to damaging ONLY). Inherits from Damageable.cs.

BASIC

MaxHealth: Initial and maximum health of your player. (From Damageable.cs)

MaxShield: Initial and maximum shield of your player. (From Damageable.cs)

EFFECTS CUSTOMIZATION

DeathScoreMin: Minimum amount to add to the global score on dead. Notice that this will only work if ScoreManage exists in the scene (The player prefab brings it already).

DeathScoreMax: Maximum amount to add to the global score on dead. Notice that this will only work if ScoreManage exists in the scene (The player prefab brings it already).

FlashesOnDamage: If true, blink / flash effect will be applied on take damage.

FlashSpeed: How fast the blink effect is performed.

FlashDuration: How long it takes for the effect to finish. Notice that the material goes back to its original state when finished.

Sprites: Array that contains every sprite on which the effect will be applied. These sprites must have the Blink material assigned.

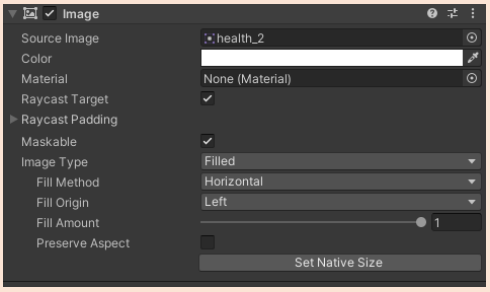
DeathVFX: Visual effect played on death.

HitSFX: Array that stores audio clips for taking damage.

UI

DisplayUI: if true, the enemy will display states through the user interface.

healthSlider: Image that displays the amount of health. It must have a sprite assigned and set to Filled.



shieldSlider: Image that displays the amount of shield. It must have a sprite assigned and set to Filled.

DamagePopUp: Assign a damage pop up prefab here. Bullet Hell Engine brings some examples that you can use.

XVariation: Variation on the x axis (locally) each time the pop up is instantiated.

PopUpDestroyTime: Damage pop up lifetime.

EVENTS

Events

SimpleEnemyExample.cs

Custom Enemy behaviour. This is an example so you can see how to make new enemies. Do not inherit from Enemy.cs (you can do it, but it is not needed).

PatternSpawner: Attach a pattern spawner. The enemy will shoot from there.

WeaponPivot: Pivot used to rotate the weapon sprite. This will aim towards the player.

MoveSpeed: Movement Speed of the enemy. Note that this enemy example performs random movements.

MinTimeToChangeDir: Minimum time to invert directions.

MaxTimeToChangeDir: Maximum time to invert directions.

MovementDirection: Starting movement direction of the enemy.

BossExample.cs

Custom Enemy behaviour (Boss). This is an example so you can see how to make new enemies. Do not inherit from Enemy.cs (you can do it, but it is not needed).

`muzzleSpawners:` Attach the spawners from the sides (array).

`GeneralSpawner:` Attach the central spawner.

04.3 – EFFECTS

CamShake.cs

Allows the player to perform a camera shake effect.

<table>
<tr><td>Shake()</td><td>Void
PARAMETERS:<table><tr><td>amount</td><td>Amount of camera shake</td></tr><tr><td>_power</td><td>Lifetime intensitty</td></tr><tr><td>_movementAmount</td><td>Position change amount</td></tr><tr><td>_rotationAmount</td><td>Rotation change amount</td></tr></table></td><td>Apply a camera shake effect with full control of the variables.</td></tr>
<tr><td>ShootShake()</td><td>Void
PARAMETERS:<table><tr><td>amount</td><td>Amount of camera shake</td></tr></table></td><td>Apply a camera shake effect set for a shot.</td></tr>
<tr><td>ExplosionShake()</td><td>Void
PARAMETERS:<table><tr><td>distance</td><td>Amount of camera shake</td></tr></table></td><td>Apply a camera shake effect set for explosions.</td></tr>
</table>

NOTE: YOU CAN CALL THESE METHODS ANYWHERE:
CamShake.instance.ShootShake(1);

ScrollingBackground.cs

Allows an object to scroll infinitely.

Requires Rigidbody2D and BoxCollider2D

moveOnStart: If true, the object will automatically move from start.

xScroll: You MUST turn this on if your scroll works on the x axis. You MUST turn this off if it only works on the y axis.

maxVelocity: maximum velocity allowed.

initialVelocity: initial velocity of the object.

acceleration: How fast the velocity increases.

currentVelocity	Vector2 (private set)	Returns the current velocity vector of the object.
canMove	Bool (private set)	Returns true if the object can be moved. False if not. You can change this value using AllowMovement() and DisallowMovement()
AllowMovement()	Void. No parameters.	Allows the object to start moving.
DisallowMovement()	Void. No parameters.	Stops the object from moving.

04.4 – EXTRA

ApplyConstantMovement.cs

Applies a certain constant movement to an object. This is used for asteroids in the demos etc…

`moveOnStart:` If true, the object will automatically move from start.

`movement:` Movement directions applied. It is recommendable to set a normalized vector.

`rotation:` Rotations applied. (No need to normalize it)

`maxSpeed:` Maximum speed allowed.

`initialSpeed:` Initial speed of the object.

`acceleration:` Capacity of gaining speed.

canMove	Bool (private set)	Returns true if the object can move.
currentSpeed	Float (private set)	Returns the current speed.
AllowMovement()	Void. No parameters.	Allows the object to start moving.
DisallowMovement()	Void. No parameters.	Stops the object from moving.

DestroyMe.cs

Destroys an object when the current animation is done. Requires an animator component attached.

GetGameInformation.cs

Displays game information such as current FPS, minimum FPS, maximum FPS. You can also set the target framerate. By default this is set to 144.

LookAtCamera.cs

Forces an object to look at the main camera in the scene.

PauseMenu.cs

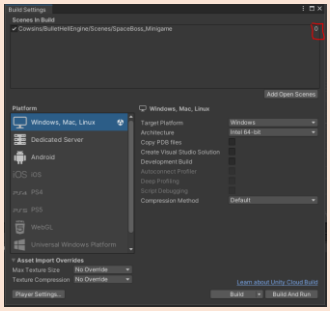
Handles the pause and unpause behaviour and UI (Pause Menu)

Note that PauseMenu is a singleton and its public methods/variables can be accessed like the following:

`PauseMenu.pauseMenu.isPaused`

`PauseMenu.pauseMenu.Pause()`

`mainMenuOrder:` Assign the order ID of your main menu in the Build Settings (File/BuildSettings)



isPaused	Bool (private set)	Returns true if the game is paused.
Pause()	Void. No parameters	Pauses the game.
Unpause()	Void. No parameters	Unpauses the game
TogglePause()	Void. No parameters	If the game is paused, un-pause it. If it is not paused, pause it.
MainMenu()	Void. No parameters	Load the main menu scene.

Asteroid.cs

Handles the asteroid behaviour shown in the demos. Notice that there is an asteroid prefab that you can use if needed.

This damages the player on trigger.

Requires a SpriteRenderer component attached.

damage: Damage dealt to the player on trigger.

randomSprites: Array that contains sprites. At start, the sprite will be randomly selected among these.

ExtraSpawnables.cs

Instantiates extra objects in the scene. In the demos, this is placed on the game managers and spawns health packs, asteroids etc…

extras: Array that contains the game objects to spawn.

spawns: Spawn points (Transforms) where the objects will be spawned.

spawnInterval: Instantiate pace. The lower this value, the faster these objects will be spawned.

04.5 – MANAGERS

PoolManager.cs

The Pool Manager is a system that helps to improve the performance of a game significantly. The pool managers stores objects in an array, and instead of generating garbage by destroying and instantiating new instances of the same object, it enables and disables them, returning them back to the pool.

Pool Manager is a singleton, it can be accessed like the following:

```
PoolManager.Instance.RequestBullet();
PoolManager.Instance.ClearPool();
```

bullet: Attach the generic prefab for the bullets: Cowsins_Bullet.prefab

bulletPool: DO NOT ATTACH ANYTHING. This is shown to control the bullets in the pool currently.

bullets: Number of bullets to store in the pool. Notice that all of these bullets will be instantiated at once on start (Pool Initialization). Notice as well that it is STRONGLY RECOMMENDED to tweak this value until you are satisfied. It makes no sense to set such a high value that a large amount of the bullets stored in the pool are never used. As well as it makes no sense to set such a low value that the system needs to spawn new instances each time.

RequestBullet()	Bullet. No Parameters	Returns a bullet from the pool
ClearPool()	Void. No parameters	Sends back all the bullets to the pool

ScoreManager.cs

Handles scoring. The object that has this component attached is generally included in the player controller prefab.

Score manager is a singleton, it can be accessed like the following:

```
PoolManager.Instance.score;
PoolManager.Instance.AddScore(amount);
PoolManager.Instance.RemoveScore(amount);
```

`canScoreNegative:` If true, the score can be negative. If false, the minimum score will be clamped to 0.

`scoreDisplay:` UI text that displays the current score. This uses TMPro (TextMeshPro).

`scoreEffectAmount:` the scoreDisplay performs a code-driven animation when the score value changes, this is the intensity of the animation.

`scoreEffectSpeed:` the scoreDisplay performs a code-driven animation when the score value changes, this is the lerp speed to its original state.

score	Integer (private set)	Returns the current score.
AddScore()	Void PARAMETERS: amount (int) — Amount to add	Adds a certain number of points to the player score.
RemoveScore()	Void PARAMETERS: amount (int) — Amount to remove	Remove a certain number of points to the player score. Score will be reset to 0 in case it is negative and that´s not allowed. You can allow negative values for the score on the ScoreManager. Check "canScoreNegative".

SoundManager.cs

Handles sounds. The object that has this component attached is generally included in the player controller prefab.

Sound manager is a singleton, it can be accessed like the following:

```
SoundManager.Instance.PlaySound(clip,delay,pitch,spatialBlend);
```

<table>
<tr>
<td>PlaySound</td>
<td>Void
PARAMETERS:
<table>
<tr><td>Clip (AudioClip)</td><td>Audio (SFX) that you want to play.</td></tr>
<tr><td>Delay</td><td>Units of time to play the sound.</td></tr>
<tr><td>Pitch</td><td>Specifies the pitch of the AudioSource.</td></tr>
<tr><td>SpatialBlend</td><td>SpatialBlend goes from 0 to 1.
0 meaning completely 2D sound and 1 meaning completely 3D sound</td></tr>
</table>
</td>
<td>Plays a custom sound on the SoundManager, given the custom parameters. In most of the cases, Bullet Hell Engine already handles this.</td>
</tr>
</table>

ShowroomBulletSpawner.cs

This script works for the bullet spawner showroom. It showcases different bullet patterns, especially for the DEMO, so it won´t have much value for your game in most of the cases.

spawner: Select the pattern spawner to work with.

displayPatternText: Text that displays the current pattern and its name. Notice that this works using TMPro (TextMeshPro).

Next()	Void. No Parameters.	Plays next pattern. If the current pattern is the last one, it will go back to the first one.
Previous()	Void. No Parameters.	Plays the previous pattern. If the current pattern is the first one, it will travel to the last one.

MiniGameManager.cs

Specially designed for the DEMO, but you can use it for your own game too! Handles basic behaviour for a simple mini game loop. Aimed at arcade style, mainly.

Player: Assign your player object. When the player is not in the scene, that will count as GameOver. (If the player is not in the scene or it is disabled, it means it died).

MainMenu: MainMenu Canvas. When leaving the main menu, this will be lerped. It MUST have a CanvasGroup assigned.

ObjectsToEnableOnPlay: On Play, objects to enable. This can be done through the events as well.

ObjectsToDisableOnPlay: On Play, objects to disable. This can be done through the events as well.

mainMenuFadeSpeed: MainMenu fade out effect speed.

Enemies: Array that contains all the possible enemies to instantiate / spawn.

EnemiesToSpawn: Maximum number of enemies to spawn. Once all the enemies have been killed, the game state will change to Victory.

SpawnInterval: Once an enemy is dead, time remaining to spawn a new one. Set this to 0 for instant spawn.

MaxSimultaneousEnemies: Maximum number of enemies that can be spawned at the same time. If this value is set to 3, you will only be able to fight against three enemies at the same time.

SpawnPoints: Transforms / Locations where these enemies will randomly be spawned.

EnemiesLeftDisplay: Text that displays the enemies left. Notice that this works using TMPro (TextMeshPro).

Events: Events

<table>
<tr><td>gameState</td><td>GameState (Private set).

GAME STATES AVAILABLE (6 IN TOTAL):
MAIN MENU › PREPARING › PLAYING
GAME OVER
VICTORY
OTHER</td><td>Returns the current GameState.

Example of return value:

GameState.Playing</td></tr>
<tr><td>Play()</td><td>Void. No Parameters.</td><td>Changes the game state to Playing and runs Play logic on game state changed.</td></tr>
<tr><td>GameOver()</td><td>Void. No Parameters.</td><td>Changes the game state to GameOver and runs GameOver logic on game state changed.</td></tr>
<tr><td>HandleVictory()</td><td>Void. No Parameters.</td><td>Changes the game state to Victory.</td></tr>
<tr><td>RestartScene()</td><td>Void. No Parameters.</td><td>Restarts the current scene.</td></tr>
<tr><td>LoadScene()</td><td>Void
PARAMETERS:
<table><tr><td>i
(integer)</td><td>Scene to load by its order ID. (check this in File/BuildSettings)</td></tr></table></td><td>Loads certain scene given its order ID (check this in File/BuildSettings)</td></tr>
</table>

BossMiniGameManager.cs

Specially designed for the DEMO, but you can use it for your own game too! Handles basic behaviour for a simple mini game loop for a boss scene. Aimed at arcade style, mainly.

MainMenu: MainMenu Canvas. When leaving the main menu, this will be lerped. It MUST have a CanvasGroup assigned.

mainMenuFadeSpeed: MainMenu fade out effect speed.

Boss: Boss object.

initialBossPosition: What the initial position of the boss should be. You can place it anywhere else, and the boss will move towards the position you specify here. This is a very simple animation through code.

bossSpeed: For the animation stated before, lerp speed from the current position to the initial position of the boss.

Events: Events

gameState	GameState (Private set). GAME STATES AVAILABLE (6 IN TOTAL): MAIN MENU › PREPARING › PLAYING GAME OVER VICTORY OTHER	Returns the current GameState. Example of return value: GameState.Playing
Play()	Void. No Parameters.	Changes the game state to Playing and runs Play logic on game state changed.
Prepare()	Void. No Parameters.	Changes the game state to Preparing and runs Prepare logic on game state changed.
GameOver()	Void. No Parameters.	Changes the game state to GameOver and runs GameOver logic on game state changed.

Victory()	Void. No Parameters.	Changes the game state to Victory and runs Victory logic on game state changed.

04.6 – PATTERN

This section handles everything related to the pattern creation, shooting etc... This is the core of the asset.

Before getting into details, let´s summarize how this system works.

For you to be able to shoot bullets using Bullet Hell Engine´s system, you MUST have an object in your scene with a PatternSpawner.cs component attached. (Depending on what type of pattern you want, you may also want to attach the ImagePattern.cs). You can have many of these pattern spawners around the scene.

When you call the shoot function on a Pattern Spawner, this will read the information of your Pattern Scriptable Object (Pattern_SO), and instantiate bullets depending the rules and settings of it. These bullets have a Bullet.cs component attached. By default, there is a bullet prefab that you can use for your bullets, this is called Cowsins_Bullet.prefab.

You can check a visual scheme of how this works below. Notice that this is explained in a section above. Click here to read.

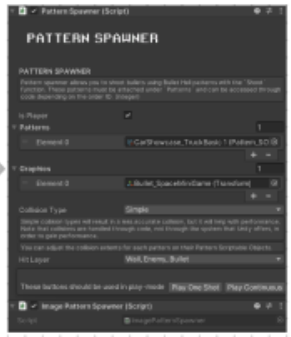
OBJECT IN THE SCENE

Your game object.
COMPONENTS ATTACHED:
- PatternSpawner.cs
- ImagePattern.cs (only if you have ImagePattern patterns to shoot in your patterns array)

Reads Pattern_SO information from the Patterns Array

Instantiates Bullets. These Bullets have the Bullet.cs attached.
RECOMMENDATION: Use the Cowsins_Bullet prefab. This is assigned on the PoolManager

EXAMPLE OF PATTERN SPAWNER OBJECT



COMMON ERROR

YOU MUST ASSIGN A GRAPHIC ELEMENT TO EACH OF YOUR PATTERNS. IF YOU HAVE 4 PATTERNS, ASSIGN 4 GRAPHIC ELEMENTS. THIS WILL BE...

EXAMPLE OF PATTERN SCRIPTABLE OBJECT

Bullet.cs

Attached to the Cowsins_Bullet.prefab. You can create new bullet prefabs though, but it is highly recommended to use this prefab, since it is already made and there is basically no reason to make a new prefab. The bullet is just the object that moves and handles the logic for both player and enemy Bullets.

ShootingModes.cs

Static class used by the PatternSpawner to define the shooting mode when shooting certain pattern. The Shooting Method is defined in an enumerator called Mode.

`ShootingModes.Mode.DefinedAmount;`

(Allows you to set a specific number of shots)

`ShootingModes.Mode.Continuous;`

(Used to stablish continuous shooting depending on your pattern fire rate. This will be continuous until you call StopShooting() on your PatternSpawner or call a new Shoot() function on your PatternSpawner)

PatternSpawner.cs

Manages the bullets instantiating through patterns (Pattern_SO). It also helps you defining the visuals of these. Pattern spawner allows you to shoot bullets using Bullet Hell patterns with the `Shoot` function. These patterns must be attached under `Patterns` and can be accessed through code depending on the order ID. (integer).

`isPlayer:` Set this to true if the pattern spawner belongs to the player.

`Patterns:` Array that contains all the pattern_SO that this PatternSpawner is capable of shooting.

`Graphics:` Graphic element for the bullet of each pattern in the Patterns array. There MUST be the same number of patterns and graphics elements.

CollisionType: Selects collision method. Simple collision types will result in a less accurate collision, but it will help with performance.

Accurate collision types will result in a more precise collision, but it will harm the performance, since it uses circular / spherical collisions.

Note that collisions are handled through code, not through the system that Unity offers, to gain performance.

HitLayer: Layers that this pattern spawner bullets can collide with. It is very important to highlight that this layer mask ONLY AFFECTS the bullets instantiated by THIS pattern spawner.

<table>
<tr><td>readyToShoot</td><td>Bool (Private set)</td><td>Returns true if the system is ready to shoot.</td></tr>
<tr><td>currentPattern</td><td>Integer (Public set)</td><td>Returns the current pattern set.</td></tr>
<tr><td>spinAmount</td><td>Float (Private set)</td><td>Returns the current spin amount.</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>Shoot()</td><td>Void.
PARAMETERS:<table><tr><td>Pattern (int)</td><td>Order ID of the pattern in the patterns array of the PatternSpawner.</td></tr><tr><td>_mode (ShootingMode.Mode)</td><td>Shooting Mode required. Choose between DefinedAmount and Continuous shooting. Check ShootingModes.cs.</td></tr></table></td><td>Allows a pattern spawner to shoot a pattern. It shoots only once for defined amount shooting modes.</td></tr>
<tr><td>Shoot()

Recommended.</td><td>Void.
PARAMETERS:<table><tr><td>Pattern (int)</td><td>Order ID of the pattern in the patterns array of the PatternSpawner.</td></tr><tr><td>_mode (ShootingMode.Mode)</td><td>Shooting Mode required. Choose between DefinedAmount and Continuous shooting. Check ShootingModes.cs.</td></tr><tr><td>Times (int)</td><td>For defined amount shooting modes, amount of times to shoot.</td></tr></table></td><td>Allows a pattern spawner to shoot a pattern. It also allows to set a finite number of times to shoot, in case the shooting mode selected is not continuous.</td></tr>
<tr><td>StopShooting()</td><td>Void. No parameters.</td><td>Stops the shots no matter what.</td></tr>
</table>

Pattern_SO.cs

This is the main part of the pattern system. This handles the scriptable object that stores patterns. You can create a new pattern scriptable object by right clicking on the projects folder / Create / COWSINS / New Pattern

patternName: You can assign a name to your pattern for better organization. This is optional though.

patternMethod: Determines the style of the pattern.

> pattern: Image that describes the shape of the pattern. Check this out for more information regarding this topic.
>
> patternSize: Size of the pattern. Adaptative distance from the center.
>
> xStep: The step determines the amount of pixels that the system will check. The higher this value is, the lower number of pixels it checks, so it´s more performant. This defines the horizontal step.
>
> yStep: The step determines the number of pixels that the system will check. The higher this value is, the lower number of pixels it checks, so it´s more performant. This defines the horizontal step.

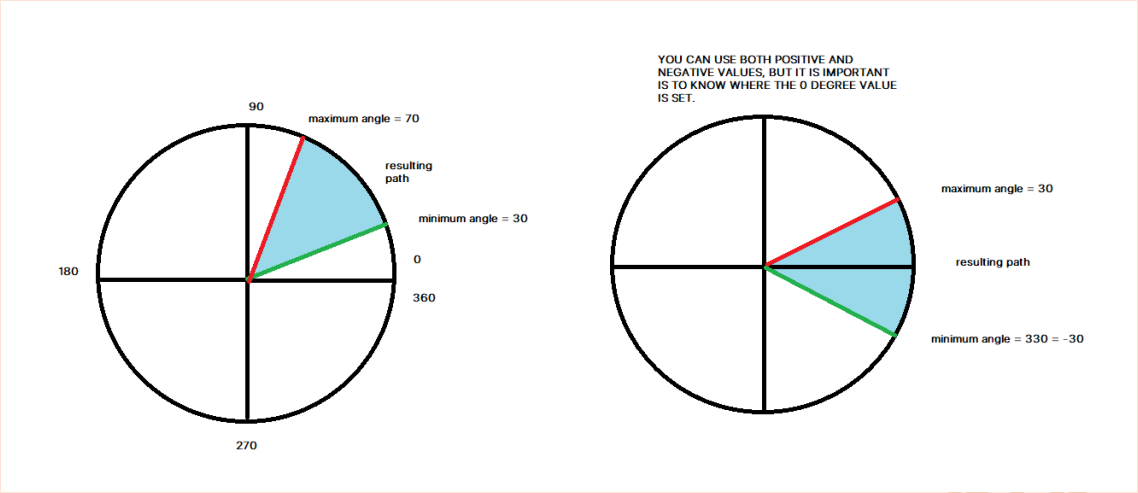
angleStyle: Automatic angle style will instantiate the bullets all around 360 degrees, while you will be able to set them manually for Manual Angle Styles.

increaseAngle: Set to true if you want to modify the angle during gameplay.

influenceRotationByParent: Set this to true if the pattern should depend on the parent rotation.

initialMinimumRotation: For manual angle styles, minimum degree value for the angle.

initialMaximumRotation: For manual angle styles, maximum degree value for the angle.



`minimumRotationStep:` Amount in degrees that the minimum rotation changes each time you shoot.

`maximumRotationStep:` Amount in degrees that the maximum rotation changes each time you shoot.

`clampMinimumRotation:` Limit value for the minimum rotation.

`clampMaximumRotation:` Limit value for the maximum rotation.

`numberOfProjectiles:` Projectiles shot (per fire)

`shootCooldown:` For continuous shooting, shooting interval.

`spawnDistanceFromEmitter:` Distance from the center of the emitter. (PatternSpawner location, since it acts as the emitter)

`spinAmount:` Bullet spin amount per fire. Set to 0 for null spin.

`invertSpinDirections:` Set this to true if the pattern spawner should invert spin directions. (Used to create effects such as flowers etc…)

`spinAccelerationAmount:` Bullet spin acceleration.

`maxSpinAmount:` limit spin velocity.

`projectileDuration:` Projectile lifetime. Keep this as low as possible for better performance results.

`applyRandomness:` If true, random rotations will be applied, always respecting the limits set.

`useSpread:` Set to true if you want to use spread.

`minSpread:` Minimum amount of spread.

`maxSpread:` Maximum amount of spread.

`idleTime:` Time for the bullet to start moving once it gets spawned.

damagePerHit: Damage dealt to the IDamageable that it collides with.

projectileSpeed: Velocity of the bullet. Set to negative speed to invert velocity values.

adaptativeSpeed: Set this to true for the system to calculate the appropriate speed based on the distance from the emitter. This is most used for image-based patterns or patterns with vertex numbers greater than 0.

setMaxProjectileSpeed: Limit the projectile velocity.

maxProjectileSpeed: maximum allowed projectile velocity.

projectileAcceleration: Set projectile acceleration. 0 for null acceleration = constant speed.

projectileSizeMultiplierOverTime: Amount of growth of the player per frame. Positive values make the bullet bigger, while negative values make it smaller.

changeDirections: Set this to true if the pattern should change directions based on the timer.

changeDirectionTimer: Amount of time before the directions change. Note that, this value is divided by two for the first direction change, since it comes from the emitter and the distance required to travel is half.

vertexNumber: Allows to simulate polygons. 0 means no vertex

(circular), 3 vertex number is a triangle, 4 vertex number is a square etc… Notice that it is based on sinus shapes, so it will not be a perfect polygon shape.

extentsSize: Collision amount. Enable Gizmos in play mode to see the size of the collisions. Notice that the collisions are handled through code to improve performance. Bullet Hell Engine does not use the Collider system by Unity for the bullets of the pattern spawner.

shotSFX: Audio clip played when shooting. Set to null if you do not want to play an audio.

pitchVariationSFX: Pitch variation of the shotSFX.

ImagePattern.cs

This MUST be attached to a pattern spawner, since it is complementary. It should only be attached when the pattern spawner has a pattern in its array that uses ImagePattern method.

Handles the reading, storing, and initialization of the pattern based on images.

04.7 - PLAYER

Crosshair.cs

Handles the crosshair object behaviour. It automatically gets destroyed for Android and iOS builds.

PlayerModifiers.cs

Stores the modifiers used by the player and power ups systems.

Notice that it is a singleton. It can be accessed like this:

```
PlayerModifiers.Instance.movementSpeedModifier;
```

movementSpeedModifier	Float (Public set)	Multiplies the player movement speed.
damageDealtModifier	Float (Public set)	Multiplies the damage dealt by the player.
damageReceivedModifier	Float (Public set)	Multiplies the damage received by the player.
healReceivedModifier	Float (Public set)	Multiplies the heal received by the player.

PlayerMovement.cs

Manages the player movement, rotation, and advanced movement actions such as dashing.

Requires a box collider 2D attached.

cam: Assign the player camera.

allowXMovement: Set to true if the player should be able to move along the horizontal axis.
allowYMovement: Set to true if the player should be able to move along the vertical axis.

acceleration: Capacity of gaining velocity.

rotates: Set to true if the player can aim towards the mouse (or joystick if using a gamepad).

- rotationSpeed: For gamepads, rotation speed.

canDash: Set to true if the player can dash.

- amountOfDashes: Maximum amount of dashes you can perform / store. These will be regenerated after being used.
- dashCooldown: Regeneration time after dashing.
- dashSpeed: Player speed while dashing.
- dashTime: Amount of time the player is in the dash state.
- canShootWhileDashing: If true, the player will be able to shoot while dashing.
- receiveDamageWhileDashing: If true, you will be able to receive damage while dashing.
- dashUIContainer: Contains as a layout all the dash UI objects.
- dashUIObject: Used to display the number of dashes left.

events

dashTimer	Float (Private set)	Dash Time remaining.
dashing	Bool (Private set)	Returns true if dashing.

PlayerHealth.cs

Manages the player damage and health system. Inherits from Damageable.cs

maxHealth: Maximum and initial health of the player.

maxShield: Maximum and initial shield of the player.

damagedSFX: Audio Clip played on damage.

healSFX: Audio Clip played on healing.

pitchVariationSFX: SFX pitch variation (random between negative and positive value)

damageCamShakeAmount: CamShake amount on hit (damage)

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

- sprites: Array that contains all the spriteRenderers that will blink on damage. These must have the Blink material assigned.
- flashDuration: How long the effect is played.
- flashSpeed: How fast the effect is played.

healthSlider: Image that displays health. This must have a sprite assigned and set to Filled.

shieldSlider: Image that displays shield. This must have a sprite assigned and set to Filled.

playerStateUI: Image used to display effects like hurt or heal.

damageColor: playerStateUI color when damage.

healColor: playerStateUI color when heal.

fadeSpeed: Speed of fade out effect for the playerStateUI color.

events: Assign the player camera.

Check Damageable functions for more info:

controllable	Bool. Private set	Returns true if the player can perform actions.
GrantControl()	Void. No Parameters	Allow the player to perform actions.
LoseControl()	Void. No Parameters	Disallow the player to perform actions.

PlayerAttack.cs

Handles player shooting. This system uses the Pattern_SO.

spawner: Attach a pattern spawner, we´ll use that to manually shoot.

fireRate: Assign a fire rate for your patterns. For this system, we won´t use the fireRate in the Weapon_SO as we won´t shoot continuously.

camShakeAmount: Amount of camera shake effect on shot.

muzzle: Array that contains all the points where muzzleFlashVFX will be instantiated. You can orientate the muzzlePoint so the muzzleFlashVFX will math that rotation.

muzzleFlashVFX: Visual effect played on each muzzle point on shot.

fireSFX: Audio Clip played on Shot.

pitchVariationFireSFX: Fire sfx pitch variation (random between negative value and positive value)

events

04.8 – POWER UPS

IncreaseMovementSpeedPowerUp.cs

`movementSpeed:` value added to the movement speed modifier in PlayerModifiers.cs

IncreaseHealReceivedPowerUp.cs

`healReceived:` value added to the heal received modifier in PlayerModifiers.cs

DecreaseDamageReceivedPowerUp.cs

`damageReceived:` value added to the damage received modifier in PlayerModifiers.cs

HealthPack.cs

`health:` Heal amount to take (for Player).

IncreaseDamageDealthPowerUp.cs

`damageDealt:` value added to the damage dealt modifier in PlayerModifiers.cs

05.THIRD-PARTY RECOGNITION

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

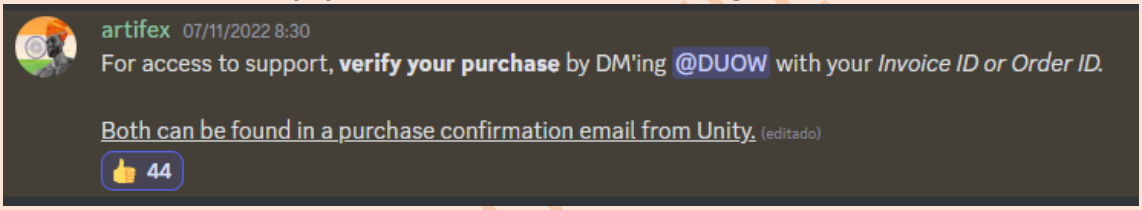
https://sfxr.me/

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. Note that you will need to verify your purchase before doing that.



1ST OPTION: Contact the owner of the server.

2ND OPTION: Go under "BULLET HELL 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