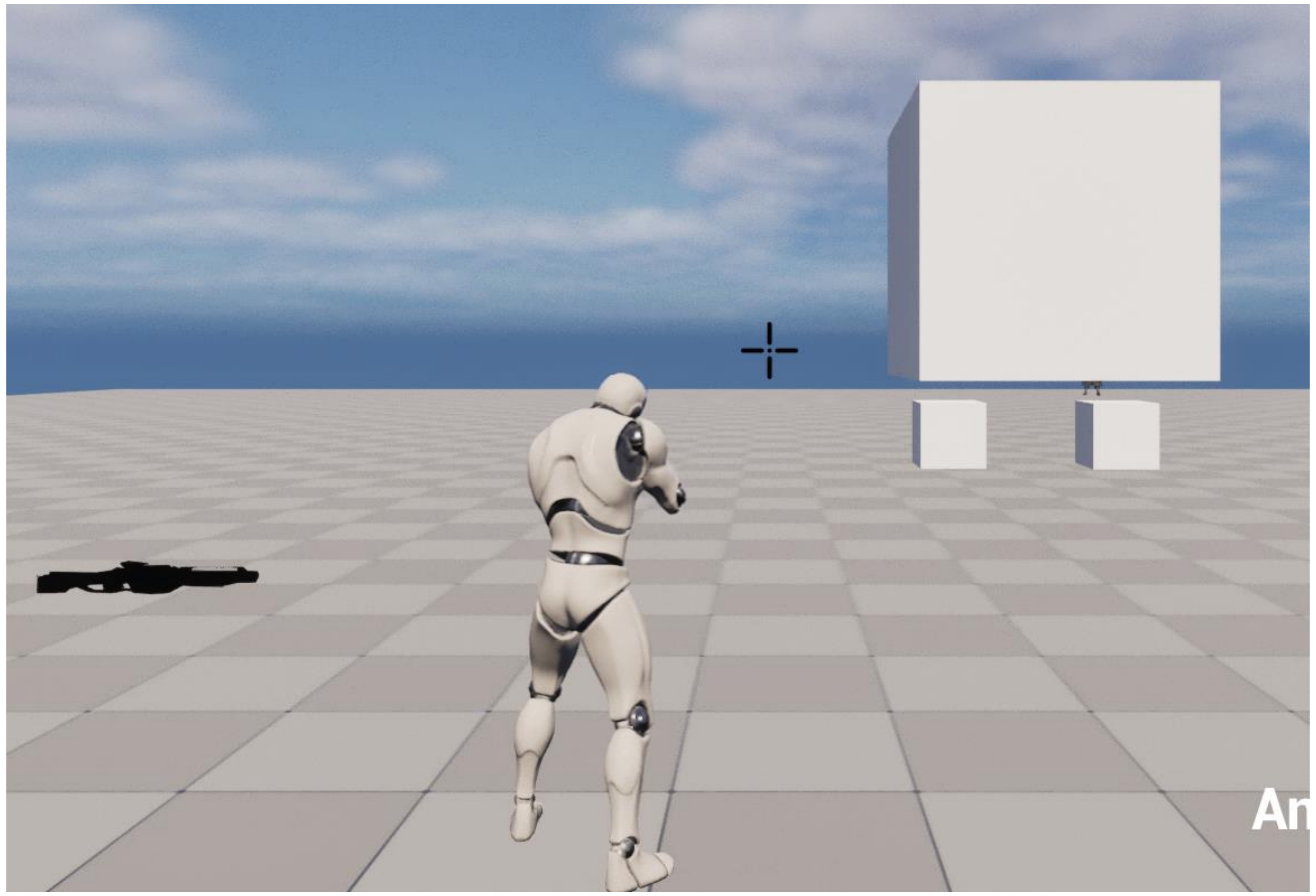
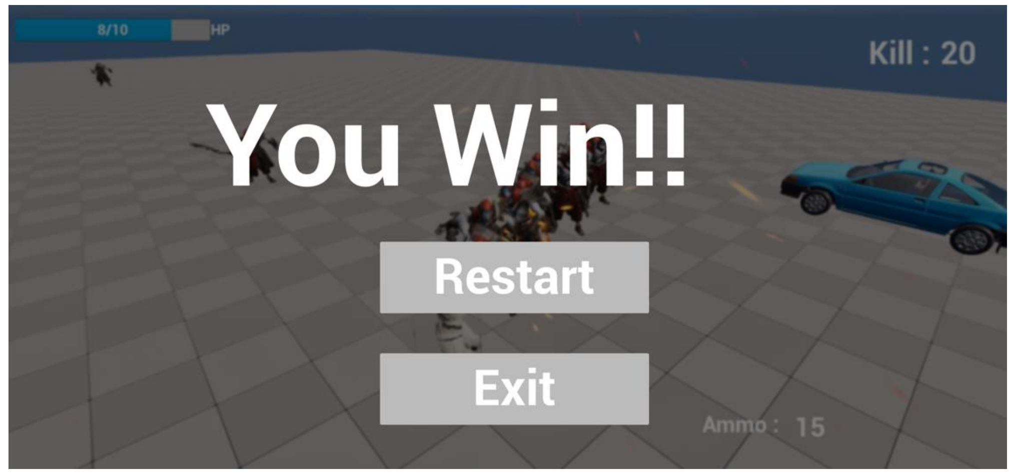
TPS Project



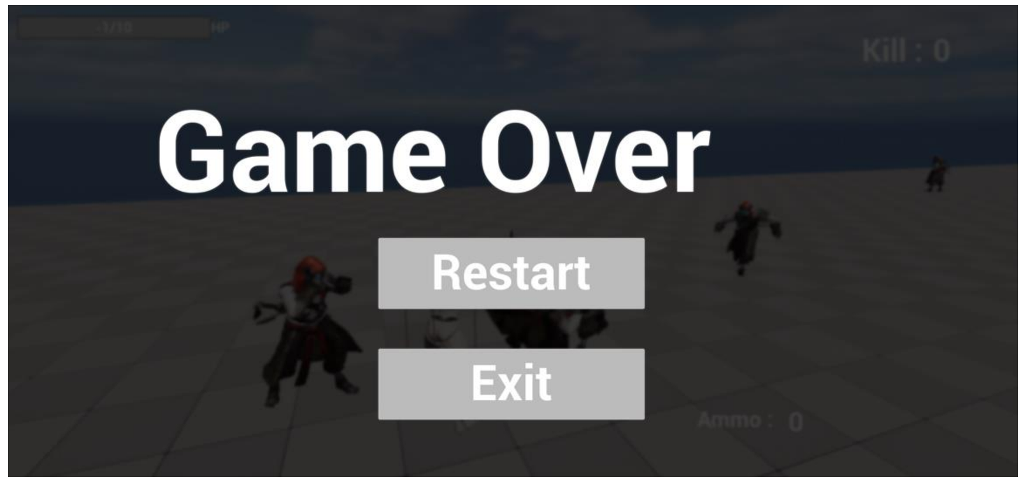
Victory Condition:

Kill 20 enemies



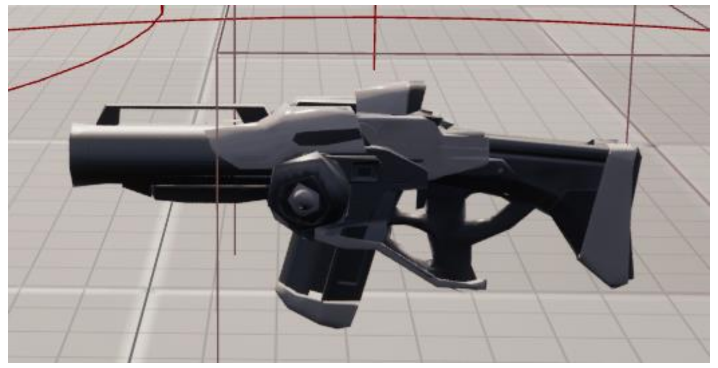
Defeat Condition:

When the character's HP becomes 0



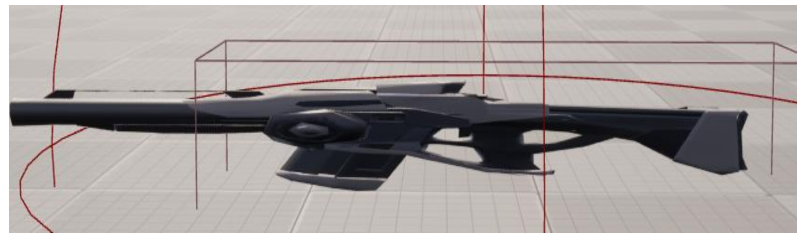
1. Weapons

1) Rifle



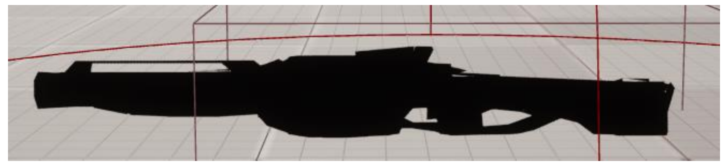
- Implement single/burst fire functionality
- Implement fire using line trace

2) Sniper



- Implement sniper mode UI
- Implement firing using line trace

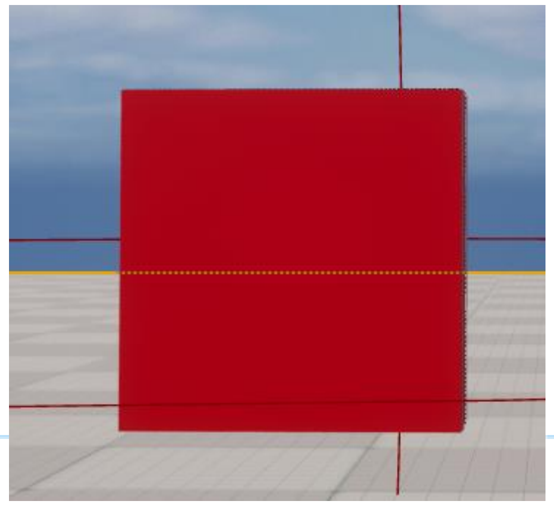
3) Bazooka



- Implement firing by directly spawning the Bullet class

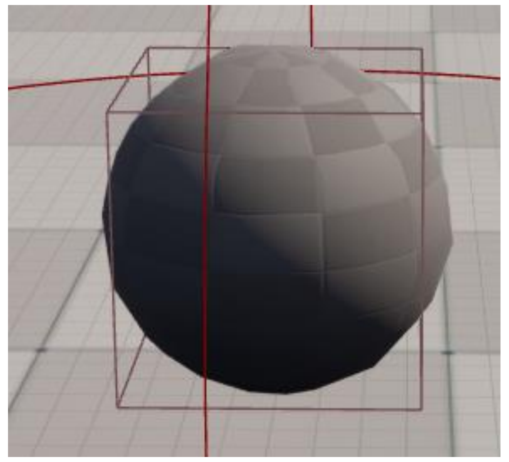
2. Items

1) HPPack



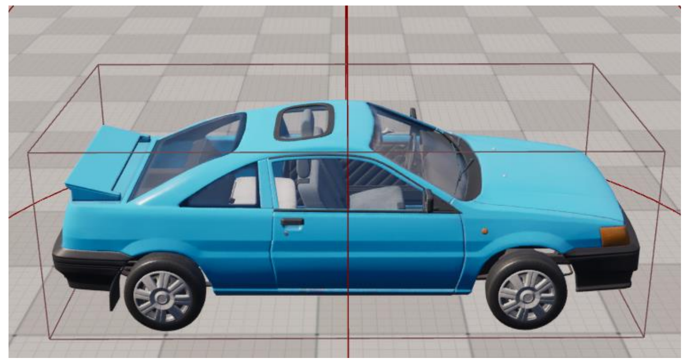
- Player HP recovery

2) Ammo



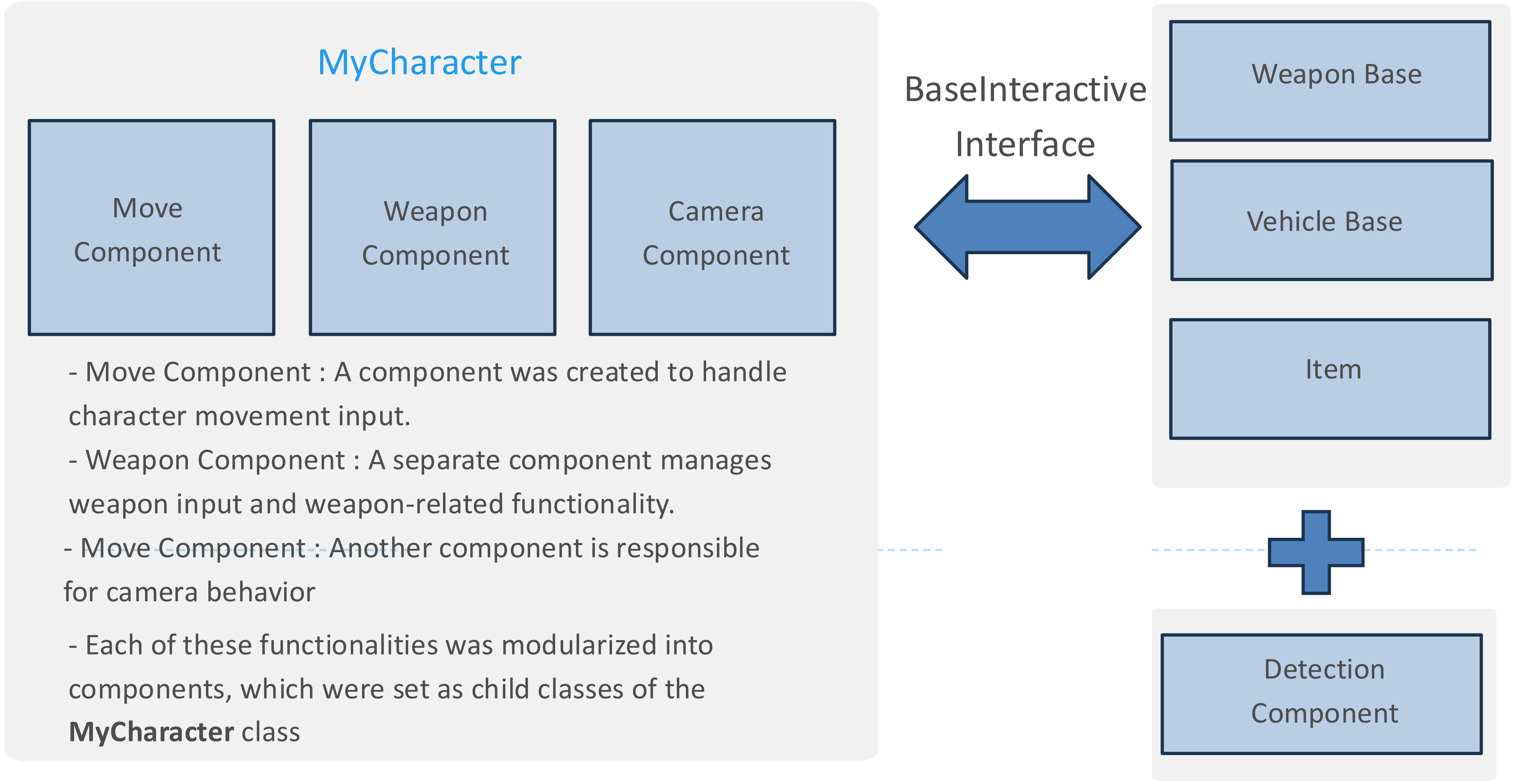
- Reload the magazine of the gun which the player is currently holding

3) Car



- Player possess and controls the vehicle

Overall code structure



- Move Component : A component was created to handle character movement input.
- Weapon Component : A separate component manages weapon input and weapon-related functionality.
- Move Component : Another component is responsible for camera behavior
- Each of these functionalities was modularized into components, which were set as child classes of the **MyCharacter** class

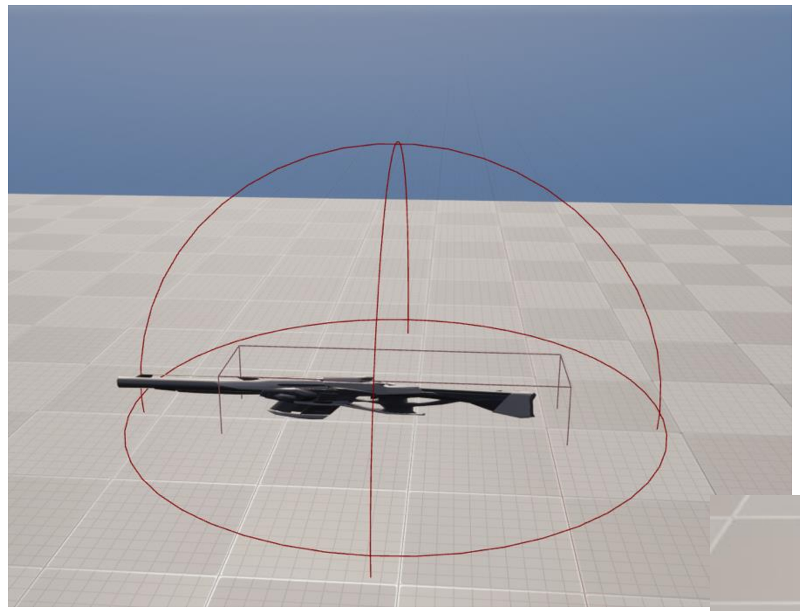
- I implemented interactions between **MyCharacter** and various items, such as picking up, ridding, and equipping

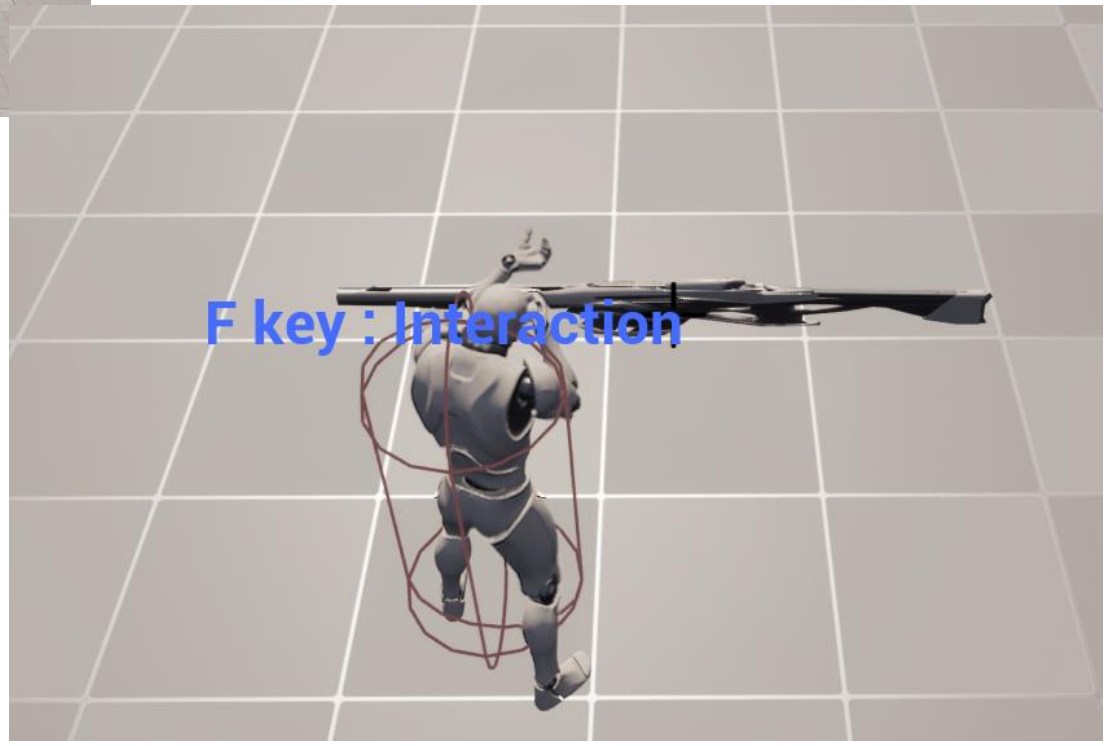
- To handle interactions with different types of classes (**Pawn, Actor**), I used an interface-based approach by inheriting and implementing appropriate interfaces for each interaction type.

- All weapons inherit from the **WeaponBase** class, and vehicles inherit from the **VehicleBase** class

- Additionally, to trigger UI when an item is detected by **Mycharacter**, I implemented a detection component and added it as a child component to each item

Detection Component





- The Detection Component is responsible for displaying the interaction UI when the player is detected.

 - A DetectionSphere (Sphere Component) is set in the parent class, and when the player enter the sphere (triggering the **OnOverlapBegin** egent), a timer is started that calls the **CheckIfInView()** fuction every 0.1 seconds.

 - The **CheckIfInView()** function performs a line trace from the player's viewpoint, and if the trace hits the item, the interaction UI is displayed accordingly.

Weapon Equip and Unequip

Weapon Equip

Pressing the **F key** equips a weapon.

The weapon is attached to a predefined socket on the player’s skeleton using the **AttachToComponent** function.
This way, the already spawned weapon actor is reused directly instead of creating a new one.

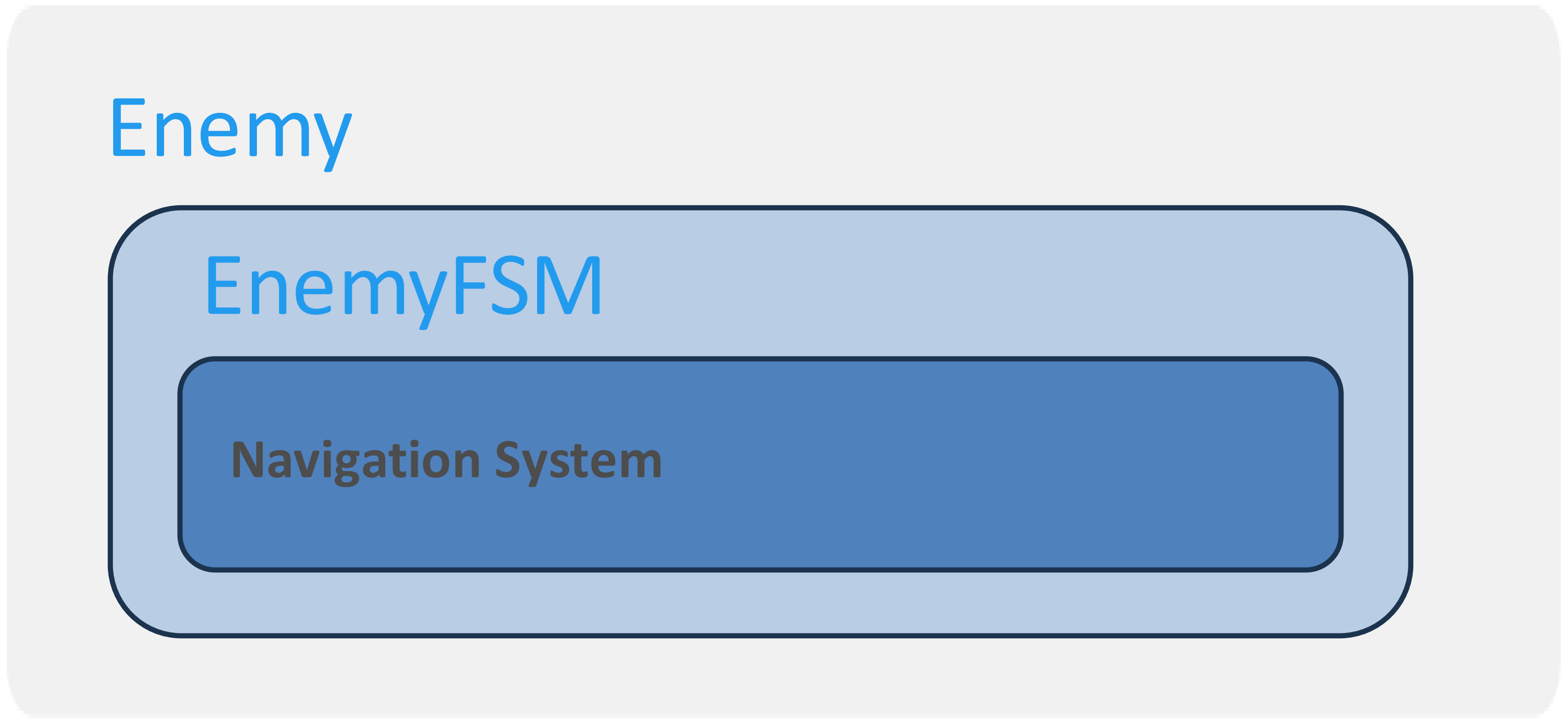
The player can equip two weapons.
Each weapon is assigned to either Slot1 or Slot2.
Pressing the 1 key equips the weapon in Slot1, and pressing the 2 key equips the weapon in Slot 2

Weapon Unequip

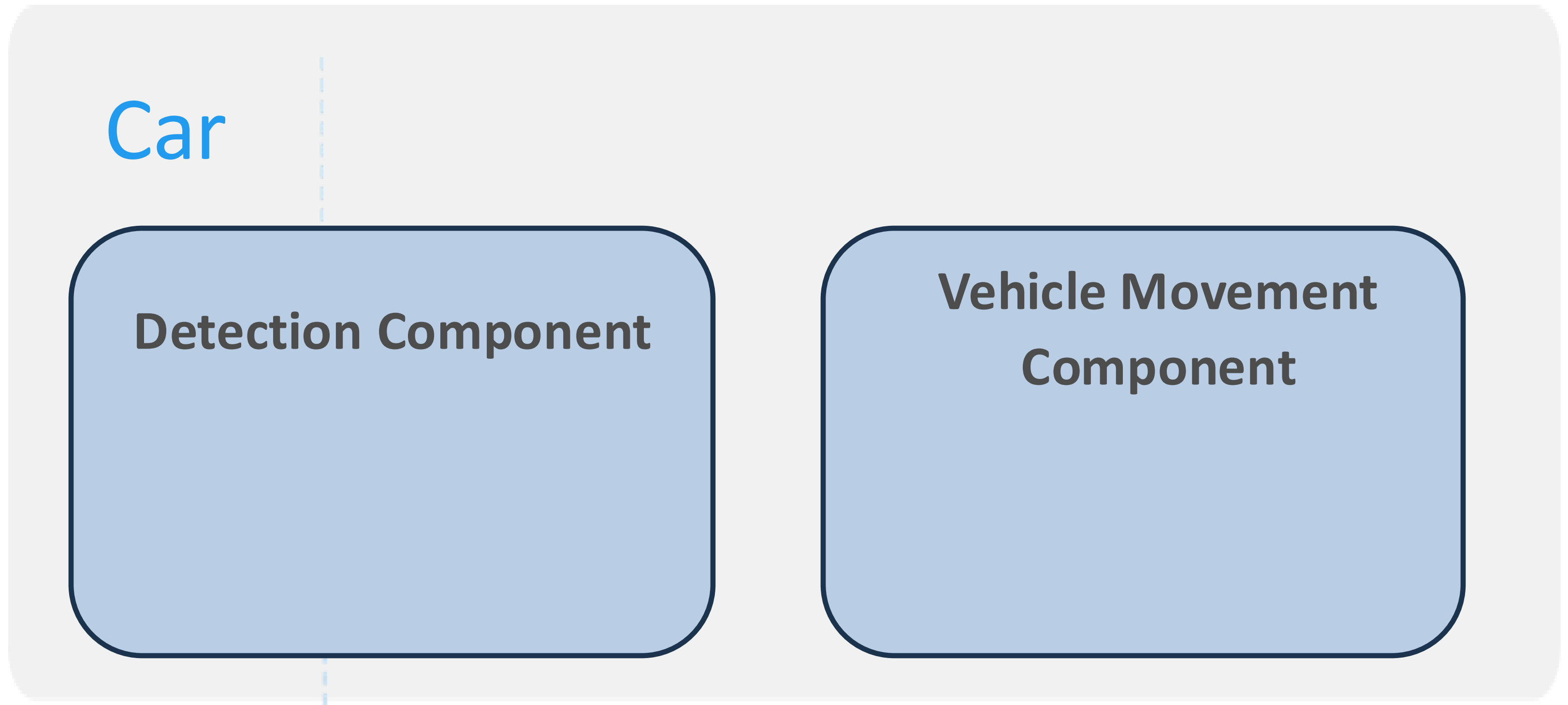
Pressing the **G key** unequips the currently equipped weapon

When unequipping, a vector is calculated pointing downward from a point in front of the player. Using a line trace, the weapon is placed at the location where this vector hits the ground.

Enemy & Car



- The Enemy actor includes an **EnemyFSM** class as a child component, which implements a finite state machine to manage the enemy's behavior states: Idle, Move, Attack, Damage, and Dead
- Using Unreal's Navigation System, the enemy automatically navigates toward the player upon detection, following the optima path.
- If the player is within a certain range, the enemy performs a melee (short-range) attack.



- Vehicles were implemented using the Chaos Vehicles Plugin

- To allow the player to posses a vehicle, input was handled through the player cotroller. However, when exiting the vehicle, the controller is till set to the vehicle, so the unposses input logic was handled within the vehicle class itself.

Thank you!