Weapon Management System

Version 1.0

Parvesh Yadav

December 12, 2025

Contents

1 Introduction

The Weapon Management System is a modular framework developed in Unity to demonstrate first-person shooter (FPS) mechanics. This documentation describes the design, implementation, usage, and extension of the system. The system is intended to be simple to understand and easy to extend, providing a solid foundation for additional gameplay mechanics in FPS games.

The system supports shooting using both raycast-based and projectile-based methods, weapon swapping, pickups, dropping mechanics, and basic user interface feedback. The primary purpose is educational; it provides a clear example of component-based game architecture in Unity.

2 System Overview and Problem Description

First-person weapons require multiple interconnected mechanics: firing, reloading, recoil, swapping, pickup, and drop interactions. Traditional implementations often mix gameplay logic across several classes and systems, resulting in tightly coupled code. This system adopts a modular approach using ScriptableObjects and well-separated components to isolate responsibilities.

This documentation serves as both a description of the current implementation and a reference for future expansions. It addresses the following primary problems:

- How to fire weapons with configurable behaviors.
- How to manage ammunition and reloading.
- How to support swapping between multiple weapons.
- How to interact with weapons in the game world (pickup and drop).
- How to maintain clean and maintainable architecture for future extension.

3 Feature List

The system implements the following core features:

- Weapon firing with left mouse button input.
- Reloading weapon using the `R` key.
- Weapon swapping via mouse scroll wheel input.
- Weapon dropping using the `G` key.
- Weapon pickup using the `E` key.
- User interface feedback for current weapon and pickup messages.
- Recoil feedback on the camera for visual realism.
- Support for both hitscan and projectile firing modes.

4 Installation and Usage

4.1 Unity Package Import

To use the system, import the provided Unity package into a clean Unity project:

1. Open Unity Hub and create a new *3D (Built-In Render Pipeline)* project.
2. In the Unity Editor, navigate to `Assets > Import Package > Custom Package`.
3. Select the provided `WeaponManagementSystem_ParveshYadav.unitypackage`.
4. In the import dialog, ensure the following are selected:
    - Documentation folder
    - Prefabs including pistol, cube and sphere
    - ScriptableObjects (weapon definitions)
    - Scenes
    - Scripts
    - UI prefabs

5. Click *Import*.

After import, open the scene located at `Assets/Scenes/SampleScene.unity`. This scene contains an example setup demonstrating all system features.

4.2 Controls and Interaction

The following controls are supported:

- **Left Click** – Fire the current weapon.
- **R** – Reload the weapon's magazine.
- **Mouse Scroll** – Swap between all available weapons.
- **G** – Drop the current weapon into the world.
- **E** – Pick up a nearby weapon.

The player starts with a single pistol. A black cube in the scene represents a pistol pickup that can be acquired by the player.

5 System Components

This section describes the major components of the weapon system.

5.1 WeaponDefinition (ScriptableObject)

A `WeaponDefinition` ScriptableObject stores all relevant parameters for a weapon:

- Weapon name
- Damage per shot
- Fire rate (shots per second)
- Magazine size and reserve ammo capacity
- Recoil parameters
- Spread angle for firing
- Boolean indicating hitscan versus projectile mode

New weapons can be added via the Unity Editor: `right-click → Create → Weapon System → Weapon Definition`.

5.2 WeaponController

The `WeaponController` script acts as the central manager for weapon logic. It:

- Reads input from the `IInputProvider`.
- Manages firing, reloading, and swapping logic.
- Handles weapon pickup and drop events.
- Applies recoil effects using the `SimpleRecoilModel`.
- Updates UI elements to reflect current state.

5.3 HitScanBehaviour

This component performs a raycast from the player camera to simulate instant weapon hits. It reads the weapon's spread angle and damage value and applies impacts accordingly.

5.4 ProjectileBehaviour

In projectile mode, a sphere projectile is instantiated at the weapon's muzzle and propelled using rigidbody physics. This mode allows for visible bullets and physics-based interactions.

5.5 SimpleRecoilModel

A simple camera rotation is applied each time the weapon fires. This provides visual feedback for firing events without complex animation curves.

5.6 LegacyInputProvider

An abstraction layer that provides input events such as fire, reload, scroll wheel, and pickup commands. This allows for future expansion to other input systems.

5.7 WeaponPickup

World objects that enable the player to acquire new weapons. When the player is near the pickup and presses E, the associated weapon is added to the inventory.

6 Algorithms, Models, and Mathematics

This section explains the underlying logic without needing code references.

6.1 Hitscan Shooting

Hitscan firing uses a forward raycast with a spread angle. The direction is perturbed randomly within the spread range to simulate inaccuracy. If the ray hits an object implementing `IDamageable`, the damage value from the weapon definition is applied.

6.2 Projectile Shooting

Projectile firing instantiates a physics-based sphere at the weapon's muzzle. Rigidbody velocity is applied in the forward direction of the weapon, causing the projectile to travel and impact other objects over time.

6.3 Weapon Swapping

Weapon swapping uses the mouse scroll wheel input, which returns a positive or negative integer. The current weapon index is modified using modular arithmetic to cycle through the available weapons:

$$\text{newIndex} = (\text{currentIndex} + \text{scrollDelta}) \bmod N,$$

where N is the number of weapons.

6.4 Pickup Logic

Pickup logic stores a reference to the nearby pickup on trigger enter. When the player presses E, the weapon is appended to the inventory and the pickup object is destroyed.

6.5 Dropping Weapons

Dropping logic instantiates a pickup prefab at the player's muzzle. The selected weapon is removed from the inventory and the next weapon is equipped automatically.

7 Demo Scene Description

The demo scene includes:

- A first-person player with movement and camera scripts.
- A default pistol equipped at start.
- A pistol pickup represented by a black cube.
- Multiple white cube targets which demonstrate the damage system.
- UI elements that display the current weapon and pickup messages.

Users can:

- Shoot and damage targets.
- Pick up the pistol cube and swap weapons.
- Reload weapons and experience recoil.
- Drop and re-pick weapons.

8 Extending the Weapon System

This system is designed to be easily expanded.

8.1 Adding New Weapons

To add a new weapon:

1. Create a new `WeaponDefinition`.
2. Set appropriate stats such as damage and fire rate.
3. Add the new definition to the weapon list on the `WeaponController`.

4. If using projectile mode, assign a suitable projectile prefab.
5. For world pickups:
    - Create a game object with a mesh.
    - Add the `WeaponPickup` script.
    - Assign the weapon definition asset.
    - Add a trigger collider.

8.2 Adding Sound and Animation

Future improvements could include:

- Weapon firing sound effects.
- Reload animations.
- Muzzle flash particle effects.
- Advanced recoil curves tied to the animation system.

9 Conclusion

The Weapon Management System provides a solid foundation for modular FPS gameplay. Its clear separation of components, extensible definition assets, and simple interaction model make it suitable for both educational and practical use in game projects.