BLInformatique — Real FPS

⚓ 💪TEAM VICTOR & BORIS
Unity 6000.2.3f1 • Generated 2026-04-18 12:13

Theme

📖 Overview

🌍 Overview

RealFPS is a complete and modular First Person Shooter framework built for Unity 6000.

It provides all the core systems required to build immersive FPS gameplay quickly, including:

🎯 Weapon System – shooting, reloading, ammo and energy management

🎒 Inventory System – item collection, stacking, UI and world interaction

🖱️ Interaction System – unified raycast-based interaction for all gameplay elements

📦 Carry System – pick up, move, and throw physical objects with dedicated arms

🤖 AI System – enemies with patrol, detection, chase and attack behaviors

🔐 Door & Interaction Systems – keys, locks, switches and event-driven gameplay

All systems are designed to work together while remaining fully modular, allowing you to use only what you need.

RealFPS is built with a data-driven approach using ScriptableObjects, making it easy to configure and extend.

🧠 Core Philosophy

RealFPS focuses on:

Modularity – each system can be used independently

Clarity – simple setup with powerful results

Scalability – suitable for prototypes and full games

Runtime Tuning – adjust gameplay directly in Play Mode

🧩 System Overview

Player

├── InteractionRaycaster

├── GeneralInventoryManager

├── FPSWeaponManager

├── FPSCarryManager

└── FPSControllerPro

Everything revolves around the player and connects through clean, reusable systems.

🛠️ How to Use

🚀 How To Use

This section guides you through the basic setup of a functional RealFPS scene.

1️⃣ Create the Player 🎮

Add your player GameObject

Add the following components:

FPSControllerPro

InteractionRaycaster

GeneralInventoryManager

FPSWeaponManager

FPSCarryManager

Assign your Main Camera

2️⃣ Setup Interaction 🖱️

Configure InteractionRaycaster:

assign camera

set interaction distance

define layer mask

Add interactable components to objects:

ItemPickup

CarryableObject

DoorLockInteractable

ToggleEventInteractable

👉 All interactions will now work automatically.

3️⃣ Setup Inventory 👜

Create ItemData assets:

define item type (Weapon, Food, Key, etc.)

assign icon and prefab

Add items to the scene using ItemPickup

Configure GeneralInventoryManager

👉 Items can now be collected, stored, and used.

4️⃣ Setup Weapons 🔫

Create an ArmsAndWeaponData asset

Configure:

animations

ammo or energy

shooting behavior

Add weapon to:

WeaponInventoryManager

Assign references in FPSWeaponManager

👉 You can now equip, shoot, and reload weapons.

5️⃣ Setup Carry System 📦

Add CarryableObject to objects

Configure:

offsets

physics settings

(Optional) assign a CarryArmsData for custom arms

👉 Player can now pick up, carry, and throw objects.

6️⃣ Setup AI 👹

Add EnemyAI and EnemyHealth to an enemy

Add:

NavMeshAgent

Animator

Configure:

patrol points

detection range

attack settings

👉 Enemy will now patrol, detect, and attack the player.

7️⃣ Setup Doors & Events 🚪

Add DoorLockInteractable to doors

Configure:

lock state

required key

Add switches using ToggleEventInteractable

👉 You now have interactive environments.

8️⃣ Test in Play Mode ▶️

Interact with objects

pick up items

use inventory

shoot weapons

carry objects

fight enemies

⚓💪 TEAM VICTOR & BORIS Note

RealFPS is designed to let you build gameplay step by step.

Start simple:

➞ Player + Interaction

Then expand:

➞ Inventory → Weapons → AI → Carry → Missions

💡 Pro Tip:

👉 Use ArmsLiveTuner in Play Mode to adjust arms and weapon alignment visually

instead of tweaking values manually.

🧭 Recommended Workflow

Interaction → Inventory → Weapons → Carry → AI → Polish

🏁 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.

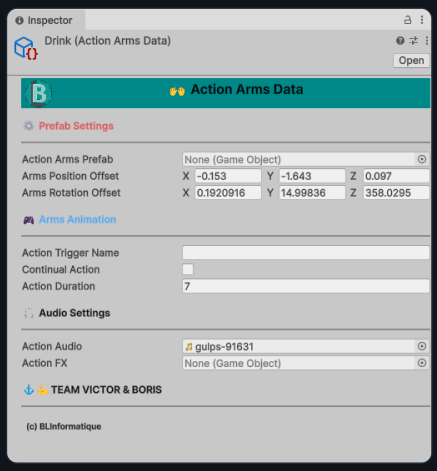
Action FX 💥

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.

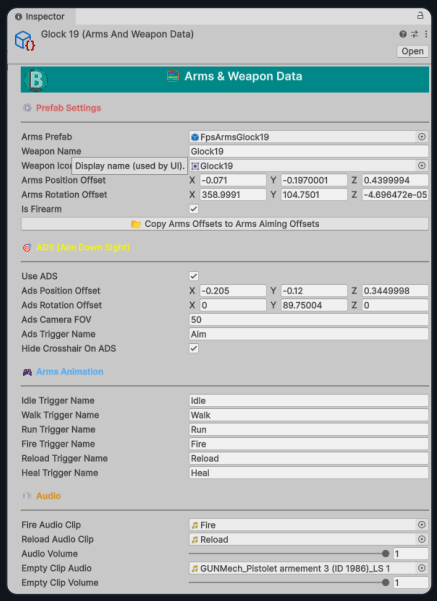
Weapon behaves like a tool? Check that "Is Firearm" is enabled.

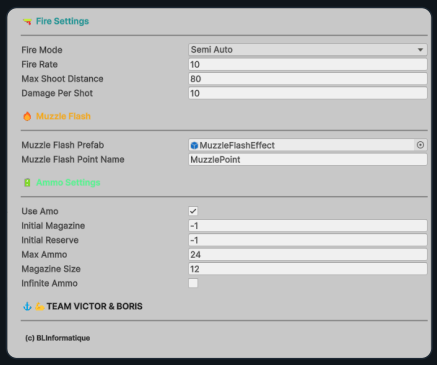
Energy not draining? Make sure “Requires Energy” is enabled and the device is switched on.

Compatibility ✅

Works with New Input System.

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





Field	Type	Tooltip
⬇		
armsPrefab	`GameObject`	Fully configured arms+weapon 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 for firearms (pistols, rifles...). FALSE for tools/flashlights unless you tick melee below.
isMeleeWeapon	`Boolean`	TRUE for melee weapons (knife, sword, crowbar...). Impacts only when close enough to actually touch.
boutonDummy	`Int32`	Copies current arms offsets to ADS offsets (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 when entering idle (optional).
walkTriggerName	`String`	Animator trigger/state when walking (optional).
runTriggerName	`String`	Animator trigger/state when running (optional).
fireTriggerName	`String`	Animator trigger/state when firing/using (optional).
reloadTriggerName	`String`	Animator trigger/state when reloading (optional).
healTriggerName	`String`	Animator trigger/state when healing (optional).
⬇		
fireAudioClip	`AudioClip`	Sound to play when firing (optional).
reloadAudioClip	`AudioClip`	Sound to play when reloading (optional).
audioVolume	`Single`	Volume for firearm sounds (0–1). • Range [0..1]
emptyClipAudio	`AudioClip`	Sound to play when magazine is empty (optional).
emptyClipVolume	`Single`	Volume for empty magazine sound (0–1). • Range [0..1]
⬇		
fireMode	`FireMode`	SemiAuto (click) or FullAuto (hold).

Field	Type	Tooltip
fireRate	`Single`	Shots per second (FullAuto only).
maxShootDistance	`Single`	Raycast max range (meters).
damagePerShot	`Single`	Damage per shot.
⬇		
muzzleFlashPrefab	`GameObject`	Prefab spawned at every shot (particles/FX).
muzzleFlashPointName	`String`	Transform name in the arms+weapon prefab used as muzzle spawn point.
⬇		
useAmo	`Boolean`	If false, ammo is not used.
initialMagazine	`Int32`	Ammo in magazine on first pickup (-1 = full).
initialReserve	`Int32`	Reserve ammo on first pickup (-1 = max).
maxAmmo	`Int32`	Max total reserve ammo.
magazineSize	`Int32`	Ammo per magazine/clip.
infiniteAmmo	`Boolean`	If true, ammo is infinite.
⬇		
meleeRange	`Single`	Short hit test range (meters). Impacts only if a surface is within this distance.
meleeHitRadius	`Single`	Sphere radius for hit test (meters). Use 0 for a thin ray; 0.1–0.25 feels forgiving.
meleeConeAngle	`Single`	Optional extra spread angle (degrees) around forward to allow slight aim error. • Range [0..20]
meleeDamage	`Single`	Damage applied on a successful hit.
meleeImpactFX	`GameObject`	FX prefab spawned on hit (sparks/blood/debris). Optional.
meleeImpactAudio	`AudioClip`	Audio played on hit (optional).
meleeRequireHitForImpact	`Boolean`	If true, impact FX & audio trigger only if we actually hit something.
meleeHitMask	`LayerMask`	LayerMask used by melee hit test (0 = Everything). Leave 0 to use the manager defaults.
⬇		
requiresEnergy	`Boolean`	If true, this device requires energy to operate (flashlight, tools, goggles, etc.).
energyMax	`Single`	Max energy capacity.
energyDrainPerSecond	`Single`	Energy drained per second while ON.
initialEnergyOnEquip	`Single`	Initial energy when equipped (clamped to energyMax).
autoSwitchOffOnDeplete	`Boolean`	Auto switch OFF when energy hits zero?
energyDesignNote	`String`	Designer note (UI helper only). • 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 Play Mode utility used to fine-tune weapon arms offsets and carry arms offsets directly at runtime.

It removes the usual trial-and-error workflow in the Inspector and gives you a fast overlay-based setup for positioning arms visually while the game is running.

It now supports both the classic weapon workflow and the new carry object arms workflow.

Target 🎯

Weapon Manager 🧩

Reference to the FPSWeaponManager.

If left empty, it is auto-resolved at runtime.

Carry Manager 📦

Reference to the FPSCarryManager.

If left empty, it is auto-resolved at runtime.

This is used when tuning carry arms while the player is transporting an object.

Edit Set 🛠

Choose which offsets you want to edit:

IdleOffsets – default weapon resting pose.

ADSOffsets – aim-down-sight pose.

CarryOffsets – carry arms pose used while transporting an object.

Smart Mode 🧠

The tuner now includes a smart carry workflow:

Auto Switch To Carry When Carrying – automatically switches to CarryOffsets when a carried object with carry arms is active.

Auto Leave Carry When Not Carrying – automatically exits carry mode when no carried object is active anymore.

Default Non Carry Edit Set – defines which mode to return to after leaving carry mode, usually IdleOffsets.

↑

This is very useful when testing crates, boxes, lamps, or any carryable object without manually changing mode every time.

Hotkeys ⌨️

Toggle Key (default: T) – Enable/disable the tuner. Can also show/hide the overlay depending on your overlay settings.

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

Slow Key (default: LeftAlt) – Apply the slow multiplier to movement/rotation steps.

Fast Key (default: LeftShift) – Apply the fast multiplier to movement/rotation steps.

F1 – Switch to IdleOffsets

F2 – Switch to ADSOffsets

F3 – Switch to CarryOffsets

R – Apply stored data back to the currently tuned transform.

Steps 📐

Position Step – Base movement increment in meters.

Rotation Step – Base rotation increment in degrees.

Slow Multiplier – Multiplies the current step while holding the slow key.

Fast Multiplier – Multiplies the current step while holding the fast key.

Controls in Play Mode 🎮

Move

A / D = X

W / S = Y

Q / E = Z

Rotate

Up / Down = X

Left / Right = Y

Page Up / Page Down = Z

Carry Arms Support 📦👐

This is the big new part.

When CarryOffsets is active, the tuner edits the currently instantiated carry arms visual from FPSCarryManager.currentCarryArmsInstance.

The source data is resolved like this:

If the carried object uses a CarryArmsData asset, offsets are written to that asset.

If no CarryArmsData is assigned, the tuner falls back to the FPSCarryManager fallback carry values (carryArmsLocalPosition and carryArmsLocalEuler).

In the overlay, the tuner shows:

the current carried object

the current carry source

whether it is using a real CarryArmsData

or the manager fallback values instead.

That means you can now tune:

a global fallback carry arms setup in FPSCarryManager

or a per-object carry arms setup through CarryArmsData assigned on CarryableObject.

Carry Requirements ✅

To tune carry arms correctly:

the player must currently be carrying a valid object

the carried object must have an active carry arms visual

FPSCarryManager must be able to spawn that visual

If a carried object exists but no carry arms visual is spawned, the overlay warns you and reminds you to enable Use Carry Arms Visual and assign a carry prefab in CarryArmsData or in the fallback fields of FPSCarryManager.

Freeze Options 🧊

When freezing the world, the tuner can also:

disable the FPS controller

disable weapon systems

disable the inventory UI

pause all Animators in the scene

This makes it much easier to adjust arms without movement, sway, or gameplay noise getting in the way.

Overlay 🪟

The runtime overlay includes:

current status: Active / Frozen

current Edit Set

live position / rotation

quick action buttons

drag support

optional resizing

optional scrolling

saved position and size via PlayerPrefs.

Overlay options

Show Overlay

Overlay Draggable

Overlay Save Position

Overlay Resizable

Overlay Scrollable

Hide Overlay On Play

Overlay Follows Active – if enabled, the overlay visibility follows the tuner active state.

Quick Buttons ⚡

Available in all modes

🧊 Freeze / Unfreeze

💾 Save (Editor only)

⟲ Reset Set

Apply From Data (R)

↖ Reset Overlay Pos/Size

Available only for weapon modes

Idle → ADS

ADS → Idle

These copy functions are intentionally hidden in carry mode, because carry tuning has its own dedicated data flow.

Save Behavior 💾

Weapon modes

When editing IdleOffsets or ADSOffsets, the tuner updates the currently equipped ArmsAndWeaponData.

Carry mode

When editing CarryOffsets, the tuner saves to:

CarryArmsData if one is assigned to the carried object

otherwise to the fallback values stored directly on FPSCarryManager.

This matches the carry system design, where each object can override the default carry setup with its own CarryArmsData.

↑

Related Systems 🔗

CarryArmsData

CarryArmsData stores:

carry arms prefab

local position offset

local rotation offset

optional parent mode

optional sway override values

It also includes an editor button to save the current carry arms instance back into the ScriptableObject during Play Mode.

FPSCarryManager

FPSCarryManager provides:

fallback carry arms prefab

fallback carry offsets

carry anchor handling

sway support for carry arms

runtime current carry arms instance and current carry data references.

Debug 🐞

Runtime debug fields include:

isActive – true when the tuner is actively applying inputs

isFrozen – true when the world is frozen

usingCarryManagerFallback – true when carry tuning uses fallback values instead of a CarryArmsData

autoUsingCarryMode – true when Smart Mode automatically switched the tuner to carry mode.

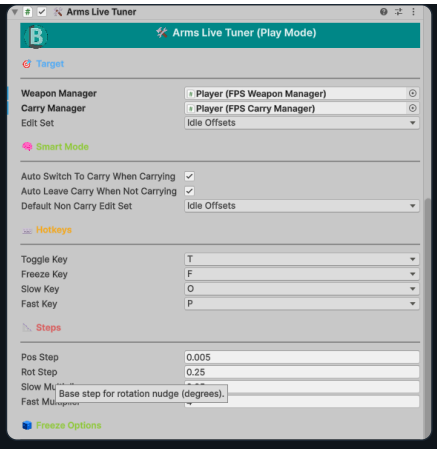
⚓ 💪 TEAM VICTOR & BORIS Note

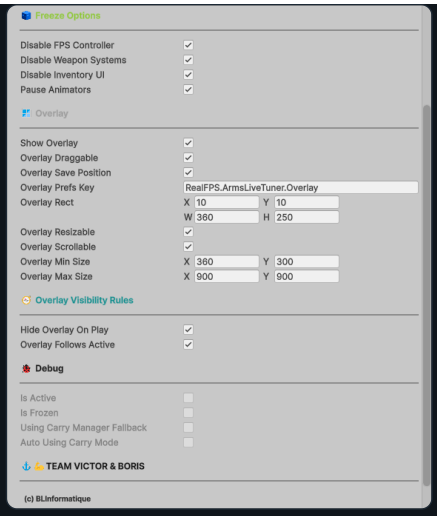
Arms Live Tuner is now the fastest way to align both weapon arms and carry arms in RealFPS.

For weapons, use IdleOffsets and ADSOffsets.

For transported objects, switch to CarryOffsets or let Smart Mode do it for you automatically.

Tune live, freeze the world, save, and stop fighting the Inspector like a pirate fighting a storm with a spoon.


Arms Live Tuner
Arms Live Tuner (Play Mode)
Target
Weapon Manager
Player (FPS Weapon Manager)
Carry Manager
Player (FPS Carry Manager)
Edit Set
Idle Offsets
Smart Mode
Auto Switch To Carry When Carrying
Auto Leave Carry When Not Carrying
Default Non Carry Edit Set
Idle Offsets
Hotkeys
Toggle Key
T
Freeze Key
F
Slow Key
O
Fast Key
P
Steps
Pos Step
0.005
Rot Step
0.25
Base step for rotation nudge (degrees).
Freeze Options


Freeze Options
Disable FPS Controller
Disable Weapon Systems
Disable Inventory UI
Pause Animators
Overlay
Show Overlay
Overlay Draggable
Overlay Save Position
Overlay Prefs Key
RealFPS.ArmsLiveTuner.Overlay
Overlay Rect
X 10
Y 10
W 360
H 250
Overlay Resizable
Overlay Scrollable
Overlay Min Size
X 360
Y 300
Overlay Max Size
X 900
Y 900
Overlay Visibility Rules
Hide Overlay On Play
Overlay Follows Active
Debug
Is Active
Is Frozen
Using Carry Manager Fallback
Auto Using Carry Mode
TEAM VICTOR & BORIS
(c) BLInformatique


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.
carryManager	`FPSCarryManager`	FPSCarryManager to control. If null, will auto-find at runtime.
editSet	`EditSet`	Which set of offsets are you editing right now (Idle, ADS or Carry)?
autoSwitchToCarryWhenCarrying	`Boolean`	If true, automatically switches to CarryOffsets when carrying an object and back to IdleOffsets when carry ends.
autoLeaveCarryWhenNotCarrying	`Boolean`	If true, automatically leaves CarryOffsets when no carried object exists anymore.
defaultNonCarryEditSet	`EditSet`	Default edit set used when leaving carry mode automatically.
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.

Field	Type	Tooltip
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).
⬇		
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?
usingCarryManagerFallback	`Boolean`	True if carry tuning currently uses FPSCarryManager fallback values instead of a CarryArmsData asset.
autoUsingCarryMode	`Boolean`	True when the tuner auto-switched to CarryOffsets because an object is being carried.

↑ Back to top

Copy fields

📦 CarryableObject MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Carry/📦 Carryable Object`

Carryable Object 📦

The CarryableObject component makes any scene object compatible with the RealFPS carry system.

It allows the player to pick up, move, carry, drop, and throw objects such as crates, boxes, lamps, or mission cargo.

This script works together with FPSCarryManager and optionally with CarryArmsData to control how the object behaves and how the carry arms are displayed while it is being transported.

Purpose 🎯

Use this component when you want an object to:

be picked up by the player

follow the carry anchor while being transported

optionally use custom carry arms

optionally become transparent while carried

optionally behave as mission cargo

restore its physics correctly when dropped.

Carry Settings 📦

Display Name

Name shown in interaction prompts.

Interaction Verb

Verb used in contextual prompts, for example:

carry crate

carry box

Can Be Carried

If enabled, the object can be picked up by the carry system.

Carry Weight

Weight value used by the carry system.

This can be used to influence player movement or balancing.

Carried Local Position

Local position offset of the object while carried.

Carried Local Euler

Local rotation offset of the object while carried.

Can Be Thrown

If enabled, the object can be thrown instead of only dropped.

Carry Root

Optional root transform moved by the carry system.

If left empty, the script uses the current GameObject transform.

Carry Arms Data

Optional CarryArmsData asset used to override the default carry arms visual for this specific object.

This is useful if a crate, lamp, or special object needs its own arms prefab and offsets.

Transparency While Carried 👁️

This script can make the carried object transparent while the player is holding it.

That is especially useful for large crates or boxes that would otherwise block the camera view.

Make Transparent While Carried

If enabled, the object becomes transparent during carry.

Use Transparent Override Material

If enabled, a dedicated transparent material is assigned while the object is carried.

Transparent Override Material

Material used when override mode is enabled.

Carried Transparency Alpha

Target alpha used while carried when transparency is applied.

Include Child Renderers For Transparency

If enabled, transparency is also applied to child renderers.

Apply Transparency Immediately On Pickup

If enabled, transparency is applied as soon as the object is picked up by gameplay or code.

Mission Cargo 📦🧭

CarryableObject can also be used as delivery cargo for missions.

Mission Cargo Id

Identifier used by mission delivery objectives.

Examples:

crate

supply_crate

food_box

Mission Delivery Amount

Amount contributed by this object when counted inside a delivery zone.

This makes the script compatible with delivery-style objectives such as loading crates into a truck, warehouse, or mission area.

Physics Settings 🔧

Target Rigidbody

Main Rigidbody controlled while carrying.

If left empty, it is automatically resolved.

Target Colliders

Optional list of colliders explicitly controlled by the carry system.

If left empty, all child colliders are used.

Restore Gravity On Drop

If enabled, gravity is restored when the object is dropped.

Use Kinematic While Carried

If enabled, the Rigidbody becomes kinematic while the object is carried.

Disable Colliders While Carried

If enabled, colliders are disabled during carry for maximum stability.

These options help prevent unwanted jitter, collision issues, or unstable physics while the object is attached to the player.

Sway Settings

Carryable objects can also have a subtle sway motion while being held.

Enable Sway

Enable idle sway while carrying.

Sway Amount X

Horizontal sway amount.

Sway Amount Y

Vertical sway amount.

Sway Speed

Speed of the sway motion.

This creates a more natural handheld feel, especially for larger carried objects.

Runtime Fields

Is Currently Carried

True while the object is actively carried.

Current Carry Manager

Reference to the current FPSCarryManager holding the object.

These fields are mainly useful for debugging.

How It Works

When the player picks up the object:

references are auto-assigned if needed
the Rigidbody is prepared for carry
gravity is disabled
the Rigidbody can become kinematic
colliders can optionally be disabled
the object is parented to the active carry anchor
local carry offsets are applied
optional transparency is applied.

While the object is being carried:

it follows the carry anchor
sway can be applied
Rigidbody velocity is kept stable
the object remains tracked by the carry manager.

When the object is dropped:

parent is restored
Rigidbody settings are restored
release velocity is applied
gravity can be restored

transparency is removed

colliders can be restored.

Interaction 🔗

CarryableObject implements the IInteractable interface.

That means it can work directly with InteractionRaycaster and show contextual prompts such as:

Press E to carry Crate

When interacted with, it automatically looks for a valid FPSCarryManager through the player inventory or in the scene.

Recommended Setup ✅

For a typical carryable crate:

add a Collider

add a Rigidbody

add CarryableObject

optionally assign a CarryArmsData

configure local carry position and rotation

enable transparency if the object blocks the view

configure mission cargo fields if used in objectives.

Best Use Cases 🧰

This script is ideal for:

crates

supply boxes

carried lamps

mission cargo

throwable props

heavy environmental objects moved by the player.

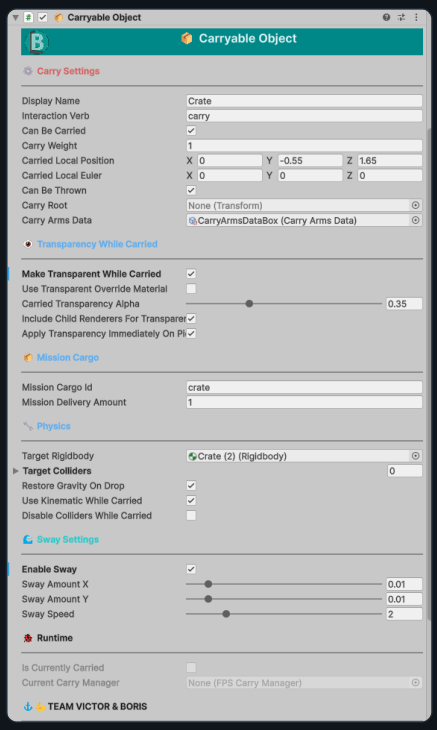
⚓💪 TEAM VICTOR & BORIS Note

CarryableObject is the bridge between the world object and the RealFPS carry system.

It does not just say “this object can be picked up” — it defines how the object behaves in the player’s hands, how stable it feels, how visible it remains, and how it can integrate into missions.

In short: it turns a dumb box into a proper gameplay box. Which is still a box… but now with ambitions.

Works with: FPSCarryManager, CarryArmsData, InteractionRaycaster, MissionDeliveryZone



Field	Type	Tooltip
⬇		
displayName	`String`	Display name used in interaction prompts.
interactionVerb	`String`	Interaction verb used in interaction prompts.
canBeCarried	`Boolean`	If true, this object can be carried.
carryWeight	`Single`	Weight used by the carry system to optionally slow down the player.
carriedLocalPosition	`Vector3`	Local position offset while carried.
carriedLocalEuler	`Vector3`	Local rotation offset while carried.
canBeThrown	`Boolean`	If true, this object can be thrown.
carryRoot	`Transform`	Optional root object moved by the carry system. Leave empty to use this GameObject.
carryArmsData	`CarryArmsData`	Optional carry arms data used to override the default carry arms visual for this object.
⬇		
makeTransparentWhileCarried	`Boolean`	If true, the object becomes transparent while it is being carried.
useTransparentOverrideMaterial	`Boolean`	If true, uses a transparent override material while carried. If false, modifies the current materials alpha at runtime.
transparentOverrideMaterial	`Material`	Transparent material applied to all renderer slots while the object is carried.
carriedTransparencyAlpha	`Single`	Target alpha applied to materials while carried. Lower values are more transparent. • Range [0,05..1]
includeChildRenderersForTransparency	`Boolean`	If true, child renderers are also affected when transparency is applied.
applyTransparencyImmediatelyOnPickup	`Boolean`	If true, transparency is also applied when the object is picked up at runtime through scripts.
⬇		
missionCargoId	`String`	Cargo identifier used by mission delivery objectives. Example: crate_small, supply_crate, food_box.
missionDeliveryAmount	`Int32`	Amount contributed by this object when counted inside a delivery zone.

Field	Type	Tooltip
⬇		
targetRigidbody	`Rigidbody`	Main Rigidbody used while carrying. If empty, it is auto-resolved.
targetColliders	`Collider[]`	Optional colliders explicitly controlled by the carry system. If empty, all child colliders are used.
restoreGravityOnDrop	`Boolean`	If true, gravity is restored when the object is dropped.
useKinematicWhileCarried	`Boolean`	If true, the Rigidbody is set to kinematic while carried.
disableCollidersWhileCarried	`Boolean`	If true, colliders are disabled while carried for maximum stability.
⬇		
enableSway	`Boolean`	Enable idle sway motion while carrying.
swayAmountX	`Single`	Sway amplitude on X axis. • Range [0..0,1]
swayAmountY	`Single`	Sway amplitude on Y axis. • Range [0..0,1]
swaySpeed	`Single`	Sway speed. • Range [0,1..10]
⬇		
isCurrentlyCarried	`Boolean`	True while this object is currently carried by a player.
currentCarryManager	`FPSCarryManager`	Current carry manager holding this object.

↑ Back to top

Copy fields

📘 CarryArmsData ScriptableObject

Carry Arms Data 👐

The CarryArmsData ScriptableObject defines how arms are displayed and positioned when the player carries a specific object.

It allows each carryable object to have its own dedicated arms prefab, offsets, and behavior, instead of relying on a global fallback.

This system works in combination with:

CarryableObject

FPSCarryManager

ArmsLiveTuner

Purpose 🎯

Use CarryArmsData to:

assign a specific arms prefab for a carried object

define position and rotation offsets

control how the arms are attached

optionally override sway behavior

save offsets directly from Play Mode using the tuner

Arms Setup 🧩

Carry Arms Prefab

Prefab instantiated when the object is carried.

This is typically a pair of arms holding the object (crate, lamp, etc.).

Parent To Arms Holder

If enabled, the arms are parented to the FPSWeaponManager armsHolder.

If disabled, the system uses the carry anchor logic instead.

Offsets 📐

Arms Position Offset

↑

Local position of the carry arms relative to their parent.

Arms Rotation Offset

Local rotation of the carry arms.

These values define how the arms align with the camera and the carried object.

Sway Override

Carry arms can either:

use the global sway from FPSCarryManager

or override it locally here

Override Sway

If enabled, this ScriptableObject overrides sway settings.

Sway Position Multiplier

Multiplier applied to position sway.

Sway Rotation Multiplier

Multiplier applied to rotation sway.

Extra Rotation Sway X

Additional sway on X axis.

Extra Rotation Sway Z

Additional sway on Z axis.

Quick Save (Play Mode)

Save From Instance Button

This button allows saving the current transform values of the instantiated carry arms directly into this ScriptableObject.

👉 This is designed to be used with ArmsLiveTuner.

Workflow:

Enter Play Mode

Carry an object

Adjust arms using ArmsLiveTuner

Click Save From Instance

Offsets are written into the ScriptableObject

Runtime Behavior ⚙️

When an object is carried:

FPSCarryManager checks if the object has a CarryArmsData

if yes → it instantiates the prefab defined here

applies position & rotation offsets

applies sway (override or global)

keeps the instance synced with the carry anchor

If no CarryArmsData is assigned:

➞ the system falls back to FPSCarryManager default carry arms settings

Integration 🔗

With CarryableObject

Each object can reference a CarryArmsData to define its own arms.

With FPSCarryManager

Handles instantiation, parenting, and lifecycle of the arms instance.

With ArmsLiveTuner

Allows real-time tuning and saving of offsets.

Best Practices ✅

Use one CarryArmsData per object type (crate, lamp, etc.)

Keep offsets clean and centered for consistency

Use ArmsLiveTuner instead of editing values manually

Only override sway when necessary

Test in motion (walking + looking) to validate alignment

⚓ 💪 TEAM VICTOR & BORIS Note

CarryArmsData is what gives personality to carried objects.

Without it:

→ all objects feel the same

With it:

→ every object can feel unique, weighty, and believable in the player's hands

It's the difference between:

"carrying a box"

and

"carrying THIS box"

CarryableObject

↓

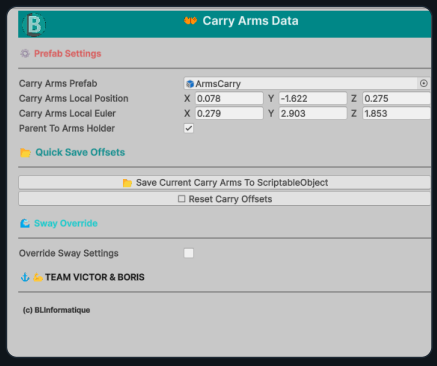
CarryArmsData (optional)

↓

FPSCarryManager

↓

Arms Instance (runtime)



Field	Type	Tooltip
⬇		
carryArmsPrefab	`GameObject`	Carry arms prefab used for this carried object.
carryArmsLocalPosition	`Vector3`	Local position offset for the carry arms prefab.
carryArmsLocalEuler	`Vector3`	Local rotation offset for the carry arms prefab.
parentToArmsHolder	`Boolean`	If true, the carry arms visual is parented to ArmsHolder. Otherwise, it is parented to the camera.
⬇		
saveCarryOffsetsButton	`Int32`	Click to save the current carry arms instance position and rotation from the scene into this ScriptableObject. Play Mode only.
resetCarryOffsetsButton	`Int32`	Click to reset carry offsets to zero.
⬇		
overrideSwaySettings	`Boolean`	If true, override the default sway settings from FPSCarryManager.
swayPositionMultiplier	`Single`	Multiplier applied to sway position on carry arms. • Range [0..2]
swayRotationMultiplier	`Single`	Multiplier applied to sway rotation on carry arms. • Range [0..5]
rotationSwayX	`Single`	Extra rotation sway on X axis for carry arms. • Range [0..15]
rotationSwayZ	`Single`	Extra rotation sway on Z axis for carry arms. • Range [0..15]

↑ Back to top

Copy fields

📦 DoorLockInteractable MonoBehaviour

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

Door Lock Interactable 🚪🔐

The DoorLockInteractable component allows you to create doors that can be opened, locked, and unlocked using keys from the inventory.

It integrates seamlessly with:

InteractionRaycaster

GeneralInventoryManager

ItemData (Key system)

Animator-based or transform-based doors

Purpose 🎯

Use this component to:

create locked/unlocked doors

require specific keys from the inventory

trigger animations or rotations

fire events when opening or unlocking

support both manual and automatic door behavior

Interaction 🖱️

This script implements IInteractable, so it works directly with:

👉 InteractionRaycaster

When the player looks at the door:

Press E to open Door

Press E to unlock Door

The behavior depends on:

whether the door is locked

whether the player has the correct key

Lock System 🔐

↑

Is Locked

If enabled, the door starts locked.

Required Key ID

String used to match a key from the inventory.

The system checks keys using:

exact key ID match

OR a "MASTER" key (universal key)

👉 This is handled through GeneralInventoryManager.HasKeyId()

Key Consumption 🔑

Consume Key On Use

If enabled:

the key is removed from inventory after unlocking

If disabled:

the key can be reused infinitely

Door Behaviour 🚪

The script supports multiple ways to open a door:

1. Animator-Based Door 🎬

If an Animator is assigned:

triggers are used to play open/close animations

Common setup:

Open trigger

Close trigger

2. Transform-Based Door 🔄

If no Animator is used:

the door rotates or moves manually

Typical use:

hinge doors

simple prototypes

sliding doors

Open State ⚙️

Is Open

Current state of the door.

Used internally to prevent duplicate triggers.

Events 🔔

OnOpened

Called when the door is fully opened.

👉 Important:

You already improved this so it triggers when the door is actually open, not just when interaction starts (good job 😍)

OnLocked

Called when trying to open without the correct key.

OnUnlocked

Called when the door is successfully unlocked.

Audio 🔊

Locked Sound

Played when trying to open a locked door without the key.

Unlock Sound

Played when unlocking the door.

Open Sound

Played when the door opens.

Integration 🔗

With Inventory System

Uses:

GeneralInventoryManager

ItemData with ItemType.Key

Keys are matched using:

keyId == requiredKeyId

or

keyId == "MASTER"

With InteractionRaycaster

Handles player input

Displays contextual prompt

Calls Interact() automatically

Typical Workflow 🧱

Setup a Locked Door

Add DoorLockInteractable

Enable Is Locked

Set Required Key ID (ex: "HouseKey")

Create an ItemData:

Type = Key

keyId = "HouseKey"

Add Animator or rotation setup

Done ✅

Advanced Usage 💡

Master Key System

Set a key with:

keyId = "MASTER"

➞ This key opens all doors

Event Driven Gameplay

Use events to:

trigger cutscenes

spawn enemies

update missions

play VFX

Mission Integration

Doors can be used as:

progression blockers

objectives (unlock door)

triggers for mission events

Debug 🐞

Helpful checks:

Door doesn't open → check Animator or rotation setup

Key not working → check keyId match

Prompt not showing → check InteractionRaycaster

⚓ 💪 TEAM VICTOR & BORIS Note

DoorLockInteractable is the backbone of progression in RealFPS.

It connects:

the world (doors)

the player (interaction)

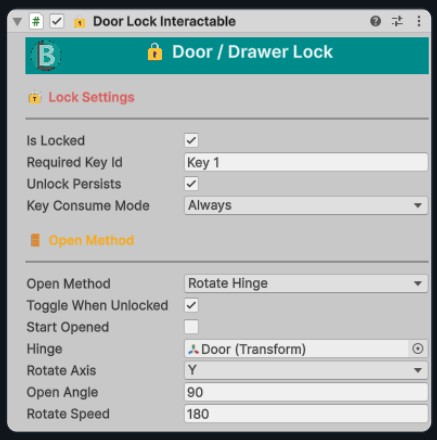
the inventory (keys)

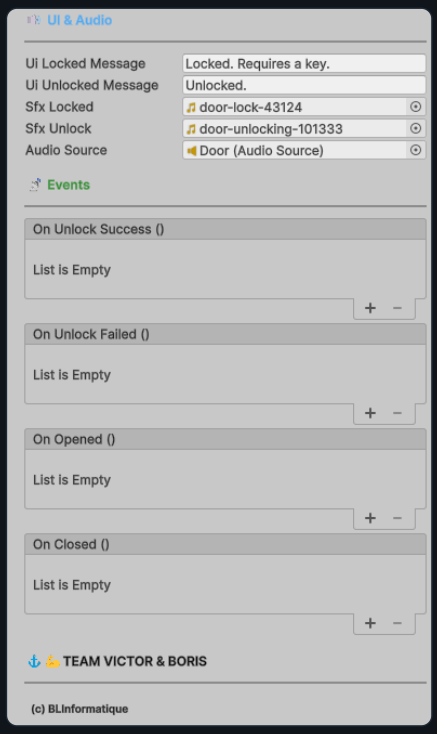
A simple door becomes:

➞ a gameplay decision

➞ a reward system

➞ a narrative tool





Field	Type	Tooltip
missionManager	`Object`	Internal MissionManager reference resolved automatically when the optional addon is installed.
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.
interactionMode	`DoorInteractionMode`	Choose how this door can be used. - ManualOnly: only works via Interact. - AutomaticOnly: opens/closes from trigger. - AutomaticAndManual: supports both.
requirePlayerInsideTriggerForManual	`Boolean`	If true, manual interaction is allowed only while the player is inside the trigger zone.
triggerTag	`String`	Tag required to activate the automatic trigger.
openOnEnter	`Boolean`	If true, the door opens automatically when a valid object enters the trigger.
closeOnExit	`Boolean`	If true, the door closes automatically when a valid object exits the trigger.
forceBoxColliderAsTrigger	`Boolean`	If true, the trigger collider is forced to isTrigger on Reset/Awake when a BoxCollider is found.
controlMode	`DoorControlMode`	How the door is controlled.
toggleWhenUnlocked	`Boolean`	If true, interaction toggles open/close when already unlocked.
startOpened	`Boolean`	Start opened at play.

Field	Type	Tooltip
animatorTrigger	`Animator`	Animator that drives the open animation in Trigger mode.
triggerOpenName	`String`	Trigger parameter used to open the door.
triggerCloseName	`String`	Optional trigger parameter used to close the door.
↓		
animatorBool	`Animator`	Animator that drives the open animation in Bool mode.
animatorBoolName	`String`	Bool parameter used to represent open(true) or closed(false).
↓		
leaves	`DoorLeaf[]`	List of door leaves/panels. For double doors, add 2 leaves. Each leaf can rotate or slide.
motionSpeed	`Single`	Motion smoothing speed for manual leaves.
forceClosedOnAwake	`Boolean`	If true, the CLOSED pose is applied on Awake.
↓		
fireEventsOnlyWhenStateFullyReached	`Boolean`	If true, OnOpened and OnClosed are fired only when the door has fully reached its target state.
rotationReachedThreshold	`Single`	Rotation tolerance in degrees used to consider a rotating leaf fully reached.
positionReachedThreshold	`Single`	Position tolerance used to consider a sliding leaf fully reached.
animatorOpenCompleteDelay	`Single`	Delay in seconds before considering the OPEN state complete in Animator modes.
animatorCloseCompleteDelay	`Single`	Delay in seconds before considering the CLOSED state complete in Animator modes.
animatorBoolOpenCompleteDelay	`Single`	Delay in seconds before considering the OPEN state complete in Animator modes.
animatorBoolCloseCompleteDelay	`Single`	Delay in seconds before considering the CLOSED state complete in Animator modes.
notifyMissionOnOpened	`Boolean`	If true, this door notifies the mission system when it opens successfully.

Field	Type	Tooltip
notifyMissionOnUnlockSuccess	`Boolean`	If true, this door notifies the mission system when it is successfully unlocked.
missionObjectiveId	`String`	Objective ID sent to the mission system. Recommended workflow for mission validation.
notifyMissionOnlyForManualUse	`Boolean`	If true, mission notification is sent only for manual/player interaction, not for automatic trigger opening.
⬇		
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.
sfxOpen	`AudioClip`	Audio when the object starts opening.
sfxClose	`AudioClip`	Audio when the object starts closing.
audioSource	`AudioSource`	Audio source used to play sounds. If null, AudioSource.PlayClipAtPoint is used.
⬇		
OnUnlockSuccess	`UnityEvent`	Invoked when the door is successfully unlocked.
OnUnlockFailed	`UnityEvent`	Invoked when unlock fails.
OnOpeningStarted	`UnityEvent`	Invoked when the object starts opening.
OnOpened	`UnityEvent`	Invoked when the object is fully opened.
OnClosingStarted	`UnityEvent`	Invoked when the object starts closing.
OnClosed	`UnityEvent`	Invoked when the object is fully closed.
OnToggled	`UnityEvent`	Invoked after every open or close request.
OnTriggerEntered	`UnityEvent`	Invoked when a valid trigger enters the automatic zone.
OnTriggerExited	`UnityEvent`	Invoked when a valid trigger exits the automatic zone.
⬇		
showDebugLogs	`Boolean`	Enable debug logs in the Console.

Field	Type	Tooltip
runtimeIsOpen	`Boolean`	True if the door target state is open.
runtimeIsLocked	`Boolean`	True if the door is currently locked.
runtimePlayerInsideTrigger	`Boolean`	True if a valid trigger object is currently inside the zone.
runtimeIsTransitioning	`Boolean`	True while the door is moving toward its target state.
runtimeIsFullyOpen	`Boolean`	True when the door has fully reached the open state.
runtimeIsFullyClosed	`Boolean`	True when the door has fully reached the closed state.

↑ Back to top

Copy fields

EnemyAI MonoBehaviour

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

Enemy AI

The EnemyAI component controls the behavior of enemies in RealFPS.

It provides a complete system for patrolling, detecting, chasing, and attacking the player, with flexible options for different enemy types such as humans or animals.

Purpose

Use this component to create enemies that can:

patrol between points (or randomly)

detect the player within a range

chase the player (walking or running)

attack the player when close

return to patrol when the player is lost

support different movement styles (biped / quadruped ready)

AI States

The enemy operates using a simple state system:

Patrol

Moves between waypoints or random positions

Default idle behavior

Chase

Triggered when the player is detected

Moves toward the player

Attack

Triggered when within attack range

Plays attack animation and applies damage

Return

Triggered when the player is lost

Returns to patrol route

Detection System

Detection Range

Distance at which the enemy can detect the player.

Field of View (FOV)

Angle of vision used to detect the player.

Line of Sight

Enemy must have a clear raycast to the player.

Movement ⚙️

Patrol Speed

Speed used during patrol.

Chase Speed

Speed used when chasing the player.

Use Run When Chasing

If enabled:

enemy runs when chasing

otherwise walks toward the player

👉 This is the option you added récemment 👌

Patrol System 🗺️

Patrol Points

List of waypoints used for patrol.

Random Patrol

If enabled:

enemy chooses random positions instead of fixed points

Wait Time

Time to wait at each patrol point.

Attack System ⚔️

Attack Range

Distance required to attack the player.

Attack Damage

Damage applied to the player.

Attack Rate

Delay between attacks.

Animation 🎬

The script is designed to work with an Animator:

Typical parameters:

MoveState (int)

0 = Idle

1 = Walk

2 = Run

Attack (trigger)

IsDead (bool)

👉 Compatible avec ton système actuel dans FPSWeaponManager / Animator

Player Detection 🎯

The AI looks for:

player transform

usually via tag or reference

Detection is based on:

distance

angle (FOV)

raycast (visibility)

Attack Behavior 💥

When the player is in range:

enemy stops moving

triggers attack animation

applies damage to player (via FPSHealth)

Death Handling 💀

Works with EnemyHealth (or equivalent):

enemy stops all behavior

animation switches to death state

AI is disabled

Quadruped Support 🐕

The system can be adapted for:

animals

creatures

quadrupeds

This mainly affects:

animations

movement style

👉 No hard dependency, flexible setup

Integration 🔗

With FPSHealth

Applies damage to the player.

With Animator

Handles movement and attack animations.

With Navigation

Typically uses NavMeshAgent for movement.

Debug 🪬

Common issues:

Enemy not moving → check NavMesh

Enemy not attacking → check attack range

Enemy stuck in attack → check state transitions

Enemy not detecting → check FOV / raycast

Typical Setup 🧱

Add EnemyAI

Add NavMeshAgent

Add Animator

Assign:

player target

patrol points

Configure:

speeds

detection range

attack settings

Best Use Cases 👛

guards

zombies

animals

NPC enemies

stealth gameplay

survival games

⚓ 💪 TEAM VICTOR & BORIS Note

EnemyAI is designed to be simple, flexible, and expandable.

It gives you:

a solid base AI

easy configuration

compatibility with RealFPS systems

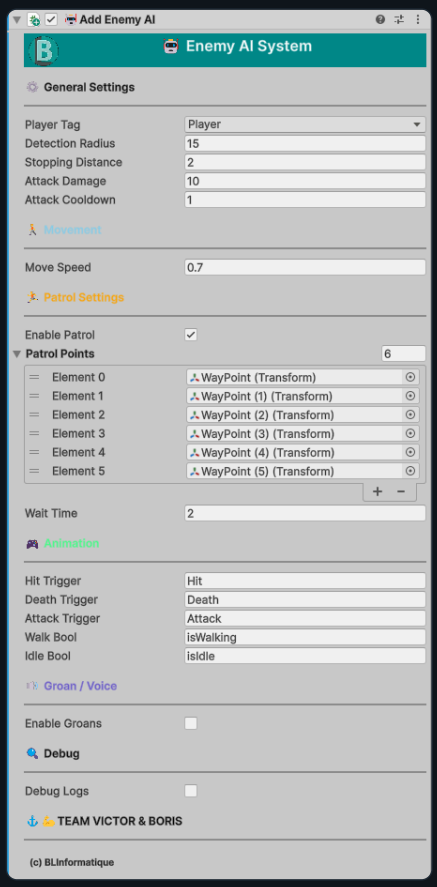
From there, you can build:

stealth AI

aggressive AI

patrol guards

wild animals



Field	Type	Tooltip
useAttackHitbox	`Boolean`	If true, damage uses a short OverlapSphere in front of the enemy. More reliable than direct GetComponent.
attackHitRadius	`Single`	Radius of the attack sphere in meters.
attackForwardOffset	`Single`	Forward offset in meters from the attack origin where the sphere is centered.
attackVerticalOffset	`Single`	Additional vertical offset applied to the attack sphere center.
damageMask	`LayerMask`	Layers checked for FPSHealth. Make sure player layers are included.
⬇		
stoppingDistanceExtra	`Single`	Extra margin added to stopping distance so the enemy can still hit when slightly short.
rotationSpeed	`Single`	Rotation speed used while turning toward the player.
faceTargetWhileAttacking	`Boolean`	If true, the enemy rotates toward the player while attacking at close range.
cancelAttackWhenOutOfRange	`Boolean`	If true, any pending attack is cancelled as soon as the player leaves attack range.
animLayer	`Int32`	Animator layer index for locomotion and attack.
attackStateName	`String`	Exact name of the Attack state as it appears in Animator.
attackDamageWindowStart	`Single`	Normalized attack damage window start. • Range [0..1]
attackDamageWindowEnd	`Single`	Normalized attack damage window end. • Range [0..1]
⬇		
drawAttackGizmo	`Boolean`	Draw the attack hitbox sphere in Scene view.
gizmoAlpha	`Single`	Color alpha for gizmo sphere preview. • Range [0,05..1]
⬇		
bodyType	`EnemyBodyType`	Defines the enemy body style. Humanoid is suited for humans and zombies. Quadruped is suited for four-legged animals.
bodyCenterOverride	`Transform`	Optional transform used as the body center reference for distance checks and gizmos. Leave empty to use this transform.
attackOrigin	`Transform`	Optional transform used as the attack origin. Recommended for quadrupeds, for example head or mouth.
fallbackAttackHeightOffset	`Single`	Fallback height offset used for the attack origin when no dedicated transform is assigned.
⬇		
playerTag	`String`	Tag used to detect the player target.

Field	Type	Tooltip
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.
↓		
patrolSpeed	`Single`	Base speed used while patrolling or when the enemy chases without running.
runWhenChasing	`Boolean`	If true, the enemy uses the chase speed when the player has been detected. If false, the enemy keeps walking toward the player.
chaseSpeed	`Single`	Speed used while chasing the player when running is enabled.
↓		
enablePatrol	`Boolean`	Enable patrol mode when the player is not detected.
patrolPoints	`Transform[]`	Waypoints the enemy will patrol between.
waitTime	`Single`	Time in seconds the enemy waits at each waypoint.
startPatrolOnStart	`Boolean`	If true, the enemy starts patrolling automatically on Start.
randomPatrol	`Boolean`	If true, the next patrol point is chosen randomly instead of sequentially.
avoidImmediateRepeat	`Boolean`	If true, the next patrol point cannot be the same as the current one in random mode.
patrolArrivalDistance	`Single`	Distance used to consider that the enemy has reached a patrol point.
repathIfPathLost	`Boolean`	If true, the patrol destination is re-applied automatically if the agent lost its path.
↓		
hitTrigger	`String`	Animator trigger name for hit animation.
deathTrigger	`String`	Animator trigger name for death animation.
attackTrigger	`String`	Animator trigger name for attack animation.
attackBool	`String`	Animator bool name used to mark the attack state.
walkBool	`String`	Animator bool name for walking state.
runBool	`String`	Animator bool name for running state while chasing.
idleBool	`String`	Animator bool name for idle state.
forceExitAttackState	`Boolean`	If true, the script forces the Animator to leave the attack state when attack is cancelled.

Field	Type	Tooltip
locomotionStateName	`String`	CrossFade target state used when forcing an exit from attack while moving.
idleStateName	`String`	CrossFade target state used when forcing an exit from attack while idle.
forceExitBlendDuration	`Single`	Blend duration used for CrossFade when forcing attack exit. • Range [0,01..0,5]
⬇		
enableGroans	`Boolean`	Enable enemy groans or 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 played randomly while idle.
idleVolume	`Single`	Volume used for idle groans. • Range [0..1]
idleInterval	`Vector2`	Random interval between idle groans in seconds.
⬇		
chaseGroans	`AudioClip[]`	Audio clips played randomly while chasing or attacking the player.
chaseVolume	`Single`	Volume used for chase groans. • Range [0..1]
chaseInterval	`Vector2`	Random interval between chase groans in seconds.
⬇		
debugLogs	`Boolean`	Enable debug logs in the Console.
⬇		
isAttacking	`Boolean`	True while the enemy is currently considered in attack state.
runtimePatrolIndex	`Int32`	Current patrol index.
runtimeInDetection	`Boolean`	True if player is currently inside detection range.
runtimeInAttackRange	`Boolean`	True if player is currently inside attack range.
runtimeIsRunning	`Boolean`	True if the enemy is currently using the running chase mode.
runtimeCurrentSpeed	`Single`	Current NavMeshAgent speed used at runtime.

↑ Back to top

Copy fields

📦 EnemyHealth MonoBehaviour

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

Enemy Health 💝💀

The EnemyHealth component handles all aspects of an enemy's health system in RealFPS.

It manages:

damage reception

hit reactions

death logic

visual and audio feedback

integration with other systems (AI, VFX, save system)

Purpose 🎯

Use this component to:

give enemies a health system

apply damage from weapons or gameplay

trigger hit effects

handle death animations

disable enemy behavior when dead

Core Settings ⚙️

Max Health

Maximum health value of the enemy.

Current Health

Runtime value updated when damage is applied.

Damage System 💥

Take Damage

Called by weapons, explosions, or scripts.

When damage is applied:

health is reduced

hit feedback is triggered

death is checked

Damage Sources 🔫

Typically called from:

FPSWeaponManager (shooting)

explosion systems

melee or interaction scripts

Hit Feedback 🎯

Hit Effects

Optional visual effects (particles, sparks, blood).

Hit Sound

Audio played when the enemy is hit.

Reaction Animation

Triggers animation such as:

flinch

hit reaction

Death System 💀

Is Dead

Runtime flag indicating the enemy is dead.

Once dead:

damage is ignored

AI is disabled

animations switch to death

Death Animation 🎬

Triggers Animator parameter:

IsDead (bool)

👉 Works directly with your Animator setup

Disable AI

The script automatically disables:

EnemyAI

movement / NavMesh

This prevents any further behavior after death.

Death Effects 🔥

Death FX

Optional particle effects spawned on death.

Death Sound

Audio played when the enemy dies.

Destroy / Disable Options 🧹

Destroy On Death

If enabled:

enemy GameObject is destroyed

Disable On Death

If enabled:

enemy is disabled instead of destroyed

👉 Recommended for Save System compatibility

Integration 🔗

With EnemyAI

AI stops when enemy dies

attack and movement are disabled

With FPSHealth

Enemy applies damage → Player loses health

With Save System

disabling instead of destroying allows restoration

keeps enemy state consistent

Runtime Flow ⚙️

When Taking Damage

Receive damage

Reduce health

Play hit feedback

Check for death

When Health Reaches Zero

Set isDead = true

Trigger death animation

Disable AI

Play death effects

Destroy or disable object

Debug 🐞

Common issues:

Enemy not dying → check maxHealth / damage values

Animation not playing → check Animator parameter

Enemy still moving → check EnemyAI disable

No hit feedback → check FX/audio references

Typical Setup 🧱

Add EnemyHealth

Set:

max health

hit effects

death effects

Link Animator

Combine with:

EnemyAI

NavMeshAgent

Best Use Cases 🧰

enemies

creatures

destructible NPCs

bosses

turrets

⚓ 💪 TEAM VICTOR & BORIS Note

EnemyHealth is the “truth” of your enemy.

EnemyAI decides what the enemy does

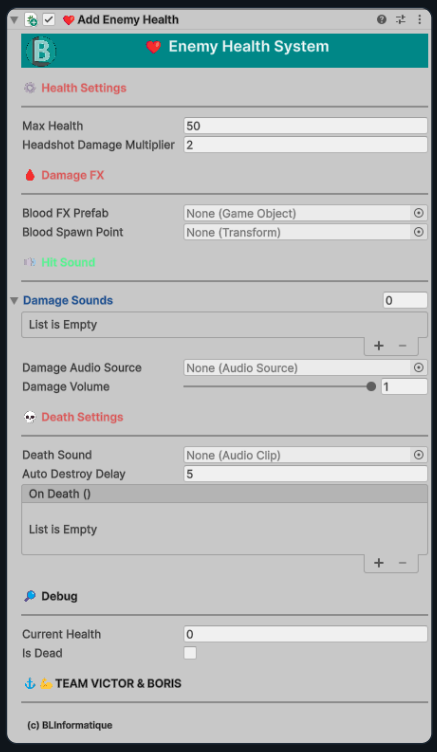
EnemyHealth decides how long it survives

Together:

➞ they create believable combat

Without it...

➞ your enemy is immortal 😅



Field	Type	Tooltip
missionManager	`Object`	Internal MissionManager reference resolved automatically when the optional addon is installed.
maxHealth	`Single`	Maximum health of this enemy.
headshotDamageMultiplier	`Single`	Damage multiplier applied when a headshot is detected.
bloodFXPrefab	`GameObject`	FX prefab spawned when the enemy takes damage.
bloodSpawnPoint	`Transform`	Optional transform used as the blood FX spawn point.
damageSounds	`AudioClip[]`	Optional audio clips played when the enemy takes damage.
damageAudioSource	`AudioSource`	Audio source used to play damage and death sounds.
damageVolume	`Single`	Volume used for damage sounds. • Range [0..1]
notifyMissionOnDeath	`Boolean`	If true, this enemy automatically notifies the mission system on death.
deathSound	`AudioClip`	Optional death sound played when the enemy dies.
deathCleanupDelay	`Single`	Delay in seconds before applying death cleanup. This allows the death animation to play.
disableCollidersOnDeath	`Boolean`	If true, all colliders on this enemy are disabled immediately on death.
disableBehavioursOnDeath	`Boolean`	If true, selected behaviours are disabled after the cleanup delay.
hideRenderersOnDeath	`Boolean`	If true, renderers are hidden after the cleanup delay.
disableObjectsOnDeath	`Boolean`	If true, objects in the list are disabled after the cleanup delay.
behavioursToDisableOnDeath	`Behaviour[]`	Optional behaviours to disable when the enemy dies. Do not include the Animator if you want the death animation to play.
objectsToDisableOnDeath	`GameObject[]`	Optional objects to disable when the enemy dies.
onDeath	`UnityEvent`	Events invoked when the enemy dies.
showDebugLogs	`Boolean`	If true, prints detailed debug logs in the Console.

Field	Type	Tooltip
currentHealth	`Single`	Current runtime health.
isDead	`Boolean`	True if this enemy is already dead.

↑ 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 🔗

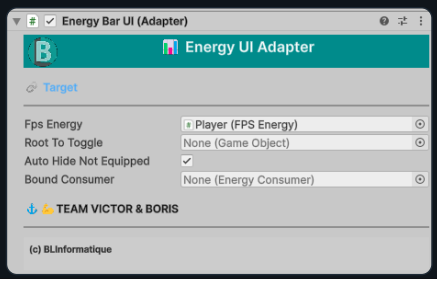
EnergyConsumer: runtime device interface/impl that owns energy, raises events.

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 🔗

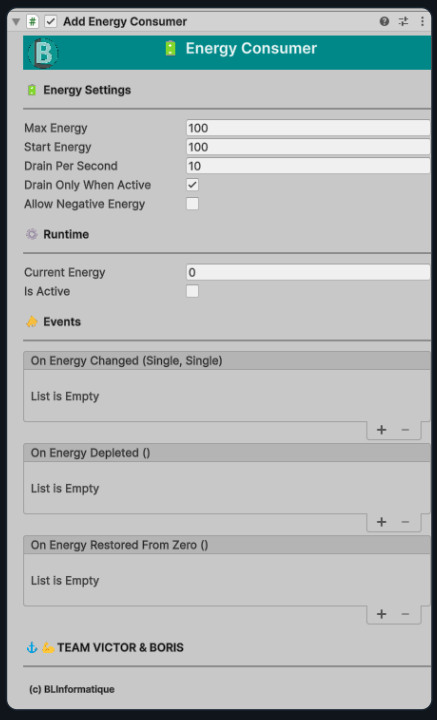
EnergyBarUI – Displays energy in UI.

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`

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.

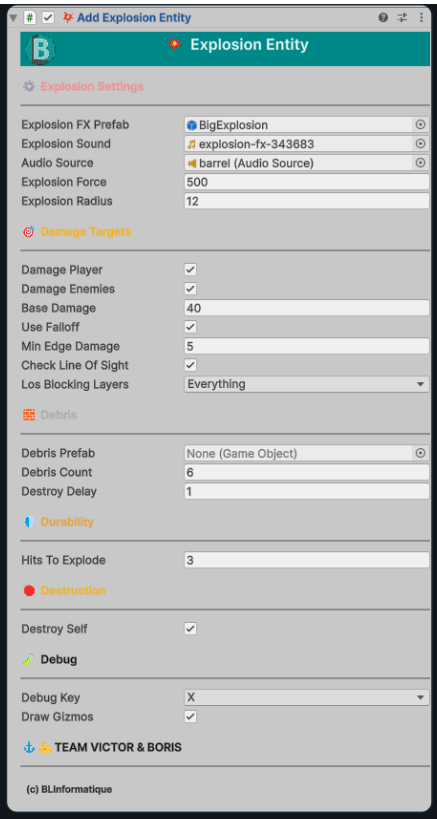
Use Check Line Of Sight when you want walls/cover to block damage.

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 spawned when the explosion occurs.
explosionSound	`AudioClip`	Sound played on explosion.
explosionForce	`Single`	Explosion force applied to rigidbodies.
explosionRadius	`Single`	Explosion radius.
explosionUpwardModifier	`Single`	Upward force modifier.
explosionLayers	`LayerMask`	Physics layers affected.
damagePlayer	`Boolean`	Damage applied to player.
damageEnemies	`Boolean`	Damage applied to enemies.
baseDamage	`Single`	Base damage at explosion center.
useFalloff	`Boolean`	Enable damage falloff.
minEdgeDamage	`Single`	Minimum damage at edge.
triggerNearbyExplosives	`Boolean`	Trigger nearby explosives.
instantChainExplosion	`Boolean`	Explode instantly nearby explosives.
hitsToExplode	`Int32`	Hits required before explosion.
destroyInsteadOfDisable	`Boolean`	If true, object is destroyed instead of disabled.
customDisableTarget	`GameObject`	Optional custom object to disable instead of root.
debugKey	`KeyCode`	Debug key to trigger explosion.
debugLogs	`Boolean`	Show debug logs.

↑ 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.

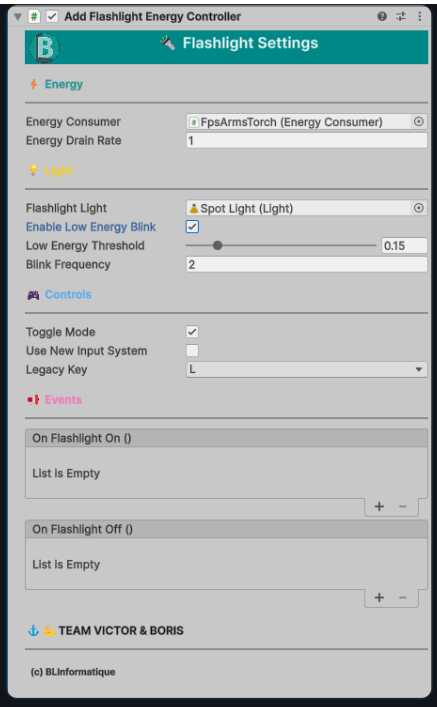
Use blink effect sparingly (avoid player frustration).

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

📦 FPSCarryManager MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Carry/👐 FPS Carry Manager`

FPS Carry Manager 👐 📦

The FPSCarryManager is the central system that handles object carrying in RealFPS.

It allows the player to:

pick up objects

carry them in front of the camera

display dedicated carry arms

drop or throw objects

modify movement behavior while carrying

It works in combination with:

CarryableObject

CarryArmsData

FPSWeaponManager

FPSControllerPro

Purpose 🎯

Use this component to create a full object carrying system where:

objects follow the player smoothly

arms visuals adapt to the object

gameplay is impacted (speed, weapons, etc.)

References 🔗

Player Camera

Used for:

positioning carried objects

throw direction

Custom Carry Anchor

Optional custom transform used as carry anchor.

↑

Arms Holder

Transform used to attach carry arms.

Weapon Manager

Used to unequip/restore weapons.

FPS Controller

Used to modify player movement while carrying.

Carry Mode 🧭

Carry Anchor Mode

Defines how the object is positioned:

CameraBased → follows camera directly

CustomAnchor → uses assigned transform

ArmsHolderAnchor → attaches to arms holder

👉 Default recommended: ArmsHolderAnchor

Runtime Anchor ⚙️

Auto Create Runtime Anchor

Creates a hidden anchor if none exists.

Runtime Anchor Name

Name of generated anchor.

Runtime Anchor Local Position

Default offset from parent.

Runtime Anchor Local Rotation

Default rotation.

Carry Arms Visual 🤲

Use Carry Arms Visual

If enabled, displays arms while carrying.

Carry Arms Prefab

Fallback prefab if no CarryArmsData is used.

Carry Arms Local Position

Fallback position offset.

Carry Arms Local Rotation

Fallback rotation offset.

Parent Carry Arms To ArmsHolder

If enabled, attaches arms to ArmsHolder.

Destroy Carry Arms On Drop

Automatically removes arms when dropping.

👉 This is the fallback system if no CarryArmsData is assigned

Carry Arms Sway 🌊

Apply Sway To Carry Arms

If enabled, arms move with object sway.

Position Multiplier

Controls sway intensity.

Rotation Multiplier

Controls rotation sway.

Extra Rotation X / Z

Adds additional motion.

Carry Behaviour ⚙

Max Carry Distance

If exceeded:

➞ object is automatically dropped

Unequip Weapon While Carrying

Disables current weapon.

Restore Weapon After Drop

Re-equips previous weapon.

Reduce Move Speed While Carrying

Slows the player.

Carry Walk Speed Multiplier

Defines speed reduction.

Disable Run While Carrying

Prevents sprinting.

Force Follow Every Frame

Ensures object sticks to anchor (LateUpdate).

Drop / Throw 🗑

Drop Key

Key used to drop object.

Throw Key

Key used to throw object.

Throw Force

Forward force applied.

Throw Upward Force

Adds vertical motion.

Drop Forward Impulse

Small push even on normal drop.

Input Safety 🛡️

Input Block After Pickup

Delay before drop input is allowed.

Require Input Release After Pickup

Prevents accidental instant drop.

New Input System 🎮

Use New Input System

Switch between old/new input systems.

Drop Action

InputAction for drop.

Throw Action

InputAction for throw.

Runtime Fields 🧠

Current Carried Object

Reference to the carried object.

Active Carry Anchor

Current anchor used.

Current Carry Arms Instance

Instantiated arms visual.

Is Carrying

True if player is carrying.

Last Pickup Time

Used for input safety.

Waiting Input Release

Prevents accidental drop.

Current Carry Arms Data

Resolved from object or fallback.

Core Workflow ⚙️

Pickup 📦

When picking an object:

Validate object (CarryableObject)

Resolve anchor

Attach object to anchor

Disable collisions with player

Unequip weapon (optional)

Spawn carry arms visual

Apply movement modifiers

Carry 🔁

During carry:

object follows anchor

sway is applied

distance is monitored

input is checked

Drop / Throw 🗑️

When dropping:

Restore collisions

Apply velocity

Remove carry arms

Restore movement

Restore weapon

Integration 🔗

With CarryableObject

Defines object behavior and offsets.

With CarryArmsData

Overrides arms visuals per object.

With FPSWeaponManager

Handles weapon state.

With FPSControllerPro

Controls player movement.

Typical Setup 🧱

Add FPSCarryManager to player

Assign:

camera

armsHolder

Configure:

carry mode

movement modifiers

Add CarryableObject to objects

(Optional) assign CarryArmsData

Best Use Cases 🧰

crates

mission cargo

physics objects

puzzle objects

throwable props

⚓ 💪 TEAM VICTOR & BORIS Note

FPSCarryManager is the brain of the carry system.

CarryableObject = what you carry

CarryArmsData = how it looks

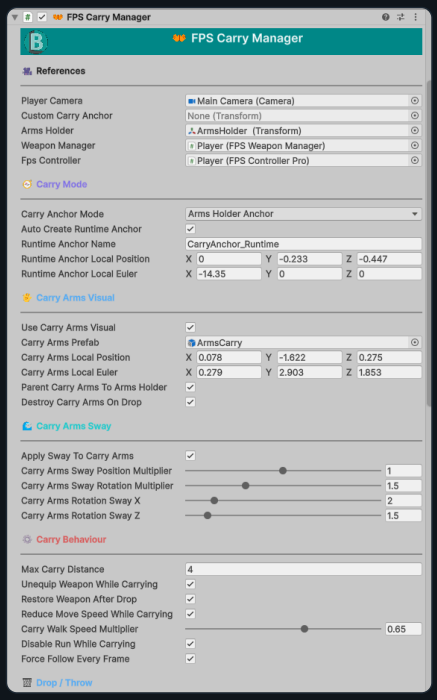
FPSCarryManager = how it behaves

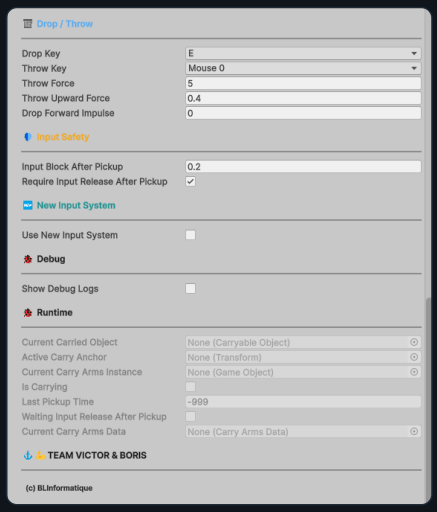
Without it:

➞ objects don't follow

➞ arms don't exist

➞ gameplay breaks





Field	Type	Tooltip
playerCamera	`Camera`	Main FPS camera used for carry positioning and throw direction.
customCarryAnchor	`Transform`	Optional custom anchor used when CarryAnchorMode = CustomAnchor.
armsHolder	`Transform`	Optional arms holder used when CarryAnchorMode = ArmsHolderAnchor.
weaponManager	`FPSWeaponManager`	Optional weapon manager. If empty, auto-resolved on Awake.
fpsController	`FPSControllerPro`	Optional FPS controller. If empty, auto-resolved on Awake.
carryAnchorMode	`CarryAnchorMode`	Choose where carried objects are attached.
autoCreateRuntimeAnchor	`Boolean`	If true, a hidden runtime anchor is created when needed.
runtimeAnchorName	`String`	Name of the generated runtime carry anchor.
runtimeAnchorLocalPosition	`Vector3`	Default local position of the runtime anchor.
runtimeAnchorLocalEuler	`Vector3`	Default local rotation of the runtime anchor.
useCarryArmsVisual	`Boolean`	If true, a dedicated carry arms prefab is shown while carrying an object.
carryArmsPrefab	`GameObject`	Fallback prefab instantiated in the arms holder while carrying.
carryArmsLocalPosition	`Vector3`	Fallback local position offset applied to the carry arms instance.
carryArmsLocalEuler	`Vector3`	Fallback local rotation offset applied to the carry arms instance.
parentCarryArmsToArmsHolder	`Boolean`	If true, the fallback carry arms visual is parented to ArmsHolder.
destroyCarryArmsOnDrop	`Boolean`	If true, the carry arms visual is automatically destroyed when dropping the object.
applySwayToCarryArms	`Boolean`	If true, the carry arms visual receives the same sway motion as the carried object.
carryArmsSwayPositionMultiplier	`Single`	Multiplier applied to sway position on carry arms. • Range [0..2]
carryArmsSwayRotationMultiplier	`Single`	Multiplier applied to sway rotation on carry arms. • Range [0..5]
carryArmsRotationSwayX	`Single`	Extra rotation sway on X axis for carry arms. • Range [0..15]

Field	Type	Tooltip
carryArmsRotationSwayZ	`Single`	Extra rotation sway on Z axis for carry arms. • Range [0..15]
⬇		
maxCarryDistance	`Single`	Maximum distance allowed between camera and carried object before forced drop.
unequipWeaponWhileCarrying	`Boolean`	If true, the weapon is unequipped while carrying.
restoreWeaponAfterDrop	`Boolean`	If true, previous weapon is restored when dropping the carried object.
reduceMoveSpeedWhileCarrying	`Boolean`	If true, the player walk speed is reduced while carrying.
carryWalkSpeedMultiplier	`Single`	Walk speed multiplier applied while carrying. • Range [0,1..1]
disableRunWhileCarrying	`Boolean`	If true, running is disabled while carrying.
forceFollowEveryFrame	`Boolean`	If true, the carried object is forced every LateUpdate to the carry anchor world transform.
⬇		
dropKey	`KeyCode`	Drop key used with the old Input System.
throwKey	`KeyCode`	Throw key used with the old Input System.
throwForce	`Single`	Forward throw force.
throwUpwardForce	`Single`	Upward throw force added on top of the forward throw.
dropForwardImpulse	`Single`	Optional additional release speed applied even on normal drop.
⬇		
inputBlockAfterPickup	`Single`	Minimum delay after pickup before drop/throw input is accepted.
requireInputReleaseAfterPickup	`Boolean`	If true, drop input is ignored until the pickup key/button is released.
⬇		
useNewInputSystem	`Boolean`	Enable the new Input System path.
dropAction	`InputAction`	Input Action used to drop the currently carried object.
throwAction	`InputAction`	Input Action used to throw the currently carried object.
⬇		
showDebugLogs	`Boolean`	Enable debug logs in Console.
⬇		
currentCarriedObject	`CarryableObject`	Currently carried object.

Field	Type	Tooltip
activeCarryAnchor	`Transform`	Runtime anchor currently used by the carry system.
currentCarryArmsInstance	`GameObject`	Current instantiated carry arms visual.
isCarrying	`Boolean`	True while the player is carrying an object.
lastPickupTime	`Single`	Time when the last pickup happened.
waitingInputReleaseAfterPickup	`Boolean`	True while waiting for input release after pickup.
currentCarryArmsData	`CarryArmsData`	Current carry arms data resolved from the carried object or fallback.

↑ 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.

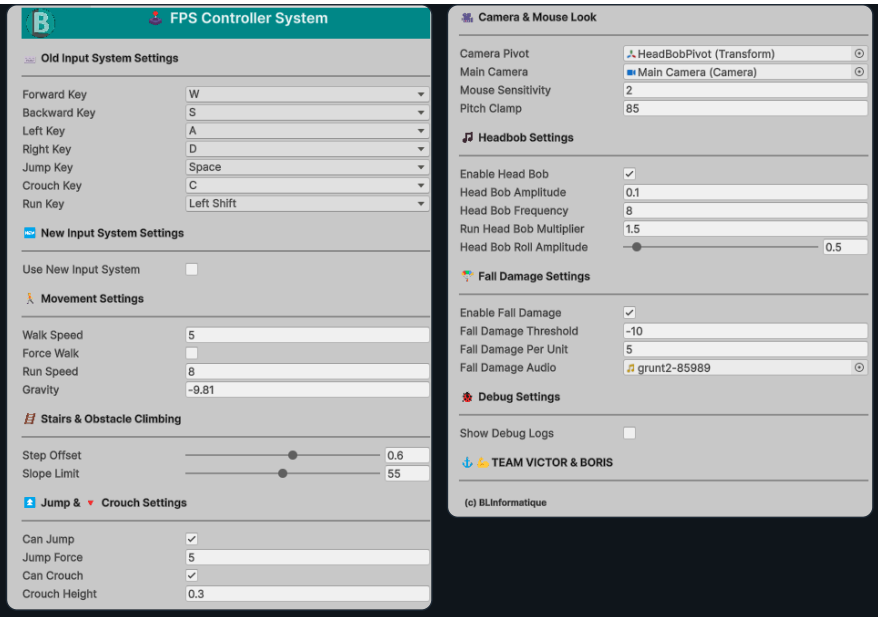
For survival gameplay, combine with FPSFood + FPSEnergy + **FPSHealth`.

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).
mainCamera	Camera	Camera component.
mouseSensitivity	Single	Mouse sensitivity.

Field	Type	Tooltip
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.

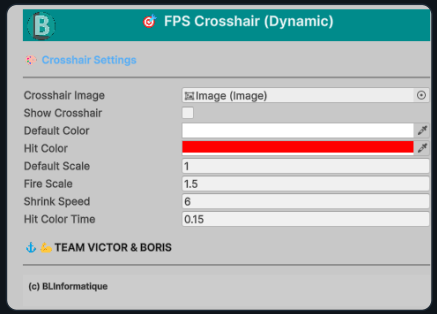
Use subtle Fire Scale (1.2–1.5) for feedback without distraction.

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.

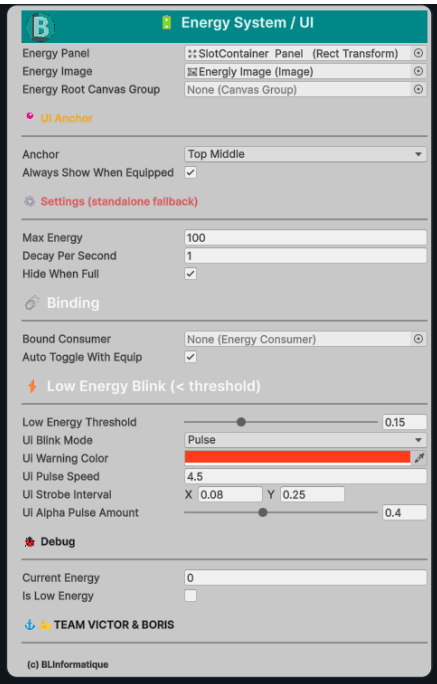
EnergyBarUI – Adapter that binds EnergyConsumer → FPSEnergy.

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.

↑


Energy System / UI
Energy Panel
SlotContainer Panel (Rect Transform)
Energy Image
Energy Image (Image)
Energy Root Canvas Group
None (Canvas Group)
UI Anchor
Anchor
Top Middle
Always Show When Equipped
Settings (standalone fallback)
Max Energy
100
Decay Per Second
1
Hide When Full
Binding
Bound Consumer
None (Energy Consumer)
Auto Toggle With Equip
Low Energy Blink (< threshold)
Low Energy Threshold
0.15
Ui Blink Mode
Pulse
Ui Warning Color
Ui Pulse Speed
4.5
Ui Strobe Interval
X 0.08
Y 0.25
Ui Alpha Pulse Amount
0.4
Debug
Current Energy
0
Is Low Energy
TEAM VICTOR & BORIS
(c) BLInformatique

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.

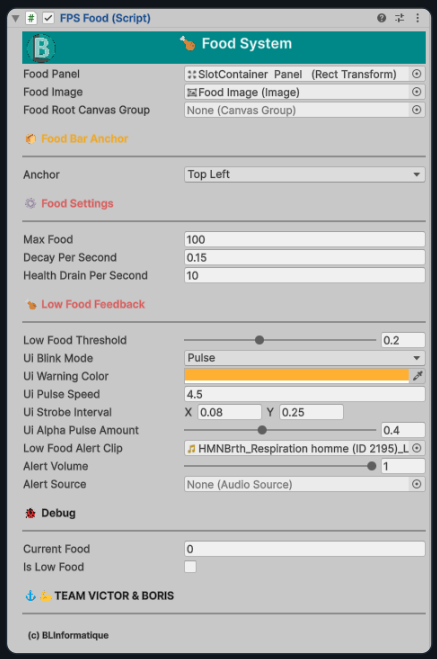
Use Pulse for subtle warning, Strobe + Sound for urgent state.

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.

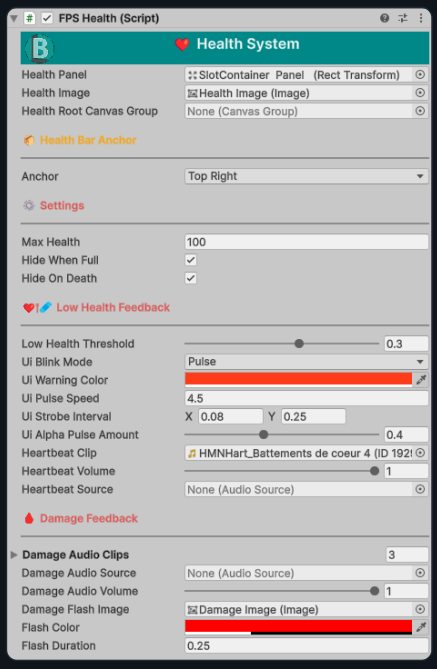
Always provide a Death Panel with “Restart” option if not using auto-respawn.

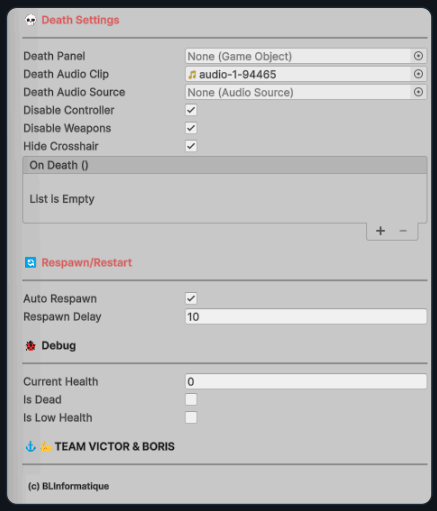
Balance fallDamage (FPSControllerPro) and health regen/healing to fit game pacing.

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).
disableController	Boolean	Disable FPSController on death.
disableWeapons	Boolean	Disable FPSWeaponManager on death.
hideCrosshair	Boolean	Hide crosshair on death.

Field	Type	Tooltip
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).

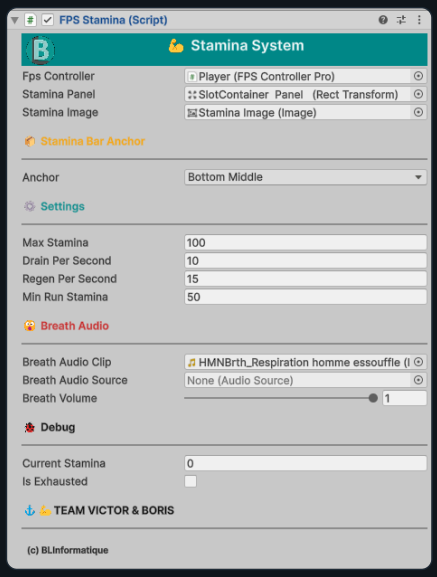
Use breathing audio sparingly to avoid annoyance.

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

FPS Weapon Manager 🔫🎯

The FPSWeaponManager is the central system that controls all weapon behavior in RealFPS.

It handles:

weapon equip / unequip

shooting logic

reloading

animations

ammo management

energy-based weapons

integration with inventory and UI

Purpose 🎯

Use this component to:

equip and switch weapons

handle firing and reloading

manage ammo and energy

control weapon animations

connect weapons to UI and gameplay systems

Core Concept 🧠

The system is built around:

ArmsAndWeaponData (ScriptableObject) → defines weapon behavior

WeaponInventoryManager → stores weapons

FPSWeaponManager → executes gameplay logic

👉 It acts as the runtime controller of the equipped weapon

Weapon Handling 🔫

Equip Weapon

Equips a weapon from:

inventory

script

startup logic

Unequip Weapon

Removes the current weapon.

Restore Previous Equipment

Used when exiting carry mode or other systems.

👉 Important for integration with FPSCarryManager

Shooting System 💥

Fire Input

Triggered by:

old Input System

new Input System

Fire Modes

Supports:

single shot

automatic

burst fire

Shooting Flow

Check fire conditions

Check ammo or energy

Play animation

Play sound

Spawn projectile or raycast

Apply damage

Ammo System 🔋

Magazine Ammo

Current bullets in magazine.

Reserve Ammo

Total available ammo.

Reload

Transfers ammo from reserve to magazine.

Reload Flow

Check if reload is possible

Play reload animation

Update ammo values

Energy System ⚡

Supports weapons that use energy instead of ammo.

Uses:

EnergyConsumer

FPSEnergy UI

Behavior

energy decreases on use

stops firing when empty

can be restored (battery, etc.)

Animation System 🎬

Works with Animator.

Typical parameters:

Fire

Reload

IsHoldingWeapon

Arms Integration

Works with:

weapon arms

ADS transitions

IK system

ADS (Aim Down Sight) 🎯

Right Click / Input

Switches between:

Idle

ADS

Behavior

changes arms position

adjusts weapon alignment

can affect accuracy

Input System 🎮

Old Input System

Uses:

Input.GetKey

mouse buttons

New Input System

Uses:

InputAction

👉 Fully compatible with Unity 6000 Input System

Integration 🔗

With WeaponInventoryManager

Handles weapon switching and ammo persistence.

With FPSEnergy

Displays energy for energy-based weapons.

With FPSCarryManager

weapon is unequipped when carrying

restored when dropping object

With EnemyHealth

Applies damage to enemies.

Runtime Flow ⚙️

Equip

Load weapon data

Spawn or activate weapon

bind energy system (if needed)

update UI

Fire

check input

check ammo/energy

play animation

apply damage

Reload

check ammo

play animation

refill magazine

Debug 🪲

Common issues:

Weapon not firing → check input setup

No damage → check raycast / damage logic

No animation → check Animator parameters

Ammo not updating → check inventory sync

Typical Setup 🧱

Add FPSWeaponManager to player

Assign:

Animator

weapon holder

Create ArmsAndWeaponData

Add weapons via WeaponInventoryManager

Configure input

Best Use Cases 🧰

FPS games

survival games

shooters

tactical gameplay

energy weapons

⚓ 💪 TEAM VICTOR & BORIS Note

FPSWeaponManager is the backbone of combat in RealFPS.

It connects:

input

animations

ammo

damage

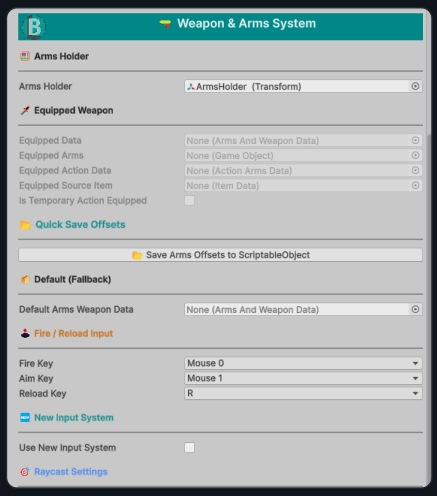
UI

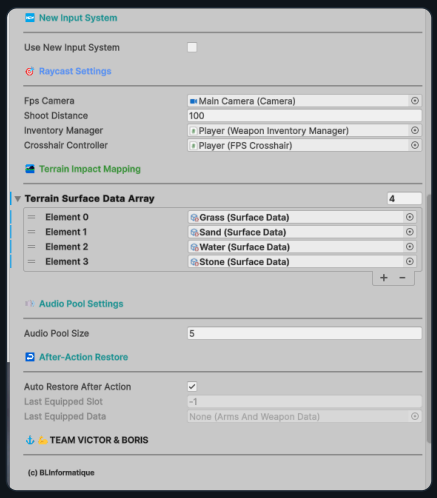
Without it:

➞ no shooting

➞ no weapons

➞ no fun





Field	Type	Tooltip
⬇		
armsHolder	`Transform`	Parent transform where arms and weapon visuals are instantiated.
⬇		
equippedData	`ArmsAndWeaponData`	Currently equipped arms and weapon data.
equippedArms	`GameObject`	Currently equipped arms instance.
equippedActionData	`ActionArmsData`	Currently equipped temporary action arms data.
equippedSourceItem	`ItemData`	Currently equipped source inventory item.
isTemporaryActionEquipped	`Boolean`	True if current equipment comes from a temporary action item.
⬇		
boutonDummy	`Int32`	Click to save the current arms prefab position and rotation in the ScriptableObject. Play Mode only.
⬇		
defaultArmsWeaponData	`ArmsAndWeaponData`	Default arms and weapon data used as fallback when unequipping.
⬇		
fireKey	`KeyCode`	Fire key when using the old Input System.
aimKey	`KeyCode`	Aim key when using the old Input System.
reloadKey	`KeyCode`	Reload key when using the old Input System.
⬇		
useNewInputSystem	`Boolean`	Enable the New Input System actions.
fireAction	`InputAction`	Fire action for the New Input System.
aimAction	`InputAction`	Aim action for the New Input System.
reloadAction	`InputAction`	Reload action for the New Input System.
⬇		
fpsCamera	`Camera`	Camera used for shooting raycasts.
shootDistance	`Single`	Default shoot distance if weapon data does not override it.
inventoryManager	`WeaponInventoryManager`	Reference to WeaponInventoryManager for

Field	Type	Tooltip
		ammo management.
crosshairController	`FPSCrosshair`	Reference to the crosshair controller UI.
terrainSurfaceDataArray	`SurfaceData[]`	SurfaceData array matching the terrain texture order. Used for bullet impact FX and sounds on Terrain.
audioPoolSize	`Int32`	Number of pooled AudioSources used for fire and reload sounds.
autoRestoreAfterAction	`Boolean`	If true, after a temporary one-shot action the previous weapon is automatically re-equipped.
lastEquippedSlot	`Int32`	Last selected weapon slot before temporary action.
lastEquippedData	`ArmsAndWeaponData`	Fallback weapon data used 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.

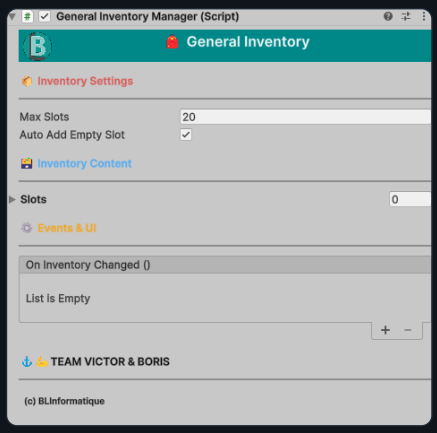
Keep UseItem() as the central usage flow: don't bypass it in gameplay.

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.

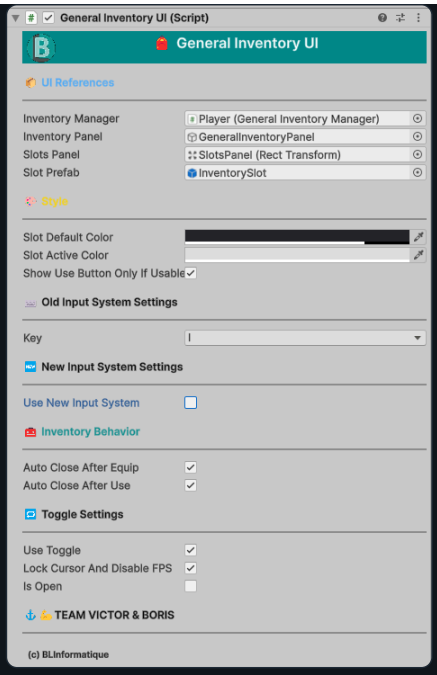
Keep autoCloseAfterEquip/use enabled for smoother gameplay.

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.
itemDropper	`ItemDropper`	Reference to the item drop system.
inventoryPanel	`GameObject`	Main inventory panel.
slotsPanel	`Transform`	Parent panel for inventory slots (use a GridLayoutGroup).
slotPrefab	`GameObject`	Prefab for a single slot (icon + quantity + buttons).
⬇		
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.
autoCloseAfterDrop	`Boolean`	If enabled, the inventory UI will close automatically after dropping an item.
⬇		
useToggle	`Boolean`	If true, pressing the key toggles between open/close inventory.
lockCursorAndDisableFPS	`Boolean`	Lock/unlock cursor and disable player controls 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.

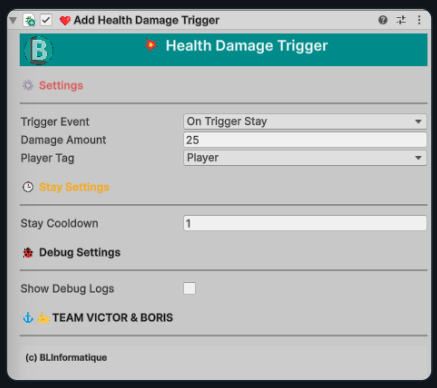
Use OnTriggerEnter for one-time hazards (spike traps).

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`

Interaction Raycaster 🖱️🎯

The InteractionRaycaster is the core interaction system in RealFPS.

It uses a center-screen raycast to detect objects in front of the player and allows interaction through a unified system.

It works with:

IInteractable interface

ItemPickup

CarryableObject

DoorLockInteractable

PickupUIManager

Purpose 🎯

Use this component to:

detect interactable objects in front of the player

display contextual interaction prompts

trigger interactions using a single input key

unify all interactions under one system

Core Concept 🧠

The system casts a ray from the center of the screen:

Camera → Raycast → Collider → IInteractable

If a valid object is hit:

a prompt is displayed

interaction is enabled

Raycast Settings 🎥

Main Camera

Camera used for raycasting.

Ray Distance

Maximum interaction distance.

Interact Mask

LayerMask used to filter interactable objects.

Auto Exclude Weapon Layer

Automatically removes the "Weapon" layer from detection.

👉 Prevents interaction with your own weapon model

Hit Detection 🔎

Search In Parents

Find interactable components in parent objects.

Search In Children

Find interactable components in children.

Require Interactable For Prompt

Only show prompt if a valid interactable is found.

Input System 🎮

Old Input System

Uses:

KeyCode interactKey (default: E)

New Input System

Use New Input System

Enables InputAction support.

Interact Action

InputAction used for interaction.

UI Prompt 🖥️

Prompt Template

Default message:

Press {key} to interact

Auto Hide Prompt

Hides prompt when no object is detected.

Use Contextual Prompt

Builds dynamic messages:

Examples:

Press E to pick up Battery

Press E to open Door

Press E to carry Crate

Default Verb

Fallback verb if none is defined.

Contextual Interaction 🧠

The system automatically adapts based on object type:

ItemPickup

→ "pick up"

CarryableObject

→ "carry"

DoorLockInteractable

→ "open"

Toggle systems

→ "toggle"

👉 Fully dynamic and user-friendly

Behaviour ⚙️

Cone Angle

Optional angle tolerance for interaction.

0 = strict raycast

0 = interaction cone

Ignore Already Carried Objects

Prevents interaction with objects already held.

Ignore Already Picked Pickups

Prevents re-interacting with collected items.

References 🔗

Player Inventory

Used when calling Interact().

If empty:

➞ auto-resolved from player

Runtime Flow ⚙️

Detection

Raycast from camera center

Check hit object

Resolve interactable component

Validate interaction

Prompt Display

Build message

Replace {key}

Show via PickupUIManager

Interaction

Player presses key

Call Interact()

Object handles its own logic

Interaction System 🔗

IInteractable Interface

void Interact(GeneralInventoryManager playerInventory);

Any object implementing this can be used by the system.

Supported Objects 🧰

ItemPickup → pickup items

CarryableObject → carry objects

DoorLockInteractable → open doors

custom interactables

UI Integration 🖥️

Uses:

👉 PickupUIManager

displays prompt text

handles visibility

prevents overlap

Inventory Interaction 👜

Interaction can:

add items to inventory

consume keys

trigger gameplay logic

Debug 🪲

Common issues:

No prompt → check ray distance / layer mask

Wrong text → check interaction verb

No interaction → check IInteractable implementation

Interacting through UI → check inventory open state

👉 The script automatically disables interaction when inventory is open

Typical Setup 🧱

Add InteractionRaycaster to player

Assign:

camera

inventory

Configure:

distance

layer mask

Add IInteractable scripts to objects

Best Use Cases 🧰

item pickup

door interaction

carrying system

switches / toggles

mission objects

⚓ 💪 TEAM VICTOR & BORIS Note

InteractionRaycaster is the glue of RealFPS gameplay.

It connects:

player view

world objects

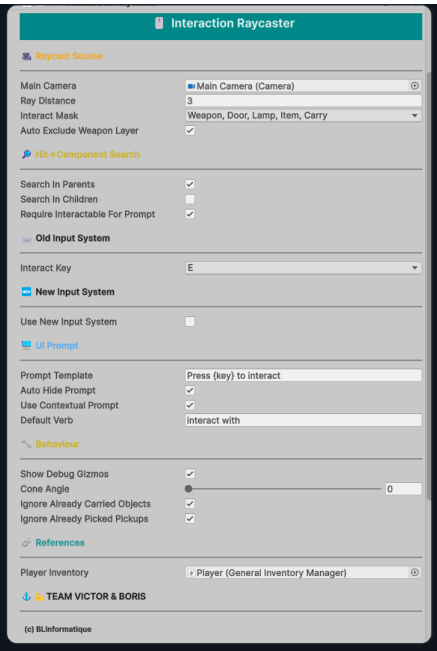
UI

gameplay systems

Without it:

➞ no interaction

➞ no immersion


Interaction Raycaster
Raycast Source
Main Camera
Main Camera (Camera)
Ray Distance
3
Interact Mask
Weapon, Door, Lamp, Item, Carry
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
Use Contextual Prompt
Default Verb
interact with
Behaviour
Show Debug Gizmos
Cone Angle
0
Ignore Already Carried Objects
Ignore Already Picked Pickups
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.
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`	Fallback 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.
useContextualPrompt	`Boolean`	If true, builds a contextual prompt such as 'Press E to pick up Battery'.
defaultVerb	`String`	Fallback interaction verb when no specific verb is resolved.
⬇		
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]
ignoreAlreadyCarriedObjects	`Boolean`	If true, objects already carried by FPSCarryManager will not show any prompt.

Field	Type	Tooltip
ignoreAlreadyPickedPickups	`Boolean`	If true, already picked pickups will not show any prompt.
⬇		
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).

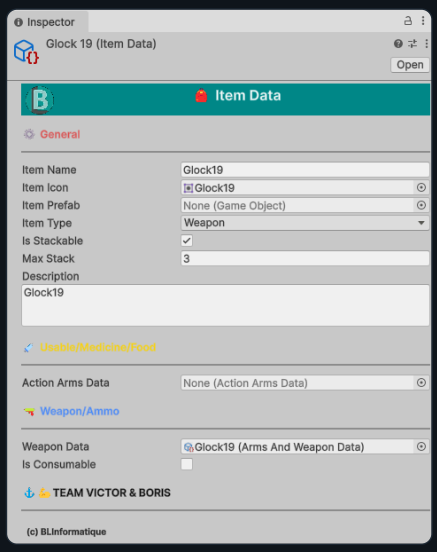
Keep keys non-consumable unless you want them single-use.

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.).
⬇		
keyId	`String`	Unique key ID for unlocking doors or objects.
consumeOnUse	`Boolean`	If true, the key is consumed when used.

Field	Type	Tooltip
isConsumable	`Boolean`	Can only be used once. If true, item will be consumed after use.

↑ Back to top

Copy fields

📦 ItemDropper MonoBehaviour

Item Dropper 📤 📦

The ItemDropper component handles dropping items from the player's inventory into the game world.

It allows the player to:

drop items from gameplay input

drop items directly from the inventory UI

spawn physical objects in the world

control drop direction, force, and quantity

Purpose 🎯

Use this component to:

convert inventory items into world objects

create a consistent drop system

support both gameplay and UI-driven drops

maintain correct quantities for stackable items

Core Concept 🧠

The system works like this:

Inventory Item → Remove from Inventory → Spawn Prefab → Apply Physics

References 👛

Inventory

Reference to GeneralInventoryManager.

Used to:

retrieve items

remove items when dropped

Player Camera

Used to:

↑

determine drop direction

position the spawned object

Inventory UI

Used to:

block drop when inventory is open (gameplay input)

Weapon Manager

Used to:

handle weapon drop behavior when dropping equipped items

Drop Settings ⚙️

Drop Distance

Distance in front of the player where the item spawns.

Drop Force

Force applied to the dropped object.

Spawn Height Offset

Prevents clipping with the ground.

Align To Ground

If enabled:

object aligns with surface normal

Use Camera Forward

If enabled:

drop direction follows camera

Otherwise:

uses player forward direction

Quantity System 📊

Default Drop Amount

Defines how many items are dropped at once.

Stackable Items

respects ItemData.isStackable

clamps amount to available quantity

Non-Stackable Items

always drops 1 item

Input System 🎮

Old Input System

Drop Key

Default: G

New Input System

Use New Input System

Enables InputAction support.

Drop Action

InputAction for drop.

Drop Methods 🧩

DropLastItem()

Drops the last item in inventory.

👉 Used for gameplay input

DropItem(ItemData item)

Drops a specific item via code.

respects inventory UI state

DropItemFromUI(ItemData item)

Drops an item directly from UI.

👉 Important:

works even when inventory is open

Internal Logic ⚙️

Safe Drop Amount

if not stackable → 1

if stackable → clamp(defaultAmount, available)

Drop Flow

Validate item

Remove from inventory

Spawn prefab

Position in front of player

Apply force

Configure runtime pickup

Runtime Pickup Compatibility 🔄

When an item is dropped:

it spawns a prefab with ItemPickup

amount is configured using:

ConfigureRuntimePickup(itemData, amount)

👉 This prevents duplication bugs (important fix 👍)

Weapon Integration 🔫

When dropping a weapon:

FPSWeaponManager is notified

weapon visuals are removed if equipped

Save System Compatibility 💾

The system ensures:

dropped items have correct identity

items can be restored correctly

👉 Important for persistence

Interaction 🔗

Dropped items automatically:

become ItemPickup

can be picked up again

respect quantity

Runtime Flow ⚙️

Gameplay Drop

Press drop key

Check inventory state

Drop last item

UI Drop

Click drop in inventory

Drop specific item

Ignore UI blocking

Debug 🐞

Common issues:

Item not spawning → check prefab in ItemData

Wrong quantity → check stack settings

Item duplicates → check runtimeConfiguredAmount

Cannot drop → check inventory UI state

Typical Setup 🧱

Add ItemDropper to player

Assign:

inventory

camera

Configure:

drop distance

drop force

Ensure ItemData has:

prefab

correct type

Best Use Cases 👜

survival games

loot systems

crafting systems

inventory management

sandbox gameplay

⚓ 💪 TEAM VICTOR & BORIS Note

ItemDropper is what makes your inventory feel real.

Without it:

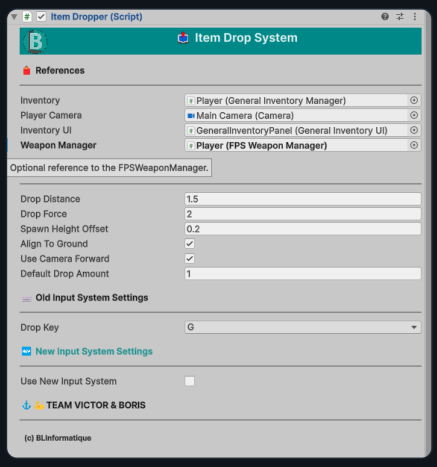
➞ items are just UI icons

With it:

➞ items exist in the world

➞ can be dropped, picked, thrown

➞ gameplay becomes tangible 😍



Field	Type	Tooltip
⬇		
inventory	`GeneralInventoryManager`	Reference to the GeneralInventoryManager.
playerCamera	`Camera`	Camera used to determine drop direction.
inventoryUI	`GeneralInventoryUI`	Optional reference to inventory UI.
weaponManager	`FPSWeaponManager`	Optional reference to the FPSWeaponManager.
⬇		
dropDistance	`Single`	Distance in front of the player where the item will be dropped.
dropForce	`Single`	Force applied to the dropped item.
spawnHeightOffset	`Single`	Height offset to avoid ground clipping.
alignToGround	`Boolean`	If true, item will align with ground normal.
useCameraForward	`Boolean`	If true, use camera forward. Otherwise uses player forward.
defaultDropAmount	`Int32`	Default quantity dropped per action for stackable items.
⬇		
dropKey	`KeyCode`	Key used to drop the last item in inventory when using the old Input System.
⬇		
useNewInputSystem	`Boolean`	Enable this if using the new Input System.
dropAction	`InputAction`	Input action used to drop the last item in inventory when using the new Input System.

↑ Back to top

Copy fields

📦 ItemPickup MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Pickup/📦 Add Item Pickup`

Item Pickup 📦👛

The ItemPickup component allows objects in the world to be collected and added to the player's inventory.

It supports multiple pickup methods and integrates seamlessly with:

InteractionRaycaster

GeneralInventoryManager

ItemDropper

PickupUIManager

optional Mission System

Purpose 🎯

Use this component to:

create collectible items in the world

add items to the inventory

support multiple pickup methods

display pickup prompts

integrate with mission objectives

work with save/load systems

Core Concept 🧠

World Object → Pickup → Inventory → Drop → World Object

This creates a complete gameplay loop between world and inventory.

Pickup Settings ⚙️

Player Tag

Tag used to detect the player.

Item Data

Reference to ItemData.

Defines:

item type

icon

prefab

behavior

Amount

Number of items given when picked.

Pickup Modes 🔁

Pickup By Trigger

Automatically picks up the item when entering the trigger.

Pickup By Key

Uses a local key input.

Pickup By Interaction Raycaster

Uses the global interaction system.

👉 Recommended mode for RealFPS

Input System 🎮

Old Input System

Pickup Key

Default: E

New Input System

Use New Input System

Enable InputAction.

Pickup Action

InputAction for pickup.

Prompt System 🖥️

Prompt Message

Default:

Press {key} to pick up

Show Local Prompt

If enabled:

displays prompt using PickupUIManager

Interaction Verb

Custom verb used by InteractionRaycaster.

Examples:

"pick up"

"take"

"grab"

Mission System 🧭

Notify Mission On Pickup

If enabled:

notifies mission system when collected

Mission Manager (Runtime)

Automatically resolved if addon is installed.

👉 Fully compatible but optional

Save-Friendly Behavior 💾

Disable Instead Of Destroy

If enabled:

object is disabled instead of destroyed

👉 Required for Save System compatibility

Disable Target

Object to disable (optional override).

Is Picked Up

Runtime flag.

Runtime Drop Compatibility 🔁

Runtime Configured Amount

Used when item is spawned via ItemDropper.

Prevents duplication bugs.

ConfigureRuntimePickup()

ConfigureRuntimePickup(itemData, amount)

Overrides default prefab values.

Feedback 🎧✨

Pickup Sounds

Random sound played on pickup.

Pickup FX Prefab

Visual effect spawned when collected.

FX Duration

Lifetime of the effect.

Runtime Fields 🐞

Is Player Nearby

True when player is in trigger.

Interaction 🔗

IInteractable Interface

void Interact(GeneralInventoryManager playerInventory);

Used by InteractionRaycaster

Runtime Flow ⚙️

Trigger Pickup

Player enters trigger

Check tag

Add item to inventory

Disable or destroy object

Key Pickup

Player near object

Prompt displayed

Player presses key

Item is collected

Interaction Raycaster Pickup

Player looks at object

Prompt displayed

Player presses interact

Item is collected

State Management 🔄

SetPickedState()

Handles:

enabling/disabling object

restoring colliders

resetting state

Used by:

save/load system

UI Integration 🖥️

Uses:

👉 PickupUIManager

shows prompt

hides prompt when needed

Inventory Integration 👛

Adds items to:

👉 GeneralInventoryManager

Handles:

stacking

quantities

item types

Debug 🐞

Common issues:

Item not picked → check tag

No prompt → check InteractionRaycaster

Duplicate items → check runtimeConfiguredAmount

Item not reappearing → check disableInsteadOfDestroy

Typical Setup 🧱

Add ItemPickup to object

Assign:

ItemData

amount

Add:

Collider (trigger)

Configure pickup mode

Best Use Cases 💼

loot items

consumables

keys

ammo

weapons

mission items

⚓ 💪 TEAM VICTOR & BORIS Note

ItemPickup is the bridge between the world and the inventory.

Without it:

➞ items are just objects

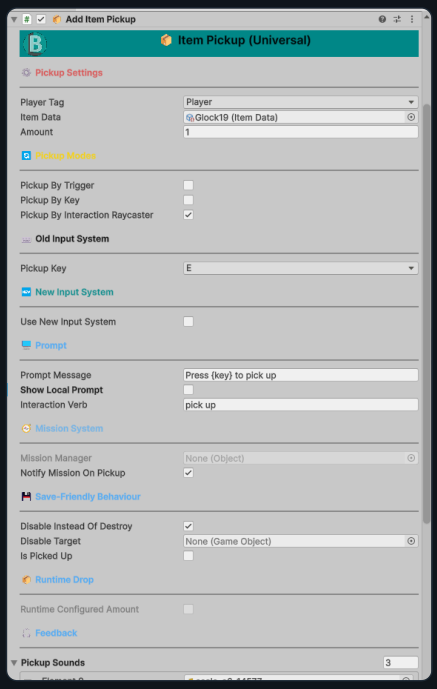
With it:

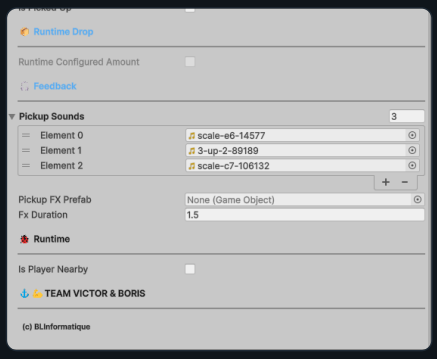
➞ items become gameplay

➞ collectible

➞ usable

➞ meaningful





Field	Type	Tooltip
missionManager	`Object`	Internal MissionManager reference resolved automatically when the optional addon is installed.
playerTag	`String`	Tag used to identify the player collider.
itemData	`ItemData`	ItemData asset added to the player's inventory.
amount	`Int32`	Amount given when this pickup is collected.
pickupByTrigger	`Boolean`	Enable automatic pickup when the player enters the trigger.
pickupByKey	`Boolean`	Enable pickup by local key handling on this object.
pickupByInteractionRaycaster	`Boolean`	Enable pickup through InteractionRaycaster via IInteractable.
pickupKey	`KeyCode`	Key used to pick up the item when using the old Input System.
useNewInputSystem	`Boolean`	Enable this if using the new Input System for local pickup key.
pickupAction	`InputAction`	Input action used to pick up the item when using the new Input System.
promptMessage	`String`	Default prompt text shown for local pickup. {key} will be replaced by the current key label.
showLocalPrompt	`Boolean`	If true, this pickup shows its own prompt when using local key pickup.
interactionVerb	`String`	Optional custom interaction verb used by InteractionRaycaster.
notifyMissionOnPickup	`Boolean`	If true, this pickup automatically notifies the mission system when collected.
disableInsteadOfDestroy	`Boolean`	If true, the pickup is disabled instead of destroyed when collected. Recommended for save/load systems.
disableTarget	`GameObject`	Optional root object to disable when collected. Leave empty to disable this GameObject.
isPickedUp	`Boolean`	True when this pickup has already been collected.

Field	Type	Tooltip
runtimeConfiguredAmount	`Boolean`	True if this pickup amount was configured at runtime by the drop system.
↓		
pickupSounds	`AudioClip[]`	Optional pickup sounds played randomly on collect.
pickupFXPrefab	`GameObject`	Optional visual FX prefab spawned on pickup.
fxDuration	`Single`	Lifetime of the spawned pickup FX in seconds.
↓		
isPlayerNearby	`Boolean`	True when the player is currently inside the trigger range.

↑ 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).

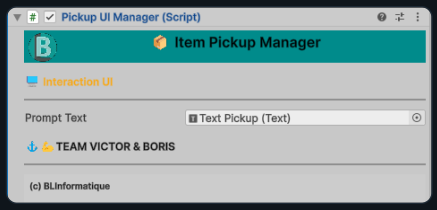
Assign a UI Text in your HUD canvas for promptText.

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.

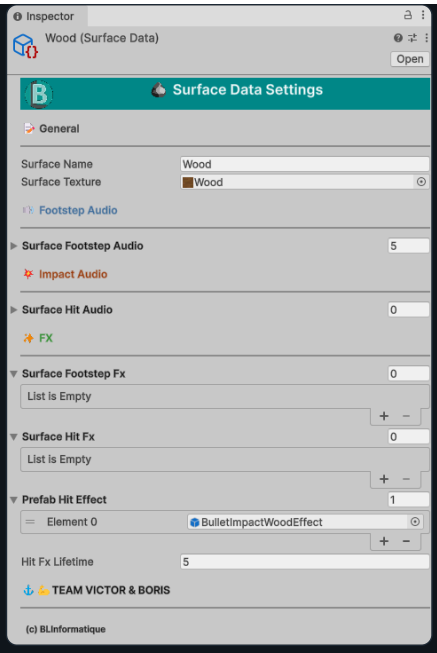
Compatibility ✅

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.

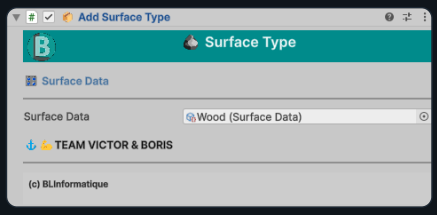
For large terrains, use FPSFootstepPlayer's terrain mapping instead of per-object SurfaceType.

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.

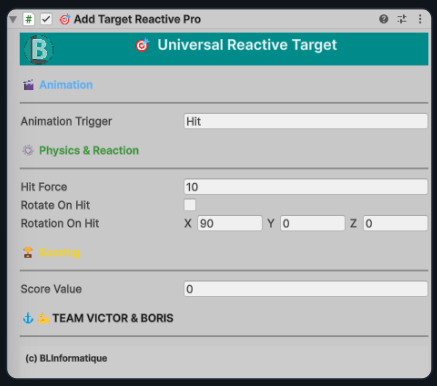
Tie scoreValue into your scoring system for training/shooting range levels.

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

📦 ToggleEventInteractable MonoBehaviour

AddComponentMenu: `BLInformatique/Real FPS/Interactions/💡 Toggle Event Interactable` • Requires: AudioSource

Toggle Event Interactable ⚫ ⚡

The ToggleEventInteractable component is a flexible interaction system used to toggle events ON and OFF.

It allows the player to:

activate or deactivate objects

toggle lights and emissions

trigger custom UnityEvents

control interactive elements such as switches, buttons, and levers

It integrates with:

InteractionRaycaster

Unity Events system

optional Save System

Purpose 🎯

Use this component to:

create switches and buttons

toggle lights or effects

control gameplay elements

trigger custom events

build reusable interaction logic

Core Concept 🧠

Interact → Toggle State → Apply Effects → Invoke Events

Interaction 🖱️

Implements:

👉 IInteractable

Works directly with:

↑

InteractionRaycaster

Example:

Press E to toggle Light

Toggle Behavior ⚙️

Is On

Current state of the toggle.

true → ON

false → OFF

Toggle Mode

Each interaction switches:

ON → OFF → ON → OFF

Events 🔔

On Toggle On

Called when switching to ON state.

On Toggle Off

Called when switching to OFF state.

👉 Use these to trigger:

doors

sounds

animations

gameplay logic

Light Control 💡

Supports toggling multiple lights.

Lights Array

List of Light components to control.

Affect Lights

If enabled:

toggles all assigned lights ON/OFF

Emission Control ✨

You added a super feature here 👌

Affect Emission

If enabled:

controls material emission

Emission Renderers

Renderers affected by emission changes.

Emission Color

Color applied when ON.

Emission Intensity

Brightness multiplier.

Emission Keyword

Shader keyword (usually _EMISSION)

👉 Works with Standard / URP shaders

Advanced Behavior ⚙️

Affect GameObjects

Toggle active state of objects.

Example:

turn on machines

enable effects

Objects To Toggle

List of GameObjects affected.

Save System Compatibility 💾

Works with:

👉 SaveableToggleEvent

Allows:

saving ON/OFF state

restoring correct state after load

Runtime Flow ⚙️

Interaction

Player interacts

State is toggled

Lights updated

Emission updated

Objects toggled

Events invoked

Integration 🔗

With InteractionRaycaster

Handles input and prompt display.

With Lighting System

Controls real-time lights.

With Materials

Controls emission in shaders.

With Save System

Maintains state persistence.

Typical Setup 🧱

Example: Light Switch

Add ToggleEventInteractable

Assign:

Light(s)

Renderer(s)

Enable:

Affect Lights

Affect Emission

Set emission color/intensity

Done ✅

Example: Machine Button

Add script

Add GameObjects to toggle

Add custom UnityEvents

Configure ON/OFF states

Best Use Cases 🧰

light switches

control panels

generators

alarms

doors (alternative to DoorLockInteractable)

environmental interactions

Debug 🪲

Common issues:

Nothing happens → check references

Emission not working → check material supports emission

Lights not toggling → check Light references

State not saved → check Save system integration

⚓ 💪 TEAM VICTOR & BORIS Note

ToggleEventInteractable is your universal interaction tool.

Instead of writing custom scripts for every interaction...

➞ you use THIS

➞ and plug events

➞ and it just works

Field	Type	Tooltip
⬇		
missionManager	`Object`	Internal MissionManager reference resolved automatically when the optional addon is installed.
⬇		
isOn	`Boolean`	Current runtime ON/OFF state.
⬇		
allowInteraction	`Boolean`	If true, this interactable can be triggered by the interaction system.
useToggle	`Boolean`	If true, each interaction toggles between ON and OFF.
startOn	`Boolean`	Initial ON/OFF state when the scene starts.
applyStateOnAwake	`Boolean`	If true, current state is applied automatically on Awake.
suppressEventsOnAwake	`Boolean`	If true, UnityEvents and audio are not fired automatically on Awake.
showDebugLogs	`Boolean`	If true, logs state changes in the Console.
⬇		
targetGameObject	`GameObject`	Optional GameObject enabled when the toggle state is ON.
targetBehaviour	`Behaviour`	Optional Behaviour enabled when the toggle state is ON.
targetLight	`Light`	Optional Light enabled when the toggle state is ON.
⬇		
targetGameObjects	`GameObject[]`	GameObjects enabled when the toggle state is ON.
targetBehaviours	`Behaviour[]`	Behaviours enabled when the toggle state is ON.
targetLights	`Light[]`	Lights enabled when the toggle state is ON.
⬇		
emissionRenderers	`Renderer[]`	Renderers whose material emission is controlled by this toggle.
emissionOnColor	`Color`	Emission color used when the toggle is ON. HDR colors are recommended.
emissionOffColor	`Color`	Emission color used when the toggle is OFF.

Field	Type	Tooltip
emissionIntensity	`Single`	Emission intensity multiplier applied to ON and OFF emission colors.
enableEmissionKeywordWhenOn	`Boolean`	If true, enables the _EMISSION keyword when turning ON.
disableEmissionKeywordWhenOff	`Boolean`	If true, disables the _EMISSION keyword when turning OFF.
useInstancedMaterials	`Boolean`	If true, uses Renderer.materials to affect only this renderer instance at runtime.
notifyMissionOnStateChanged	`Boolean`	If true, this interactable automatically notifies the mission system after state changes.
⬇		
audioSource	`AudioSource`	Optional AudioSource used to play ON and OFF sounds.
turnOnClip	`AudioClip`	Optional clip played when turning ON.
turnOffClip	`AudioClip`	Optional clip played when turning OFF.
⬇		
onTurnOn	`UnityEvent`	Invoked when the state changes to ON.
onTurnOff	`UnityEvent`	Invoked when the state changes to OFF.
onInteracted	`UnityEvent`	Invoked after each interaction.
onStateChanged	`UnityEvent<Boolean>`	Invoked whenever the state changes. Bool parameter = current ON/OFF state.

↑ 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.

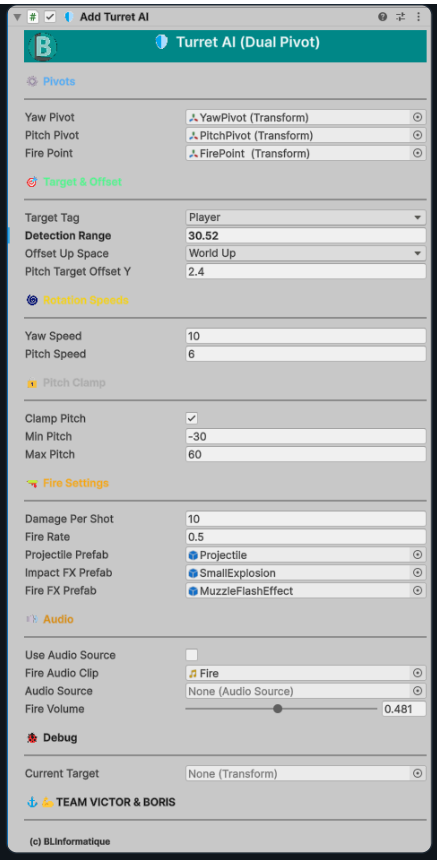
Attach SurfaceType to walls/targets for realistic impact FX.

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.

↑


Add Turret AI
Turret AI (Dual Pivot)
Pivots
Yaw Pivot
YawPivot (Transform)
Pitch Pivot
PitchPivot (Transform)
Fire Point
FirePoint (Transform)
Target & Offset
Target Tag
Player
Detection Range
30.52
Offset Up Space
World Up
Pitch Target Offset Y
2.4
Rotation Speeds
Yaw Speed
10
Pitch Speed
6
Pitch Clamp
Clamp Pitch
Min Pitch
-30
Max Pitch
60
Fire Settings
Damage Per Shot
10
Fire Rate
0.5
Projectile Prefab
Projectile
Impact FX Prefab
SmallExplosion
Fire FX Prefab
MuzzleFlashEffect
Audio
Use Audio Source
Fire Audio Clip
Fire
Audio Source
None (Audio Source)
Fire Volume
0.481
Debug
Current Target
None (Transform)
TEAM VICTOR & BORIS
(c) BLInformatique

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]
⬇		
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.

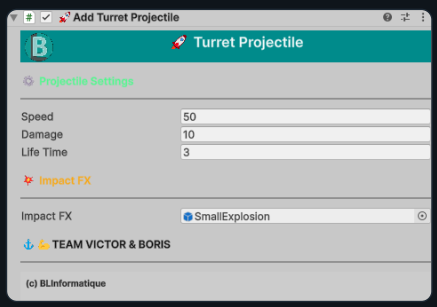
For big explosions, pair with ExplosionEntity triggered on impact.

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.

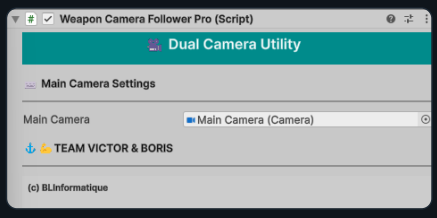
↑

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.

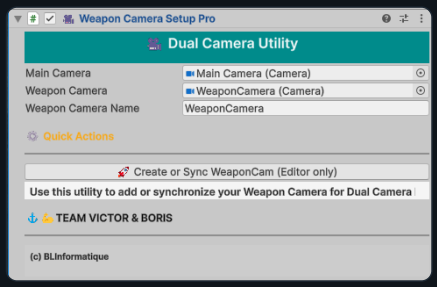
Keep background = clear to overlay seamlessly on MainCam.

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.

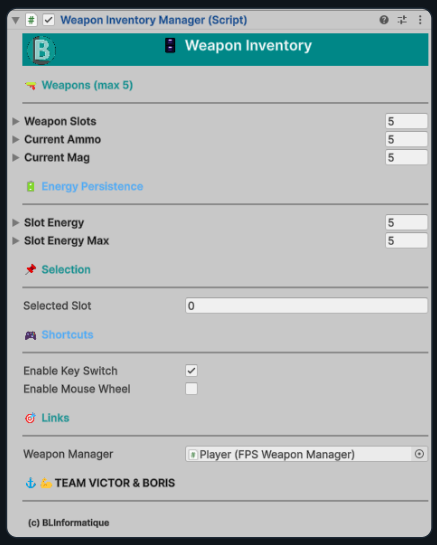
For energy-based tools (flashlight, scanner), test slot switching to confirm persistence works.

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[]`	Current reserve ammo per slot.
currentMag	`Int32[]`	Current magazine ammo per slot.
slotEnergy	`Single[]`	Per-slot energy snapshot. -1 means uninitialized.
slotEnergyMax	`Single[]`	Per-slot max energy cache.
selectedSlot	`Int32`	Current selected slot index. -1 means none selected.
enableKeySwitch	`Boolean`	Enable weapon switch by keyboard (keys 1-5).
enableMouseWheel	`Boolean`	Enable weapon switch by mouse wheel.
weaponManager	`FPSWeaponManager`	Reference to 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.

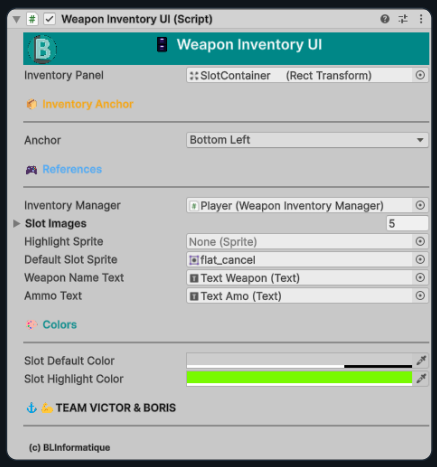
Keep UpdateUI() connected to WeaponInventoryManager events for efficiency (optional optimization).

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 (recommended size: 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