BLInformatique — Real FPS

⚓ 💪TEAM VICTOR & BORIS
Unity 6000.2.3f1 • Generated 2025-09-11 13:18

Theme

↑

📖 Overview

RealFPS ⚓💪 – Complete FPS Framework for Unity

🎮 RealFPS is a modular and extensible First-Person Shooter system built for Unity 6000.

It provides everything you need to create immersive and responsive FPS gameplay.

✨ Key Features

🎮 Complete FPS character controller

❤️‍🔥 Stamina, health & energy systems

🎒 Inventory with equipable, usable & weapon items

🔑 Dynamic interactions (doors, drawers, ladders, tools…)

🥽 IK integration for arms & weapons

🛠️ Custom editor tools (Arms Live Tuner, Sprite Creator, Manual Builder)

🚀 Modular add-ons (climbing, swimming, vehicles…)

⚙️ Fully configurable, scalable, and designed to save you months of development.

Created with passion by ⚓💪 TEAM VICTOR & BORIS.

🛠️ How to Use

1. Importing RealFPS 📦

Download and import the RealFPS package into your Unity project.

Make sure you are using Unity 6000.2.3f1 or higher for full compatibility.

All scripts and tools will be placed under:

Assets/BLInformatique/RealFPS/

2. Setting Up the Scene 🧱

Drag the RealFPS Player Prefab (FPSControllerPro or RealFPSController) into your scene.

Add a Main Camera 📷 if not already present.

Ensure your project uses the New Input System 🎮 (enable in Project Settings).

3. Configuring Layers & Tags ⚙️

Player layer: Player

Interaction layer: Interactable

Weapon camera layer: WeaponCam

Fire & torch interactions: Fire (default tag: Torch)

4. Testing the Demo Scene 🚀

Open DemoScene.unity inside the RealFPS folder.

Press Play ▶️ and explore:

Movement & stamina system

Picking up & equipping items

Using weapons and tools

Interacting with doors, drawers, ladders

5. Editor Tools 🛠️

Arms Live Tuner 🖐️: Adjust arm position & IK live during play mode.

Sprite Creator 🖼️: Capture and generate inventory icons.

Manual Builder 📖: Generate your own project documentation.

6. Extending RealFPS 🔧

Create new items via ItemData ScriptableObject.

Add new interactions by attaching InteractionBase-derived scripts.

Plug in your own animations or switch to IK-only mode.

⚓💪 Tip from TEAM VICTOR & BORIS: Start with the demo, then customize step by step. RealFPS is modular — you don't need everything at once.

🏁 Conclusion

RealFPS is more than just a set of scripts – it is a foundation for building immersive FPS games.

From movement 🎮 to weapons 🔫, from stamina ❤️‍🔥 to interactions 🔑, everything is designed to be modular, customizable, and developer-friendly.

By combining IK 🤖, editor tools 🛠️, and smart design choices, RealFPS helps you focus on what truly matters: creating unique gameplay experiences.

Whether you are a solo indie dev or part of a larger crew 🚀, RealFPS adapts to your workflow and scales with your project.

And remember: RealFPS is built with passion by ⚓💪 TEAM VICTOR & BORIS.

We've charted the course — now it's your turn to sail and create worlds where players will live unforgettable adventures. 🌊🏴‍☠️

CONTENTS

🔎 Filter components…

Tooltips are in **English**. Tables list serialized fields. Optional notes (Markdown) and images are shown below each component when present.

↑

Copy fields

📘 ActionArmsData ScriptableObject

ActionArmsData 🙌

The ActionArmsData is a ScriptableObject used to define how the player's arms behave during special actions (healing, drinking, eating, using tools, etc.).

It centralizes animation, prefab positioning, audio, and FX settings for a given arm action.

Prefab Settings ⚙️

Action Arms Prefab 🧩

Prefab that contains the arm(s), hand, and optionally an object (like a bottle or medkit).

Arms Position Offset 📍

Local position adjustment to correctly align the arms prefab with the player view.

Arms Rotation Offset 🔄

Local rotation adjustment for fine-tuning alignment.

Arms Animation 🎮

Action Trigger Name ▶️

Animator trigger that plays the action animation (e.g., "Heal", "Drink", "Eat").

Continual Action 🔁

If enabled, the action loops or continues as long as the input is held (useful for drinking or eating).

Action Duration ⏱️

Defines how long the action lasts (ignored if continual action is enabled).

Audio & FX 🎧

Action Audio 🔊

Audio clip that plays during the action.

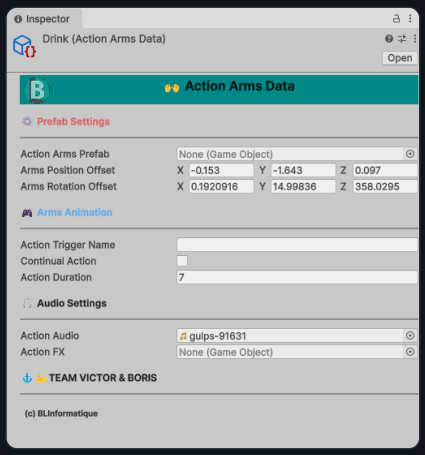
↑

Action FX 💥

Prefab instantiated during the action (for example, particles, visual effects).

⚓💪 TEAM VICTOR & BORIS Note

ActionArmsData makes arm actions modular and reusable. Instead of hard-coding every animation and sound, you create multiple ActionArmsData assets — one for each type of action — and assign them to items or systems in RealFPS.



Field	Type	Tooltip
⬇		
actionArmsPrefab	`GameObject`	Action arms prefab (example: arm(s)+hand+object).
armsPositionOffset	`Vector3`	Local position offset for arms prefab.
armsRotationOffset	`Vector3`	Local rotation offset for arms prefab.
⬇		
actionTriggerName	`String`	Animator trigger name to play (Heal, Drink, Eat, etc.).
continualAction	`Boolean`	If enabled, action will loop or play as long as the trigger is held (for continuous actions like drinking, eating, etc.).
actionDuration	`Single`	Duration of the animation in seconds (ignored if 'continualAction' is true).
⬇		
actionAudio	`AudioClip`	Audio clip to play during the action.
actionFX	`GameObject`	FX prefab to spawn during the action.

↑ Back to top

Copy fields

📘 ArmsAndWeaponData ScriptableObject

ArmsAndWeaponData 🧮

Purpose. ArmsAndWeaponData is a ScriptableObject that defines a complete “arms + weapon/tool” setup (pistol, flashlight, melee, gadgets…).

It centralizes prefab offsets, ADS (Aim Down Sight) settings, animation triggers, audio, shooting, ammo, and — for non-firearms — energy usage.

Prefab Settings ⚙️

Arms Prefab 🧩

Prefab containing arms, hand(s), and optionally the object (weapon/tool).

Weapon Name 🏷️

Display name used in UI.

Weapon Icon 🖼️

Optional sprite for inventory/HUD.

Arms Position Offset / Rotation Offset 📍🔄

Local offsets for proper alignment with the camera.

Is Firearm 🔫

Enables weapon-specific settings (shooting, ammo, crosshair). Keep disabled for tools or devices (flashlight, goggles, etc.).

ADS – Aim Down Sight 🎯 (firearms only)

Use ADS 🎯

Enables aiming (typically bound to right mouse button).

ADS Position / Rotation Offsets 📍🔄

Adjusts pose when aiming down sights.

ADS Camera FOV 🔭

Camera zoom while aiming.

ADS Trigger Name ▶️

Animator trigger/state for aiming pose.

↑

Hide Crosshair On ADS ❌✚

Optionally hides crosshair while aiming.

📂 Copy Arms Offsets to Arms Aiming Offsets (Editor Only) 🧰

Copies normal offsets into ADS offsets as a starting point.

Arms Animation 🎮

Animator Trigger Names ▶️

Idle, Walk, Run, Fire, Reload, Heal, etc.

Use consistent naming across prefabs for easier Animator reuse.

Audio 🔊 (firearms)

Fire / Reload Audio Clips 🔫🔁

Audio Volume 🎚️

Empty Clip Audio 🪫 (played when magazine is empty).

Fire Settings 🔫 (firearms)

Fire Mode (SemiAuto / FullAuto).

Fire Rate ⏱️ (shots per second in FullAuto).

Max Shoot Distance 📐.

Damage Per Shot 💥.

Muzzle Flash 🔥

Muzzle Flash Prefab ✨ – Particles/VFX instantiated on shot.

Muzzle Flash Point Name 🎯 – Transform name in prefab (default: root if empty).

Ammo Settings 🔋 (firearms)

Use Ammo ✅ – Toggles ammo system.

Initial Magazine / Reserve 📦 (-1 = start full).

Max Ammo / Magazine Size 📈.

Infinite Ammo ♾️ – Ignores reserve, useful for demos/tests.

Energy Settings ⚡ (non-firearms)

Requires Energy ⚡ – For flashlight, goggles, or devices.

Energy Max / Drain Per Second 🔋⏱️.

Initial Energy On Equip 🚀 – Energy amount when equipped (clamped to Max).

Auto Switch Off On Deplete 📴.

Energy Design Note 📝 – UI-only memo for designers (e.g. "Use Battery items to recharge").

Best Practices 🧠

Start with normal offsets, then copy them to ADS offsets before fine-tuning.

Use separate assets (ArmsAndWeaponData) for weapon variants (iron sight, scope, silencer) instead of one bloated asset.

Keep Animator triggers optional if you rely mostly on IK.

For non-firearms, disable Is Firearm and enable Requires Energy.

Always double-check Muzzle Flash Point Name spelling.

Maintain consistent trigger naming across all weapons.

Troubleshooting 🧯

↑

ADS not working? Ensure “Use ADS” is enabled and offsets are set.

No muzzle flash? Verify the Muzzle Point Transform name.

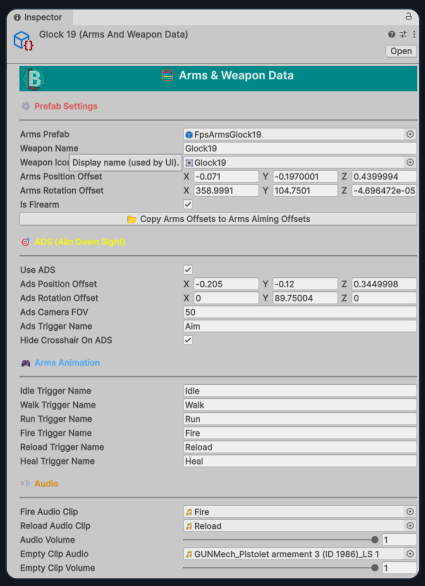
Weapon behaves like a tool? Check that “Is Firearm” is enabled.

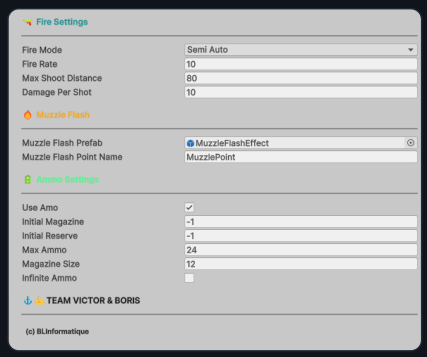
Energy not draining? Make sure “Requires Energy” is enabled and the device is switched on.

Compatibility ✅

Works with New Input System.

Fully integrated with EditorTools (StyledBox, Separator, tooltips in English).





Field	Type	Tooltip
⬇		
armsPrefab	`GameObject`	Prefab of the arms + weapon/device (fully configured prefab).
weaponName	`String`	Display name (used by UI).
weaponIcon	`Sprite`	Icon for UI (optional).
armsPositionOffset	`Vector3`	Local position offset for arms prefab.
armsRotationOffset	`Vector3`	Local rotation offset for arms prefab.
isFirearm	`Boolean`	True if this is a firearm (enables crosshair/weapon features). Set FALSE for flashlight or tools.
boutonDummy	`Int32`	Copies the current arms position/rotation offsets to the ADS aiming offsets below (Editor only).
⬇		
useADS	`Boolean`	Enable Aim Down Sight for this weapon (right-click to aim).
adsPositionOffset	`Vector3`	Arms/weapon position offset when aiming.
adsRotationOffset	`Vector3`	Arms/weapon rotation offset when aiming.
adsCameraFOV	`Single`	Camera FOV when aiming (smaller = more zoom).
adsTriggerName	`String`	Animator trigger/state for aiming (optional).
hideCrosshairOnADS	`Boolean`	Hide crosshair when aiming?
⬇		
idleTriggerName	`String`	Animator trigger/state to play when entering idle (optional).
walkTriggerName	`String`	Animator trigger/state to play when walking (optional).
runTriggerName	`String`	Animator trigger/state to play when running (optional).
fireTriggerName	`String`	Animator trigger/state to play when firing/using (optional).
reloadTriggerName	`String`	Animator trigger/state to play when reloading (optional).
healTriggerName	`String`	Animator trigger/state to play when healing (optional).
⬇		
fireAudioClip	`AudioClip`	Sound to play when firing/using (optional).
reloadAudioClip	`AudioClip`	Sound to play when reloading (optional).
audioVolume	`Single`	Volume for all sounds (0–1). • Range [0..1]
emptyClipAudio	`AudioClip`	Sound to play when out of ammo / empty magazine (optional).

Field	Type	Tooltip
emptyClipVolume	`Single`	Volume for empty magazine sound (0–1). • Range [0..1]
⬇		
fireMode	`FireMode`	Select fire mode: SemiAuto (click) or FullAuto (hold).
fireRate	`Single`	Fire rate (shots per second, only for FullAuto mode).
maxShootDistance	`Single`	Maximum range for raycast (meters).
damagePerShot	`Single`	Damage per shot.
⬇		
muzzleFlashPrefab	`GameObject`	Prefab to spawn as muzzle flash at every shot (particles/FX).
muzzleFlashPointName	`String`	Transform name used as muzzle flash spawn point inside the arms+weapon prefab. If empty, use prefab root.
⬇		
useAmo	`Boolean`	If true, ammo is infinite.
initialMagazine	`Int32`	Ammo in magazine on first pickup (-1 = full).
initialReserve	`Int32`	Reserve ammo on first pickup (-1 = max).
maxAmmo	`Int32`	Max total ammo the player can carry.
magazineSize	`Int32`	Ammo per magazine/clip.
infiniteAmmo	`Boolean`	If true, ammo is infinite.
⬇		
requiresEnergy	`Boolean`	If true, this device requires energy to operate (flashlight, tools, etc.).
energyMax	`Single`	Max energy capacity for this device.
energyDrainPerSecond	`Single`	Energy drained per second while the device is ON.
initialEnergyOnEquip	`Single`	Initial energy when equipped/instantiated (clamped to energyMax).
autoSwitchOffOnDeplete	`Boolean`	If true, the device should auto switch OFF when energy hits zero (Flashlight recommendation).
energyDesignNote	`String`	Optional note for designers (UI helper, not used by code). • TextArea

↑ Back to top

Copy fields

📦 ArmsLiveTuner MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Utility/🛠️ Arms Live Tuner`

Arms Live Tuner 🛠️

The Arms Live Tuner is a powerful runtime utility that lets you fine-tune arms and weapon offsets directly in Play Mode.

It avoids trial-and-error in the Inspector and provides a draggable overlay with hotkeys for fast iteration.

Target 🎯

Weapon Manager 🧩

Reference to the FPSWeaponManager. If left empty, it auto-finds at runtime.

Edit Set 🛠️

Choose which offsets you are editing:

IdleOffsets – default resting pose.

ADSOffsets – aim-down-sight pose.

Hotkeys ⌨️

Toggle Key (default: T) – Enable/disable tuner and overlay.

Freeze Key (default: F) – Freeze/unfreeze the entire game while tuning.

Slow Key (default: LeftAlt) – Apply slow multiplier to step size.

Fast Key (default: LeftShift) – Apply fast multiplier to step size.

F1 / F2 – Switch between Idle and ADS sets.

R – Apply values from current ArmsAndWeaponData back to the transform.

Steps 📐

↑

Position Step 📍 – Base movement increment in meters.

Rotation Step 🔁 – Base rotation increment in degrees.

Slow / Fast Multipliers ⚡ – Scale the step when holding modifiers.

Controls in Play Mode:

Move: A/D = X, W/S = Y, Q/E = Z

Rotate: ↑↓ = X, ←→ = Y, PgUp/PgDn = Z

Freeze Options 🧊

When freezing the world:

Optionally disable the FPS Controller.

Optionally disable weapon systems (switch/fire/reload).

Optionally disable inventory UI.

Optionally pause all Animators in the scene.

This lets you lock the game state and adjust offsets without distraction.

Overlay 🪟

Show Overlay – Toggle runtime overlay window.

Overlay Draggable – Drag window with the mouse.

Overlay Resizable / Scrollable – Resize with grip, enable vertical scroll.

Overlay Save Position – Store rect in PlayerPrefs.

Hide Overlay On Play – Start hidden when entering Play Mode.

Overlay includes live status (Active, Frozen), position/rotation values, and quick buttons:

Idle ➜ ADS / ADS ➜ Idle – Copy offsets between sets.

💾 Save SO (Editor only) – Save offsets back to ScriptableObject.

⟲ Reset Set – Reset offsets to zero.

Apply From Data (R) – Apply stored values to current transform.

↖ Reset Overlay Pos/Size – Restore default overlay rect.

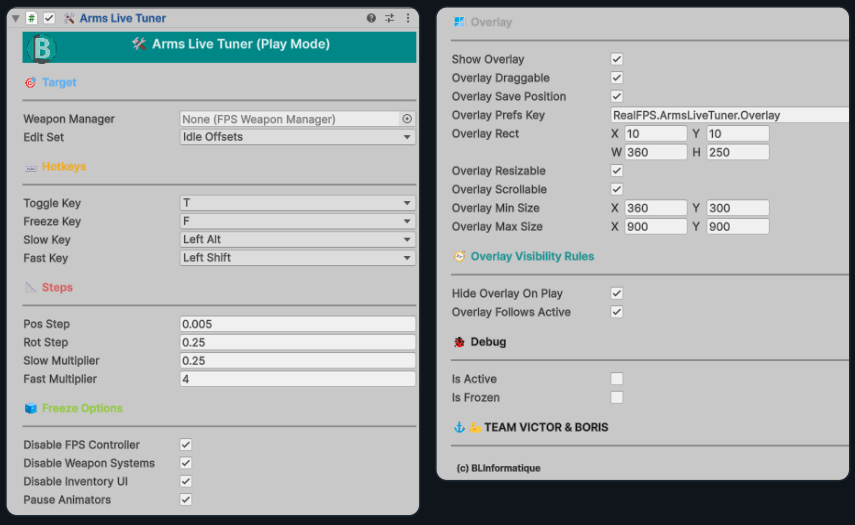
Debug

isActive – Whether tuner is currently applying inputs.

isFrozen – Whether the world is currently frozen.

⚓ 💪 TEAM VICTOR & BORIS Note

Arms Live Tuner is the fastest way to align your arms and weapons in RealFPS. Activate, freeze the world, nudge your offsets with hotkeys, then save back to the ScriptableObject — no more tedious guessing.




Arms Live Tuner
Active: YES Frozen: YES
Edit Set: IdleOffsets (F1=Idle, F2=ADS)
Pos: -0,1650, -0,0600, 0,4880
Rot: 5,79, 84,00, 354,75
Move A/D=X W/S=Y Q/E=Z
Rotate ↑↓=X ←→=Y PgUp/PgDn=Z
Steps Pos=0,005 Rot=0,25 (Alt=Slow x0,25, Shift=Fast x4)
Unfreeze (F)
Save SO
Reset Set
Idle → ADS
ADS → Idle
Apply From Data (R)
Reset Overlay Pos/Size

Field	Type	Tooltip
↓		
weaponManager	`FPSWeaponManager`	FPSWeaponManager to control. If null, will auto-find at runtime.
editSet	`EditSet`	Which set of offsets are you editing right now (Idle or ADS)?
↓		
toggleKey	`KeyCode`	Toggle tuner ON/OFF + overlay show/hide in Play Mode.
freezeKey	`KeyCode`	Freeze / Unfreeze the whole game while tuning.
slowKey	`KeyCode`	Hold to multiply step by slowMultiplier.
fastKey	`KeyCode`	Hold to multiply step by fastMultiplier.
↓		
posStep	`Single`	Base step for position nudge (meters).
rotStep	`Single`	Base step for rotation nudge (degrees).
slowMultiplier	`Single`	When holding Slow key, step = step * slowMultiplier.
fastMultiplier	`Single`	When holding Fast key, step = step * fastMultiplier.
↓		
disableFPSController	`Boolean`	Also disable FPS controller while frozen.
disableWeaponSystems	`Boolean`	Also disable weapon systems (switch/fire/reload) while frozen.
disableInventoryUI	`Boolean`	Also disable inventory UI while frozen.
pauseAnimators	`Boolean`	Pause all Animators in scene (speed=0) while frozen.
↓		
showOverlay	`Boolean`	Show the runtime overlay window.
overlayDraggable	`Boolean`	Allow dragging the overlay window with the mouse.
overlaySavePosition	`Boolean`	Save overlay position & size in PlayerPrefs.
overlayPrefsKey	`String`	Overlay PlayerPrefs key (per project).
overlayRect	`Rect`	Initial overlay rect (x,y,width,height).
overlayResizable	`Boolean`	Make the overlay resizable with a bottom-right grip.
overlayScrollable	`Boolean`	Enable a vertical scroll if content exceeds the window height.
overlayMinSize	`Vector2`	Minimum window size (w,h).
overlayMaxSize	`Vector2`	Maximum window size (w,h).

Field	Type	Tooltip
↓		
hideOverlayOnPlay	`Boolean`	Start with overlay hidden when Play begins (recommended).
overlayFollowsActive	`Boolean`	If true, overlay visibility follows 'isActive' when you press ToggleKey.
↓		
isActive	`Boolean`	Is tuner currently active (inputs applied)?
isFrozen	`Boolean`	Is the world currently frozen?

↑ Back to top

Copy fields

📦 DoorLockInteractable MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Interactions/🔒 Door Lock Interactable`

DoorLockInteractable 🔒

The DoorLockInteractable is an interaction script for locked objects such as doors 🚪 and drawers 🗄️.

It handles unlocking with keys, playing sounds, triggering animations, or physically rotating/sliding transforms.

Lock Settings 🔐

Is Locked ✅

If enabled, the object requires a matching key.

Required Key ID 🔑

String identifier of the required key (matches ItemData.keyId). Supports "MASTER" for universal keys.

Unlock Persists 🔓

Once unlocked, remains unlocked for all future interactions.

Key Consume Mode 📜

Always – Key is always consumed.

IfItemConsumable – Consumed only if ItemData.isConsumable is true.

Never – Key is never consumed.

Open Method 🚪

Choose how the object opens:

AnimatorTrigger 🎬 – Calls Animator.SetTrigger(triggerName).

AnimatorBool 🎬 – Toggles a boolean parameter on the Animator.

RotateHinge 🔄 – Rotates around a local axis (like a door).

↑

SlideLocal 📦 – Slides along a local axis (like a drawer).

Extra options:

Toggle When Unlocked 🔁 – Alternates open/close on each interaction.

Start Opened 🚪 – Object begins open at runtime.

Rotate / Slide Parameters – Axis, angle/distance, and speed.

UI & Audio 🔊

UI Locked Message 📝 – Text shown when no key is found.

UI Unlocked Message 📝 – Text shown when unlocking succeeds.

Locked SFX / Unlock SFX 🔊 – Audio clips for feedback.

Audio Source 🎚️ – Custom source for playback (optional).

Events 📡

UnityEvents fired at runtime:

OnUnlockSuccess 🎉

OnUnlockFailed ❌

OnOpened 🚪

OnClosed 🛑

Usage Flow 🧭

Player interacts → If locked:

Checks inventory (GeneralInventoryManager.HasKeyId).

Consumes key depending on KeyConsumeMode.

Plays feedback and opens if successful.

If already unlocked → Opens/closes depending on toggleWhenUnlocked.

Supports both animation-based and transform-based movement.

Best Practices 🧠

Use AnimatorTrigger for cinematic doors, RotateHinge for quick physics-like behavior, SlideLocal for drawers.

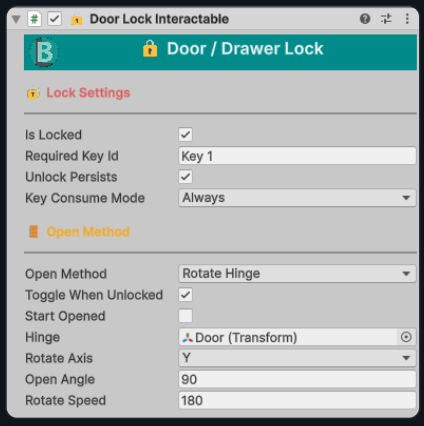
Set Unlock Persists = false if you want doors to relock each time.

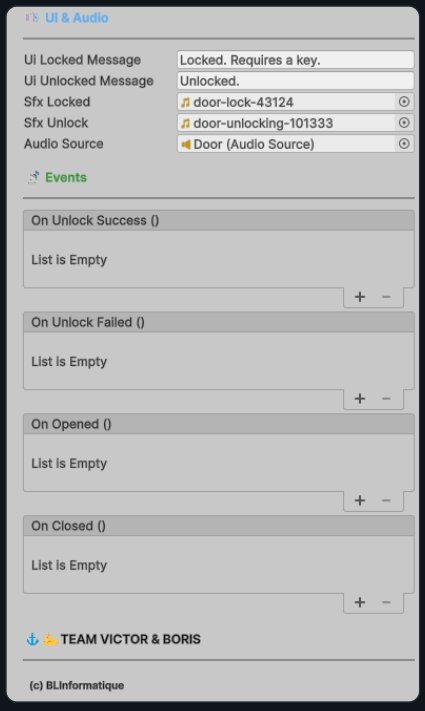
Use the "MASTER" key ID for debugging or universal keys.

Pair with UI prompts (ex. “Press E to Use Key”) via your Pickup UI system.

⚓💪 TEAM VICTOR & BORIS Note

DoorLockInteractable unifies keys, doors, and drawers into one flexible system. You don’t need separate scripts for every lockable object — just configure the open method and lock behavior.





Field	Type	Tooltip
↓		
isLocked	`Boolean`	If true, interaction requires a matching key to unlock.
requiredKeyId	`String`	Key ID required to unlock (matches ItemData.keyId or 'MASTER').
unlockPersists	`Boolean`	If true, once unlocked it remains unlocked for future interactions.
keyConsumeMode	`KeyConsumeMode`	How the key should be consumed when unlocking.
↓		
openMethod	`OpenMethod`	Choose how this object opens (Animator Trigger/Bool, Rotate hinge, or Slide).
toggleWhenUnlocked	`Boolean`	Toggles open/close each interaction when unlocked.
startOpened	`Boolean`	Start opened at play (applies to Transform-based and Animator Bool).
animatorTrigger	`Animator`	Animator that drives the open animation (Trigger).
triggerName	`String`	Trigger name to fire when opening.
animatorBool	`Animator`	Animator that drives the open animation (Bool).
boolName	`String`	Bool parameter that indicates open(true)/closed(false).
hinge	`Transform`	Transform pivot that rotates (door hinge). If null, uses this transform.
rotateAxis	`LocalAxis`	Local axis to rotate around.
openAngle	`Single`	Angle in degrees when fully opened (relative to start).
rotateSpeed	`Single`	Opening/closing speed in deg/s.
slider	`Transform`	Transform that slides (drawer). If null, uses this transform.
slideAxis	`LocalAxis`	Local axis to slide along.
openDistance	`Single`	Distance in meters when fully opened (relative to start).
slideSpeed	`Single`	Opening/closing speed in m/s.
↓		
uiLockedMessage	`String`	Shown when locked and no matching key is found.
uiUnlockedMessage	`String`	Shown when successfully unlocked.
sfxLocked	`AudioClip`	Audio when locked without key.
sfxUnlock	`AudioClip`	Audio when successfully unlocked.
audioSource	`AudioSource`	Audio source to play SFX. If null, a one-shot will be used.

Field	Type	Tooltip
OnUnlockSuccess	`UnityEvent`	
OnUnlockFailed	`UnityEvent`	
OnOpened	`UnityEvent`	
OnClosed	`UnityEvent`	

↑ Back to top

Copy fields

📦 EnemyAI MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/AI/🚨Add Enemy AI` • Requires: Animator, CapsuleCollider, NavMeshAgent, EnemyHealth

EnemyAI 🚨

The EnemyAI script controls NPC enemies using Unity's NavMeshAgent, animations, and optional groan/voice system.

It supports patrolling, chasing the player, attacking, and reacting to damage.

General Settings ⚙️

Player Tag 🎯 – Tag used to detect the player (default: "Player").

Detection Radius 📡 – Distance in meters at which the enemy detects and starts chasing the player.

Stopping Distance ⛔ – Minimum distance to the player before attacking.

Attack Damage 💥 – Amount of damage dealt per hit.

Attack Cooldown ⏱️ – Delay between consecutive attacks.

Movement 🚶

Move Speed 🏃 – Speed of the NavMeshAgent (chasing and patrol).

Patrol Settings 🏃

Enable Patrol ✅ – If enabled, enemy will patrol between waypoints when not detecting the player.

Patrol Points 📍 – Array of waypoint transforms.

Wait Time ⏳ – Seconds to wait at each waypoint before moving to the next.

Animation 🎮

Hit Trigger 😳 – Animator trigger name for "Hit" animation.

↑

Death Trigger ☠️ – Animator trigger name for “Death” animation.

Attack Trigger ⚔️ – Animator trigger name for “Attack” animation.

Walk Bool 🚶 – Animator bool controlling walking state.

Idle Bool 💤 – Animator bool controlling idle state.

Groan / Voice System 📢

Enable Groans ✅ – Toggles zombie/enemy groan system.

Voice Source 🎙️ – AudioSource to play groans (auto-created if missing).

Pitch Range 🎚️ – Random pitch variation for groans.

Idle Groans 😴

Idle Groans Clips 🎵 – Clips played randomly while idle.

Idle Volume 🔉 – Volume for idle groans.

Idle Interval ⏳ – Random delay between idle groans.

Chase Groans 🩸

Chase Groans Clips 🎵 – Clips played while chasing/attacking.

Chase Volume 🔉 – Volume for chase groans.

Chase Interval ⏳ – Random delay between chase groans.

Debug 🔍

Debug Logs 📝 – Prints runtime information in Console.

Runtime Behaviour 🧭

Detection: When the player enters detection radius, the enemy chases them.

Attack: At stopping distance, the enemy damages the player at each cooldown.

Patrol: If patrol is enabled and no player is detected, enemy moves between waypoints.

Animations: Animator parameters (walk, idle, attack, death, hit) are updated automatically.

Groans: Randomized idle or chase groans add atmosphere.

Best Practices 🧠

Use consistent Animator parameter names across enemies to reuse controllers.

Adjust Detection Radius and Stopping Distance per enemy type (fast vs. slow enemies).

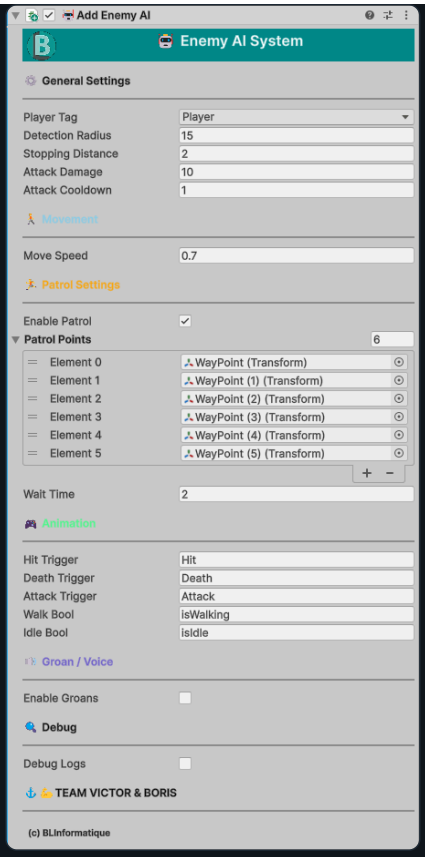
Keep Pitch Range narrow for realism, wider for creepy zombie-like variation.

Patrol works best with NavMesh baked and multiple waypoints.

For performance, disable groans if many enemies are active at once.

⚓💪 TEAM VICTOR & BORIS Note

EnemyAI makes it simple to create patrol/chase/attack behaviour without custom coding. With groans and patrols, even a few enemies can bring your FPS world alive with tension.


Add Enemy AI
Enemy AI System
General Settings
Player Tag
Player
Detection Radius
15
Stopping Distance
2
Attack Damage
10
Attack Cooldown
1
Movement
Move Speed
0.7
Patrol Settings
Enable Patrol
Patrol Points
6
Element 0
WayPoint (Transform)
Element 1
WayPoint (1) (Transform)
Element 2
WayPoint (2) (Transform)
Element 3
WayPoint (3) (Transform)
Element 4
WayPoint (4) (Transform)
Element 5
WayPoint (5) (Transform)
Wait Time
2
Animation
Hit Trigger
Hit
Death Trigger
Death
Attack Trigger
Attack
Walk Bool
isWalking
Idle Bool
isIdle
Groan / Voice
Enable Groans
Debug
Debug Logs
TEAM VICTOR & BORIS
(c) BLInformatique

Field	Type	Tooltip
↓		
playerTag	`String`	Tag used to detect the player target.
detectionRadius	`Single`	Detection radius for chasing the player.
stoppingDistance	`Single`	Minimum distance to the player before attacking.
attackDamage	`Single`	Amount of damage dealt to the player per attack.
attackCooldown	`Single`	Cooldown time in seconds between each attack.
↓		
moveSpeed	`Single`	Movement speed of the enemy (NavMeshAgent component).
↓		
enablePatrol	`Boolean`	Enable patrol mode (enemy moves between waypoints when not detecting player).
patrolPoints	`Transform[]`	Waypoints (Transforms) the enemy will patrol between.
waitTime	`Single`	Time (seconds) the enemy waits at each waypoint during patrol.
↓		
hitTrigger	`String`	Animator trigger name for hit animation.
deathTrigger	`String`	Animator trigger name for death animation.
attackTrigger	`String`	Animator trigger name for attack animation.
walkBool	`String`	Animator bool name for walking state.
idleBool	`String`	Animator bool name for idle state.
↓		
enableGroans	`Boolean`	Enable zombie groans/voice system.
voiceSource	`AudioSource`	AudioSource used for playing groans (auto-created if null).
pitchRange	`Vector2`	Random pitch range for each groan played.
↓		
idleGroans	`AudioClip[]`	Audio clips to play randomly while idle.
idleVolume	`Single`	Volume for idle groans. • Range [0..1]
idleInterval	`Vector2`	Random interval between idle groans (seconds).
↓		
chaseGroans	`AudioClip[]`	Audio clips to play randomly while chasing or attacking the player.
chaseVolume	`Single`	Volume for chase groans. • Range [0..1]
chaseInterval	`Vector2`	Random interval between chase groans (seconds).

Field	Type	Tooltip
⬇		
debugLogs	`Boolean`	Enable debug logs in the Console.

↑ Back to top

Copy fields

📦 EnemyHealth MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/AI/❤️Add Enemy Health`

EnemyHealth ❤️

The EnemyHealth component manages an enemy’s hit points, damage reactions (FX + audio), and death flow.

It plugs into EnemyAI for hit/death animations and exposes UnityEvents for scoring, loot, or custom logic.

Health Settings ⚙️

Max Health 🩺

Total HP of the enemy. Initializes currentHealth at Start.

Headshot Damage Multiplier 🎯

Multiplier used by weapon logic when a headshot is detected.

Damage FX 🩸

Blood FX Prefab 💥

Optional FX instantiated on hit.

Blood Spawn Point 📍

Where to spawn FX (fallback: enemy position + Vector3.up * 1.5f).

Hit Sound 🔊

Damage Sounds 🎵

Array of clips randomly chosen on each hit.

Damage Audio Source 🚹

If none is assigned, one is auto-created (3D, non-looping).

Damage Volume 🔊

Volume for hit sounds (0–1).

Death Settings 💀

↑

Death Sound ☠️

Optional clip played on death.

Auto Destroy Delay 🕛

Seconds before destroying the enemy object after death (0 = keep corpse).

On Death (UnityEvent) 📡

Hook for score, loot drops, counters, etc.

Runtime State 🔍

currentHealth (read-only)

isDead (read-only)

Damage & Death Flow 🧭

TakeDamage(amount)

Ignores if already dead.

Subtracts health and clamps [0, maxHealth].

If EnemyAI present → triggers hit animation via PlayHitAnimation().

Plays random damage sound (if any).

Spawns blood FX (if any).

If health ≤ 0 → Die().

Die()

Guards against double death.

Invokes OnDeath event.

Plays death sound (if any).

Notifies EnemyAI → triggers death animation, stops agent, disables collider.

Optionally Destroy(gameObject, autoDestroyDelay).

Integration Tips 🧠

Pair with EnemyAI for animations and NavMesh behaviour.

Drive headshots from your weapon hit logic by multiplying damage before calling TakeDamage.

Use OnDeath to:

Increment score / kill counters,

Spawn loot pickups,

Notify wave or spawner systems.

Keep FX lightweight; auto-destroyed FX (2s) minimize leaks in long sessions.

Troubleshooting 🧯

No hit reactions? Ensure EnemyAI is on the same GameObject (for hit trigger).

No sounds? Make sure clips are assigned; verify damageAudioSource exists (auto-added at Start).

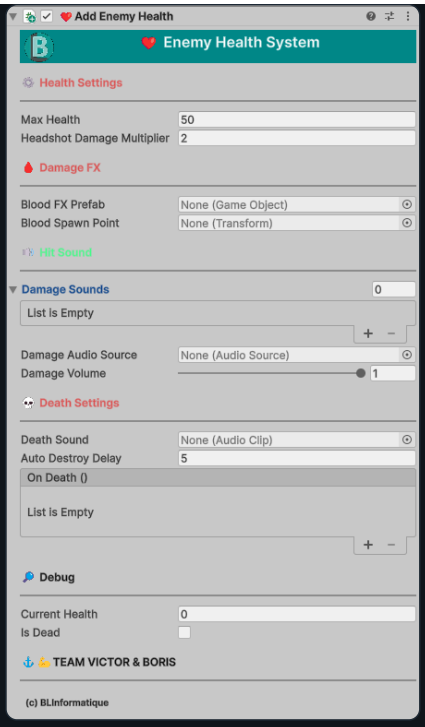
Enemy never dies? Confirm your weapon calls TakeDamage() with a positive amount and that colliders/tags align with your hit detection.

Corpse vanishes too fast? Increase Auto Destroy Delay or set to 0 to keep it.

⚓ 💪 TEAM VICTOR & BORIS Note

EnemyHealth is intentionally simple: all presentation (FX/audio/anim) is local, while scoring and drops are exposed via OnDeath. Glue it to your gameplay loop without rewriting core health logic.

↑



Field	Type	Tooltip
⬇		
maxHealth	`Single`	Max health of this enemy.
headshotDamageMultiplier	`Single`	Multiplier applied to damage if a headshot is detected (used by weapon logic).
⬇		
bloodFXPrefab	`GameObject`	FX to spawn when taking damage (blood, sparks, etc).
bloodSpawnPoint	`Transform`	Transform to use as spawn point (optional).
⬇		
damageSounds	`AudioClip[]`	Optional damage sound (random if multiple).
damageAudioSource	`AudioSource`	AudioSource to play damage sounds.
damageVolume	`Single`	Volume for damage sounds. • Range [0..1]
⬇		
deathSound	`AudioClip`	Optional death sound.
autoDestroyDelay	`Single`	Auto destroy object after death (seconds). Set 0 to keep.
onDeath	`UnityEvent`	Events called when enemy dies.
⬇		
currentHealth	`Single`	
isDead	`Boolean`	

↑ Back to top

Copy fields

📦 EnergyBarUI MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Energy/Energy Bar UI (Adapter)`

EnergyBarUI 📊 (Adapter)

Purpose. EnergyBarUI is a lightweight adapter that binds any runtime device implementing energy (e.g., flashlight, goggles, tools) to a UI bar.

It listens to energy change events and updates a FPSEnergy UI component. It also auto-shows/hides a UI root when the device is equipped/unequipped.

Works with:

Energy source → EnergyConsumer (runtime device: exposes CurrentEnergy/MaxEnergy + onEnergyChanged).

UI widget → FPSEnergy (fills an Image, holds currentEnergy / maxEnergy, no draining here).

Equipment flow → FPSWeaponManager (calls Bind(), OnEquipped(), OnUnequipped() on item switch).

Inspector Fields ⚙️

FPSEnergy 🖼️

Reference to the UI bar component that displays energy. If missing, Reset() tries to auto-grab it on the same GameObject.

Root To Toggle 🪟

Optional UI panel to enable/disable when the device is equipped/unequipped.

Auto Hide Not Equipped 🙈

If true, the root panel is hidden when the device is not equipped.

Bound Consumer (ReadOnly) 🔗

The currently bound EnergyConsumer at runtime (set by Bind()).

Public API 🧩

void Bind(EnergyConsumer consumer)

↑

Subscribes to consumer.onEnergyChanged, copies its current values into FPSEnergy, and stops any self-consumption on FPSEnergy (UI must not drain energy).

If another consumer was previously bound, it unsubscribes first.

void OnEnergyChanged(float current, float max)

Event handler that clamps and forwards values to FPSEnergy, updating the fill amount.

void OnEquipped() / void OnUnequipped()

Shows/hides the rootToToggle (if autoHideNotEquipped), and refreshes UI from the bound device.

Typical Flow 🧭

Equip an item that requires energy (flashlight):

FPSWeaponManager detects equip → calls EnergyBarUI.Bind(device) → EnergyBarUI.OnEquipped().

During use, the device modifies its energy and raises onEnergyChanged.

EnergyBarUI receives the event → updates FPSEnergy (bar fill).

Unequip the device:

FPSWeaponManager calls EnergyBarUI.OnUnequipped() → optional panel hides.

Setup Checklist ✅

Place a UI Canvas with your Energy Bar (Image fill) and add FPSEnergy on it.

Add EnergyBarUI on the same GameObject (or a parent that references the FPSEnergy).

(Optional) Drag the bar's root panel into Root To Toggle and enable Auto Hide Not Equipped.

Ensure your device implements/uses EnergyConsumer and invokes onEnergyChanged.

↑

From your equipment flow (e.g., FPSWeaponManager), call Bind() when equipping and OnEquipped()/OnUnequipped() on state changes.

Best Practices 🧠

Single source of truth: only the device changes energy; the UI never drains. EnergyBarUI explicitly calls fpsEnergy.StopConsumption().

Late refresh: on equip, call OnEquipped() after the device initialized its energy values (prevents 0% flashes).

Pooling-safe: if your devices are pooled, always unsubscribe previous listeners (handled by Bind() automatically).

Multi-device HUD: you can have multiple EnergyBarUI instances, each bound by different managers (e.g., tool energy vs. suit energy).

Troubleshooting 🧯

Bar doesn't update? Confirm the device fires onEnergyChanged and that Bind() was called at equip time.

Wrong max/current? Ensure you pass device values into Bind() before calling OnEquipped(); OnEnergyChanged clamps with max > 0.

Panel never shows? Check Auto Hide Not Equipped and that rootToToggle is assigned; verify OnEquipped() is called by your manager.

UI drains energy?! That's a design smell. Keep draining logic inside the device; EnergyBarUI is display-only.

Related Components 🔗

EnergyConsumer: runtime device interface/impl that owns energy, raises events.

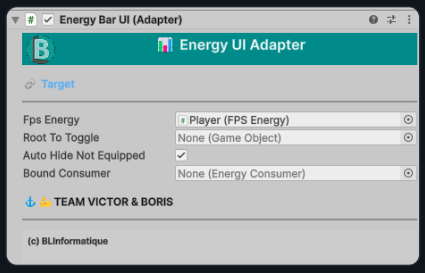
FPSEnergy: UI bar component (Image fill).

FPSWeaponManager: orchestrates equips/unequips, should call Bind(), OnEquipped(), OnUnequipped().

⚓ 💪 TEAM VICTOR & BORIS Note

Keep UI passive. Let gameplay drive the numbers; let EnergyBarUI simply mirror them with style. For a clean UX, auto-hide the panel when no energy

device is equipped, then pop it in when the flashlight is out.



Field	Type	Tooltip
⬇		
fpsEnergy	`FPSEnergy`	The FPSEnergy UI component that displays the bar.
rootToToggle	`GameObject`	Optional: UI root panel to toggle when equipped/unequipped.
autoHideNotEquipped	`Boolean`	If true, hides the root panel when not equipped (OnUnequipped).
boundConsumer	`EnergyConsumer`	Current EnergyConsumer device bound to this UI (set at runtime, do not modify manually).

↑ Back to top

Copy fields

📦 EnergyConsumer MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Energy/Add Energy Consumer`

EnergyConsumer 🔋

The EnergyConsumer is a runtime component that manages an energy pool for any device (flashlight, goggles, tools, gadgets).

It controls draining, recharging, events, and integrates with UI via EnergyBarUI.

Energy Settings ⚙️

Max Energy 🔝 – Maximum capacity (default: 100).

Start Energy 🚀 – Initial energy when the device is created/equipped.

Drain Per Second ⏱️ – Energy drained per second while active.

Drain Only When Active 🔌 – If true, only drains while IsActive is true.

Allow Negative Energy ➖ – If false, device auto-stops at 0. If true, energy may go below zero (special cases/debug).

Runtime State 📊

CurrentEnergy (read-only) – Current runtime value.

MaxEnergy (read-only) – Maximum capacity.

IsActive (read-only) – True if the device is active.

Events 📣

onEnergyChanged(float current, float max) 🔄 – Raised every time the value changes. Used by EnergyBarUI to update UI.

onEnergyDepleted 🪫 – Fired when energy hits 0.

onEnergyRestoredFromZero 🔋 – Fired when energy goes from 0 back above 0.

↑

Public API 🧩

SetActive(bool active) – Toggles the device's active state (controls draining).

Recharge(float amount) – Adds energy (clamped), returns actual added amount. Triggers onEnergyRestoredFromZero if reviving from 0.

SetEnergy(float value) – Directly sets energy value (clamped).

CurrentEnergy / MaxEnergy – Properties to read state.

Runtime Behaviour 🧭

On Awake: initializes currentEnergy from startEnergy. Fires onEnergyChanged, and onEnergyDepleted if starting at 0.

On Update: drains energy based on settings.

If drainOnlyWhenActive = true, drains only when active.

If allowNegativeEnergy = false, clamps at 0 and fires onEnergyDepleted.

Always fires onEnergyChanged after updates.

Integration with UI 🔗

Pair with EnergyBarUI: call Bind(EnergyConsumer) when the item is equipped.

EnergyBarUI listens to onEnergyChanged and updates the FPSEnergy bar.

Recommended: call SetActive(true/false) when toggling the device (ex. flashlight ON/OFF).

Best Practices 🧠

Keep Start Energy ≤ Max Energy.

Always raise gameplay logic (e.g. flashlight turns OFF) via onEnergyDepleted. Don't rely solely on Update.

Use Recharge() to handle batteries or pickups.

For debug/devices with infinite power, set drainPerSecond = 0.

Troubleshooting 🧯

Energy not updating? Ensure Update runs (component enabled), and drainPerSecond > 0.

UI not moving? Check that EnergyBarUI.Bind() is called when equipping.

Device not turning off at 0? Verify allowNegativeEnergy = false and hook into onEnergyDepleted to stop behaviour.

Related Components 🔗

EnergyBarUI – Displays energy in UI.

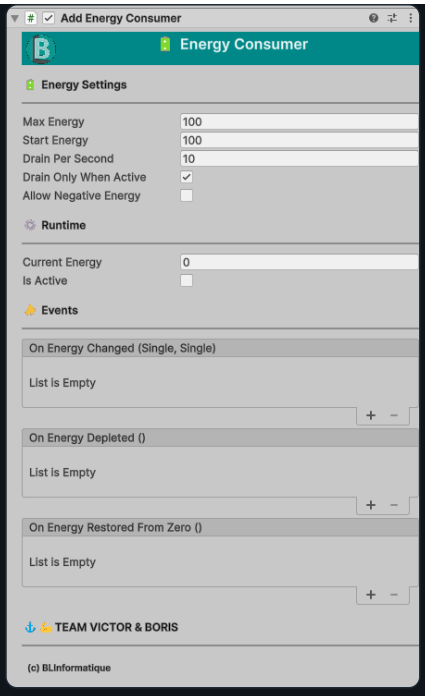
FPSEnergy – Fills a UI bar.

FPSWeaponManager – Calls Bind() when equipping items with energy.

⚓💪 TEAM VICTOR & BORIS Note

EnergyConsumer is the universal energy logic for RealFPS. Any device — flashlight, scanner, goggles — can run on it. Just add the script, set the drain rate, and wire it to the UI with EnergyBarUI.

↑



Field	Type	Tooltip
⬇		
currentEnergy	`Single`	Current runtime energy value.
isActive	`Boolean`	True if device is currently active (and draining energy if configured to).
⬇		
maxEnergy	`Single`	Maximum energy capacity for this device.
startEnergy	`Single`	Initial energy when device is started or equipped.
drainPerSecond	`Single`	Energy drained per second when device is active.
drainOnlyWhenActive	`Boolean`	If true, device only drains energy while 'IsActive' is true.
allowNegativeEnergy	`Boolean`	If false, device automatically stops when energy reaches zero.
⬇		
onEnergyChanged	`UnityEvent<Single, Single>`	Invoked whenever energy value changes (current, max).
onEnergyDepleted	`UnityEvent`	Invoked when energy reaches zero.
onEnergyRestoredFromZero	`UnityEvent`	Invoked when energy is restored from zero (was empty, now > 0).

↑ Back to top

Copy fields

📦 ExplosionEntity MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/💥Add Explosion Entity` • Requires: BoxCollider

ExplosionEntity 💥

The ExplosionEntity simulates an explosion effect in RealFPS.

It combines FX, sound, debris, physics forces, and damage into one reusable component.

Useful for explosive barrels, grenades, vehicles, or scripted traps.

Explosion Settings ⚙️

Explosion FX Prefab ✨ – Instantiated at explosion point.

Explosion Sound 🔊 – Audio clip played on detonation.

Audio Source 🎚️ – Optional; if null, a temporary 3D source is created.

Explosion Force 💨 – Physics impulse applied to rigidbodies in radius.

Explosion Radius 📏 – Range of effect (meters).

Damage Targets 🎯

Damage Player 👤 – If true, damages FPSHealth components in range.

Damage Enemies 🤖 – If true, damages EnemyHealth components in range.

Base Damage 💥 – Damage at explosion center.

Use Falloff 📉 – Damage decreases with distance.

Min Edge Damage 🔻 – Minimum damage at radius edge when falloff is enabled.

Check Line Of Sight 👀 – Blocks damage if an obstacle is between explosion and target.

↑

LOS Blocking Layers 🧱 – LayerMask defining which layers block line of sight.

Debris 🧱

Debris Prefab 🪨 – Optional prefab spawned with physics.

Debris Count 🔢 – Number of debris pieces to spawn.

Destroy Delay ⏲️ – Lifetime of FX and debris before auto-destroy.

Durability 🛡️

Hits To Explode 🔫 – Number of hits before explosion (supports bullet triggers).

Destruction 🛑

Destroy Self ❌ – If true, destroys object after explosion. Otherwise disables it.

Debug 🧪

Debug Key ⌨️ – Press key in Play Mode to force explosion (default: X).

Draw Gizmos 🖊️ – Draw explosion radius sphere in Scene view.

Runtime Behaviour 🧭

Hit Registration – Call RegisterHit() (e.g., from bullet impact). When hits ≥ threshold, explosion is queued.

Explosion Sequence – On Explode() call:

Disables colliders (prevents double trigger).

Spawns FX + sound.

Spawns debris (if set).

Applies AddExplosionForce to rigidbodies.

Deals damage to FPSHealth (player) and EnemyHealth (AI).

Destroys or disables the object.

Integration 🔗

Combine with bullet impact logic → call RegisterHit() when projectile collides.

Damage flows into:

FPSHealth for the player,

EnemyHealth for AI.

Use OnDrawGizmos for level design to visualize radius.

Best Practices 🧠

Use useFalloff = true for realistic grenades; set minEdgeDamage = 0 for sharp falloff.

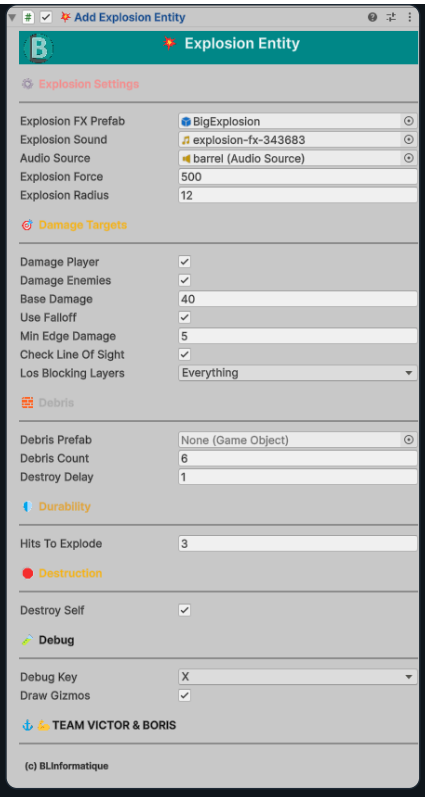
Use Check Line Of Sight when you want walls/cover to block damage.

Keep debris lightweight; auto-destroy prevents performance leaks.

If using many explosives, lower explosionForce to avoid excessive physics overhead.

⚓💪 TEAM VICTOR & BORIS Note

ExplosionEntity unifies visuals + physics + gameplay damage. With just one script, your FPS gains explosive barrels, grenades, and destructible props — ready to shake the battlefield.


Add Explosion Entity
Explosion Entity
Explosion Settings
Explosion FX Prefab
BigExplosion
Explosion Sound
explosion-fx-343683
Audio Source
barrel (Audio Source)
Explosion Force
500
Explosion Radius
12
Damage Targets
Damage Player
Damage Enemies
Base Damage
40
Use Falloff
Min Edge Damage
5
Check Line Of Sight
Los Blocking Layers
Everything
Debris
Debris Prefab
None (Game Object)
Debris Count
6
Destroy Delay
1
Durability
Hits To Explode
3
Destruction
Destroy Self
Debug
Debug Key
X
Draw Gizmos
TEAM VICTOR & BORIS
(c) BLInformatique

Field	Type	Tooltip
↓		
explosionFXPrefab	`GameObject`	FX prefab to spawn on explosion.
explosionSound	`AudioClip`	Sound to play on explosion.
audioSource	`AudioSource`	Optional AudioSource to reuse (3D). If null, a temp source is created.
explosionForce	`Single`	Force applied to rigidbodies in the radius.
explosionRadius	`Single`	Explosion effect radius.
↓		
damagePlayer	`Boolean`	Deal damage to the player (FPSHealth).
damageEnemies	`Boolean`	Deal damage to enemies (EnemyHealth).
baseDamage	`Single`	Base damage at the explosion center.
useFalloff	`Boolean`	Use damage falloff from center to radius.
minEdgeDamage	`Single`	Minimum damage at radius edge when falloff is enabled.
checkLineOfSight	`Boolean`	Only deal damage if line of sight is clear (no obstacle).
losBlockingLayers	`LayerMask`	Physics layers considered as blocking line of sight.
↓		
debrisPrefab	`GameObject`	Debris prefab spawned at explosion (optional).
debrisCount	`Int32`	Number of debris instances.
destroyDelay	`Single`	Lifetime of FX and debris (seconds).
↓		
hitsToExplode	`Int32`	Number of hits before exploding.
↓		
destroySelf	`Boolean`	Destroy this object right after the explosion.
↓		
debugKey	`KeyCode`	Press this key to force explosion (play mode).
drawGizmos	`Boolean`	Draw gizmos for radius, etc.

↑ Back to top

Copy fields

📦 FlashlightEnergyController MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Energy/Add Flashlight Energy Controller` • Requires: EnergyConsumer

FlashlightEnergyController 🔦

The FlashlightEnergyController manages a flashlight device that consumes energy.

It integrates EnergyConsumer for draining, supports legacy Input and the new Unity Input System, and can blink when energy is low.

Energy ⚡

Energy Consumer 🔋 – Reference to the EnergyConsumer on the same object. Controls energy pool.

Energy Drain Rate ⏱️ – Amount drained per second while the flashlight is ON.

Light 💡

Flashlight Light 💡 – The Light component toggled by the script.

Enable Low-Energy Blink ⚠️ – If enabled, light blinks when energy is low.

Low Energy Threshold 📉 – Percentage (0–1) of max energy that triggers blinking.

Blink Frequency 🔁 – Blink rate in Hz when under threshold.

Controls 🎮

Toggle Mode ⏯️ –

ON: Press key once to toggle.

OFF: Hold key to keep flashlight ON.

Use New Input System 🆕 – Enables Unity's Input System support.

↑

Legacy Input: Choose KeyCode (default L).

New Input System: Bind an InputAction (flashlightAction).

Events 📢

On Flashlight On 🌟 – UnityEvent invoked when flashlight turns ON.

On Flashlight Off 🟣 – UnityEvent invoked when flashlight turns OFF.

Runtime Behaviour 🧭

Input: Key press or InputAction toggles ON/OFF (or hold, depending on mode).

Energy: While ON, EnergyConsumer drains energy per second.

If energy hits 0, flashlight switches OFF automatically.

Blinking: If enabled and energy ≤ threshold, the Light component toggles rapidly.

Events: External systems (UI, SFX) can subscribe to On/Off events.

Integration 🔗

Requires EnergyConsumer (manages energy).

Works with EnergyBarUI + FPSEnergy for HUD display.

Use OnFlashlightOn/Off to trigger sounds, animations, or UI indicators.

Best Practices 🧠

Keep drainPerSecond small for smoother consumption.

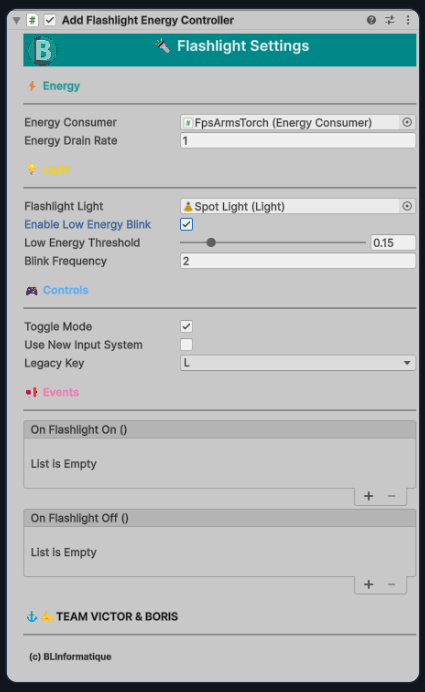
Use blink effect sparingly (avoid player frustration).

If you support both Input Systems, wrap with a toggle in your project settings.

For multiplayer, sync isOn state and energy events over the network.

⚓💪 TEAM VICTOR & BORIS Note

This controller turns a simple Light into a gameplay-aware flashlight. With energy drain, blinking feedback, and input support, it fits right into the RealFPS survival toolkit.



Field	Type	Tooltip
↓		
energyConsumer	`EnergyConsumer`	EnergyConsumer component for this flashlight.
energyDrainRate	`Single`	Energy used per second when light is on.
↓		
flashlightLight	`Light`	The actual Light component to toggle.
enableLowEnergyBlink	`Boolean`	Enable low-energy blink effect.
lowEnergyThreshold	`Single`	Energy threshold (0-1) to start blinking. • Range [0..1]
blinkFrequency	`Single`	Blink frequency in Hz when low on energy.
↓		
toggleMode	`Boolean`	If true, pressing the key toggles the flashlight on/off. If false, hold to keep on.
useNewInputSystem	`Boolean`	Use Unity's New Input System instead of legacy Input.
legacyKey	`KeyCode`	Key used in legacy Input System mode.
flashlightAction	`InputAction`	Input Action used in New Input System mode.
↓		
onFlashlightOn	`UnityEvent`	Invoked when the flashlight turns on.
onFlashlightOff	`UnityEvent`	Invoked when the flashlight turns off.

↑ Back to top

Copy fields

📦 FPSControllerPro MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/🕹️Add FPS Controller Pro Behaviour` • Requires: CharacterController, FPSFootstepPlayer, FPSStamina, FPSFood, FPSHealth, GeneralInventoryManager, FPSWeaponManager, WeaponInventoryManager, FPSCrosshair, FPSEnergy, EnergyBarUI

FPSControllerPro 🕹️

The FPSControllerPro is the core first-person character controller of RealFPS.

It manages movement, camera, jumping, crouching, stamina, health, fall damage, headbob, and integrates with inventory, weapons, and UI.

Input Systems ⌨️ / 🎮

Supports both Legacy Input System and New Unity Input System.

Legacy Input – Uses KeyCode for movement, jump, crouch, run.

New Input System – Uses InputAction references (moveAction, jumpAction, crouchAction, runAction).

Toggle via: useNewInputSystem.

Movement 🚶

Walk Speed 🪶 – Base movement speed.

Run Speed 🏃 – Sprint speed (disabled if forceWalk = true).

Force Walk 🚫 – Forces walking only (useful with stamina depletion).

Gravity ⬇️ – Gravity applied each frame.

Step Offset 🪜 – Maximum step height climbable.

Slope Limit ⛰️ – Maximum slope angle.

Jump & Crouch ⏫ 🔻

Can Jump ✅ – Enables/disables jump ability.

↑

Jump Force 🦘 – Upward velocity when jumping.

Can Crouch ✅ – Enables/disables crouch ability.

Crouch Height 📏 – CharacterController height when crouching.

Public API:

ToggleCrouch() – Switch crouch/stand and update height.

Camera & Look 🎥

Camera Pivot 🎯 – Transform pivot for rotation/headbob.

Main Camera 📷 – Player camera reference.

Mouse Sensitivity 🖱️ – Look sensitivity.

Pitch Clamp 📐 – Vertical rotation clamp (up/down look).

Headbob 🎵

Enable HeadBob ✅ – Simulated camera motion while walking/running.

Amplitude / Frequency 🌊 – Strength and speed of bob.

Run Multiplier ⚡ – Extra bobbing when running.

Roll Amplitude 🔄 – Camera roll (Z-axis sway).

Fall Damage 🪂

Enable Fall Damage ✅ – Turns fall damage on/off.

Threshold 📉 – Minimum downward speed to start damage.

Damage Per Unit 💥 – Scales damage beyond threshold.

Audio 🎵 – Optional sound when taking fall damage.

Logic:

When landing after a fall, damage = (impactVelocity - threshold) * damagePerUnit.

Applies to FPSHealth on the same object.

Debug 🐞

Show Debug Logs 📜 – Prints runtime info (movement, velocity, grounded state, etc.)

Public Properties 🔗

moveInput – Current input vector.

isCrouching – True if crouching.

isRunning – True if running.

SetRunning(bool) – Force running state externally.

Runtime Behaviour 🧭

Mouse Look – Rotates player (yaw) and camera pivot (pitch).

Movement – Applies walk/run speed, gravity, and jump.

Crouch – Toggles height of CharacterController.

Headbob – Oscillates pivot position/rotation for immersion.

Fall Damage – On landing, applies damage and optional sound.

Required Components 🛠️

FPSControllerPro automatically requires and integrates with:

CharacterController

FPSFootstepPlayer (footstep sounds)

FPSStamina (sprint stamina)

FPSFood (hunger system)

FPSHealth (player health)

GeneralInventoryManager (inventory)

FPSWeaponManager + WeaponInventoryManager (weapons)

FPSCrosshair (UI crosshair)

FPSEnergy + EnergyBarUI (energy display)

Best Practices 🧠

Assign CameraPivot and MainCamera before play; otherwise the controller will disable itself.

Pair with FPSStamina: set forceWalk = true when stamina = 0.

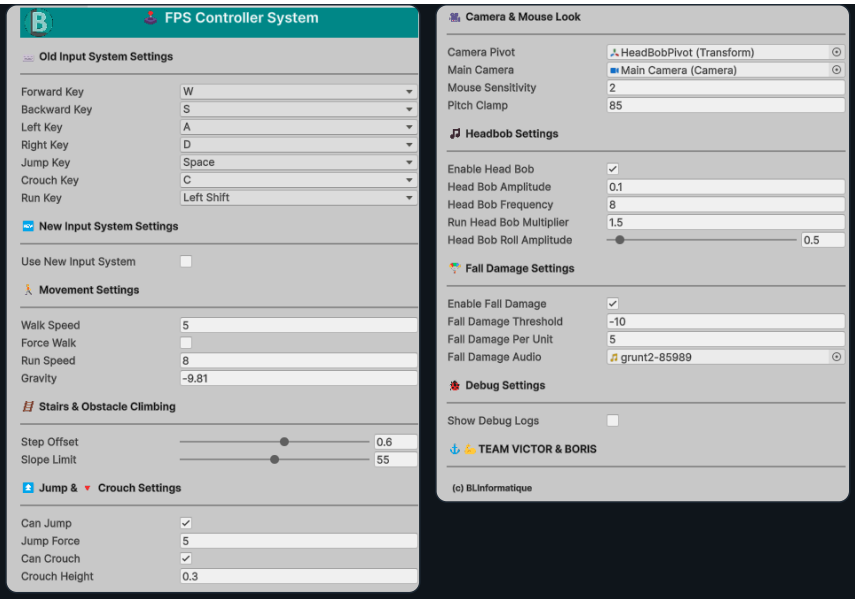
For survival gameplay, combine with FPSFood + FPSEnergy + **FPSHealth`.

Tune stepOffset and slopeLimit for stairs and terrain realism.

Keep headbob subtle for realism, but adjust amplitude/frequency for arcade feel.

⚓ 💪 TEAM VICTOR & BORIS Note

FPSControllerPro is the heart of RealFPS. Around it orbit stamina, health, weapons, inventory, and UI. Configure it once, and the rest of the RealFPS ecosystem falls into place.


FPS Controller System
Old Input System Settings
Forward Key W
Backward Key S
Left Key A
Right Key D
Jump Key Space
Crouch Key C
Run Key Left Shift
New Input System Settings
Use New Input System
Movement Settings
Walk Speed 5
Force Walk
Run Speed 8
Gravity -9.81
Stairs & Obstacle Climbing
Step Offset 0.6
Slope Limit 55
Jump & Crouch Settings
Can Jump
Jump Force 5
Can Crouch
Crouch Height 0.3
Camera & Mouse Look
Camera Pivot HeadBobPivot (Transform)
Main Camera Main Camera (Camera)
Mouse Sensitivity 2
Pitch Clamp 85
Headbob Settings
Enable Head Bob
Head Bob Amplitude 0.1
Head Bob Frequency 8
Run Head Bob Multiplier 1.5
Head Bob Roll Amplitude 0.5
Fall Damage Settings
Enable Fall Damage
Fall Damage Threshold -10
Fall Damage Per Unit 5
Fall Damage Audio grunt2-85989
Debug Settings
Show Debug Logs
TEAM VICTOR & BORIS
(c) BLInformatique

Field	Type	Tooltip
⬇		
forwardKey	`KeyCode`	Forward key (old input).
backwardKey	`KeyCode`	Backward key (old input).
leftKey	`KeyCode`	Left key (old input).
rightKey	`KeyCode`	Right key (old input).
jumpKey	`KeyCode`	Jump key (old input).
crouchKey	`KeyCode`	Crouch key (old input).
runKey	`KeyCode`	Run key (old input).
⬇		
useNewInputSystem	`Boolean`	Enable new Input System.
⬇ 🎮 New Input System Actions		
moveAction	`InputAction`	Move action.
jumpAction	`InputAction`	Jump action.
crouchAction	`InputAction`	Crouch action.
runAction	`InputAction`	Run action.
⬇		
walkSpeed	`Single`	Walk speed.
forceWalk	`Boolean`	If true, the player will always walk (never run), even if the run key is pressed. Useful for stamina or special gameplay states.
runSpeed	`Single`	Run speed.
gravity	`Single`	Gravity (should be negative).
⬇		
stepOffset	`Single`	Step offset for the Character Controller (max step height the player can climb). • Range [0,1..1]
slopeLimit	`Single`	Slope limit for the Character Controller (maximum climbable slope in degrees). • Range [30..80]
⬇		
canJump	`Boolean`	Can jump?
jumpForce	`Single`	Jump force.
canCrouch	`Boolean`	Can crouch?
crouchHeight	`Single`	Crouch height.
⬇		
cameraPivot	`Transform`	Camera pivot (for pitch/headbob/roll).

Field	Type	Tooltip
mainCamera	`Camera`	Camera component.
mouseSensitivity	`Single`	Mouse sensitivity.
pitchClamp	`Single`	Pitch clamp (vertical look limit).
↓		
enableHeadBob	`Boolean`	Enable headbob?
headBobAmplitude	`Single`	Headbob amplitude.
headBobFrequency	`Single`	Headbob frequency.
runHeadBobMultiplier	`Single`	Headbob run multiplier.
headBobRollAmplitude	`Single`	Max roll Z for headbob. • Range [0..10]
↓		
enableFallDamage	`Boolean`	Enable fall damage for player.
fallDamageThreshold	`Single`	Minimum vertical speed (negative) to trigger fall damage.
fallDamagePerUnit	`Single`	Amount of damage per extra unit of vertical speed (over threshold).
fallDamageAudio	`AudioClip`	Play this sound when taking fall damage.
↓		
showDebugLogs	`Boolean`	Show debug logs.

↑ Back to top

Copy fields

📦 FPSCrosshair MonoBehaviour

FPSCrosshair 🎯

The FPSCrosshair manages the dynamic crosshair in RealFPS.

It handles visibility, size animation when firing, and color feedback on hits.

Crosshair Settings 🎨

Crosshair Image 🖼️ – The UI.Image used as crosshair (anchor at screen center).

Show Crosshair 👁️ – Toggles visibility.

Default Color ⚪ – Normal state.

Hit Color 🔴 – Temporary color shown when a hit is detected.

Default Scale 📏 – Base scale of the crosshair.

Fire Scale 🔫 – Expanded size when firing.

Shrink Speed ⏱️ – How fast the crosshair returns to default size.

Hit Color Time ⏳ – Duration (seconds) of hit color before reverting.

Runtime Behaviour 🧭

At Start: crosshair is initialized with default color and scale.

At Update:

Smoothly shrinks from fireScale back to defaultScale.

Resets hit color back to default after hitColorTime.

Public API 🧩

↑

AnimateFire() 🔫 – Expands crosshair to fireScale (call when firing).

AnimateHit() 🎯 – Sets hit color and starts the hit timer (call when a shot hits).

SetCrosshairVisible(bool visible) 👁️ – Shows or hides crosshair.

UpdateCrosshair() 🔄 – Manually refreshes state (color, visibility, scale).

ResetCrosshairSize() 📏 – Forces target scale back to defaultScale.

Integration 🔗

Weapon systems (FPSWeaponManager) should call:

AnimateFire() on shot.

AnimateHit() when a projectile successfully hits an enemy or surface.

UI logic can toggle crosshair visibility (e.g., hide in ADS).

Best Practices 🧠

Always anchor the crosshair Image to the center of the screen.

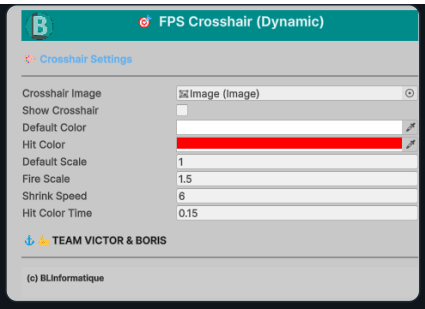
Use subtle Fire Scale (1.2–1.5) for feedback without distraction.

Use short Hit Color Time for responsive feedback (0.1–0.2s).

Hide crosshair during ADS (Aim Down Sight) if using realistic scopes.

⚓💪 TEAM VICTOR & BORIS Note

The crosshair is more than a dot — it's instant player feedback. Expanding on fire, flashing red on hits, disappearing in ADS: small touches that make your FPS feel alive.



Field	Type	Tooltip
⬇		
crosshairImage	`Image`	Image UI to use as crosshair (should be anchored to center).
showCrosshair	`Boolean`	Show or hide crosshair.
defaultColor	`Color`	Default crosshair color.
hitColor	`Color`	Color when hit detected.
defaultScale	`Single`	Default crosshair size (scale).
fireScale	`Single`	Scale multiplier when firing.
shrinkSpeed	`Single`	Time for the crosshair to return to default size (seconds).
hitColorTime	`Single`	Time to keep hit color (seconds).

↑ Back to top

Copy fields

📦 FPSEnergy MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Energy/Energy UI (FPSEnergy)`

FPSEnergy 🔋 (UI Manager)

The FPSEnergy is the UI manager for energy bars in RealFPS.

It displays and animates the player's or equipped device's energy state, with support for blinking warnings, anchoring, and auto-hide.

UI Elements 🖼️

Energy Panel 🟦 – Main RectTransform root of the energy bar.

Energy Image 🎨 – Fill Image representing energy level.

Energy Root CanvasGroup 🚹 – Optional group for alpha fading/pulsing effects.

Anchor Settings 📍

Anchor – Position of the energy bar on screen: TopLeft, TopRight, BottomLeft, BottomRight, TopMiddle, BottomMiddle.

Always Show When Equipped 👁️ – Keeps bar visible when a device is equipped, even if energy is full.

Standalone Fallback ⚙️

(Used when no EnergyConsumer is bound.)

Max Energy 🩺 – Maximum energy value.

Decay Per Second ⏱️ – Drain rate when StartConsumption() is active.

Hide When Full 🙈 – Automatically hides the bar when energy is full and no device is equipped.

Binding 🔗

↑

Bound Consumer 🔋 – Current EnergyConsumer (set by FPSWeaponManager on equip).

Auto Toggle With Equip ✅ – Automatically shows/hides when an item is equipped/unequipped.

Methods:

Bind(EnergyConsumer) – Connects to a consumer, subscribes to its events.

OnEquipped() – Shows the panel (if auto toggle is enabled).

OnUnequipped() – Hides the panel and resets defaults.

Low Energy Blink ⚡

Low Energy Threshold 📉 – Ratio (0–1) under which blinking starts.

Blink Mode ✨ – Options: None, Pulse, Strobe.

UI Warning Color 🟥 – Color blended in low energy state.

Pulse Speed 🔁 – Speed of pulse blinking.

Strobe Interval 🎲 – Random delay between strobe toggles.

Alpha Pulse Amount ⬜ – How much the CanvasGroup fades during pulse.

Public API 🧩

StartConsumption() / StopConsumption() – Legacy draining mode (no consumer).

RestoreEnergy(float) – Adds energy (direct or via bound consumer).

RefreshUIImmediate() – Forces UI refresh.

ApplyAnchor() – Updates bar position on screen.

Runtime Behaviour 🧭

When equipped:

FPSWeaponManager calls Bind() + OnEquipped().

Energy is updated via onEnergyChanged event from the device.

When unequipped:

OnUnequipped() hides the bar.

Low energy:

UI enters blinking (Pulse/Strobe) mode.

Legacy mode:

If no consumer is bound, FPSEnergy can still drain over time using StartConsumption().

Best Practices 🧠

Use EnergyConsumer on devices (flashlight, goggles, tools), and let FPSEnergy display their state.

Always pair with EnergyBarUI for cleaner binding from equipment flow.

For survival gameplay, set hideWhenFull = true for minimal UI clutter.

Use Pulse mode for immersion, Strobe mode for urgency (battery almost dead).

Related Components 🔗

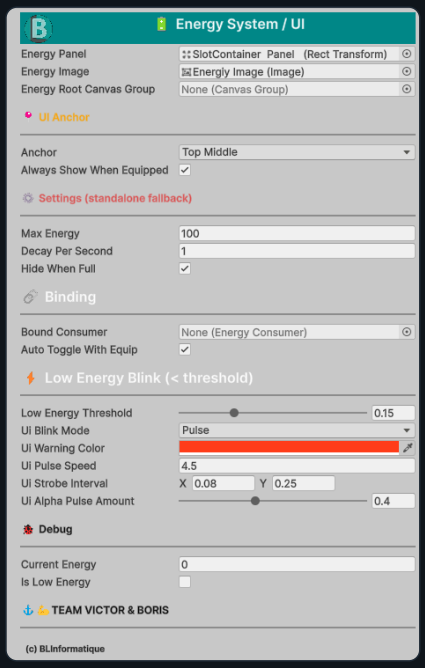
EnergyConsumer – Device-level energy logic.

EnergyBarUI – Adapter that binds EnergyConsumer → FPSEnergy.

FPSWeaponManager – Equips items and triggers binding.

⚓ 💪 TEAM VICTOR & BORIS Note

FPSEnergy ensures your UI clearly reflects gameplay states — from full power to dying batteries. Subtle blinks or urgent strobes keep players aware without distracting them from the action.



Field	Type	Tooltip
energyPanel	`RectTransform`	Main RectTransform of the energy bar UI (to move/anchor).
energyImage	`Image`	Image (fillAmount) for energy bar.
energyRootCanvasGroup	`CanvasGroup`	Optional CanvasGroup of the bar root (for alpha pulse).
↓		
anchor	`EnergyAnchor`	Anchor position for energy UI on screen.
alwaysShowWhenEquipped	`Boolean`	Keep bar visible when a device is equipped, even if full.
↓		
maxEnergy	`Single`	Max energy when not bound to an EnergyConsumer.
decayPerSecond	`Single`	Energy decay rate per second when StartConsumption() is active (legacy mode).
hideWhenFull	`Boolean`	Auto-hide bar when energy is full and nothing is equipped.
↓		
boundConsumer	`EnergyConsumer`	Bound EnergyConsumer (set by FPSWeaponManager on Equip).
autoToggleWithEquip	`Boolean`	Show bar when equipped; hide on unequip.
↓		
lowEnergyThreshold	`Single`	Blink UI when energy below this ratio (0.15 = 15%). • Range [0,01..0,5]
uiBlinkMode	`BlinkMode`	Blinking mode for the UI.
uiWarningColor	`Color`	UI color when low (lerped from original).
uiPulseSpeed	`Single`	Pulse speed (Hz-ish).
uiStrobeInterval	`Vector2`	Strobe interval range (seconds).
uiAlphaPulseAmount	`Single`	Canvas alpha pulse amount (0..1). • Range [0..1]
↓		
currentEnergy	`Single`	
isLowEnergy	`Boolean`	

↑ Back to top

Copy fields

📦 FPSFood MonoBehaviour

FPSFood 🍗

The FPSFood system tracks the player's hunger in RealFPS.

It decreases food over time, displays a UI bar, provides low-food warnings (blink + sound), and drains health when food reaches zero.

Food Bar UI 📦

Food Panel 🟦 – Root RectTransform of the bar.

Food Image 🎨 – UI fill Image representing food value.

Food Root CanvasGroup 🚹 – Optional group for alpha pulsing.

Anchor 📍 – On-screen position of the bar (TopLeft, BottomMiddle, etc.).

Food Settings ⚙️

Max Food 🍽️ – Maximum food value.

Decay Per Second ⏱️ – How fast food decreases automatically.

Health Drain Per Second ❤️ – Damage applied to FPSHealth when food = 0.

Low Food Feedback 🍗⚠️

Low Food Threshold 📉 – Ratio below which warnings start (default 15%).

Blink Mode ✨ – None, Pulse, or Strobe.

UI Warning Color 🟥 – Color blended during low food state.

Pulse Speed 🔁 – Frequency of pulse effect.

Strobe Interval 🎲 – Random delay range for strobe flicker.

↑

Alpha Pulse Amount 🌫️ – Fade intensity for CanvasGroup.

Low Food Alert Clip 🎵 – Looping sound played when low on food.

Alert Volume 🔉 – Sound volume.

Alert Source 🚰 – AudioSource used (auto-created if null).

Debug 🐞

Current Food 🍗 – Runtime food value.

Is Low Food ⚠️ – True if under threshold.

Runtime Behaviour 🧭

Decay – Food decreases each frame by decayPerSecond.

UI Update – Updates fill amount and shows bar if hidden.

Low Food – If below threshold:

Blinks UI (pulse/strobe).

Plays alert sound (looped).

Normal – Restores default UI color/alpha, stops sound.

At Zero – Calls FPSHealth.TakeDamage() each second (starvation).

Public API 🧩

RestoreFood(float amount) – Increases food (clamped to max).

ApplyAnchor() – Sets bar position on screen.

Integration 🔗

Automatically drains FPSHealth when food = 0.

Displayed UI can be anchored anywhere on screen.

Works together with FPSHealth and FPSStamina for survival gameplay.

Best Practices 🧠

Tune decayPerSecond based on desired survival pacing.

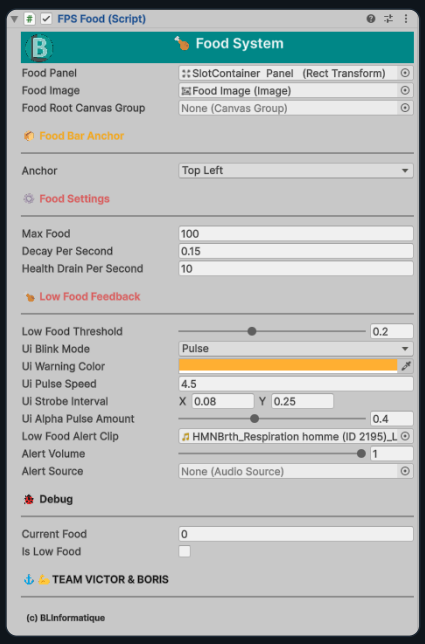
Use Pulse for subtle warning, Strobe + Sound for urgent state.

Provide items (consumables) that call RestoreFood() to replenish.

For multiplayer, sync currentFood and play warnings locally.

⚓💪 TEAM VICTOR & BORIS Note

FPSFood ties survival mechanics to player feedback: a visible bar, blinking UI, warning sounds, and direct health impact. It ensures hunger is not just a stat but a gameplay driver.



Field	Type	Tooltip
foodPanel	`RectTransform`	Main RectTransform of the food bar UI (to move/anchor).
foodImage	`Image`	Image (fillAmount) for food bar.
foodRootCanvasGroup	`CanvasGroup`	Optional CanvasGroup for alpha pulse.
↓		
anchor	`FoodAnchor`	Anchor position for food UI on screen.
↓		
maxFood	`Single`	Max food value.
decayPerSecond	`Single`	Food decay rate per second.
healthDrainPerSecond	`Single`	Health drain per second when food is 0.
↓		
lowFoodThreshold	`Single`	Blink UI when food below this ratio (0.15 = 15%). • Range [0,01..0,5]
uiBlinkMode	`BlinkMode`	Blinking mode for the UI.
uiWarningColor	`Color`	UI color when low (lerped from original).
uiPulseSpeed	`Single`	Pulse speed (Hz-ish).
uiStrobeInterval	`Vector2`	Strobe interval range (seconds).
uiAlphaPulseAmount	`Single`	Canvas alpha pulse amount (0..1). • Range [0..1]
lowFoodAlertClip	`AudioClip`	Low food alert sound to play (looped).
alertVolume	`Single`	Volume of alert audio. • Range [0..1]
alertSource	`AudioSource`	AudioSource to play alert sound (will add one if null).
↓		
currentFood	`Single`	
isLowFood	`Boolean`	

↑ Back to top

Copy fields

📦 FPSFootstepPlayer MonoBehaviour

FPSFootstepPlayer 👣

The FPSFootstepPlayer handles footstep sounds for the player.

It supports surface-dependent audio (terrain textures, collider SurfaceTypes), as well as different step intervals for walking, running, and crouching.

Audio Settings 🎧

Footstep Audio Source 🔊 – Optional. If null, uses AudioSource.PlayClipAtPoint.

Footstep Volume 🎚️ – Global volume for footstep playback.

Step Settings 🦶

Step Interval Walk 🚶 – Time (seconds) between steps while walking.

Step Interval Run 🏃 – Time between steps while running.

Step Interval Crouch 🤫 – Time between steps while crouched.

Terrain Mapping 🗺️

Terrain Surface Data Array 🗺️ – Array of SurfaceData assets mapped to terrain textures (order must match terrain texture slots).

Runtime Behaviour 🧭

Grounded check – No footsteps if airborne.

Movement detection – Only plays when movement input > threshold.

Interval – Step rate is chosen based on:

Running → stepIntervalRun.

↑

Crouching → stepIntervalCrouch.

Default → stepIntervalWalk.

Surface detection:

First, raycast beneath player.

If hit collider has SurfaceType with valid SurfaceData → use it.

Else, if hit collider is terrain → determine dominant texture at hit point and map via terrainSurfaceDataArray.

Play clip: Randomly selects one audio clip from chosen surface's surfaceFootstepAudio[].

Integration 🔗

Requires CharacterController (to check grounded state).

Works best with FPSControllerPro (uses its moveInput, isRunning, isCrouching).

Needs SurfaceData assets configured for each ground type (concrete, wood, grass, etc.).

Can be combined with SurfaceType component on colliders for non-terrain surfaces.

Best Practices 🧠

Ensure terrainSurfaceDataArray matches terrain texture order (first slot = grass, etc.).

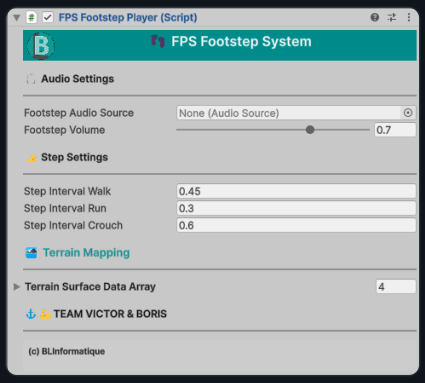
Use different intervals for stealth vs normal movement. Longer = quieter immersion.

Place SurfaceType components on floor colliders (wood planks, metal grids) for variety.

Keep clips short and varied to avoid repetition.

⚓💪 TEAM VICTOR & BORIS Note

Footsteps are subtle but powerful: they anchor the player in the world. With FPSFootstepPlayer, every step on wood creaks, metal clangs, or grass rustles, amplifying immersion in RealFPS.



Field	Type	Tooltip
⬇		
footstepAudioSource	`AudioSource`	AudioSource to use for footsteps (optional, can use PlayClipAtPoint if null).
footstepVolume	`Single`	Volume for footsteps. • Range [0..1]
⬇		
stepIntervalWalk	`Single`	Interval between steps when walking.
stepIntervalRun	`Single`	Interval between steps when running.
stepIntervalCrouch	`Single`	Interval between steps when crouched.
⬇		
terrainSurfaceDataArray	`SurfaceData[]`	SurfaceData array matching your terrain textures order. One SurfaceData per terrain texture slot!

↑ Back to top

Copy fields

📦 FPSHealth MonoBehaviour

FPSHealth ❤️

The FPSHealth system manages player health, health bar UI, low-health feedback, death logic, and respawn/restart in RealFPS.

It integrates UI, audio, and gameplay events to give clear player feedback and control.

Health Bar UI 📦

Health Panel 🟦 – Root RectTransform of the health bar.

Health Image 🎨 – UI fill Image for health.

Health Root CanvasGroup 🚺 – Optional group for alpha pulse effects.

Anchor 📍 – Position on screen (TopLeft, BottomMiddle, etc.).

Settings ⚙️

Max Health ❤️ – Maximum player health.

Hide When Full 🙈 – Optionally hides bar at full health.

Hide On Death ☠️ – Hides bar when player dies.

Low Health Feedback 💝

Low Health Threshold 📉 – Ratio below which warnings start.

Blink Mode ✨ – None, Pulse, or Strobe.

UI Warning Color 🟥 – Blend color in low state.

Pulse Speed 🔁 – Pulse oscillation speed.

Strobe Interval 🎡 – Random strobe timing.

↑

Alpha Pulse Amount 🌫️ – Fade intensity.

Heartbeat Clip ❤️‍🔥 – Looping sound on low health.

Heartbeat Volume / Source 🔊 – Audio configuration.

Damage Feedback 🩸

Damage Audio Clips 🎶 – Array of sounds for hits.

Damage Audio Source 🚹 – Source to play sounds.

Damage Audio Volume 🚹 – Volume of hit sounds.

Damage Flash Image 🟥 – Optional screen overlay flash.

Flash Color / Duration ⏱️ – Defines damage flash effect.

Death Settings 💀

Death Panel 🪟 – UI panel shown on death.

Death Audio Clip / Source 🔊 – Optional death sound.

Disable Controller 🕹️ – Disables FPSControllerPro on death.

Disable Weapons 🔫 – Disables FPSWeaponManager on death.

Hide Crosshair 🎯 – Hides FPSCrosshair when dead.

On Death (UnityEvent) 📡 – Event hook for custom logic (score, FX).

Respawn / Restart 🔁

Auto Respawn ⏱️ – Automatically restarts scene after delay.

Respawn Delay ⏳ – Delay before respawn.

RestartScene() – Reloads active scene (call from UI button if manual).

Debug 🐞

Current Health ❤️ – Runtime value.

Is Dead ☠️ – Status flag.

Is Low Health ⚠️ – Status flag.

Public API 🧩

TakeDamage(float amount) – Subtracts health, plays feedback, triggers death if ≤ 0.

Heal(float amount) – Restores health (clamped).

RestartScene() – Reloads active scene.

Runtime Behaviour 🧭

Updates UI fill and feedback each frame.

When under threshold → UI blinks, heartbeat starts.

When damaged → Plays random sound, flashes overlay.

When dead → Disables controller/weapons, hides crosshair, shows panel, plays sound, invokes onDeath.

Auto-respawn or waits for restart input.

Integration 🔗

Works with FPSControllerPro (disabled on death).

Disables FPSWeaponManager when dead.

Hides FPSCrosshair for clarity.

Combined with FPSFood and FPSEnergy for full survival loop.

Compatible with FPSStamina to link exhaustion/damage effects.

Best Practices 🧠

Use Pulse blink for immersive low-health, Strobe for panic feedback.

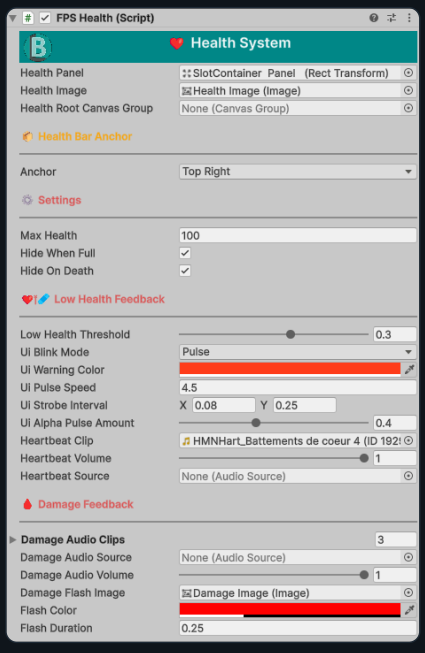
Always provide a Death Panel with “Restart” option if not using auto-respawn.

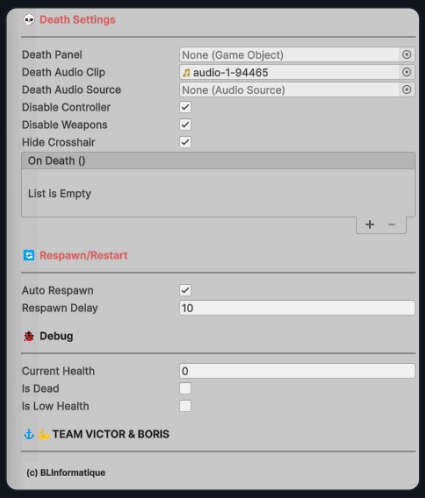
Balance fallDamage (FPSControllerPro) and health regen/healing to fit game pacing.

Pair with audio + overlay for strong feedback without overwhelming.

⚓💪 TEAM VICTOR & BORIS Note

FPSHealth ensures the player’s survival state is always visible, audible, and gameplay-relevant. It’s the anchor of the survival system — where health, food, stamina, and energy converge.





Field	Type	Tooltip
healthPanel	`RectTransform`	Main RectTransform of the health bar UI (to move/anchor).
healthImage	`Image`	Image (fillAmount) for health bar.
healthRootCanvasGroup	`CanvasGroup`	Optional CanvasGroup for alpha pulse.
↓		
anchor	`HealthAnchor`	Anchor position for health UI on screen.
↓		
maxHealth	`Single`	Max health value.
hideWhenFull	`Boolean`	Auto hide bar when health is full.
hideOnDeath	`Boolean`	Hide bar when player is dead.
↓		
lowHealthThreshold	`Single`	Blink UI when health below this ratio (0.15 = 15%). • Range [0,01..0,5]
uiBlinkMode	`BlinkMode`	Blinking mode for the UI.
uiWarningColor	`Color`	UI color when low (lerped from original).
uiPulseSpeed	`Single`	Pulse speed (Hz-ish).
uiStrobeInterval	`Vector2`	Strobe interval range (seconds).
uiAlphaPulseAmount	`Single`	Canvas alpha pulse amount (0..1). • Range [0..1]
heartbeatClip	`AudioClip`	Heartbeat sound to play on low health (looped).
heartbeatVolume	`Single`	Volume of heartbeat audio. • Range [0..1]
heartbeatSource	`AudioSource`	AudioSource to play heartbeat sound (will add one if null).
↓		
damageAudioClips	`AudioClip[]`	Sound to play when player takes damage (optional, random if several).
damageAudioSource	`AudioSource`	AudioSource to play damage sound (optional, will add one if null).
damageAudioVolume	`Single`	Volume of damage audio. • Range [0..1]
damageFlashImage	`Image`	Optional: image to flash red on damage.
flashColor	`Color`	Flash color for damage.
flashDuration	`Single`	Flash duration (seconds).
↓		
deathPanel	`GameObject`	Panel UI to show on death (ex: 'DeathPanel').
deathAudioClip	`AudioClip`	Play this sound on death (optional).
deathAudioSource	`AudioSource`	AudioSource to play death sound (optional, will add one if null).

Field	Type	Tooltip
disableController	`Boolean`	Disable FPSController on death.
disableWeapons	`Boolean`	Disable FPSWeaponManager on death.
hideCrosshair	`Boolean`	Hide crosshair on death.
onDeath	`UnityEvent`	Event called on player death. (Plug your UI/FX here)

↓		
autoRespawn	`Boolean`	Use automatic respawn after death (if false, show UI panel with button).
respawnDelay	`Single`	Delay before automatic respawn (seconds).

↓		
currentHealth	`Single`	
isDead	`Boolean`	
isLowHealth	`Boolean`	

↑ Back to top

Copy fields

📦 FPSStamina MonoBehaviour

FPSStamina 💪

The FPSStamina system manages the player's stamina bar, draining it while running and regenerating it when resting.

It also enforces forced walking when exhausted and plays breathing audio feedback.

References 🔗

FPS Controller 🕹️ – Link to your FPSControllerPro.

Stamina Panel 🟦 – Root RectTransform for the stamina bar.

Stamina Image 🎨 – Fill Image of the bar.

Anchor 📍

Anchor – Choose where the stamina bar is displayed (TopLeft, BottomMiddle, etc.).

Settings ⚙️

Max Stamina 🔝 – Maximum capacity.

Drain Per Second ⏱️ – Amount drained when running.

Regen Per Second 🌱 – Amount restored when not running.

Min Run Stamina 🏃 – Minimum stamina required to run again.

Breath Audio 😮‍💨

Breath Audio Clip 🎵 – Looping heavy breathing sound when exhausted.

Breath Audio Source 🚹 – Optional, auto-created if null.

↑

Breath Volume 🔊 – Sound volume.

Debug 🐞

Current Stamina 📊 – Current runtime value.

Is Exhausted 😵 – True when stamina = 0 and player is forced to walk.

Runtime Behaviour 🧭

Running:

Drains stamina.

If stamina reaches 0 → sets isExhausted = true, forces walk mode (fpsController.forceWalk = true).

Resting:

Regenerates stamina.

When stamina ≥ minRunStamina → allows running again (forceWalk = false).

UI:

Fills bar proportionally.

Auto-shows when running or stamina < max.

Audio:

Plays breathing sound when exhausted.

Stops when stamina recovers.

Public API 🧩

ApplyAnchor() – Repositions bar on screen.

Integration 🔗

Works directly with FPSControllerPro:

Reads isRunning and moveInput.

Calls SetRunning() and sets forceWalk when exhausted.

Combines with FPSHealth and FPSFood for survival mechanics.

Best Practices 🧠

Balance drain vs regen rates to match game pacing (arcade vs survival).

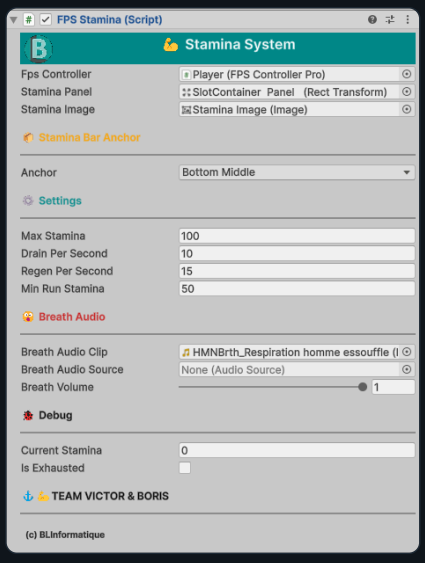
Use breathing audio sparingly to avoid annoyance.

Consider coupling stamina with actions (melee swings, jumps) for deeper gameplay.

Keep minRunStamina > 0 for smoother transitions (prevents stutter between walk/run).

⚓💪 TEAM VICTOR & BORIS Note

FPSStamina makes sprinting a tactical choice. Run too long and you're forced to slow down, with heavy breathing warning you — a simple but powerful immersion layer for RealFPS.



Field	Type	Tooltip
fpsController	`FPSControllerPro`	Reference to your FPSController.
staminaPanel	`RectTransform`	Main RectTransform of the stamina UI (to move/anchor).
staminaImage	`Image`	Image (fillAmount) for stamina bar.

↓		
anchor	`StaminaAnchor`	Anchor position for stamina UI on screen.

↓		
maxStamina	`Single`	Max stamina value.
drainPerSecond	`Single`	Stamina drain rate per second when running.
regenPerSecond	`Single`	Stamina regen rate per second when not running.
minRunStamina	`Single`	Minimum stamina required to run again.

↓		
breathAudioClip	`AudioClip`	AudioClip to play when player is exhausted (heavy breathing, looping).
breathAudioSource	`AudioSource`	AudioSource to use (optional, will add one if null).
breathVolume	`Single`	Volume of the breath audio. • Range [0..1]

↓		
currentStamina	`Single`	
isExhausted	`Boolean`	

↑ Back to top

Copy fields

📦 FPSWeaponManager MonoBehaviour

FPSWeaponManager 🔫

The FPSWeaponManager equips arms/weapon prefabs, handles fire / ADS / reload, drives arms Animator states, updates crosshair, binds energy UI for devices, and spawns surface impact FX.

It integrates tightly with WeaponInventoryManager, GeneralInventoryUI, FPSControllerPro, FPSCrosshair, FPSEnergy, and SurfaceData.

References & Holders 🔗

Arms Holder 🧩 – Parent Transform where the arms + weapon prefab is instantiated.

Equipped Data (ReadOnly) 📄 – Current ArmsAndWeaponData.

Equipped Arms (ReadOnly) 🦾 – Instantiated arms GameObject in hand.

Default Arms Data 📦 – Optional fallback equipped when nothing else is held.

FPS Camera 📷 – Used for ADS FOV and raycast shots.

WeaponInventoryManager 🎒 – Provides ammo counts per slot.

GeneralInventoryUI 🟦 – Blocks firing when inventory is open.

FPSCrosshair 🎯 – UI feedback (show/hide, fire/hit pulses).

Editor utility: 📂 Save Arms Offsets to ScriptableObject (Play Mode) writes current local position/rotation back into the active ArmsAndWeaponData (or ActionArmsData).

Input ⌨️ / 🕹️

Legacy Input: fireKey (Mouse0), aimKey (Mouse1), reloadKey (R).

New Input System: fireAction, aimAction, reloadAction (enable with useNewInputSystem).

↑

Inventory safety: if GeneralInventoryUI.isOpen → input is ignored.

Equip / Unequip 🧭

Equip(ArmsAndWeaponData data)

Spawns arms prefab under Arms Holder, applies default (idle) offsets, restores camera FOV, configures audio pool and Animator idle.

Crosshair visibility follows data.isFirearm.

Energy binding: finds EnergyConsumer in equipped arms, assigns EnergyRuntime.CurrentEnergyConsumer, notifies FPSEnergy (Bind + OnEquipped() / OnUnequipped()).

Unequip()

Destroys current arms, clears data and energy binding, hides crosshair (unless a default data is assigned, which is then equipped).

ADS – Aim Down Sight 🎯

EnterADS() / ExitADS() (internal)

Swaps arms local position/rotation to ADS offsets from ArmsAndWeaponData.

Sets camera FOV to ADS value; restores on exit.

Hides crosshair during ADS if hideCrosshairOnADS is enabled.

Optionally triggers Animator adsTriggerName / returns to idleTriggerName.

Firing 🔥

Fire modes: SemiAuto (press) / FullAuto (held, rate-limited by fireRate).

Ammo checks: If isFirearm && !infiniteAmmo, consumes magazine; plays empty clip when mag = 0.

FX & audio: Triggers fireTriggerName, plays pooled fire audio, spawns muzzle flash at muzzleFlashPointName (falls back to root if not found).

Crosshair: AnimateFire() then AnimateHit() on successful raycast hit.

Raycast: from FPS Camera forward up to maxShootDistance (or manager shootDistance).

Damage: calls FPSHealth.TakeDamage() (player) or EnemyHealth.TakeDamage() (AI).

Headshots: if hit collider has tag "Head", multiplies damage by enemyHealth.headshotDamageMultiplier.

Surface impact: looks for SurfaceType.surfaceData, spawns hit FX/audio (neutral parent is created at hit to avoid scale/rotation issues).

Explosives: if ExplosionEntity found, calls RegisterHit().

Public: Fire(), Reload(), SpawnSurfaceImpactEffect(RaycastHit hit).

Reloading 🔁

Uses WeaponInventoryManager arrays:

currentMag[selectedSlot] and currentAmmo[selectedSlot].

Calculates ammo needed vs available, updates counts, triggers reloadTriggerName and plays reload audio.

Logs “No reserve ammo!” when appropriate.

Temporary Actions (Hands-only) ✋

EquipTempActionArms(ActionArmsData)

Stores previous slot/data (lastEquippedSlot / lastEquippedData), unequips weapon without fallback, instantiates action arms, plays actionAudio, triggers actionTriggerName.

If autoRestoreAfterAction and actionDuration > 0, starts coroutine to RestorePreviousEquipment() after the delay.

RestorePreviousEquipment()

Re-equips from WeaponInventoryManager slot or falls back to lastEquippedData.

Audio Pool 🔊

audioPoolSize sources are pre-created (2D non-spatial) for overlapping one-shots (fire/reload/empty).

PlayPooledOneShot(AudioClip, volume) cycles through the pool to avoid cut-offs.

Animator & Movement 🎬

Tracks FPSControllerPro.moveInput and isRunning to drive an integer MoveState (0 Idle / 1 Walk / 2 Run) on the arms Animator.

Ensures an Idle trigger is sent on equip if defined.

Utilities 🧰

FindDeepChild(Transform parent, string name) – Recursive search by exact name (used for muzzle point).

Best Practices 🧠

Keep ADS offsets close to idle and fine-tune with Arms Live Tuner; then “📁 Save Offsets…” to write back to the SO.

Name Animator parameters consistently across weapons (Idle, Fire, Reload, MoveState, Aim).

Tag head colliders with "Head" to enable headshot multipliers.

Ensure SurfaceType is present on environment colliders and that their SurfaceData have hit FX + audio configured.

For devices (flashlight/tools), rely on EnergyConsumer + FPSEnergy; the manager will bind/unbind automatically on equip/unequip.

↑

When using the New Input System, make sure you Enable() the actions on OnEnable() and Disable() on OnDisable() (already handled here).

Troubleshooting 🧯

No fire in FullAuto? Check fireRate, ensure Time.time gating isn't too strict, and confirm input path (legacy vs new).

Muzzle flash at wrong place? Verify muzzleFlashPointName matches a child transform name; otherwise it falls back to arms root.

Crosshair stays visible in ADS? Ensure hideCrosshairOnADS is enabled in ArmsAndWeaponData.

No damage on enemies? Confirm your colliders are on the expected objects (EnemyHealth in parent), and that raycast maxShootDistance is sufficient.

Inventory open blocks firing by design—close the UI or handle a "fire-through-inventory" exception if your game needs it.

Related Components 🔗

ArmsAndWeaponData / ActionArmsData – Data for poses, audio, ADS, actions.

WeaponInventoryManager – Ammo per slot, selected slot, equip by slot.

GeneralInventoryUI – UI state that blocks firing.

FPSCrosshair – Fire/Hit UI feedback.

EnergyConsumer / FPSEnergy / EnergyBarUI / EnergyRuntime – Device energy + HUD binding.

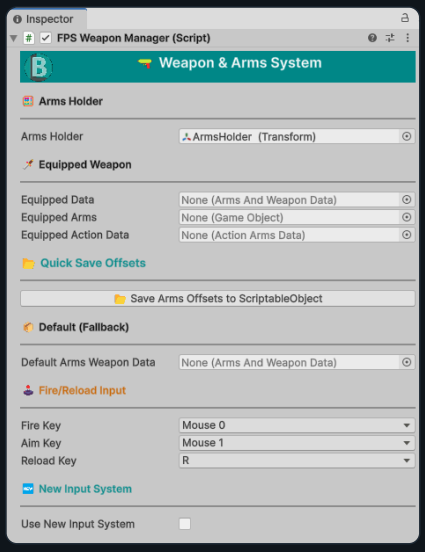
SurfaceType / SurfaceData – Impact FX + sounds.

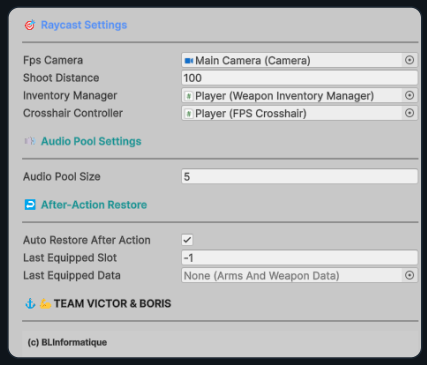
EnemyHealth / FPSHealth / ExplosionEntity – Damage targets & reactions.

FPSControllerPro – Movement state feeding arms Animator.

⚓ 💪 TEAM VICTOR & BORIS Note

FPSWeaponManager is your combat conductor: it keeps arms posing, input, ammo, FX, UI, and damage in sync—so you can focus on weapon feel, not plumbing.





Field	Type	Tooltip
↓		
armsHolder	`Transform`	Parent transform where arms + weapon are instantiated.
↓		
equippedData	`ArmsAndWeaponData`	Currently equipped arms/weapon data.
equippedArms	`GameObject`	Currently equipped arms (instance in hand).
equippedActionData	`ActionArmsData`	
↓		
boutonDummy	`Int32`	Click to save the current arms prefab position/rotation in the ScriptableObject (Play Mode only).
↓		
defaultArmsWeaponData	`ArmsAndWeaponData`	Default arms/weapon data (optional, unequip fallback).
↓		
fireKey	`KeyCode`	
aimKey	`KeyCode`	
reloadKey	`KeyCode`	
↓		
useNewInputSystem	`Boolean`	
fireAction	`InputAction`	
aimAction	`InputAction`	
reloadAction	`InputAction`	
↓		
fpsCamera	`Camera`	
shootDistance	`Single`	
inventoryManager	`WeaponInventoryManager`	Reference to WeaponInventoryManager for ammo management.
crosshairController	`FPSCrosshair`	Crosshair controller (UI)
↓		
audioPoolSize	`Int32`	Number of AudioSources to pool for firing sounds.

Field	Type	Tooltip
autoRestoreAfterAction	`Boolean`	If true, after a one-shot action (heal/drink/eat), the previous weapon is auto re-equipped.
lastEquippedSlot	`Int32`	Last selected weapon slot before temporary action.
lastEquippedData	`ArmsAndWeaponData`	Fallback to data if WeaponInventoryManager is missing.

↑ Back to top

Copy fields

📦 GeneralInventoryManager MonoBehaviour

GeneralInventoryManager 👜

The GeneralInventoryManager is the core inventory system in RealFPS.

It manages item slots, stackable logic, item usage flow, keys, ammo, and communicates with UI and gameplay systems.

Inventory Settings 📦

Max Slots 🔢 – Maximum number of slots.

Auto Add Empty Slot ➕ – If true, items are automatically placed into the first empty slot.

Inventory Content 🗃️

Slots 📋 – Runtime list of InventorySlot (each slot contains ItemData + quantity).

Inspector shows current inventory content (read-only).

Events & UI ⚙️

On Inventory Changed (UnityEvent) 🔁 – Fired when items are added, removed, or used (UI must refresh here).

Core API 🧩

GetAllItems() – Returns all ItemData currently in inventory.

AddItem(ItemData, int) – Adds item(s). Handles stacking if isStackable and maxStack.

RemoveItem(ItemData, int) – Removes item(s). Cleans empty slots.

GetItemCount(ItemData) – Returns total amount of a specific item.

HasItem(ItemData) – Returns true if at least one exists.

↑

PrintInventory() – Debug utility (prints slots and counts).

Keys Helpers 🔑

HasKeyId(string keyId, out ItemData foundKey)

Returns true if inventory contains a matching key.

"MASTER" is a universal key that unlocks all locks.

ConsumeKey(ItemData key)

Removes one instance of the key.

Used by DoorLockInteractable and other lockable objects.

Use Flow 🧭

The UseItem(int slotIndex) method handles item usage depending on ItemType:

Medicine 💊

Calls FPSHealth.Heal(amount).

If actionArmsData is set → equips via FPSWeaponManager.EquipTempActionArms().

If consumable → removes after use or after action duration.

Food 🍗

Calls FPSFood.RestoreFood(amount).

Optionally plays actionArmsData.

Removes if consumable.

Usable (batteries, tools) 🔋

If itemEnergy > 0 → recharges the currently bound EnergyConsumer (EnergyRuntime.CurrentEnergyConsumer).

Plays useAudio.

Removes if consumable.

If actionArmsData is set → temporary arms action.

Ammo 💥

If linked to a WeaponData, adds ammo to the corresponding slot in WeaponInventoryManager.

Prints debug if ammo doesn't match owned weapon.

Weapon 🔫

Adds a new weapon to WeaponInventoryManager.

Key 🔑

Keys are not used from the inventory. They are consumed automatically by lock scripts.

Coroutines ⏳

RemoveAfterAction(ItemData, float delay)

Waits for action duration, then consumes the item if it is consumable.

Runtime Integration 🔗

FPSHealth – Medicine restores health.

FPSFood – Food restores hunger.

FPSWeaponManager – Handles action arms (animations) or equipping.

WeaponInventoryManager – Adds weapons or ammo.

EnergyRuntime / EnergyConsumer – Batteries recharge devices.

GeneralInventoryUI – Displays/refreshes slots when onInventoryChanged is invoked.

DoorLockInteractable – Uses key helpers.

Best Practices 🧠

Always connect OnInventoryChanged to UI for live updates.

Use isStackable + maxStack for consumables like ammo or food.

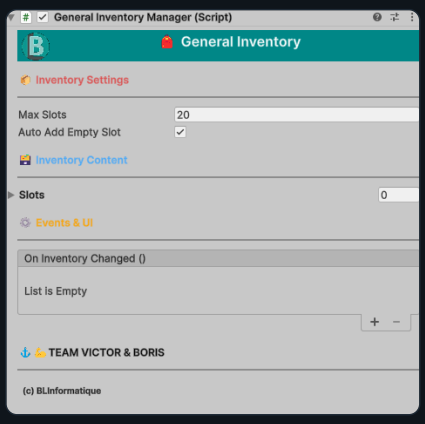
Keep UseItem() as the central usage flow: don't bypass it in gameplay.

Assign actionArmsData on items to add animations (drinking, eating, healing).

Use MASTER key for debugging locked doors.

⚓💪 TEAM VICTOR & BORIS Note

GeneralInventoryManager is the backbone of RealFPS survival gameplay: it turns raw ScriptableObjects into real, usable items. Medicines heal, food restores, ammo reloads, batteries recharge — all routed through a single flow.



Field	Type	Tooltip
↓		
maxSlots	`Int32`	Maximum number of inventory slots.
autoAddEmptySlot	`Boolean`	If true, auto-add to first empty slot.
↓		
slots	`List<InventorySlot>`	Current content of the inventory (inspector view).
↓		
onInventoryChanged	`UnityEvent`	Call this event when inventory changes (for UI refresh).

↑ Back to top

Copy fields

📦 GeneralInventoryUI MonoBehaviour

GeneralInventoryUI 👜🎛️

The GeneralInventoryUI manages the user interface for the inventory.

It displays item slots, handles Use/Equip buttons, supports both Input Systems, and locks/unlocks player controls when the inventory is open.

UI References 📦

Inventory Manager 🎭 – Link to GeneralInventoryManager.

Inventory Panel 🎛️ – Root panel GameObject for inventory.

Slots Panel 🧩 – Parent container (GridLayoutGroup recommended).

Slot Prefab 🎨 – Prefab for slot visuals (Icon + Quantity + Buttons).

Style 🎨

Slot Default Color 🟣 – Normal background color.

Slot Active Color 🟢 – Highlight color for usable items.

Show Use Button Only If Usable ✅ – If true, hides “Use” buttons for non-usable items.

Input Settings ⌨️🎮

Old Input System

Key ⌨️ – Default Tab. Toggles open/close.

New Input System

Use New Input System 🆕 – Enables InputAction workflow.

Keypress / KeyDown / KeyUp Events 🎛️ – InputActions for open/close.

↑

Toggle Settings 🔁

Use Toggle ↔️ – Press once to toggle, or hold if disabled.

Lock Cursor & Disable FPS 🔒 – Locks/unlocks mouse, disables FPSControllerPro while inventory is open.

Inventory Behavior 🧭

Auto Close After Equip 🛠️ – Closes UI automatically when equipping.

Auto Close After Use 🛠️ – Closes UI automatically when using items.

Is Open (ReadOnly) 👁️ – True if panel is visible.

Runtime Behaviour 🧭

Opening

Shows inventory panel.

Locks cursor, disables FPS controller (optional).

Calls UpdateUI().

Closing

Hides panel.

Restores cursor lock and re-enables FPS controller.

Updating UI

Clears previous slots, instantiates new ones.

Displays Icon, Quantity, and tooltips (from ItemData).

Buttons:

Use: Calls GeneralInventoryManager.UseItem(slot).

Equip: Equips via FPSWeaponManager or WeaponInventoryManager.

Colors slots according to usability.

Integration 🔗

GeneralInventoryManager – Provides slots and handles usage logic.

FPSWeaponManager – Equips ActionArmsData or weapons.

WeaponInventoryManager – Handles ammo and weapon slots.

EnergyRuntime.CurrentEnergyConsumer – Required for using battery-type Usable items.

FPSControllerPro – Disabled when UI is open.

Best Practices 🧠

Connect GeneralInventoryManager.onInventoryChanged to UpdateUI() for live refresh.

Customize slotPrefab (icons, tooltips, style) to fit your game's design.

Keep autoCloseAfterEquip/use enabled for smoother gameplay.

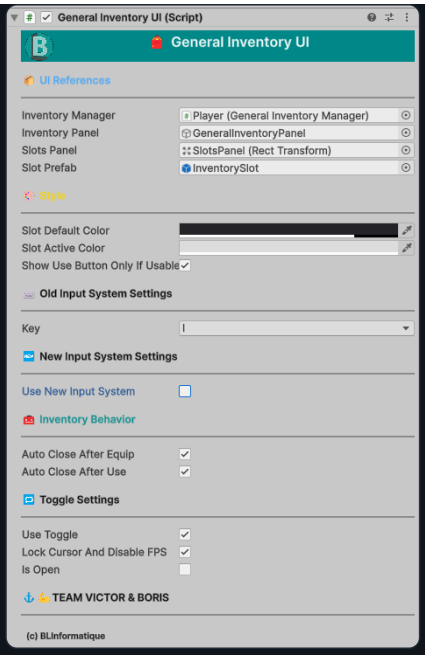
Ensure EnergyRuntime.CurrentEnergyConsumer is valid to allow battery usage from UI.

Place the UI in its own Canvas with proper sorting to overlay correctly.

⚓💪 TEAM VICTOR & BORIS Note

The GeneralInventoryUI is the front-end of your survival inventory. Smooth toggling, clear slots, and instant equip/use make players feel in control while keeping the game immersive.

↑


General Inventory UI (Script)
General Inventory UI
UI References
Inventory Manager
Player (General Inventory Manager)
Inventory Panel
GeneralInventoryPanel
Slots Panel
SlotsPanel (Rect Transform)
Slot Prefab
InventorySlot
Style
Slot Default Color
Slot Active Color
Show Use Button Only If Usable
Old Input System Settings
Key
I
New Input System Settings
Use New Input System
Inventory Behavior
Auto Close After Equip
Auto Close After Use
Toggle Settings
Use Toggle
Lock Cursor And Disable FPS
Is Open
TEAM VICTOR & BORIS
(c) BLInformatique

Field	Type	Tooltip
⬇		
inventoryManager	`GeneralInventoryManager`	Reference to the GeneralInventoryManager.
inventoryPanel	`GameObject`	
slotsPanel	`Transform`	Parent panel for inventory slots (use a GridLayoutGroup).
slotPrefab	`GameObject`	Prefab for a single slot (icon + quantity + button).
⬇		
slotDefaultColor	`Color`	Default color for slots.
slotActiveColor	`Color`	Color for selected or usable slot.
showUseButtonOnlyIfUsable	`Boolean`	Show 'Use' button only for usable items.
⬇		
key	`KeyCode`	🔘 Key used to toggle the inventory (Old Input System).
⬇		
useNewInputSystem	`Boolean`	Enable this if you're using the new Input System.
⬇ 🎮 New Input System Actions		
keypressEvent	`InputAction`	🔵 Input action for 'key held'.
keyDownEvent	`InputAction`	⬇️ Input action for 'key down'.
keyUpEvent	`InputAction`	⬆️ Input action for 'key up'.
⬇		
autoCloseAfterEquip	`Boolean`	If enabled, the inventory UI will close automatically after equipping an item.
autoCloseAfterUse	`Boolean`	If enabled, the inventory UI will close automatically after Using an item.
⬇		
useToggle	`Boolean`	If true, pressing the key toggles between open/close inventory.
lockCursorAndDisableFPS	`Boolean`	Lock/unlock cursor and disable player controls

Field	Type	Tooltip
		when inventory is open.
isOpen	`Boolean`	Is the inventory currently open?

↑ Back to top

Copy fields

📦 HealthDamageTrigger MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/❤️Add Health Damage Trigger`

HealthDamageTrigger 💥

The HealthDamageTrigger is a simple trigger-based damage system.

When the player enters, stays inside, or exits a trigger collider, it applies damage to their FPSHealth.

Useful for traps, lava pits, toxic zones, or electric fences.

Settings ⚙️

Trigger Event 🎛️ – When damage is applied:

OnTriggerEnter – Once, when entering.

OnTriggerStay – Repeatedly, while inside (uses cooldown).

OnTriggerExit – Once, when leaving.

Damage Amount 💥 – How much health to subtract per event.

Player Tag 🏷️ – Tag used to identify the player (default: "Player").

Stay Settings 🕒

Stay Cooldown ⏱️ – Minimum delay between damage ticks when using OnTriggerStay.

Debug 🐞

Show Debug Logs 📜 – Prints messages when damage is applied.

Runtime Behaviour 🧭

Detects collisions via OnTriggerEnter / Stay / Exit.

Checks if the other object has the player tag.

↑

Finds an FPSHealth component on the collider, parent, or child.

Calls TakeDamage(damageAmount) on it.

Integration 🔗

Requires a Trigger Collider (isTrigger = true) on the same GameObject.

Player must have FPSHealth.

Place this script on hazardous objects (spikes, traps, fire volumes).

Best Practices 🧠

Use OnTriggerStay with a cooldown for areas like poison gas or lava.

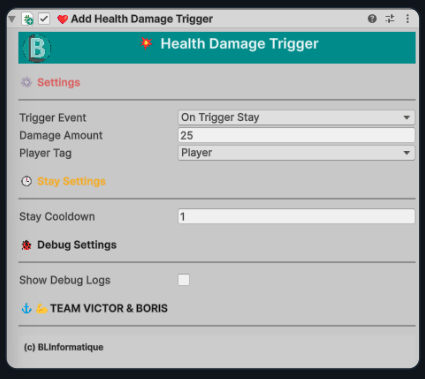
Use OnTriggerEnter for one-time hazards (spike traps).

Use OnTriggerExit for effects like electric gates or leaving a dangerous zone.

Adjust damageAmount to match survival pacing.

⚓💪 TEAM VICTOR & BORIS Note

A small script with big impact: HealthDamageTrigger makes your levels deadly. From toxic puddles to flaming barrels, it connects environment hazards directly to the player's FPSHealth.



Field	Type	Tooltip
↓		
triggerEvent	`TriggerEvent`	When to apply the damage (enter, stay, or exit).
damageAmount	`Single`	Amount of health to remove per event.
playerTag	`String`	Tag to detect as player.
↓		
stayCooldown	`Single`	Cooldown (seconds) between damages for OnTriggerStay.
↓		
showDebugLogs	`Boolean`	Show debug logs.

↑ Back to top

Copy fields

📦 InteractionRaycaster MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Interactions/🖱️ Interaction Raycaster`

InteractionRaycaster 🖱️

The InteractionRaycaster is the core system for player interactions in RealFPS.

It casts a ray from the camera center, detects interactable objects, shows a UI prompt, and invokes their interaction logic.

Raycast Source 🎥

Main Camera 📷 – Camera used for center-screen raycast (auto-fills with Camera.main if empty).

Ray Distance 📏 – Maximum reach for interactions.

Interact Mask 🎭 – LayerMask defining interactable layers.

Auto Exclude Weapon Layer 🚫🔫 – Automatically removes the "Weapon" layer from the mask.

Hit → Component Search 🔎

Search In Parents 🔼 – Also looks for IInteractable on parent objects.

Search In Children 🔽 – Also looks in children (rare).

Require Interactable For Prompt ✅ – If true, prompt is only shown if an interactable is found.

Input Settings ⌨️ / 🎮

Old Input System ⌨️

Interact Key (default E).

New Input System 🆕

Use New Input System toggle.

↑

Interact Action 🎛️ – InputAction for interaction.

UI Prompt 🖥️

Prompt Template 📝 – Text template (e.g. "Press {key} to interact"). {key} is replaced by the bound key label.

Auto Hide Prompt 🙈 – Hides prompt when no valid interactable is detected.

Prompts are displayed via PickupUIManager.Instance.ShowPrompt().

Behaviour 🔧

Show Debug Gizmos 🐞 – Visualize ray in Scene view.

Cone Angle 🎯 – Optional interaction cone (instead of strict forward ray).

References 🔗

Player Inventory 👜 – GeneralInventoryManager passed to interactions. If null, searches on the player object.

Interactable Flow 🧭

Raycast: shoots from camera center every frame.

Detection: checks for IInteractable on collider, parent, or child.

If found → cached as _currentInteractable.

If not → legacy fallback: looks for DoorLockInteractable.

UI Prompt: builds prompt from promptTemplate and current key binding.

Input: on key press (interactKey or interactAction), calls:

IInteractable.Interact(playerInventory) (preferred).

Or DoorLockInteractable.Interact() (legacy).

Public Interfaces 🧩

IInteractable

void Interact(GeneralInventoryManager playerInventory)

Implement this on your interactable scripts to plug into the system.

Integration 🔗

GeneralInventoryManager – Passed to interactions for inventory-based logic (keys, items).

DoorLockInteractable – Legacy compatibility (until migrated to IInteractable).

PickupUIManager – Displays UI prompts.

FPSControllerPro – Interaction ray is disabled when GeneralInventoryUI.isOpen.

Best Practices 🧠

Implement IInteractable on all new interactables (doors, drawers, NPCs, pickups).

Keep promptTemplate short and localized.

Use coneAngle > 0 for forgiving detection (e.g., larger objects).

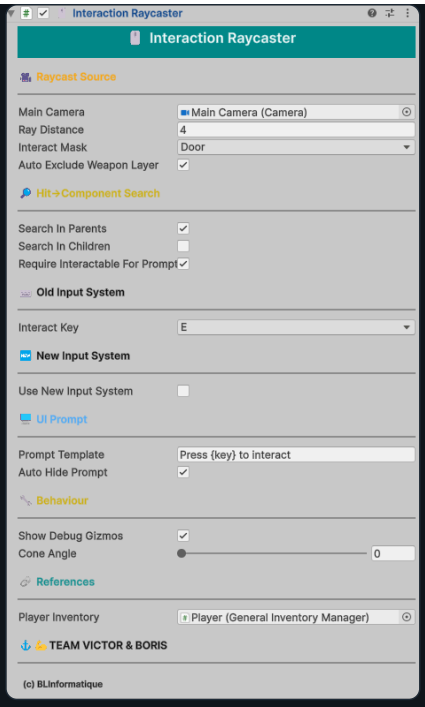
Exclude "Weapon" layer to avoid arms blocking raycast.

Ensure PickupUIManager exists in the scene for prompts.

⚓💪 TEAM VICTOR & BORIS Note

InteractionRaycaster is the bridge between player and world. A single ray connects UI, inventory, and interactables — making interaction smooth and unified across RealFPS.

↑


Interaction Raycaster
Interaction Raycaster
Raycast Source
Main Camera
Main Camera (Camera)
Ray Distance
4
Interact Mask
Door
Auto Exclude Weapon Layer
Hit→Component Search
Search In Parents
Search In Children
Require Interactable For Prompt
Old Input System
Interact Key
E
New Input System
Use New Input System
UI Prompt
Prompt Template
Press {key} to interact
Auto Hide Prompt
Behaviour
Show Debug Gizmos
Cone Angle
0
References
Player Inventory
Player (General Inventory Manager)
TEAM VICTOR & BORIS
(c) BLInformatique

Field	Type	Tooltip
↓		
mainCamera	`Camera`	Camera used for the center-screen ray.
rayDistance	`Single`	Maximum ray distance.
interactMask	`LayerMask`	Layers considered as interactable.
autoExcludeWeaponLayer	`Boolean`	Auto-exclude 'Weapon' layer from the mask at Start (if it exists).
↓		
searchInParents	`Boolean`	Search interactable on collider's parents.
searchInChildren	`Boolean`	Also search interactable on collider's children (rarely needed).
requireInteractableForPrompt	`Boolean`	Only show prompt if a real interactable is found.
↓		
interactKey	`KeyCode`	Key used to interact when using the old Input System.
↓		
useNewInputSystem	`Boolean`	Enable the new Input System path.
interactAction	`InputAction`	Input Action to trigger interaction.
↓		
promptTemplate	`String`	Prompt message template. {key} will be replaced by the bound key label.
autoHidePrompt	`Boolean`	Hide the prompt when no valid interactable is in front of the player.
↓		
showDebugGizmos	`Boolean`	Show a crosshair dot/raycast gizmo for debug.
coneAngle	`Single`	Angle in degrees for interaction cone check (0 = straight ray only). • Range [0..45]
↓		

Field	Type	Tooltip
playerInventory	`GeneralInventoryManager`	Player inventory used when invoking Interact(). If null, will search on this GameObject.

↑ Back to top

Copy fields

📘 ItemData ScriptableObject

ItemData 👛

The ItemData is a ScriptableObject that defines all the data and behaviour metadata for inventory items.

Every consumable, weapon, ammo, key, or usable tool in RealFPS is defined through ItemData.

Item Types 📑

public enum ItemType {

Food, Usable, Medicine, Ammo, Weapon, Wood, Key, Misc

}

Food 🍗 – Edible or drinkable items.

Usable 🔋 – Tools, torches, batteries, gadgets.

Medicine 💉 – Healing items.

Ammo 💥 – Ammo for specific weapons.

Weapon 🔫 – Firearms or melee.

Wood 🪵 – Crafting or fire fuel.

Key 🔑 – Unlocks doors or locks.

Misc 📦 – Any other items.

General ⚙️

Item Name 🏷️ – Display name in UI.

Item Icon 🖼️ – Sprite used in inventory.

Item Prefab 🧩 – World object prefab (optional for pickup/drop).

↑

Item Type 📑 – Category (Food, Weapon, etc.).

Is Stackable 📦 – Can stack in inventory.

Max Stack 🔢 – Maximum per slot.

Description 📝 – Tooltip text for UI.

Is Consumable 🗑️ – If true, item is consumed after use.

Medicine 💉

Heal Medicine Amount ❤️ – Health restored when used.

Food 🍗

Heal Amount 💗 – Extra health restored.

Energy Amount ⚡ – Restores food/energy stats (FPSFood).

Action Arms 🙌

ActionArmsData 🎭 – Optional arms animation for Food, Usable, Medicine (e.g., drink, eat, heal).

When set, GeneralInventoryManager.UseItem() will equip temporary arms and play the action.

Usable 🔋

Item Energy ⚡ – Duration or energy restored/added (batteries recharge devices).

Use Prefab 🧩 – Optional prefab used when activated.

Use Audio 🔊 – Sound played when used.

Weapon / Ammo 🔫

Weapon Data 🪖 – Reference to ArmsAndWeaponData if this item is a weapon.

Ammo For Weapon 💥 – Reference to weapon this ammo belongs to.

Ammo Amount 🔢 – Number of rounds provided.

Wood 🪵

Is Fuel 🔥 – Marks wood as usable for crafting or fire.

Key 🔑

Key ID 🔑 – Unique identifier for locks (DoorLockInteractable).

Consume On Use 🔑 – If true, key is consumed when unlocking.

Integration 🔗

GeneralInventoryManager – Uses itemType to decide what happens when using.

GeneralInventoryUI – Displays icons, tooltips, buttons.

FPSWeaponManager / WeaponInventoryManager – Handles Weapon and Ammo.

FPSFood – Restores hunger/energy when consuming Food.

FPSHealth – Restores health when consuming Medicine.

EnergyRuntime / EnergyConsumer – Recharge devices via batteries.

ActionArmsData – Animations when using items.

DoorLockInteractable – Matches keyId for unlocking.

Best Practices 🧠

Use stackable for ammo, food, and crafting resources.

Provide ActionArmsData for immersive use animations (drinking, healing).

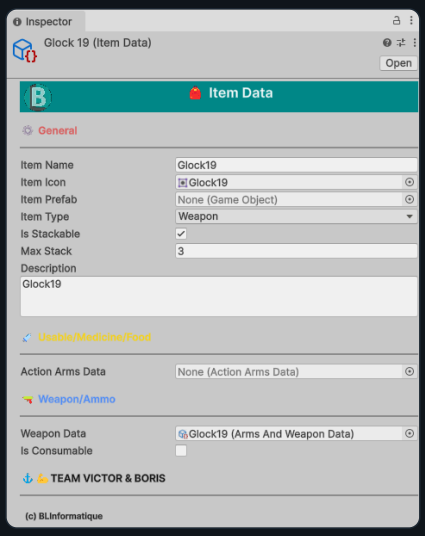
Keep keys non-consumable unless you want them single-use.

Match ammoForWeapon correctly to weapon data; otherwise ammo won't apply.

Use description for player-facing tooltips in GeneralInventoryUI.

⚓ 💪 TEAM VICTOR & BORIS Note

ItemData is the DNA of items in RealFPS. It doesn't contain logic but defines what an item is — from food to firearms. Every interaction (inventory, UI, weapons, survival systems) depends on ItemData as the central definition.



Field	Type	Tooltip
↓		
itemName	`String`	Item display name for UI.
itemIcon	`Sprite`	Main icon for inventory UI.
itemPrefab	`GameObject`	Prefab for the item (optional, for world pickup/drop).
itemType	`ItemType`	Type of this item (food, usable, ammo, weapon, etc.).
isStackable	`Boolean`	Can be stacked in inventory.
maxStack	`Int32`	Maximum stack count in inventory.
description	`String`	Tooltip description (short, for UI). • TextArea
↓		
healMedicineAmount	`Single`	Amount of health restored if item type is Medicine.
↓		
healAmount	`Single`	Amount of health restored if item type is Food/Drink.
energyAmount	`Single`	Amount of energy restored if item type is Food/Drink.
↓		
actionArmsData	`ActionArmsData`	Arms/animation setup for this item (ActionArmsData). Used for animations when using or consuming the item.
↓		
itemEnergy	`Single`	Duration or energy (for torch, battery, etc.). For batteries, this is the energy added.
usePrefab	`GameObject`	Prefab or ScriptableObject to use when activated (optional).
useAudio	`AudioClip`	Sound to play on use (optional).
↓		
weaponData	`ArmsAndWeaponData`	Reference to weapon data if this is a weapon.
ammoForWeapon	`ArmsAndWeaponData`	Reference to weapon data if this is an ammo item.
ammoAmount	`Int32`	Amount of ammo provided by this item.
↓		
isFuel	`Boolean`	Used for crafting or fire (optional, for wood, sticks, etc.).

Field	Type	Tooltip
keyId	`String`	Unique key ID for unlocking doors or objects.
consumeOnUse	`Boolean`	If true, the key is consumed when used.
isConsumable	`Boolean`	Can only be used once. If true, item will be consumed after use.

↑ Back to top

Copy fields

📦 ItemPickup MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Pickup/📦Add Item Pickup` • Requires: BoxCollider

ItemPickup 📦

The ItemPickup is the universal pickup script in RealFPS.

It lets the player collect an ItemData into their inventory, either automatically on trigger or by pressing a key with a UI prompt.

Pickup Settings ⚙️

Player Tag 🏷️ – Tag used to detect the player (default: "Player").

Item Data 👛 – Reference to the ItemData ScriptableObject.

Amount 🔢 – Quantity to give (for stackables or consumables).

Pickup Modes 🔄

Pickup By Trigger 🚪 – Auto-collects when entering the trigger collider.

Pickup By Key ⌨️ – Requires a key press (shows a UI prompt).

Input Settings

Old Input System ⌨️ – pickupKey (default E).

New Input System 🆕 – Toggle useNewInputSystem + assign an InputAction.

Feedback 🎧

Pickup Sounds 🔊 – Random clip(s) played when collected.

Pickup FX Prefab ✨ – Optional FX prefab instantiated on pickup.

FX Duration ⏱️ – Auto-destroy delay for FX.

↑

Prompt Message 💬 – Template for UI prompt (e.g., "Press {key} to pick up").

Runtime Behaviour 🧭

OnTriggerEnter

If pickupByTrigger = true, immediately gives item.

Else, marks player as nearby for key-based pickup.

OnTriggerStay / Update

If pickupByKey = true and player is nearby:

Shows prompt via PickupUIManager.Instance.ShowPrompt().

On key press → calls PickupItem().

OnTriggerExit

Clears player reference and hides prompt.

PickupItem()

Calls GeneralInventoryManager.AddItem(itemData, amount).

If success → plays sounds, FX, and destroys the pickup object.

If no GeneralInventoryManager exists on player, adds one automatically.

Integration 🔗

GeneralInventoryManager – Core system that stores the collected item.

PickupUIManager – Displays “Press E” prompt when pickup by key.

ItemData – Defines the item type and behaviour when later used.

Best Practices 🧠

Use pickupByTrigger for common items (ammo, food).

Use pickupByKey for important or story-related items.

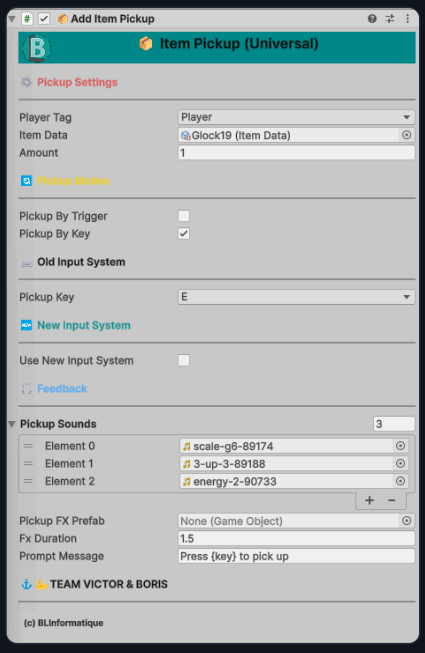
Always assign ItemData to define what is collected.

Customize promptMessage to localize text and improve immersion.

Keep collider size small so pickup triggers only when close.

⚓💪 TEAM VICTOR & BORIS Note

ItemPickup makes every object collectible — from ammo boxes to medkits and keys. Combined with ItemData and GeneralInventoryManager, it's the entry point of RealFPS's survival loop.



Field	Type	Tooltip
↓		
playerTag	`String`	Tag to detect as player.
itemData	`ItemData`	The item to add to the player's inventory.
amount	`Int32`	Amount to give (for stackable or consumable items).
↓		
pickupByTrigger	`Boolean`	Enable automatic pickup when player enters the trigger.
pickupByKey	`Boolean`	Enable pickup by pressing a key.
↓		
pickupKey	`KeyCode`	Key used to pick up the item (Old Input System).
↓		
useNewInputSystem	`Boolean`	Enable this if using the new Input System.
pickupAction	`InputAction`	Input action for picking up the item (New Input System).
↓		
pickupSounds	`AudioClip[]`	Pickup sound(s) (random, optional).
pickupFXPrefab	`GameObject`	FX to play on pickup (optional).
fxDuration	`Single`	FX duration before destroy (seconds).
promptMessage	`String`	Default text shown to prompt the player.

↑ Back to top

Copy fields

📦 PickupUIManager MonoBehaviour

PickupUIManager 🖥️📦

The PickupUIManager is the central system that shows and hides pickup/interact prompts on screen.

It is a simple singleton manager that ensures only one prompt is visible at a time, regardless of how many pickups are nearby.

UI Reference 🎨

Prompt Text 📝 – The Text UI element used to display prompts (e.g. “Press E to pick up”).

Runtime State ⚙️

Instance 📡 – Singleton reference to the active manager.

currentPickup 📦 – The pickup currently controlling the prompt.

Public API 🧩

ShowPrompt(ItemPickup pickup, string message)

Displays the message in promptText.

Sets the current pickup reference.

If another pickup tries to show, it overrides only if it’s new.

HidePrompt(ItemPickup pickup)

Hides the prompt only if it was shown by this pickup.

Clears the current reference.

Runtime Behaviour 🧭

↑

At Awake:

Sets itself as Instance.

Hides the prompt UI by default.

During gameplay:

ItemPickup calls ShowPrompt() when player is nearby (key-based pickup).

InteractionRaycaster calls ShowPrompt() when pointing at an interactable.

When action is completed or player moves away → HidePrompt() is called.

Integration 🔗

ItemPickup – Uses this manager to show “Press E to pick up”.

InteractionRaycaster – Uses this manager to show “Press {key} to interact”.

GeneralInventoryManager – Indirectly tied, since pickup actions add items to the inventory.

Best Practices 🧠

Place exactly one PickupUIManager in the scene (singleton).

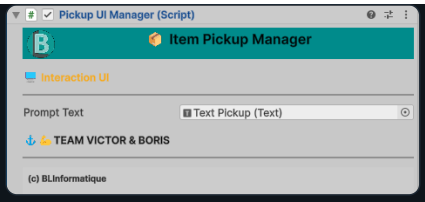
Assign a UI Text in your HUD canvas for promptText.

Customize the font, size, or style of the text for immersion.

Keep prompts short and localized (e.g., “Press F to Open”).

⚓💪 TEAM VICTOR & BORIS Note

PickupUIManager keeps prompts tidy. Whether you’re grabbing ammo, opening doors, or using tools, it ensures the player only ever sees one clear instruction at a time.



Field	Type	Tooltip
promptText	`Text`	Reference to the UI Text for pickup prompts.

↑ Back to top

Copy fields

📘 SurfaceData ScriptableObject

SurfaceData 🪨

Purpose. SurfaceData is a ScriptableObject describing a physical surface for footsteps and bullet impacts: display name, preview texture, audio sets, impact/footstep FX, and FX lifetime. Designers assign it to terrains or colliders (via SurfaceType) so gameplay can choose the correct sounds/FX at runtime.

Fields ⚙️

surfaceName 🏷️ – Human-friendly name used in UI/debug.

surfaceTexture 🖼️ – Preview/representative texture (for materials or UI previews).

surfaceFootstepAudio[] 👣🔊 – Footstep clips played when the player walks/runs/crouches on this surface.

surfaceHitAudio[] 💥🔊 – Impact clips when projectiles hit this surface.

surfaceFootstepFx[] 👣✨ – Optional FX spawned on footsteps (dust, splash…).

surfaceHitFx[] 💥✨ – Particle FX spawned on bullet/projectile impact (sparks, dust…).

prefabHitEffect[] 🧩 – Additional prefabs on impact (decals, custom marks).

hitFxLifetime ⏰ – Lifetime (seconds) before impact FX auto-destroy (0 = never).

How It’s Used 🔗

Footsteps → FPSFootstepPlayer reads terrain texture or collider’s SurfaceType to select surfaceFootstepAudio and optionally spawn surfaceFootstepFx.

Bullet Impacts → FPSWeaponManager.SpawnSurfaceImpactEffect() looks up the hit collider’s SurfaceType → SurfaceData to:

play a random surfaceHitAudio,

↑

spawn surfaceHitFx / prefabHitEffect,

schedule destruction using hitFxLifetime.

Level Authoring → Add SurfaceType to meshes/colliders and assign the right SurfaceData. For terrains, map each terrain texture index to a SurfaceData in your footstep system.

Setup Checklist ✅

Create assets via Create → BLInformatique → Real FPS → Surface Data.

Fill footstep and impact clips/FX, and set hitFxLifetime to avoid FX leaks.

On meshes/colliders, add SurfaceType and assign this SurfaceData.

On terrains, ensure your footstep player's terrain-to-SurfaceData array matches the terrain texture order.

Test with FPSFootstepPlayer (walk/run/crouch) and with FPSWeaponManager (shoot surfaces) to validate audio/FX.

Best Practices 🧠

Keep audio sets short and varied to avoid repetition; randomize pitch slightly at playback for richness.

Use lightweight FX and reasonable lifetimes for performance, especially in firefights.

Create variants (e.g., "Concrete_Dry", "Concrete_Wet") when you need different footsteps/impacts in similar environments.

Centralize common decal prefabs in prefabHitEffect so all impacts share consistent visuals.

Compatibility ✅

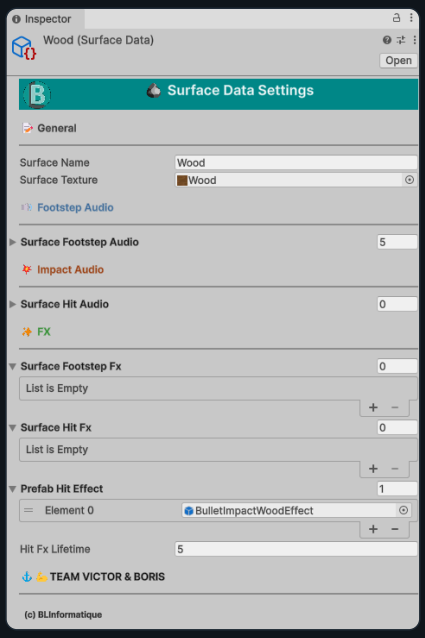
Designed for Unity 6000.2.3f1.

Integrates with BLInformatique EditorTools styling (StyledBox/Separator) and tooltips (English).

↑

⚓💪 TEAM VICTOR & BORIS Note

SurfaceData is your sound & FX palette. Tag your world once, and every step and bullet will speak the right material language — wood creaks, metal pings, stone throws sparks.



Field	Type	Tooltip
surfaceName	`String`	Friendly display name for this surface (for UI, debug, etc).
surfaceTexture	`Texture2D`	Main texture that represents this surface (for material/preview/UI).
surfaceFootstepAudio	`AudioClip[]`	Footstep sounds played when player walks/runs/crouches on this surface.
surfaceHitAudio	`AudioClip[]`	Hit sounds when a bullet/projectile impacts this surface.
surfaceFootstepFx	`GameObject[]`	Optional particle/FX objects to spawn on footsteps (dust, splash, etc).
surfaceHitFx	`GameObject[]`	Particle systems to spawn on bullet/projectile impact (ex: sparks, dust, etc).
prefabHitEffect	`GameObject[]`	Prefab(s) to spawn on hit (ex: decal, mark, or any custom effect).
hitFxLifetime	`Single`	Lifetime (seconds) of the impact FX (0 = never auto-destroyed).

↑ Back to top

Copy fields

📦 SurfaceType MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/📦 Add Surface Type` • Requires: AudioSource

SurfaceType 🪨

The SurfaceType component tags a collider or object in the scene with a specific SurfaceData.

This makes footsteps, bullet impacts, and FX context-aware (wood creaks, metal clangs, stone sparks).

Fields ⚙️

Surface Data 📑 – The SurfaceData asset assigned to this object.

Defines footstep sounds, impact sounds, FX, and preview texture.

Requirements 🛠️

AudioSource 🎧 – Required component, used to play surface-related sounds.

Runtime Behaviour 🧭

When a player walks/runs/crouches:

FPSFootstepPlayer checks if the hit collider has SurfaceType.

Uses its surfaceData.surfaceFootstepAudio for footsteps, and optionally surfaceFootstepFx.

When a bullet or projectile hits:

FPSWeaponManager.SpawnSurfaceImpactEffect() looks for SurfaceType on the hit object.

Plays random surfaceHitAudio, spawns surfaceHitFx and prefabHitEffect.

Integration 🔗

SurfaceData – ScriptableObject with all audio/FX settings.

↑

FPSFootstepPlayer – Reads SurfaceType for footstep playback.

FPSWeaponManager – Reads SurfaceType for bullet impacts and FX.

ExplosionEntity – May use SurfaceType for impact debris and sound selection.

Best Practices 🧠

Assign SurfaceType to every major collider (floors, walls, props) to get consistent audio/FX.

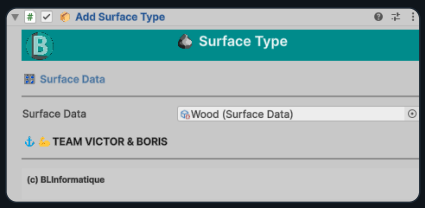
For large terrains, use FPSFootstepPlayer's terrain mapping instead of per-object SurfaceType.

Keep one SurfaceType per object; use multiple colliders if you want varied surfaces on a single mesh.

Create reusable SurfaceData assets (Wood, Metal, Concrete, Grass, Water) and assign them widely.

⚓💪 TEAM VICTOR & BORIS Note

SurfaceType is the bridge between the world and SurfaceData. With just one component, every step and bullet in RealFPS reacts authentically to the material underfoot or in front of the muzzle.



Field	Type	Tooltip
surfaceData	`SurfaceData`	Assign the SurfaceData (ScriptableObject) for this object. Defines all sounds, FX, and texture used by GameTools (steps, impacts, visuals, etc).

↑ Back to top

Copy fields

📦 TargetReactivePro MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/🎯Add Target Reactive Pro` • Requires: Rigidbody

TargetReactivePro 🎯

The TargetReactivePro is a universal reactive target script.

When hit by bullets or projectiles, it plays animations, applies physics impulses, rotates like a hinged target, and awards score.

Animation 🎬

Animation Trigger ▶️ – Animator trigger to fire when the target is hit (if Animator is present).

Physics & Reaction ⚙️

Hit Force 💥 – Impulse applied at hit point using Rigidbody.AddForceAtPosition.

Rotate On Hit 🔁 – If true, also rotates transform when hit.

Rotation On Hit ↩️ – Vector3 rotation applied when rotateOnHit is enabled (default: 90° on X).

Scoring 🏆

Score Value 🪙 – Points to award on hit (requires custom scoring system, e.g. GameManager.Instance.AddScore(scoreValue)).

Requirements 🛠️

Rigidbody is required (auto-configured in Awake with Continuous collision and Interpolate).

Collider is needed for hits (box, sphere, mesh…).

Animator is optional, used if you want animated targets.

Public API 🧩

↑

ReactToHit(Vector3 hitPoint, Vector3 hitNormal)

Called externally (e.g. by FPSWeaponManager after a raycast hit).

Plays animation, applies force, applies rotation, and can award score.

Runtime Behaviour 🧭

When ReactToHit() is called:

Triggers animation if Animator is assigned.

Adds impulse force at the hit point (if Rigidbody & hitForce > 0).

Optionally rotates object.

Score value can be sent to GameManager.

In Editor only: pressing H simulates a hit for testing.

Integration 🔗

FPSWeaponManager – Should call ReactToHit(hit.point, hit.normal) after a successful raycast hit.

SurfaceType / SurfaceData – Handles sound/FX for the hit; TargetReactivePro only handles physical/animation response.

GameManager / Scoring – Hook into scoreValue for arcade-style shooting ranges.

Best Practices 🧠

Use hitForce for physical knockbacks, and rotateOnHit for shooting gallery-style flipping targets.

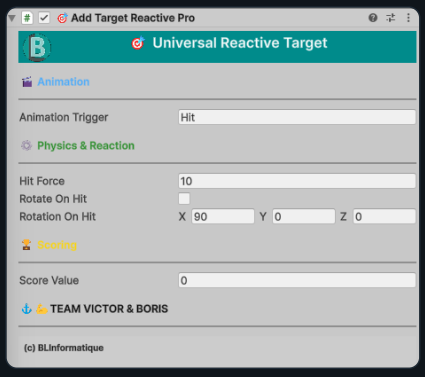
Keep animationTrigger consistent across targets for reusable Animator controllers.

Tie scoreValue into your scoring system for training/shooting range levels.

Use with SurfaceData so bullets hitting targets still spawn correct FX and sounds.

⚓ 💪 TEAM VICTOR & BORIS Note

TargetReactivePro makes your levels react. Whether you're building a training range or arcade-style scoring system, it combines physics, animation, and scoring into one modular target script.



Field	Type	Tooltip
⬇		
animationTrigger	`String`	Animator trigger to play on hit (if Animator is present).
⬇		
hitForce	`Single`	Impulse force applied when hit (0 = disable).
rotateOnHit	`Boolean`	If enabled, also rotate the object when hit (like a target on hinge).
rotationOnHit	`Vector3`	Rotation added on hit (only if rotateOnHit is true).
⬇		
scoreValue	`Int32`	Points awarded when hit (requires custom scoring system).

↑ Back to top

Copy fields

📦 TurretAI MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/🛡️ Add Turret AI`

TurretAI 🛡️

The TurretAI controls an automated turret with dual pivots (yaw + pitch).

It tracks the player, clamps rotation, and fires either hitscan rays or projectiles with optional FX and sounds.

Pivots & Points ⚙️

Yaw Pivot ↔️ – Horizontal rotation axis.

Pitch Pivot ↕️ – Vertical rotation axis (local X must be pitch axis).

Fire Point 🔫 – Where rays/projectiles/FX are spawned.

Target & Offset 🎯

Target Tag 🏷️ – Tag of target (default: "Player").

Detection Range 📏 – Maximum distance to track/fire.

Pitch Target Offset Y ⬆️ – Vertical offset applied to aim (e.g., +1 = chest/head).

Offset Up Space 🌍 – Defines which “up” is used for offset: WorldUp, TargetUp, PitchPivotUp.

Rotation Speeds 🌀

Yaw Speed 🔁 – Responsiveness (deg/sec).

Pitch Speed 🔁 – Responsiveness (deg/sec).

Pitch Clamp 🔒

Clamp Pitch ✅ – Restricts pitch rotation range.

↑

Min Pitch / Max Pitch 📐 – Limits (negative looks down, positive looks up).

Fire Settings 🔫

Damage Per Shot 💥 – Damage applied on each hit.

Fire Rate ⏱️ – Shots per second (0 disables autofire).

Projectile Prefab 🚀 – Optional projectile to spawn. If null → hitscan ray.

Impact FX Prefab ✨ – Effect spawned at hit point.

Muzzle FX Prefab 🔥 – Effect spawned at FirePoint when firing.

Audio 🔊

Use Audio Source ✅ – If true, uses a persistent AudioSource. Else, plays one-shot at FirePoint.

Fire Audio Clip 🎵 – Sound played when firing.

Audio Source 🎚️ – Reference (auto-created if null).

Fire Volume 🔉 – Volume (0–1).

Debug 🐞

Current Target 🎯 – Auto-filled reference to target transform.

OnDrawGizmos 🖊️ – Draws fire direction and detection radius in Scene view.

Runtime Behaviour 🧭

Target Acquisition: finds object by tag.

Range Gate: ignores if target too far.

Yaw: rotates yawPivot towards target horizontally.

↑

Pitch: calculates aim direction, clamps to limits, rotates pitchPivot around local X.

Fire:

Spawns muzzle FX.

Plays fire audio (persistent or one-shot).

If projectile: instantiates prefab and assigns damage/FX.

Else hitscan: raycast forward, applies damage to FPSHealth, spawns impact FX.

Integration 🔗

FPSHealth – Player component, damaged when ray/projectile hits.

TurretProjectile – Projectile behaviour script (damage + FX).

SurfaceType/SurfaceData – For impact FX/audio (when bullets hit surfaces).

GameManager – Can connect to scoring or alerts on turret kills.

Best Practices 🧠

Use clamped pitch to stop turrets rotating unrealistically.

Keep detectionRange moderate to avoid performance cost on large maps.

For sci-fi guns: use projectile prefabs. For military turrets: use hitscan.

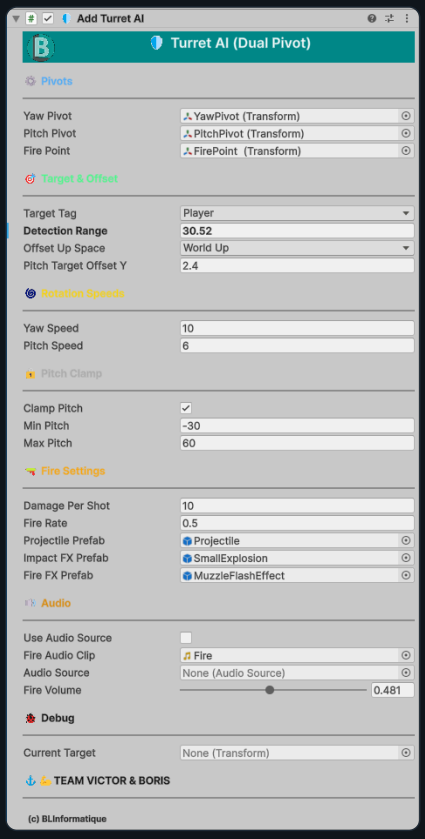
Attach SurfaceType to walls/targets for realistic impact FX.

Test with OnDrawGizmos enabled to visualize detection zones.

⚓ 💪 TEAM VICTOR & BORIS Note

↑

TurretAI turns any level into a deadly challenge. With pivots, offsets, FX, and sound, it’s a plug-and-play enemy that punishes players who stay in sight too long.



Field	Type	Tooltip
↓		
yawPivot	`Transform`	Yaw pivot (horizontal rotation).
pitchPivot	`Transform`	Pitch pivot (vertical elevation). Its local X (right) must be the rotation axis.
firePoint	`Transform`	FirePoint at the end of the barrel (for ray/FX/projectiles).
↓		
targetTag	`String`	Tag of the player/target to track.
detectionRange	`Single`	Max distance at which the turret will track and fire.
offsetUpSpace	`OffsetUpSpace`	Which 'up' vector should be used to apply the vertical offset.
pitchTargetOffsetY	`Single`	Vertical offset applied to aiming (e.g., +1 = aim at upper body).
↓		
yawSpeed	`Single`	Yaw responsiveness (deg/sec).
pitchSpeed	`Single`	Pitch responsiveness (deg/sec).
↓		
clampPitch	`Boolean`	Clamp the local X angle of the pitch pivot.
minPitch	`Single`	Minimum pitch angle (deg). Negative looks down.
maxPitch	`Single`	Maximum pitch angle (deg). Positive looks up.
↓		
damagePerShot	`Single`	Damage dealt per shot.
fireRate	`Single`	Fire rate (shots per second). Set to 0 to disable auto-fire.
projectilePrefab	`GameObject`	Optional projectile prefab (if null, uses hitscan raycast).
impactFXPrefab	`GameObject`	Impact FX prefab to spawn at hit point (optional).
fireFXPrefab	`GameObject`	Muzzle FX prefab spawned at FirePoint (optional).
↓		
useAudioSource	`Boolean`	If true, turret uses a persistent AudioSource. If false, it just plays one-shot at firePoint.
fireAudioClip	`AudioClip`	Audio clip played when firing.
audioSource	`AudioSource`	AudioSource used to play fire sound (created if null).
fireVolume	`Single`	Volume for fire sound (0..1). • Range [0..1]
↓		

Field	Type	Tooltip
currentTarget	`Transform`	Current tracked target (auto-found by tag).

↑ Back to top

Copy fields

📦 TurretProjectile MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Behaviour/🚀Add Turret Projectile`

TurretProjectile 🚀

The TurretProjectile represents a fired projectile from a turret (or any launcher).

It moves forward at a set speed, applies damage on impact, spawns FX, and then destroys itself.

Projectile Settings ⚙️

Speed 🏎️ – Travel speed in meters/second.

Damage 💥 – Damage dealt on impact (applies to FPSHealth).

Lifetime 🕰️ – Seconds before the projectile auto-destroys (safety to avoid clutter).

Impact FX 💥

Impact FX Prefab ✨ – Optional prefab spawned at collision/trigger point (sparks, explosion, etc.).

Runtime Behaviour 🧭

Start()

If Rigidbody exists → sets linearVelocity = forward * speed.

Schedules self-destroy after lifeTime.

Collision / Trigger

On impact with a collider:

Checks for FPSHealth on the hit object's parent.

Calls TakeDamage(damage).

↑

Spawns impactFX (destroyed after 1.5s).

Destroys itself.

Integration 🔗

TurretAI – Spawns this projectile at firePoint when using projectile mode.

FPSHealth – Damaged when hit.

Impact FX – Any prefab (particle, explosion, decal).

Can be reused for player-fired projectiles (grenades, rockets).

Best Practices 🧠

Always add a Rigidbody + Collider (set to trigger or not) for physics interaction.

Keep lifeTime short to avoid leaks if projectiles miss everything.

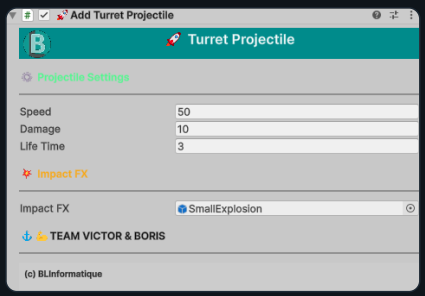
For big explosions, pair with ExplosionEntity triggered on impact.

Scale speed to match turret's role (slow plasma vs fast bullets).

Use DoImpact() for consistent FX spawn and cleanup.

⚓💪 TEAM VICTOR & BORIS Note

TurretProjectile is a simple but modular projectile. Plug it into turrets, rocket launchers, or sci-fi cannons — with custom FX, it becomes the visual punch of your RealFPS arsenal.



Field	Type	Tooltip
↓		
speed	`Single`	Projectile speed in m/s.
damage	`Single`	Damage inflicted on impact.
lifeTime	`Single`	Lifetime before auto-destroy (seconds).
↓		
impactFX	`GameObject`	Prefab for impact FX (spark, explosion, etc.).

↑ Back to top

Copy fields

📦 WeaponCameraFollowerPro MonoBehaviour

WeaponCameraFollowerPro 📹

The WeaponCameraFollowerPro is a utility script for dual-camera setups in RealFPS.

It ensures that the weapon camera (which renders only arms/weapons) always follows the main FPS camera position and rotation.

Settings ⚙️

Main Camera 📷 – Reference to the main FPS camera.

This is usually the player's view camera.

If null, no syncing occurs.

Runtime Behaviour 🧭

Executed in LateUpdate() to guarantee sync after the main camera has moved.

Copies position and rotation from mainCamera to the weapon camera holder.

Prevents visible lag or desync between the two cameras.

Integration 🔗

Used alongside WeaponCameraSetupPro to configure layers/culling for the dual-camera system.

Required when arms/weapons are rendered in a separate camera (layer = WeaponCam).

Works with FPSWeaponManager for accurate ADS (Aim Down Sight) alignment.

Best Practices 🧠

↑

Place this script on your weapon camera GameObject.

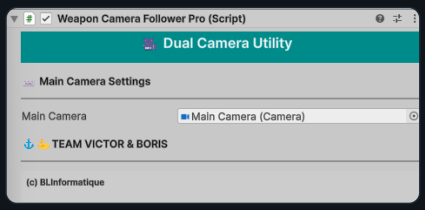
Always set mainCamera to the same camera used by FPSControllerPro.

Keep both cameras with identical FOV unless intentionally using zoom effects.

If using post-processing, apply it only to the main camera, not the weapon camera.

⚓💪 TEAM VICTOR & BORIS Note

WeaponCameraFollowerPro makes the dual-camera pipeline seamless. One camera for world rendering, one for arms — locked in sync for smooth, professional FPS visuals.



Field	Type	Tooltip
mainCamera	`Camera`	Camera to follow (Main FPS Camera).

↑ Back to top

Copy fields

📦 WeaponCameraSetupPro MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Utility/📹 Weapon Camera Setup Pro`

WeaponCameraSetupPro 📹

The WeaponCameraSetupPro is a utility script that automatically creates and configures a dual-camera setup for RealFPS.

It ensures that arms/weapons are rendered separately from the world, avoiding clipping issues and providing professional FPS visuals.

Cameras 📹

Main Camera 📷 – Reference to the player's main FPS camera. If empty, auto-detects Camera.main.

Weapon Camera 🔫 – Existing weapon camera (optional). If none, one is created.

Weapon Camera Name 🏷️ – Default name: "WeaponCamera".

Quick Actions ⚡

🚀 Create or Sync WeaponCam (Editor Only)

Button in the Inspector (via EditorTools).

Auto-detects main camera.

Creates the Weapon layer if missing.

Finds or creates the weapon camera.

Matches main camera settings.

Configures culling, depth, nearClip, farClip.

Removes extra AudioListener.

Updates culling mask so:

↑

WeaponCam renders only “Weapon” layer.

MainCam renders everything except “Weapon”.

Ensures auto-follow via WeaponCameraFollowerPro.

Runtime Behaviour 🧭

WeaponCam follows main camera via WeaponCameraFollowerPro.

Arms/weapons placed on Weapon layer are visible only to WeaponCam.

MainCam renders world, environment, enemies, etc.

WeaponCam renders last, at higher depth, avoiding clipping through walls.

Integration 🔗

WeaponCameraFollowerPro – Keeps weapon camera perfectly synced to main camera.

FPSWeaponManager – Relies on dual-camera setup for ADS and clean rendering.

Arms Prefabs – Must be set to Weapon layer.

SurfaceData / SurfaceType – Impacts remain visible on world only, not arms.

Best Practices 🧠

Place WeaponCameraSetupPro on your Main Camera.

Run Create or Sync once to set up the dual cameras.

Always assign arms/weapons to the Weapon layer.

Set WeaponCam nearClipPlane = 0.01 to avoid clipping.

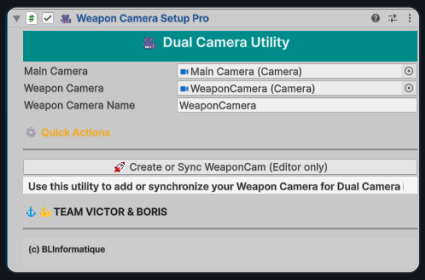
Keep background = clear to overlay seamlessly on MainCam.

Disable extra AudioListener (already handled by script).

Use identical FOV for both cameras unless doing scoped zoom.

⚓ 💪 TEAM VICTOR & BORIS Note

WeaponCameraSetupPro is your one-click dual-camera wizard. No more fighting with clipping — arms and weapons always render crisp, separate from the world, for AAA-style FPS visuals.



Field	Type	Tooltip
mainCamera	`Camera`	Reference to your main FPS Camera. Leave empty to auto-detect Camera.main.
weaponCamera	`Camera`	Optional: Assign an existing Weapon Camera (else it will auto-create).
weaponCameraName	`String`	Name for the Weapon Camera GameObject.

⬇		
btnCreate	`Int32`	
usageDummy	`String`	

↑ Back to top

Copy fields

📦 WeaponInventoryManager MonoBehaviour

WeaponInventoryManager 🗄️🔫

The WeaponInventoryManager tracks the player's weapon slots (max 5), including ammo counts and energy persistence per slot.

It integrates tightly with FPSWeaponManager for equipping and firing, and with EnergyConsumer for managing battery-powered devices.

Weapon Slots 🎮

weaponSlots[5] 🧩 – Array of ArmsAndWeaponData assets, one per slot (max 5).

currentAmmo[5] 💥 – Reserve ammo counts per slot.

currentMag[5] 🔫 – Magazine counts per slot.

slotEnergy[5] 🔋 – Energy snapshot per slot. -1 means uninitialized.

slotEnergyMax[5] 🔝 – Max energy snapshot per slot (informative).

selectedSlot 🎯 – Index (0–4) of the currently active weapon.

Input Settings ⌨️🖱️

enableKeySwitch 🔢 – Switch weapons using keys 1–5.

enableMouseWheel 🖱️ – Switch weapons with the scroll wheel.

References 🔗

weaponManager 🔫 – Reference to the FPSWeaponManager. Auto-finds if empty.

Runtime Behaviour 🧭

At Start: equips the selected slot automatically.

↑

At Update:

Listens to Alpha1–Alpha5 keys if enableKeySwitch.

Listens to MouseWheel if enableMouseWheel.

Calls EquipWeapon(slot) accordingly.

Public API 🧩

EquipWeapon(int slot)

Unequips current slot (unbinds energy events).

Sets selectedSlot.

Calls weaponManager.Equip(data) to instantiate arms prefab.

Binds to EnergyRuntime.CurrentEnergyConsumer (from FPSWeaponManager).

Restores or caches energy snapshot for the slot.

Subscribes to onEnergyChanged to keep slot energy up to date.

AddWeapon(ArmsAndWeaponData data)

Finds first empty slot, assigns data.

Initializes ammo (magazine + reserve).

Initializes energy snapshot (-1 until first equip).

Returns slot index, or -1 if full.

RemoveWeapon(int slot)

Clears weapon data, ammo, and energy snapshot from slot.

If it was bound → unsubscribes from energy events.

AddAmmo(ArmsAndWeaponData data, int amount)

Finds slot containing the weapon, increases currentAmmo up to maxAmmo.

GetWeaponSlot(ArmsAndWeaponData data)

Returns index of weapon in slots, or -1 if not found.

Energy Binding 🔋

When equipping, binds the EnergyConsumer from the equipped arms/device.

First time → captures initial prefab value.

Subsequent equips → restores previous snapshot value (keeps continuity).

OnEnergyChangedRelay() keeps snapshots updated in arrays.

Integration 🔗

FPSWeaponManager – For equipping, ADS, firing.

GeneralInventoryManager – For adding new weapons or ammo from pickups.

EnergyRuntime / EnergyConsumer – Energy persistence between slot switches.

UI Systems – Crosshair, EnergyBarUI, FPSEnergy reflect weapon states.

Best Practices 🧠

Always keep weaponSlots.Length = 5 (default).

Use AddWeapon from inventory pickups, never assign slots directly at runtime.

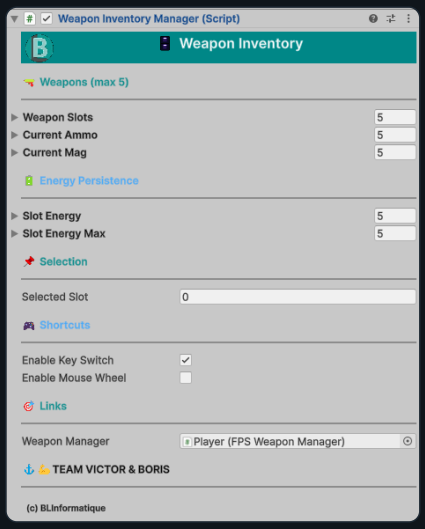
For energy-based tools (flashlight, scanner), test slot switching to confirm persistence works.

Use RemoveWeapon for discard/destroy mechanics (broken weapons, story events).

Configure both enableKeySwitch and enableMouseWheel for flexible gameplay.

⚓💪 TEAM VICTOR & BORIS Note

WeaponInventoryManager is the arsenal brain: it remembers your ammo, preserves your batteries, and swaps weapons on command. With it, RealFPS delivers the polished multi-weapon loop players expect.



Field	Type	Tooltip
↓		
weaponSlots	`ArmsAndWeaponData[]`	ArmsAndWeaponData list for all inventory slots (max 5).
currentAmmo	`Int32[]`	
currentMag	`Int32[]`	
↓		
slotEnergy	`Single[]`	Per-slot energy snapshot. -1 means 'uninitialized'.
slotEnergyMax	`Single[]`	Per-slot max energy cached (optional, informative).
↓		
selectedSlot	`Int32`	Current selected slot index (0-4).
↓		
enableKeySwitch	`Boolean`	Enable weapon switch by keyboard (keys 1-5).
enableMouseWheel	`Boolean`	Enable weapon switch by mouse wheel.
↓		
weaponManager	`FPSWeaponManager`	

↑ Back to top

Copy fields

📦 WeaponInventoryUI MonoBehaviour

WeaponInventoryUI 🗄️🎮

The WeaponInventoryUI displays the player's weapon slots (max 5) on screen.

It updates slot icons, highlights the selected weapon, and shows weapon name and ammo counts.

UI References 🎨

Inventory Panel 🟦 – Root RectTransform for positioning/anchoring.

Slot Images (5) 🎴 – Array of Image UI elements for each weapon slot.

Highlight Sprite ✨ – Sprite used to visually highlight the selected slot.

Default Slot Sprite ⬜ – Sprite shown when no weapon is in a slot.

Weapon Name Text 🏷️ – Displays the current weapon's name.

Ammo Text 🔫 – Displays magazine / reserve counts.

Anchoring 📍

Inventory Anchor – Position on screen: TopLeft, TopRight, BottomLeft, BottomRight, TopMiddle, BottomMiddle.

ApplyAnchor() – Applies the chosen anchor at runtime or in Editor (OnValidate).

Colors 🎨

Slot Default Color 🟣 – Color of non-selected slots.

Slot Highlight Color 🟢 – Color of the selected slot.

Runtime Behaviour 🧭

↑

OnEnable / Start

Calls ApplyAnchor() to position the panel.

Calls UpdateUI() to initialize slots.

Update

Continuously calls UpdateUI() (if inventoryManager != null).

UpdateUI()

Loops through 5 slots:

Displays weapon icon (or default if empty).

Applies highlight color to the selected slot.

Overrides sprite with highlightSprite if assigned.

Updates tooltip (UITooltip.text) with weapon name or “Empty Slot”.

Updates weaponNameText and ammoText based on selected slot.

Integration 🔗

WeaponInventoryManager – Provides current weapon slots, ammo counts, and selected slot.

ArmsAndWeaponData – Supplies weaponName and weaponIcon.

UITooltip – Optional component to display extra info on hover.

FPSWeaponManager – Actual equipping/unequipping logic (slots reflect its state).

Best Practices 🧠

Always assign slotImages.Length = 5 to match the manager.

Use defaultSlotSprite to clearly show empty slots.

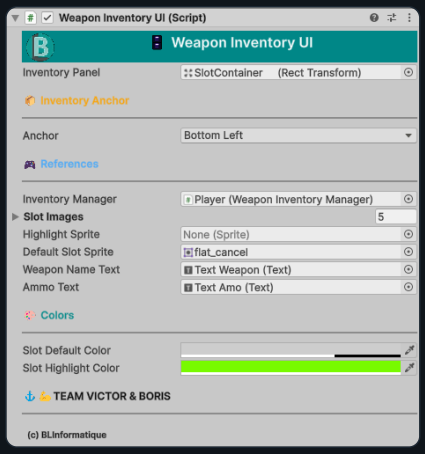
Keep UpdateUI() connected to WeaponInventoryManager events for efficiency (optional optimization).

Customize weaponNameText and ammoText to match your HUD style.

Place this UI in a dedicated Canvas with correct sorting to overlay gameplay.

⚓💪 TEAM VICTOR & BORIS Note

WeaponInventoryUI turns the arsenal into a clear, stylish HUD. Players always know what they're carrying, what's selected, and how many bullets are left — essential for survival and combat pacing.



Field	Type	Tooltip
inventoryPanel	`RectTransform`	Main inventory panel RectTransform for positioning/anchoring.
↓		
anchor	`InventoryAnchor`	Anchor position for inventory UI on screen.
↓		
inventoryManager	`WeaponInventoryManager`	WeaponInventoryManager to link with this UI.
slotImages	`Image[]`	Slot UI Images (size must be 5, one for each weapon slot).
highlightSprite	`Sprite`	Sprite to use for the selected slot highlight.
defaultSlotSprite	`Sprite`	Default background sprite for empty slots.
weaponNameText	`Text`	UI Text to display current weapon name.
ammoText	`Text`	UI Text to display current weapon ammo count.
↓		
slotDefaultColor	`Color`	Color for unselected slots.
slotHighlightColor	`Color`	Color for the selected slot.

↑ Back to top