Project Report

A First-Person Shooter Game with Puzzle-Solving

***Group:**CD project blue*

Members:

Gao Hong

Hou Mobai

Zeng hao

Outline

A. Purpose of the system and features

A.1 Purpose

The game combines first-person shooting (FPS) with puzzle-solving elements, set on a mysterious island. The player's task is to slay monsters on the island, solve the mysteries of various mechanisms, and ultimately find a way to leave the island.

The main goal of the game is to attract teenagers and adults who are passionate about action, adventure, and puzzle-solving. By integrating the excitement of shooting with the intellectual challenge of puzzle-solving, the game significantly enhances its challenge and depth. The design focuses on enhancing player engagement and the replayability of the game by providing complex puzzles and strategic gameplay, keeping players highly involved and encouraging them to explore various aspects of the game. In this way, the game offers not only the thrill of shooting but also the joy of in-depth exploration and puzzle-solving, delivering a comprehensive adventure experience to the players

A.2 Features

Our game is a linear FPS (First-Person Shooter) experience, featuring all the basic functions of an FPS. This includes character movement, sprinting, a variety of monsters, scene destruction and puzzle-solving, weapons with multiple attributes, and interactive scene-based puzzle-solving. Additionally, the game character has various movement modes and physical presence. Our game, while based on FPS combat, integrates puzzle-solving and parkour elements, making it more interesting and distinctive.
In Phase 1, the game focuses on building basic elements, including basic shooting mechanics, obstacles along paths, and basic interactions with ACTORs (characters) on the map. The core of this stage is to establish a solid foundation for the game, familiarizing players with the environment and basic operations.
Entering Phase 2 Two, it undergoes significant changes and expansions. This stage not only inherits the basic gameplay of Stage One but also introduces deeper elements, such as intellectual challenge puzzle tasks, greatly enriching the depth and exploration space of the game. Notably, Stage Two brings diversified weapon choices and a refined ammunition reloading and loading system, adding more strategy and dynamism to combat. Additionally, the game now supports direct combat between players and monsters, increasing the game's challenge. The introduction of a new mechanism system requires players to use intelligence to solve puzzles during combat, while the pet following system provides additional interaction and fun.
Overall, from Stage One to Stage Two, the game evolves from basic functionality to advanced interaction and strategic depth. By adding new weapon systems, player-monster combat mechanisms, pet assistance, and puzzle tasks, the game not only offers a richer and more in-depth gaming experience but also successfully combines the tension of traditional FPS games with the intellectual challenges of puzzle games, creating a unique and captivating gaming world.

B. Explanation of Implementation (code and interface)

B.1 Phase 1

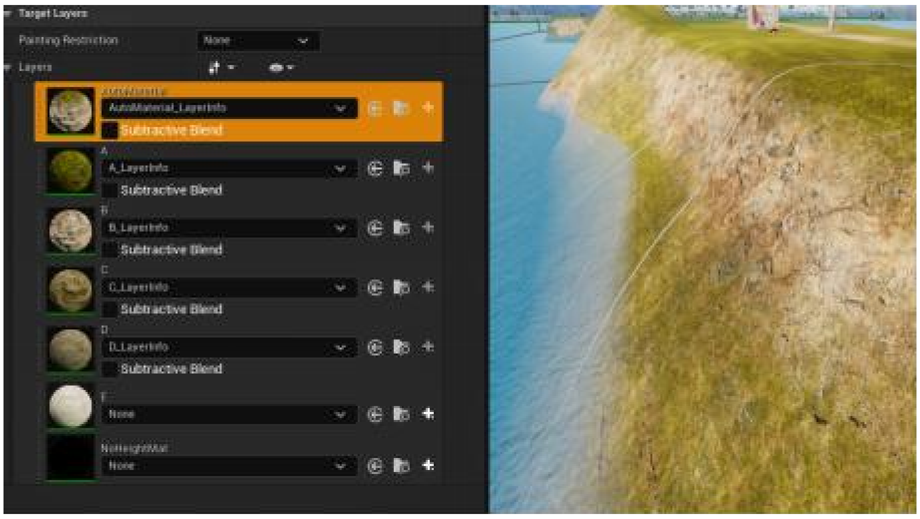
B.1.1 Map

-Overall map

About the overall design of the map, we want the game to take place in a relatively small place, but we do not want the player's field of view to be excessively blocked by obstacles, leading to a negative visual experience. Therefore, we chose to set the game location on a small island, allowing players to move only on the island's land while having a clear view of the ocean and snowy mountains. Through the design of lighting and fog, we aim to present the game scene with the best possible visual experience.

Overall look of game map

For the land area of the island, where players can play on, we used the Landscape tool to sculpt the general shape of the island and then fixed it in detail. We also used AutoMaterial to automatically adjust different materials based on the terrain. For instance, areas with steep slopes feature a stone-like material, while flat terrain is covered with grass material, and so on.


Target Layers
Painting Restriction: None
Layers
AutoMaterial: AutoMaterial_LayerInfo, Subtractive Blend
A: A_LayerInfo, Subtractive Blend
B: B_LayerInfo, Subtractive Blend
C: C_LayerInfo, Subtractive Blend
D: D_LayerInfo, Subtractive Blend
F: None
NoHeightMat: None

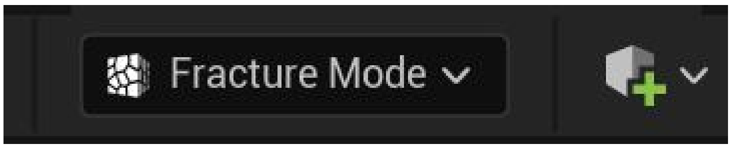
Layers of Materials

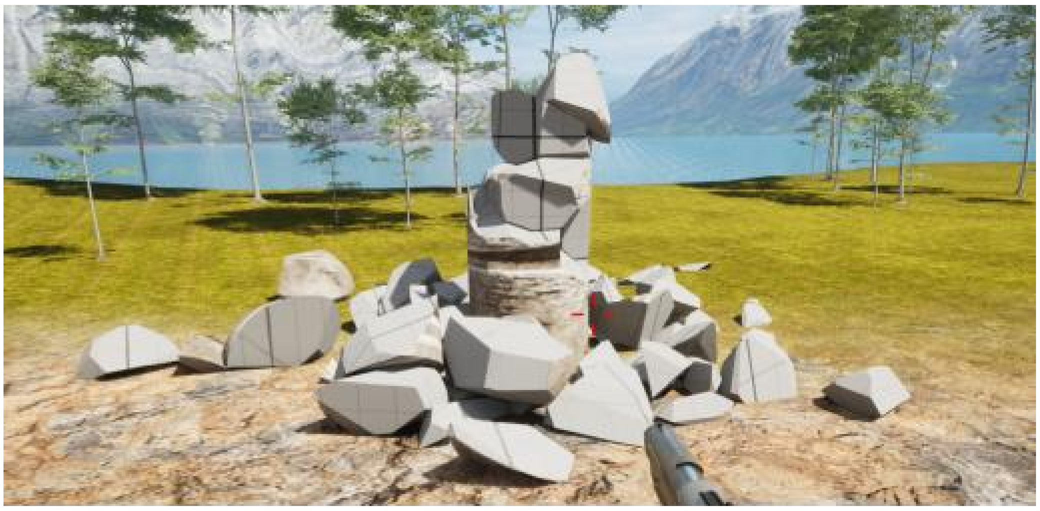
We also incorporated some non-original 3D assets from sources such as Quixel Bridge and other free online resources. For example, distant snowy mountains and components of the castle. We formed a natural distribution of these assets within the map by stretching and rotating them and putting them in the right place.

Use of 3D Assets

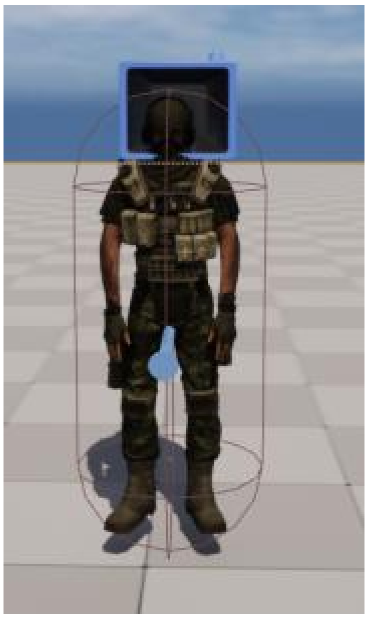
-Destructible objects

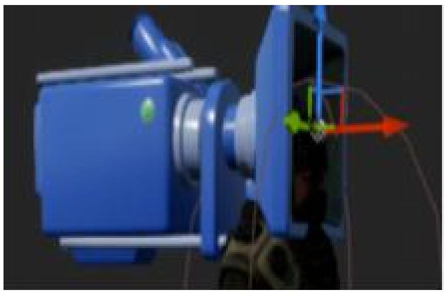
Using Fracture Mode to create destructible objects.
They can be blown up as the player fights enemies to intensify the battle atmosphere.



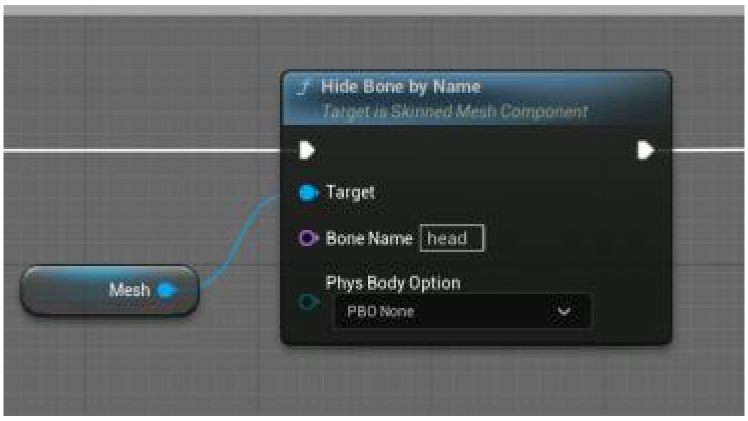


B.1.2 Character/player





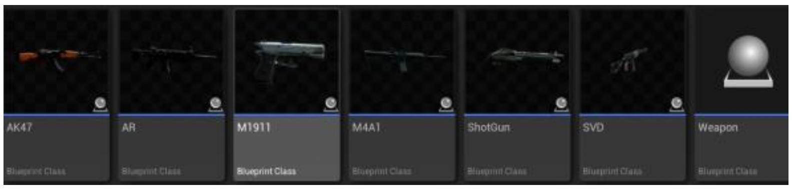
Here is our character. In this stage, we implement a simple character downloaded from the unreal marketplace for free. In this section, we demonstrate a design that sets us apart from other FPS games. Our characters have complete bodies, rather than just arms and a floating camera. Here, we use a spring arm to control the character's perspective. The spring arm is also applied in subsequent designs for death animations. Additionally, to ensure complete shadows of the characters in scenes with simulated lighting.

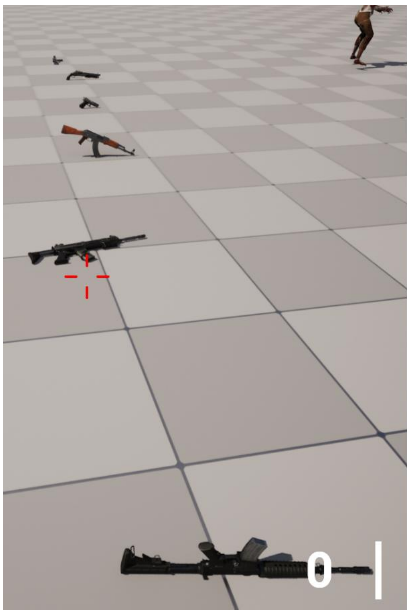


Bone Name: head
Phys Body Option: PBO None

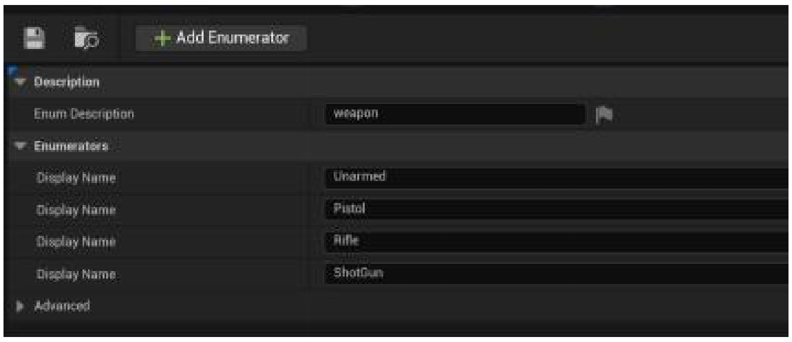
we use the 'hide bone by name' function to hide the head without affecting the shadow

B.1.3 Weapon





In this stage, we simply define the weapon by using the parent-blueprint class. So we can add weapons easily.

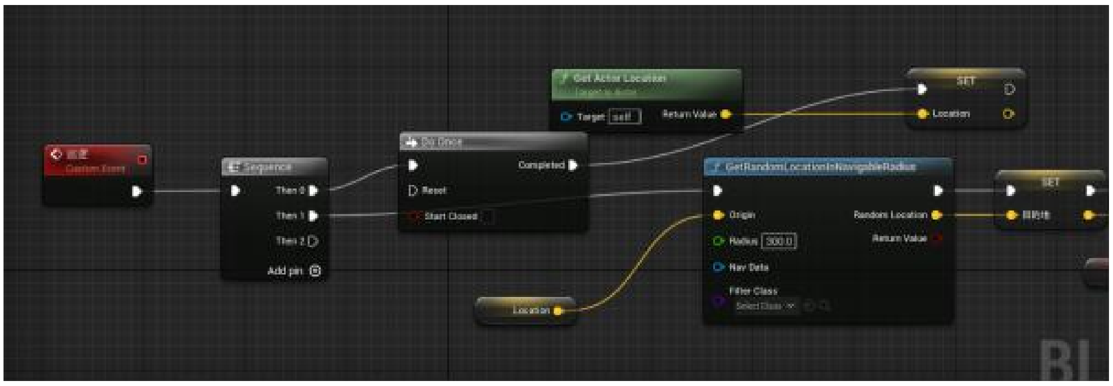


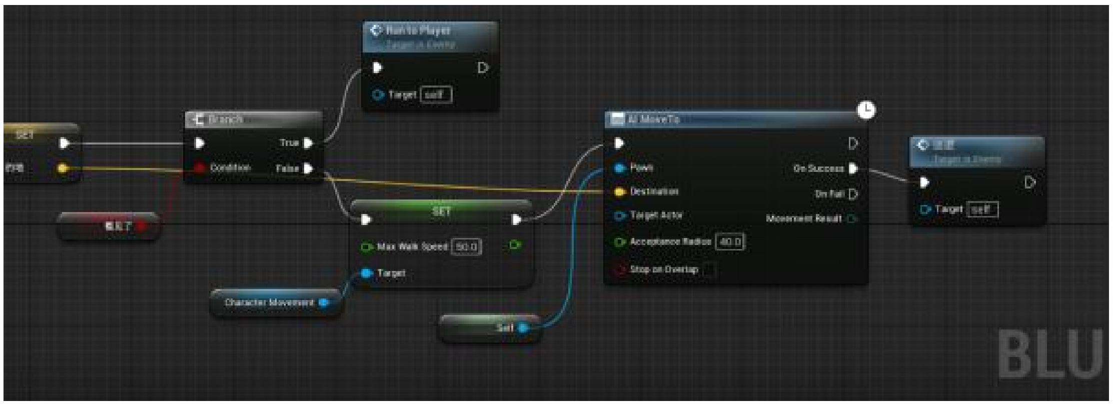
Enum Description: weapon
Display Name: Unarmed
Display Name: Pistol
Display Name: Rifle
Display Name: ShotGun

We also add an Enum to clarify different weapon types, which will help us to handle multiple weapon problems

B.1.4 Enemy

Basic Functionality: At this stage, the focus of the game is on establishing the basic environment and characters. The behavior of monsters is usually limited to basic non-interactive actions.

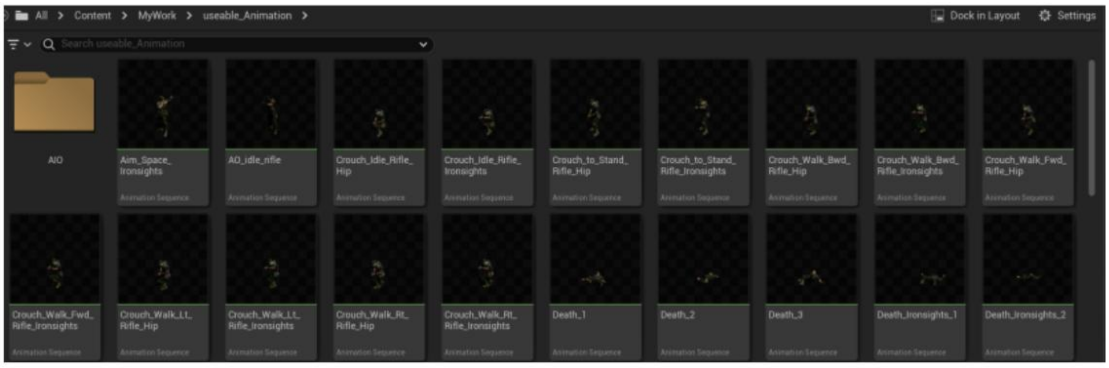




B.2 Phase 2

B2.1 Movement system

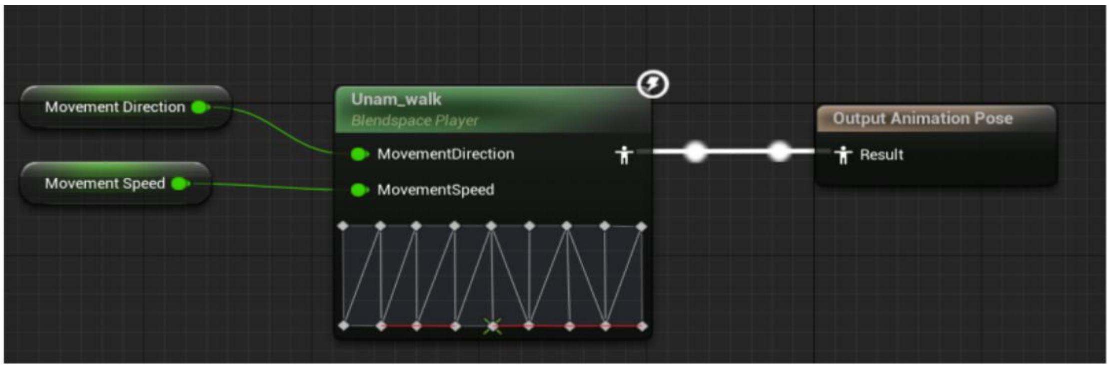
-basic movement

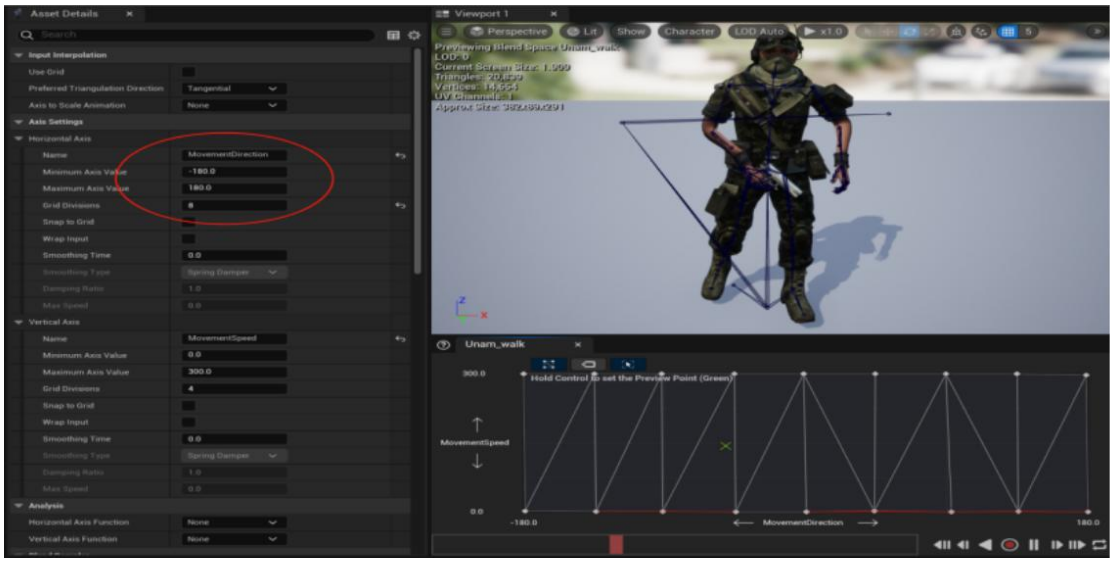

All > Content > MyWork > useable_Animation >

Dock in Layout
Settings

Search useable_Animation

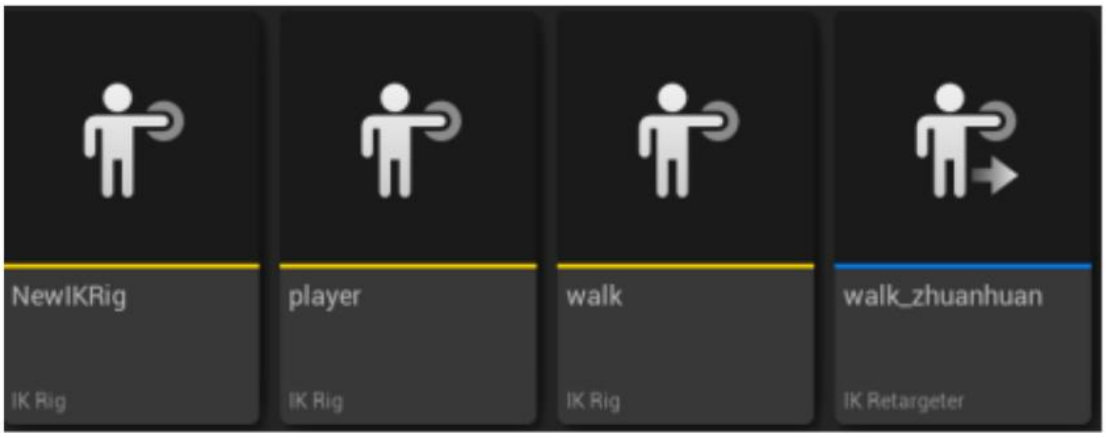
AIO
Aim_Space_Ironsights
AO_idle_rifle
Crouch_Idle_Rifle_Hip
Crouch_Idle_Rifle_Ironsights
Crouch_to_Stand_Rifle_Hip
Crouch_to_Stand_Rifle_Ironsights
Crouch_Walk_Bwd_Rifle_Hip
Crouch_Walk_Bwd_Rifle_Ironsights
Crouch_Walk_Fwd_Rifle_Hip
Crouch_Walk_Fwd_Rifle_Ironsights
Crouch_Walk_Lt_Rifle_Hip
Crouch_Walk_Lt_Rifle_Ironsights
Crouch_Walk_Rt_Rifle_Hip
Crouch_Walk_Rt_Rifle_Ironsights
Death_1
Death_2
Death_3
Death_Ironsights_1
Death_Ironsights_2





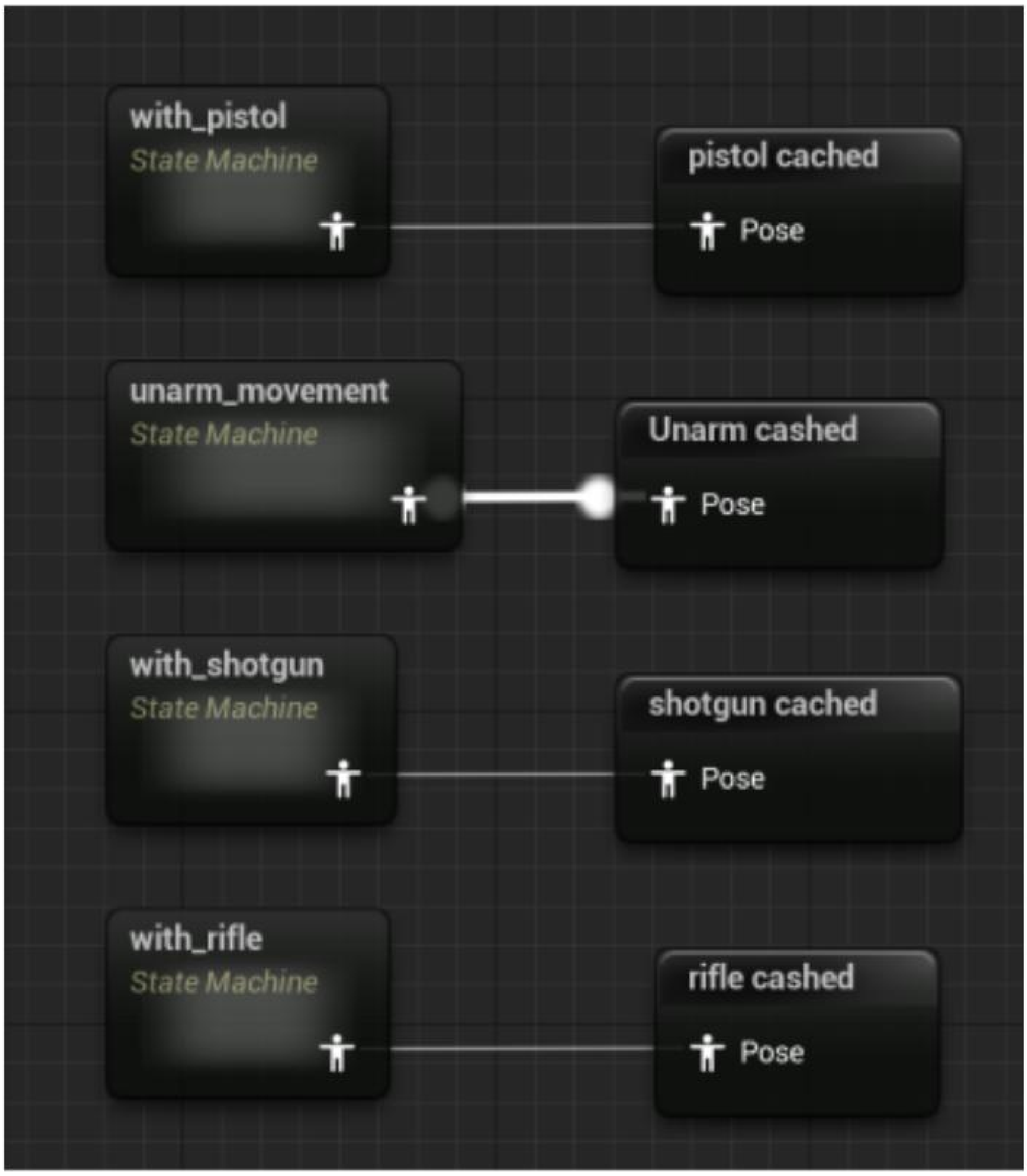
Input Interpolation	
Use Grid	
Preferred Triangulation Direction	Tangential
Axis to Scale Animation	None
Axis Settings	
Horizontal Axis	
Name	MovementDirection
Minimum Axis Value	-180.0
Maximum Axis Value	180.0
Grid Divisions	8
Snap to Grid	
Wrap Input	
Smoothing Time	0.0
Smoothing Type	Spring Damper
Damping Ratio	1.0
Max Speed	0.0
Vertical Axis	
Name	MovementSpeed
Minimum Axis Value	0.0
Maximum Axis Value	300.0
Grid Divisions	4
Snap to Grid	
Wrap Input	
Smoothing Time	0.0
Smoothing Type	Spring Damper
Damping Ratio	1.0
Max Speed	0.0
Analysis	
Horizontal Axis Function	None
Vertical Axis Function	None

Our game character has movement vectors in eight different directions, which means we need animations for walking in each of these directions. To implement character movement,



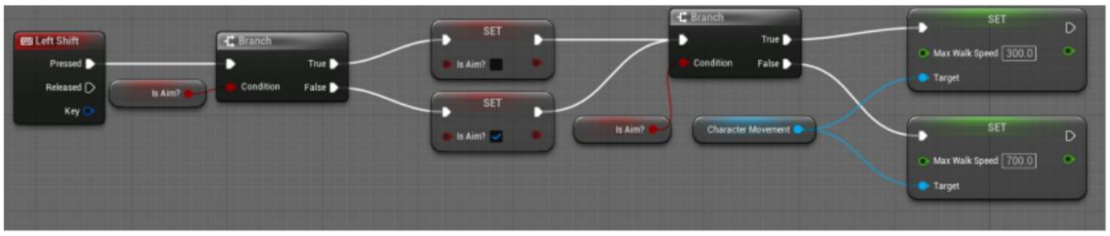
we first use IK Rigging to convert beginner animations, freely downloaded from an online store, into animations tailored to our own character's skeleton.

After acquiring the necessary animation assets for our character, we employ a blend space to mix these animations based on the character's movement speed and direction. This method is used to achieve realistic walking animations when the character is in an unarmed state. Blend space is an effective tool in game development for smoothly blending between different animations based on variable inputs, such as speed and direction, ensuring that character movements appear natural and responsive to player inputs.



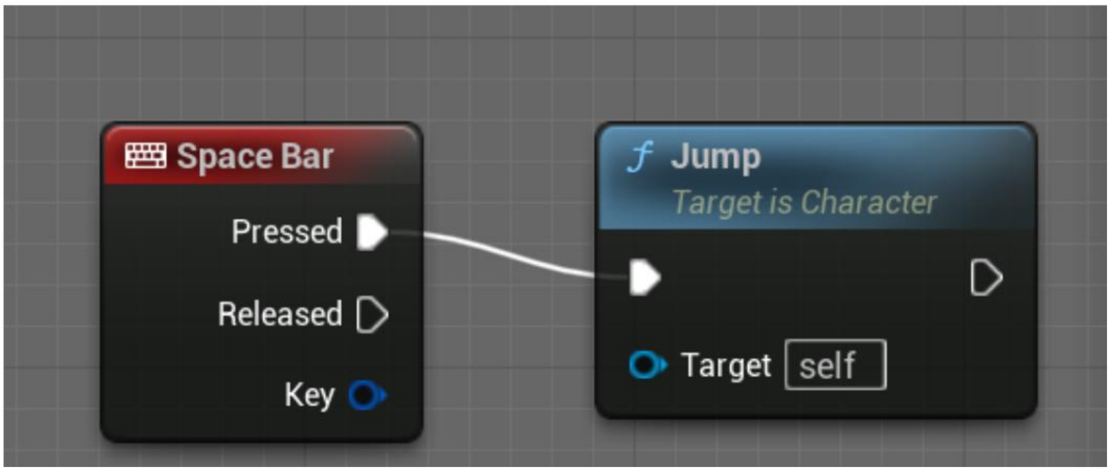
We had four different movement states. They are unarmed, with a pistol, with a shotgun, with a rifle,   each state has its own aim offset and different direction movement animation. and using a cache to reduce the cost of the system.

-Sprint



We have also implemented a sprinting feature to enable characters to sprint. While the character is sprinting, we impose certain restrictions: the character cannot shoot or reload while in the sprinting state. This adds a strategic element to the gameplay, as players must decide when to sprint, knowing that they cannot engage in shooting or reloading during this action. Such a mechanism balances the advantage of rapid movement with the inability to perform offensive actions, contributing to a more dynamic and tactical gaming experience

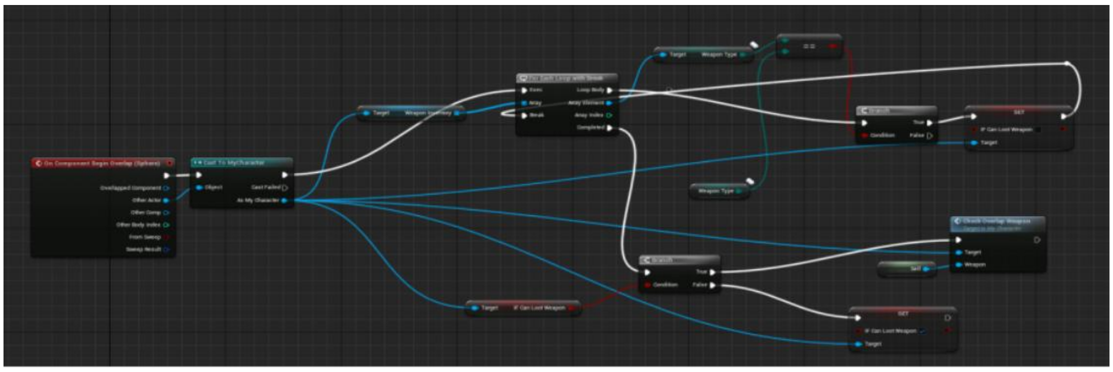
-Jump



It’s quite simple because Unreal had provided a solution to make actors jump.

B2.2 Basic weapon logic system

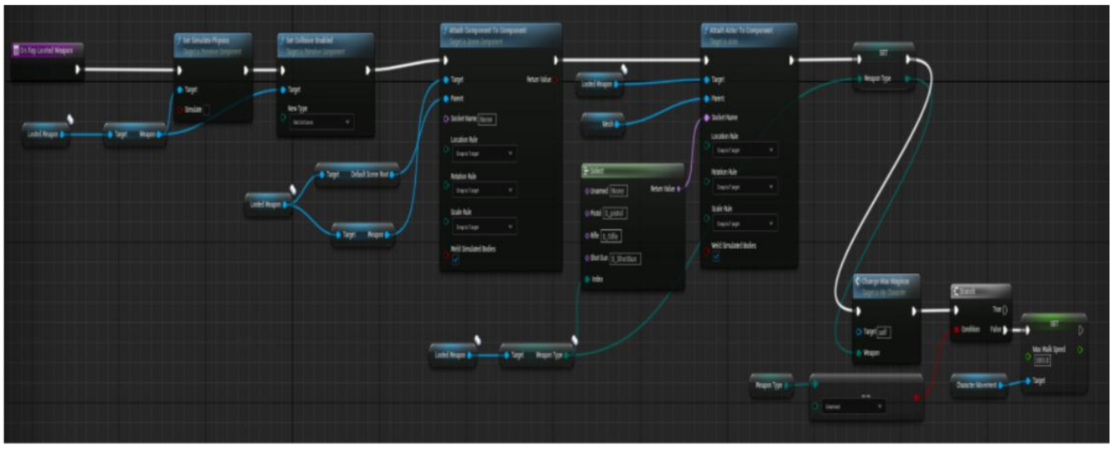
-Loot weapon

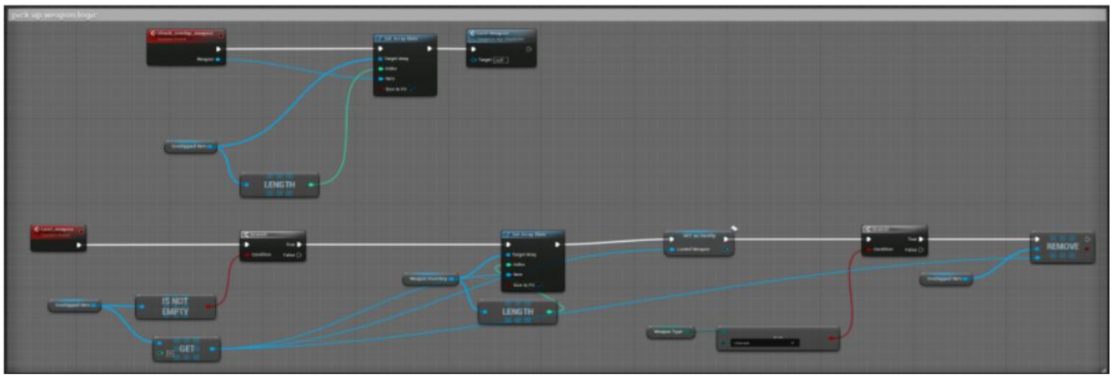


In our game, characters can use three types of weapons: a pistol, a rifle, and a shotgun. These weapons cannot be acquired simultaneously in our design. We created a weapon list and set a lapping box range for each weapon. Whenever a player's box overlaps with a weapon's box, we execute a for loop to check if the player already has an enumeration of that weapon. If they do, the 'can loot weapon' boolean value turns false.

preventing the execution of the 'attach weapon to the hand' function. Consequently, the weapon on the ground is not picked up and remains there. If the player is not already holding a weapon, then the weapon will be picked up, and the 'weapon type' enumeration variable will be set to the corresponding weapon type.

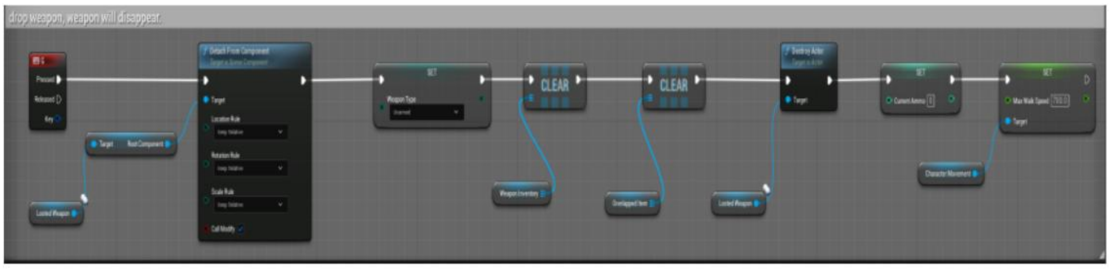
-Pick up weapon





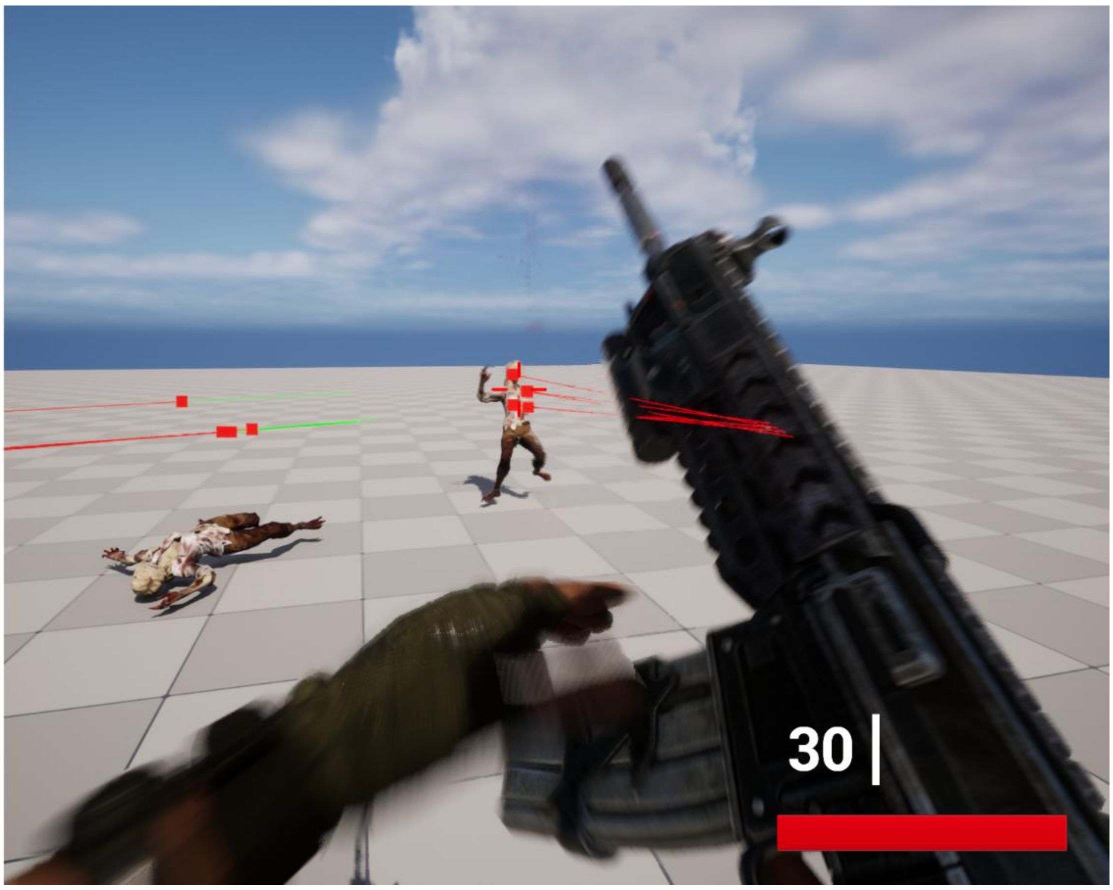
When we confirm that there is no weapon in hand, we can pick up a weapon. Picking up a weapon involves several steps. First, we need to remove the weapon's physical effects because our weapons have physical effects. The second step involves attaching the weapon to the corresponding socket on the character's skeleton. We have already set up specific sockets for each of the three types of guns on the character's skeleton. Additionally, we will perform another check to ensure the weapon type is consistent.

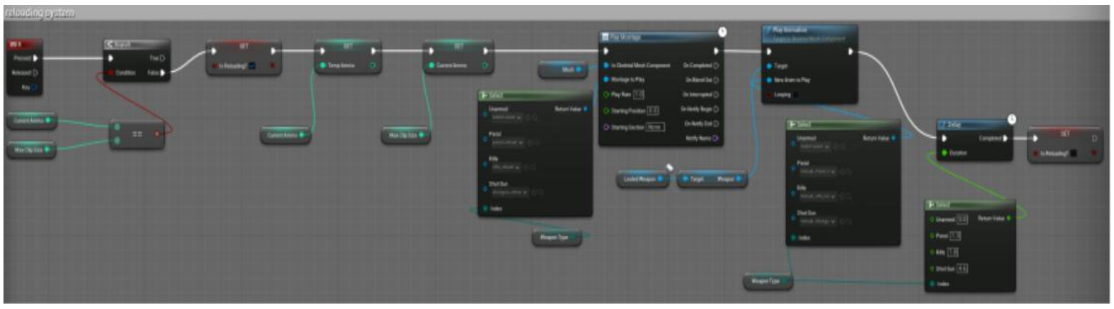
-Drop weapon



The logic for discarding weapons in our game is quite straightforward. We do not implement a mechanism to reposition discarded weapons; instead, they simply disappear. This is achieved by using a keyboard event to clear the player's list of held weapons and by destroying the actor. At the same time, the character's movement speed is restored, as different weapons affect the player's speed differently when held. This approach distinguishes between sprinting speed and the speeds associated with holding different weapons.

-Reload weapon



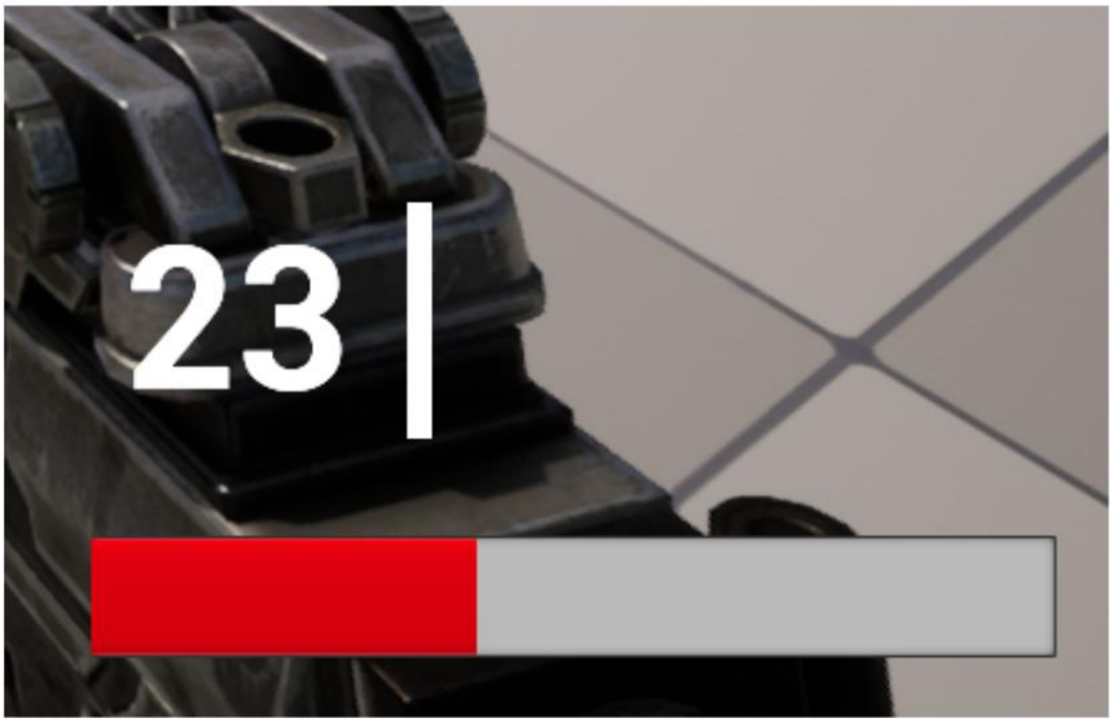


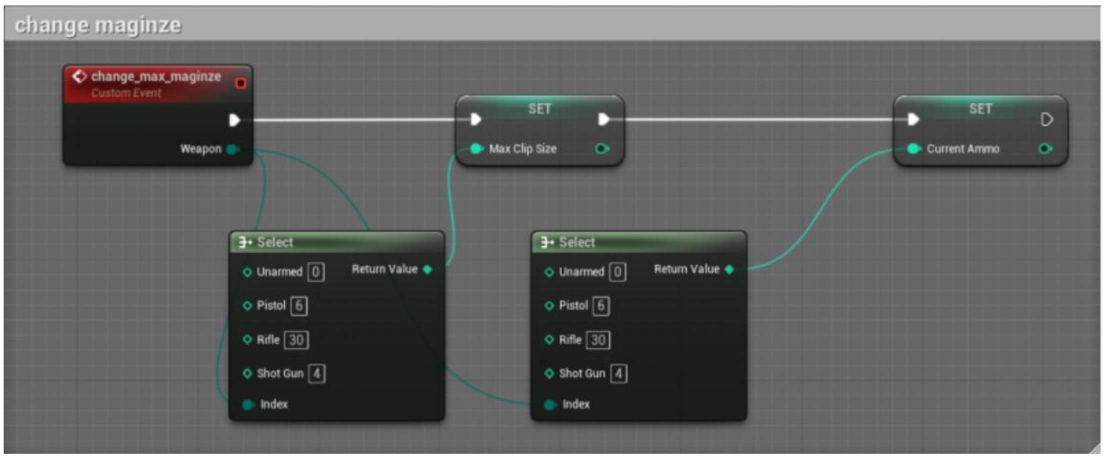
Each gun in our game has its own specific gun animations and reloading animations.

When initiating the reloading action, the function first checks if the gun is eligible for reloading. Our criteria for this are that the number of bullets must be less than the maximum magazine capacity, and the weapon should not already be in the process of reloading.

Once these conditions are met, the system plays different reloading animation montages based on the type of gun, along with the character's reloading montage. To ensure the reloading process is time-sensitive and matches the duration of the reloading animations, we have added a delay tailored to different types of guns. After the delay, the reloading status is set to false, and the bullet count is updated to the maximum magazine capacity.

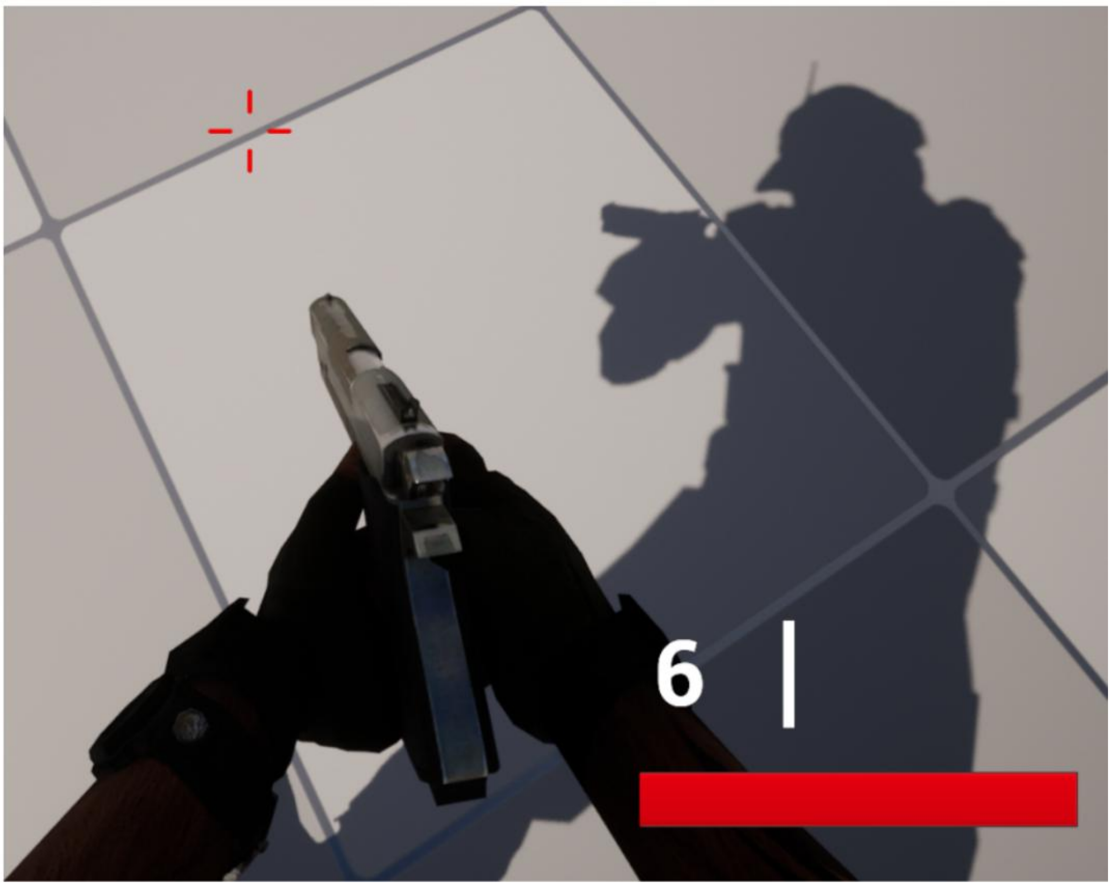
-Weapon’s Ammo count



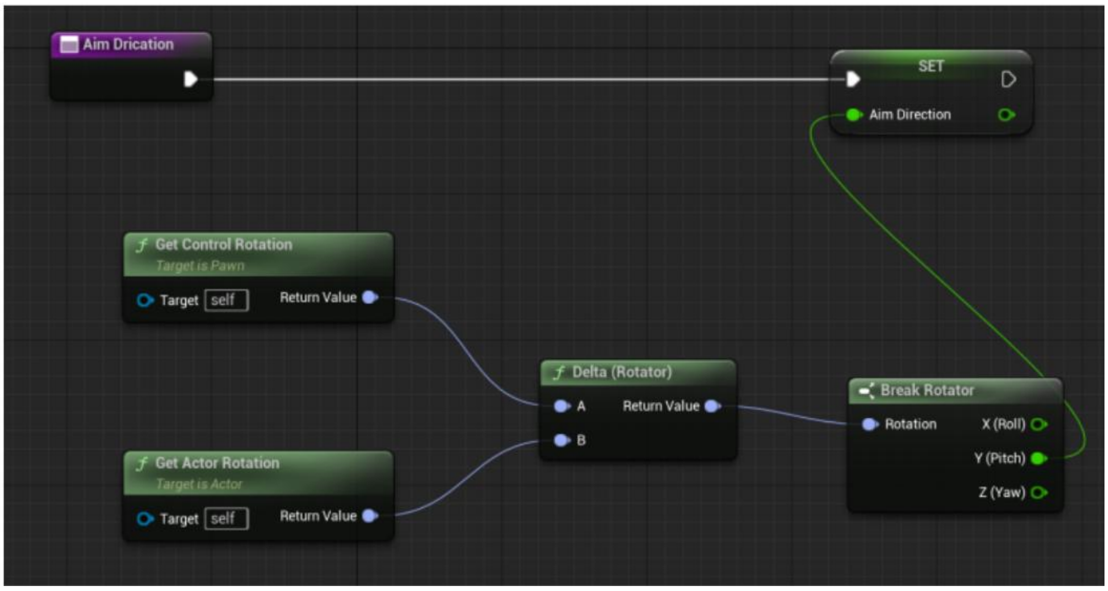


In our game, we define different bullet counts and initial bullet amounts for each firearm based on the gun's enumeration type. This means that each type of gun has its own specific total ammunition capacity and a defined number of bullets available at the start. This approach allows for variation in gameplay, as different guns offer different capacities and initial ammunition, influencing the player's strategy and choices in the game.

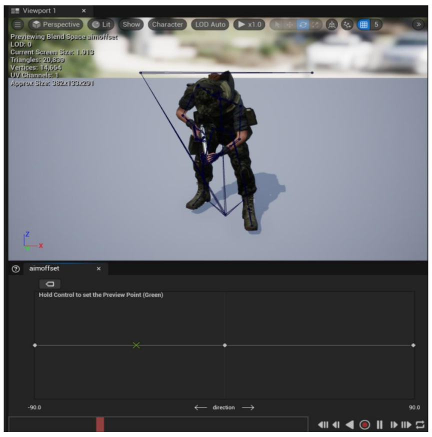
-Aim Offset

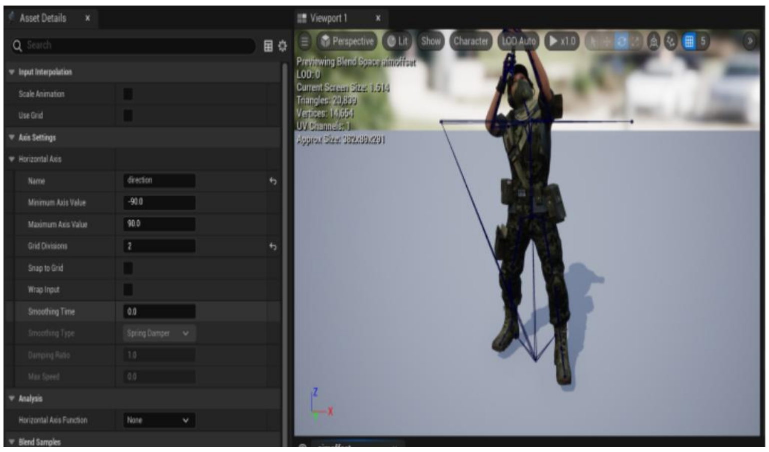


Aim offset is a crucial feature for FPS (First-Person Shooter) games. Without it, characters cannot aim based on the movement of the mouse. In our game, we have developed three different types of aim offsets corresponding to the three types of weapons we have: pistol, rifle, and shotgun. Each type of weapon has a unique aim offset that reflects its characteristics and handling, providing a more realistic and immersive aiming experience for the player. This differentiation ensures that aiming with a pistol, for instance, feels distinct from aiming with a rifle or a shotgun, both in terms of visual feedback and game mechanics.



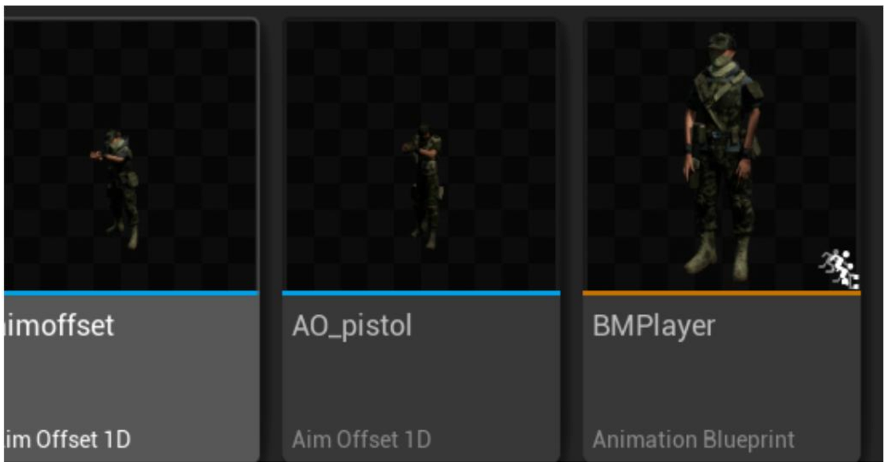
In this section, we calculate the aim direction post it to the aim offset features, and promote it to the variable.





Asset Details

Input Interpolation	
Scale Animation	☐
Use Grid	☐
Axis Settings	
Horizontal Axis	
Name	direction
Minimum Axis Value	-90.0
Maximum Axis Value	90.0
Grid Divisions	2
Snap to Grid	☐
Wrap Input	☐
Smoothing Time	0.0
Smoothing Type	Spring Damper
Damping Ratio	1.0
Max Speed	0.0
Analysis	
Horizontal Axis Function	None
Blend Samples	

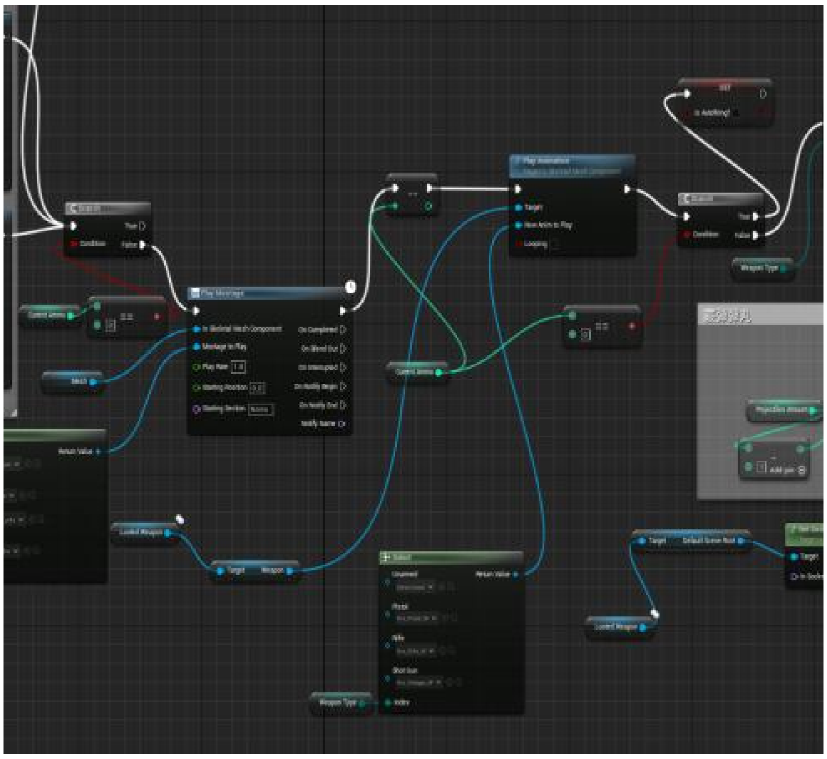


In BMPplayer, We use those vars to set down the aim offset. now our player can aim based on their mouse input. We made three types of aim offset based on different weapons types.

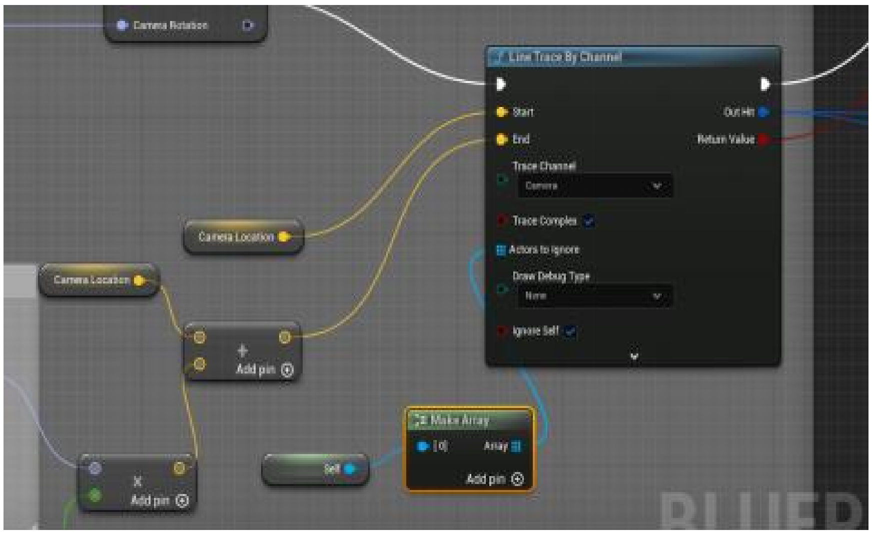
B2.3 Firearms system

-Firing

Shooting weapon is a mouse click event. we use the left mouse to shoot weapons.
First, when we shoot a weapon we will check the ammo count. if the ammo is 0, we can't shoot it. When we are reloading as well.
Then we will check the weapon type using enumeration. if the gun is AR, we will enable auto-fire logic. others will be left.   Then we will play the fire montage according to the weapon type.

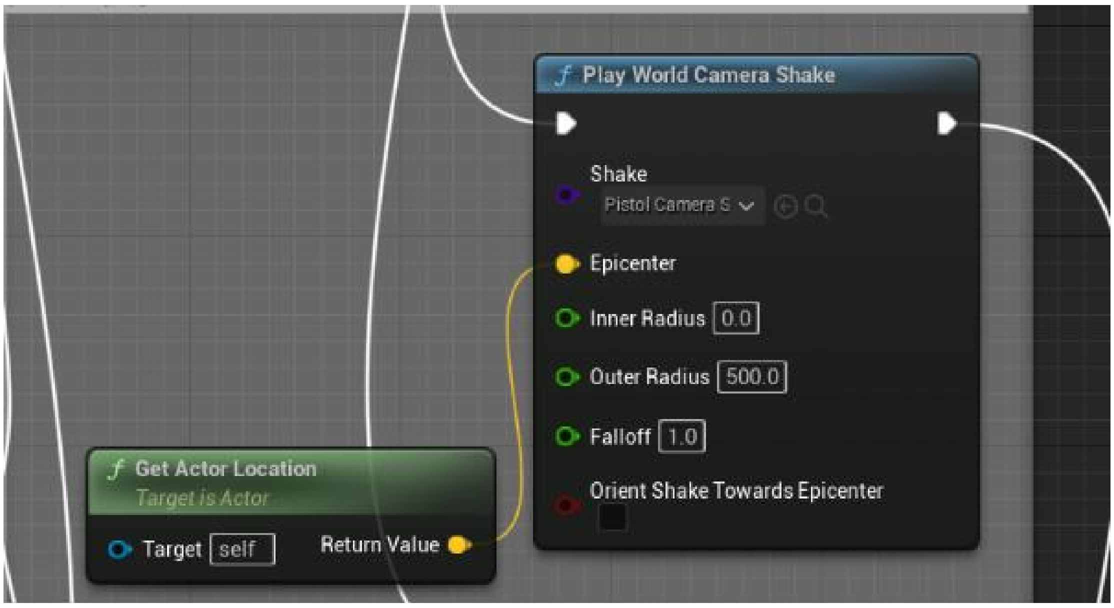


Then the system will simulate firing a gun by firing a line.
This process requires obtaining the player's coordinates and orientation.
The line will perform collision detection and return the first colliding object.



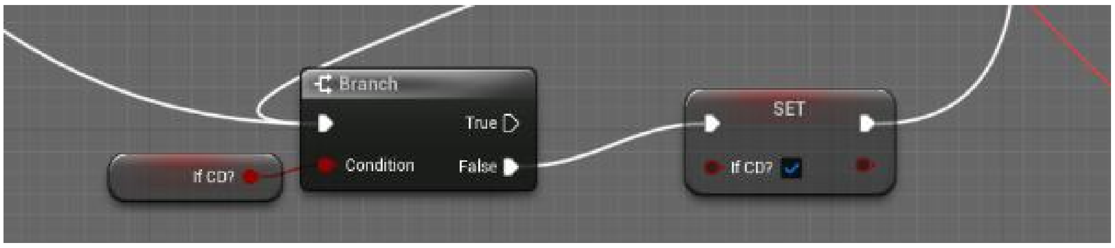
-Recoil simulation

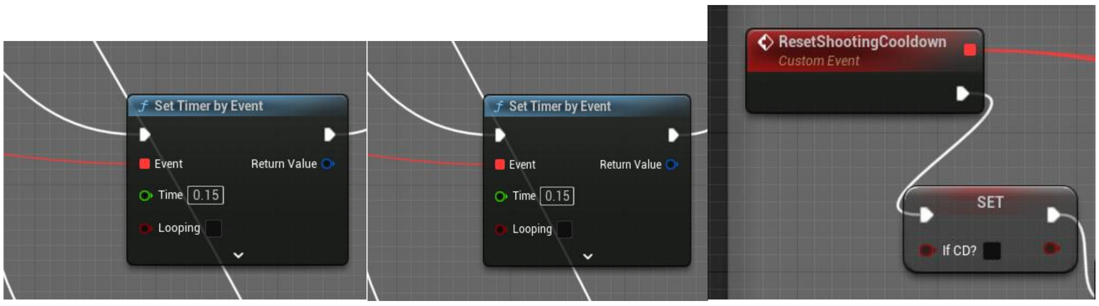
Using camera shake to simulate the gun recoil.



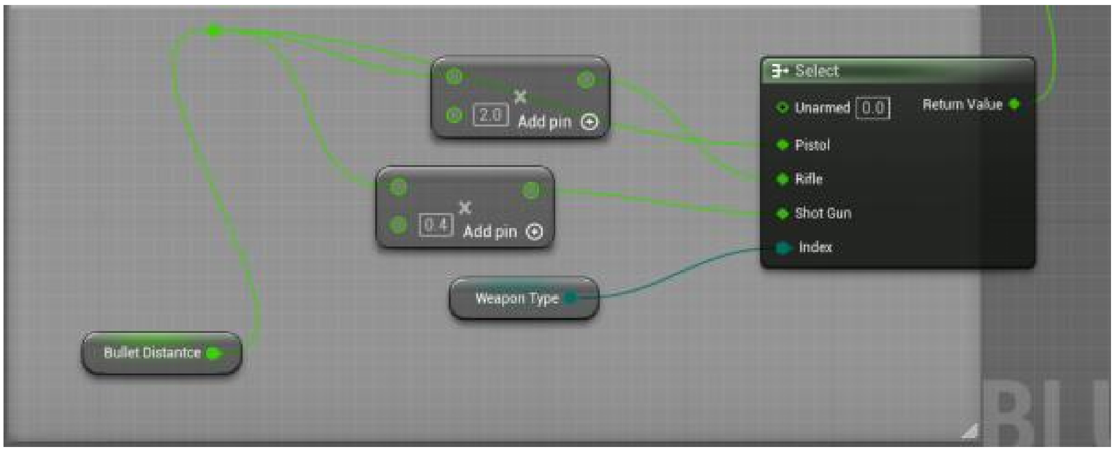
-Firing speed and effective range control

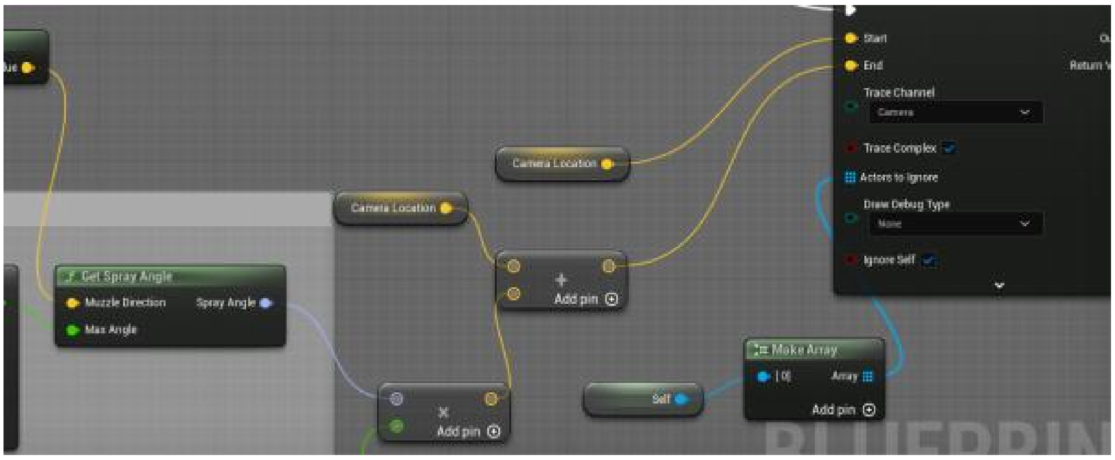
Controlling the gun's firing speed by setting a Boolean type variable and a timer.
variable becomes true after the gun fires, at which time the gun cannot fire again until the timer resets the variable to false





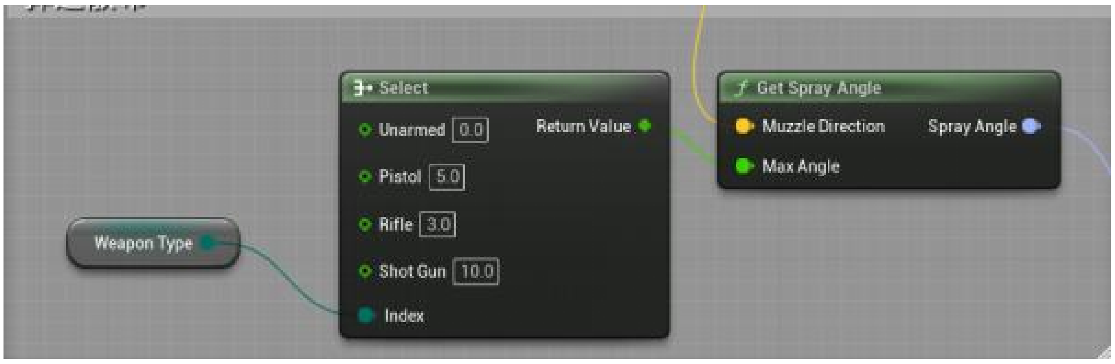
And the range of the gun is accomplished by controlling the end of the line (from line trace by channel).

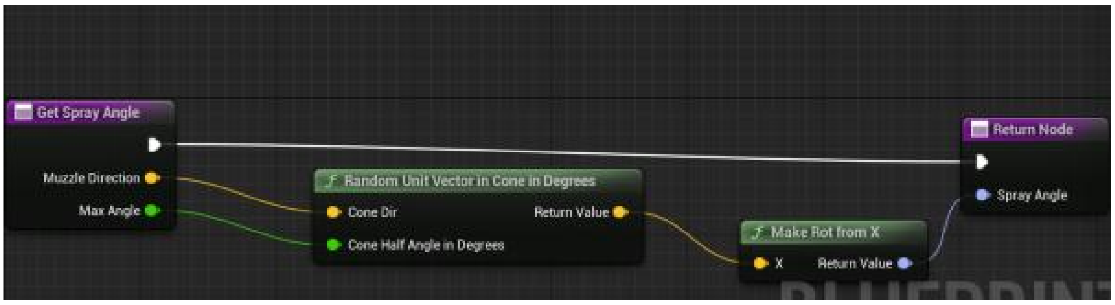




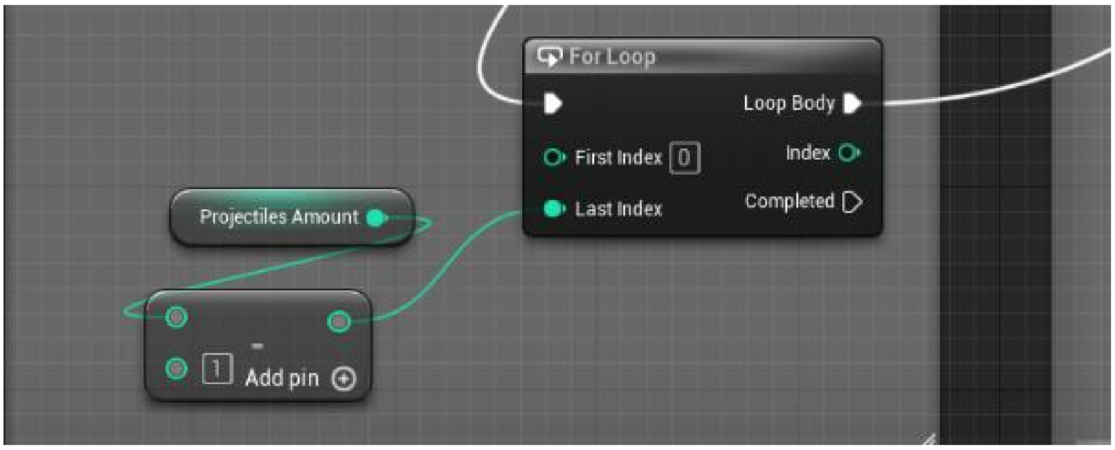
-Shooting dispersion and shotgun shooting simulation

Simulating gunshot dispersion by creating a cone angle.



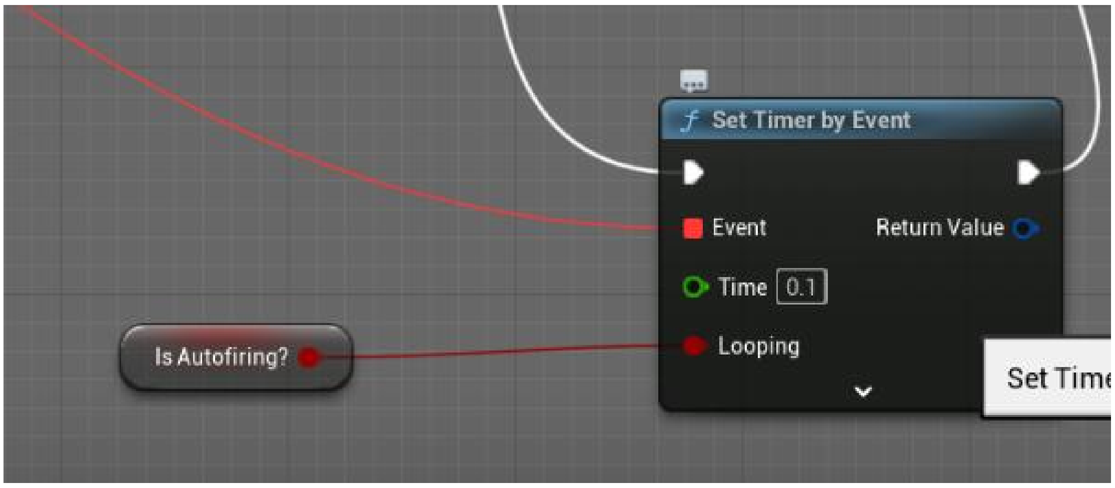


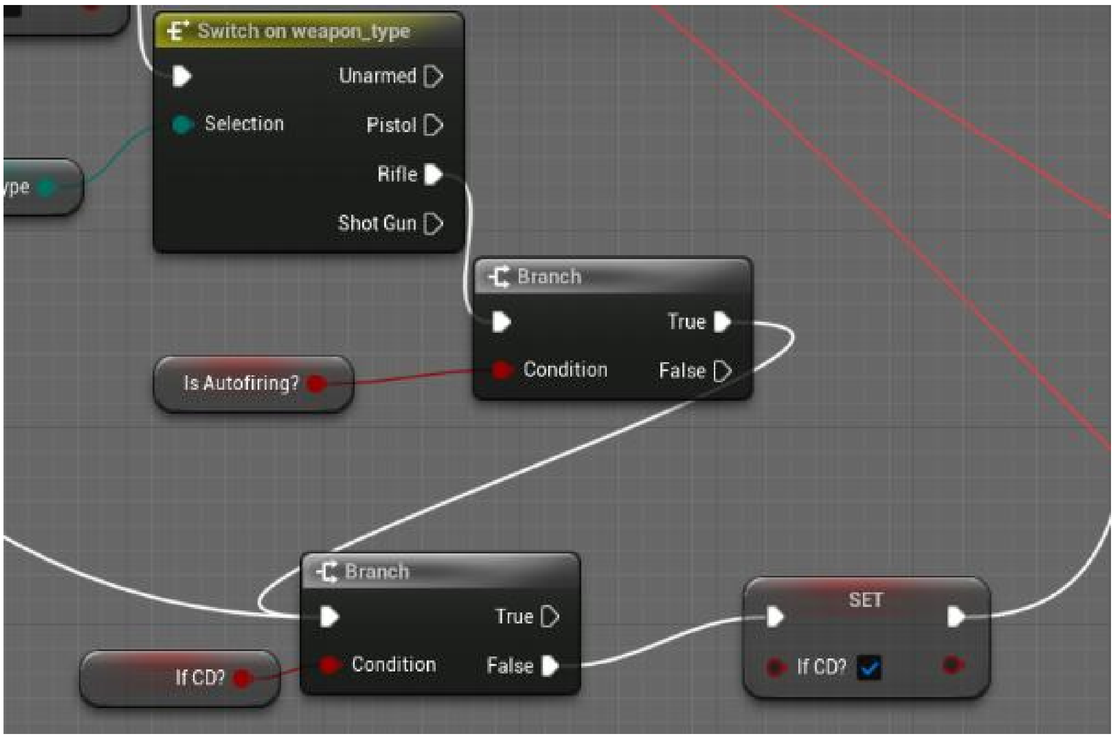
When the player is holding a shotgun, the process of looping the firing of rays is done a number of times corresponding to a set number of shotgun projectiles. This is equivalent to firing simultaneously in the player's actual experience.



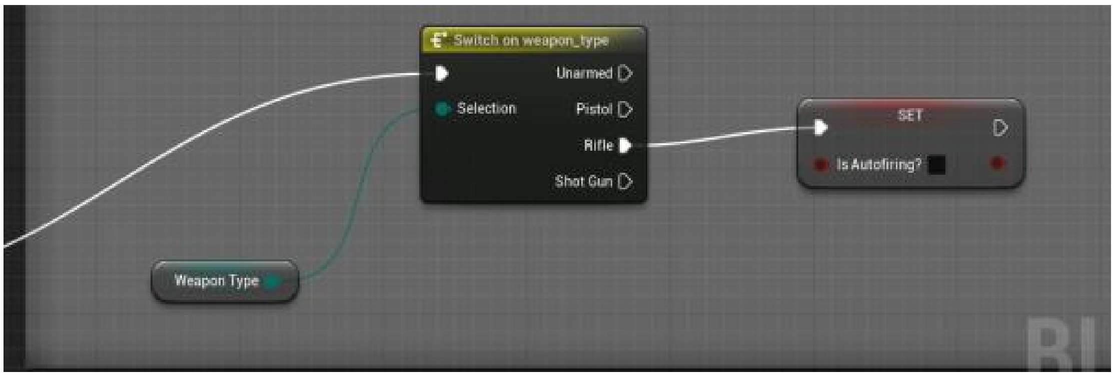
-AR auto-fire mode

When the player is holding an automatic rifle, tapping the left button triggers automatic firing.



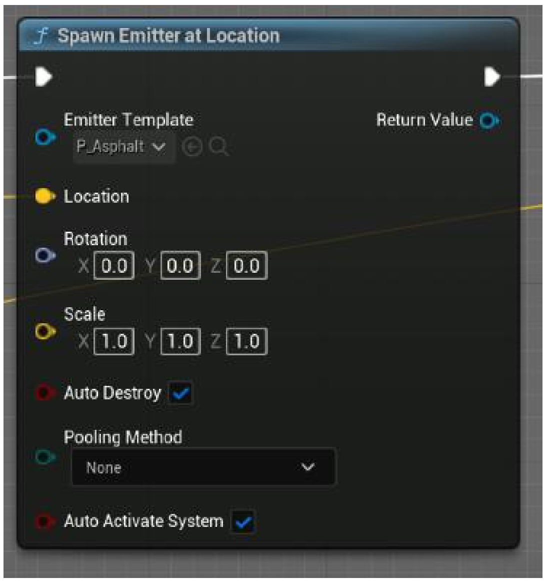


When the player releases the left mouse button, the loop of firing will stop.

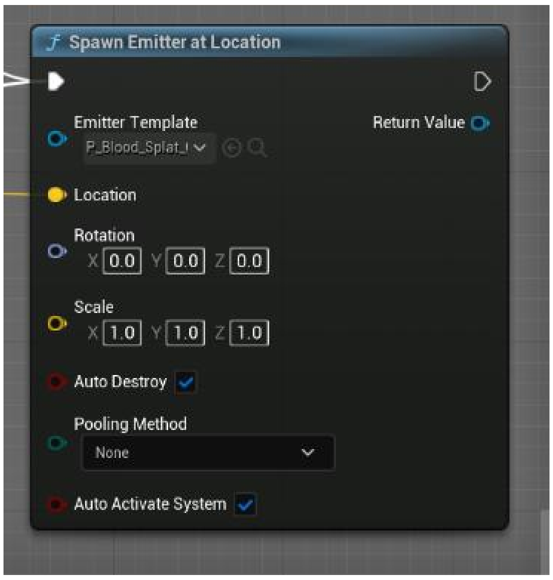


-Hit effect

Spawning emitter at where the bullet(line) hit to simulate the hit effect.
When a bullet hits the ground or other objects, it generates a rock splash effect.
And when it hits an enemy, it generates a blood spray effect.
(The FX materials here are from Epic store.)



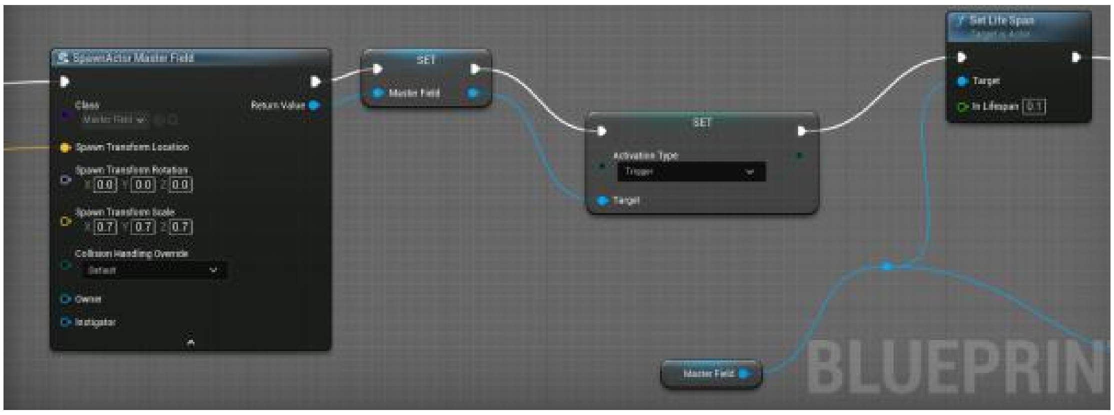
Spawn Emitter at Location
Emitter Template: P_Asphalt
Return Value
Location
Rotation: X 0.0 Y 0.0 Z 0.0
Scale: X 1.0 Y 1.0 Z 1.0
Auto Destroy ✓
Pooling Method: None
Auto Activate System ✓



Spawn Emitter at Location
Emitter Template: P_Blood_Splat_(
Return Value
Location
Rotation: X 0.0 Y 0.0 Z 0.0
Scale: X 1.0 Y 1.0 Z 1.0
Auto Destroy ✓
Pooling Method: None
Auto Activate System ✓

-Object destruction function

When a bullet hits an object, it creates a master field to blow up the destructible object, then disappears in a very short time.



SpawnActor Master Field
Class: Master Field
Return Value
Spawn Transform Location
Spawn Transform Rotation: X 0.0 Y 0.0 Z 0.0
Spawn Transform Scale: X 0.7 Y 0.7 Z 0.7
Collision Handling Override: Default
Owner
Instigator

SET
Master Field

SET
Activation Type: Trigger
Target

Set Life Span
Target is Actor
Target
In Lifespan 0.1

Master Field

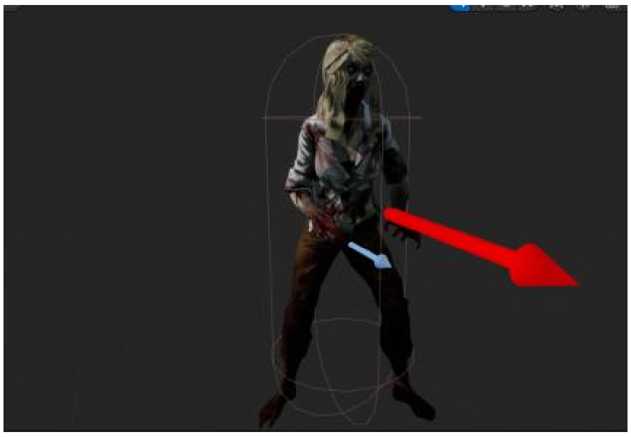
B2.4 Enemy system

-Improvements and additions to Phase 2

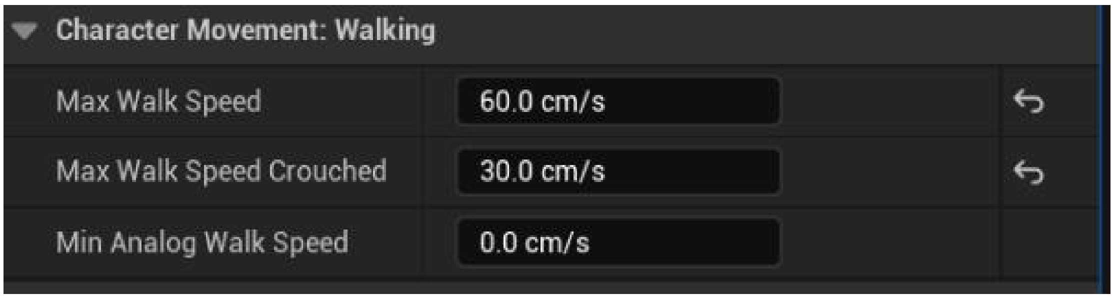
1.Enhanced Interactivity: This stage focuses on enhancing the interactivity and challenge of the game. Monsters are endowed with more complex behaviors and abilities.
2.Attack Functionality: Monsters can now attack the player, meaning they can initiate attacks and inflict damage on the player.
3.Pursuit Logic: In addition to attacking, monsters will also chase the player. This adds tension and strategic depth to the game, as players need to plan their movements and defense strategies.
4.Reaction to Being Hit: Monsters react when attacked, such as retreating, shaking, or making sounds. This enhances the immersion and realism of the game.
5.Combat Interaction: The interaction between monsters and players becomes more enriched, requiring players to use more complex tactics and skills to deal with the various behaviors of the monsters.
6.Increasing Monster Variety: Zombies will chase the player at the beginning, while tree monsters and flying insects patrol within a range and wait for the player to approach before initiating a chase.And a Boss has been added to provide players with richer combat challenges through the AI behavior tree.

- common enemy: zombie

Zombies are the most common enemy in this game.
They have a moderate life and attack damage.

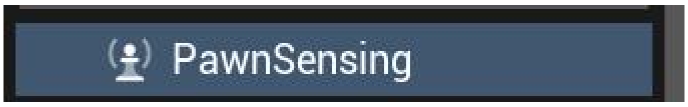


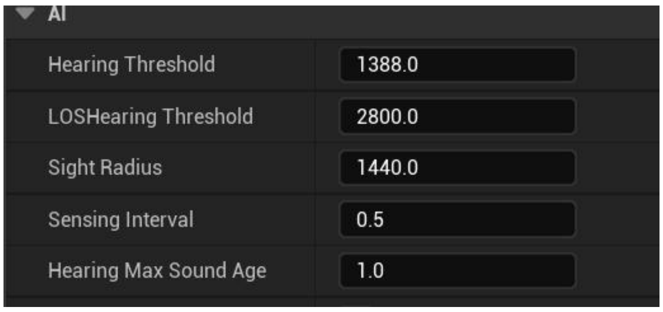
Normally it moves slowly, but when the player gets close to it, it will dash toward the player at a higher speed.


Character Movement: Walking

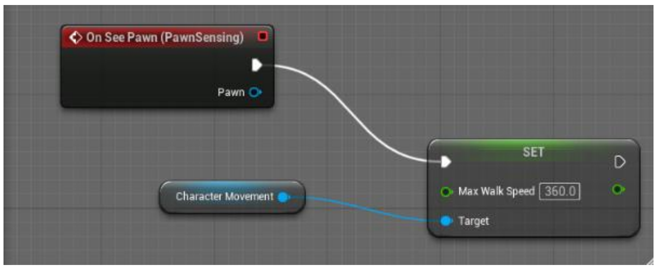
Max Walk Speed	60.0 cm/s
Max Walk Speed Crouched	30.0 cm/s
Min Analog Walk Speed	0.0 cm/s

This feature is accomplished by creating enemy PawnSensing.


PawnSensing


AI

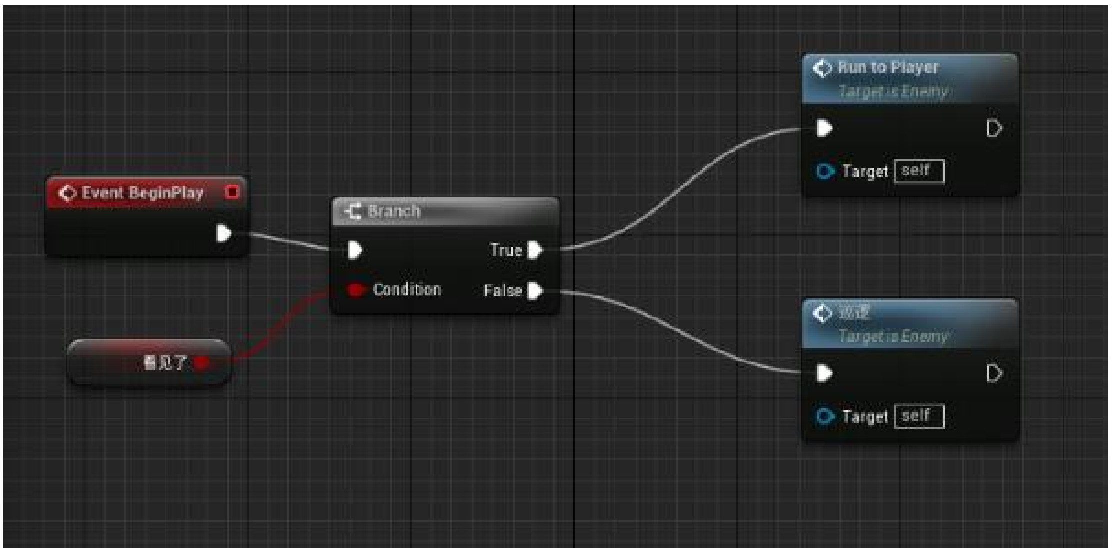
Hearing Threshold	1388.0
LOSHearing Threshold	2800.0
Sight Radius	1440.0
Sensing Interval	0.5
Hearing Max Sound Age	1.0

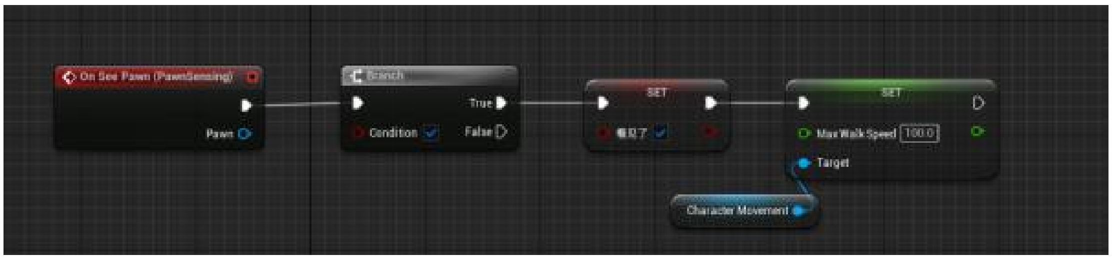


（ Model and texture and animations resource ： https://smartpoly.gumroad.com/l/ZombieFPSFiles）

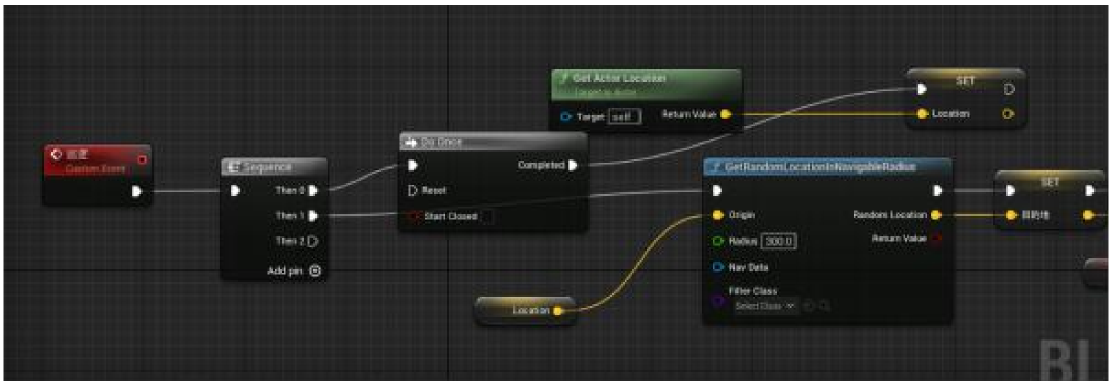
-Roaming monsters: tree monsters and flying insects

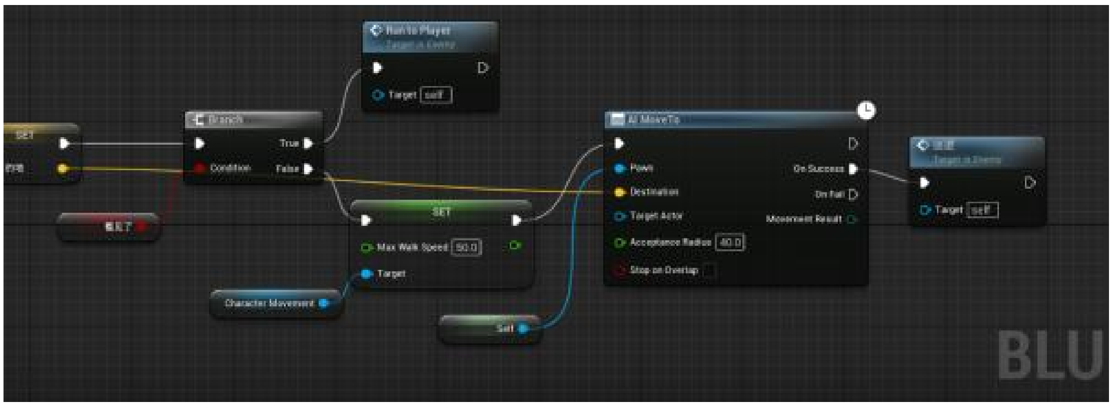
These two monsters actually use the same blueprint design. Unlike zombies, they will patrol within the range and attack the player only when the player comes into view. Among them, tree monsters move slower, have higher attack power, and have higher health volume, while flying insects move faster, but have lower attack power and have very low health volume.





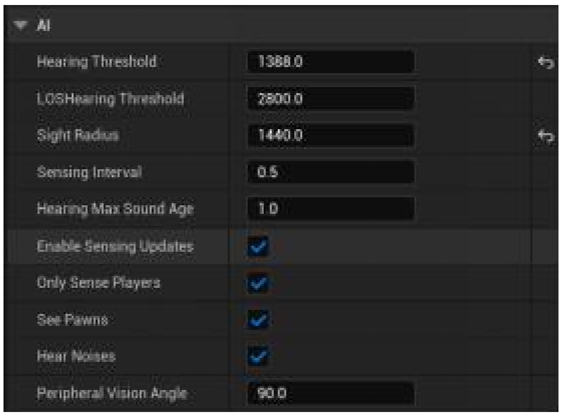
Determine behavior patterns through a variable and a branch to determine which action the monster should choose. The meaning of this variable is 'player spotted'. If true, then my monster will run towards the player; if false, the monster will choose to patrol





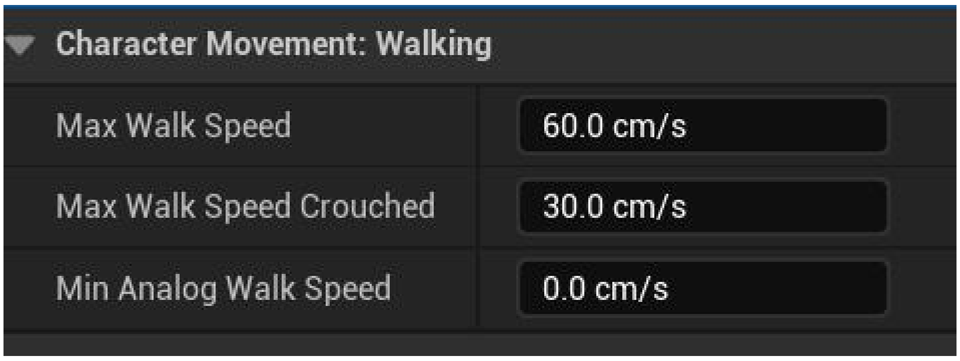
For the implementation logic of patrol, a sequence is first used to stipulate that the current position of the monster is obtained first, otherwise the monster will gradually move out of the patrol range. Then, get the random location the monster moved to

·Dashing when the player is near



AI

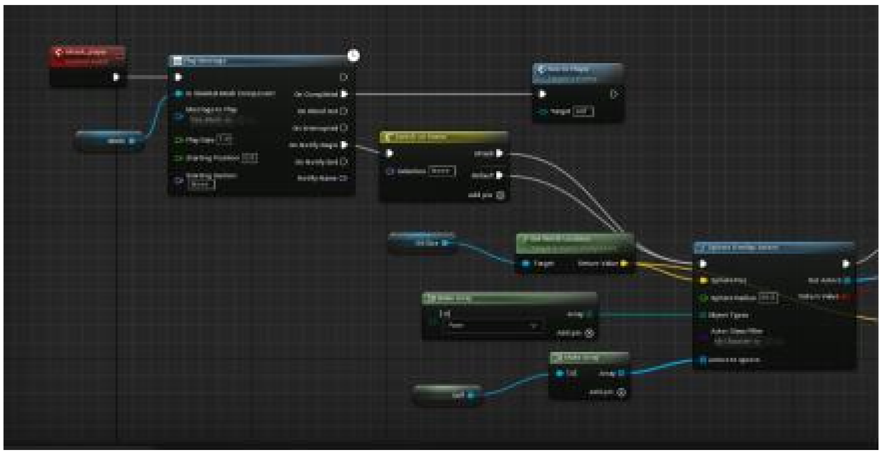
Hearing Threshold	1388.0
LOSHearing Threshold	2800.0
Sight Radius	1440.0
Sensing Interval	0.5
Hearing Max Sound Age	1.0
Enable Sensing Updates	✓
Only Sense Players	✓
See Pawns	✓
Hear Noises	✓
Peripheral Vision Angle	90.0

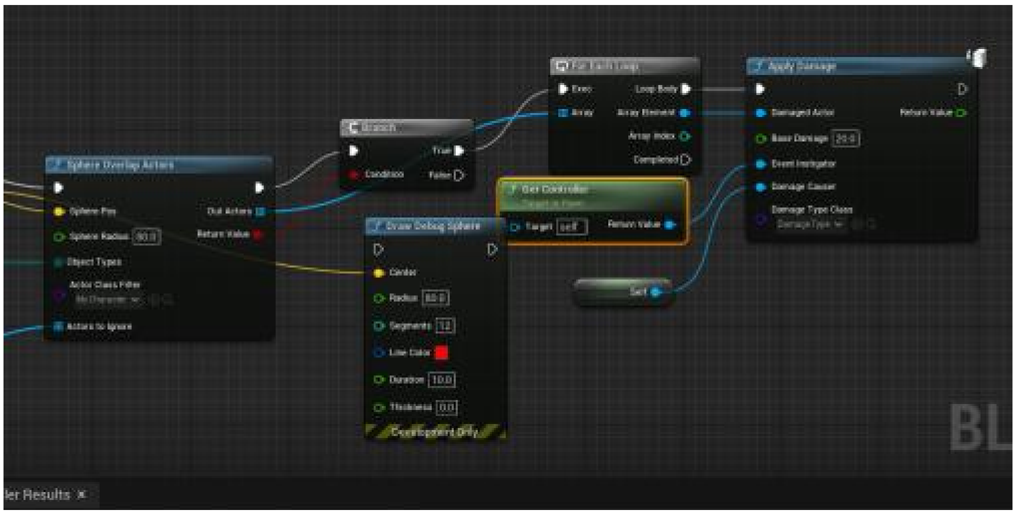


Character Movement: Walking

Max Walk Speed	60.0 cm/s
Max Walk Speed Crouched	30.0 cm/s
Min Analog Walk Speed	0.0 cm/s

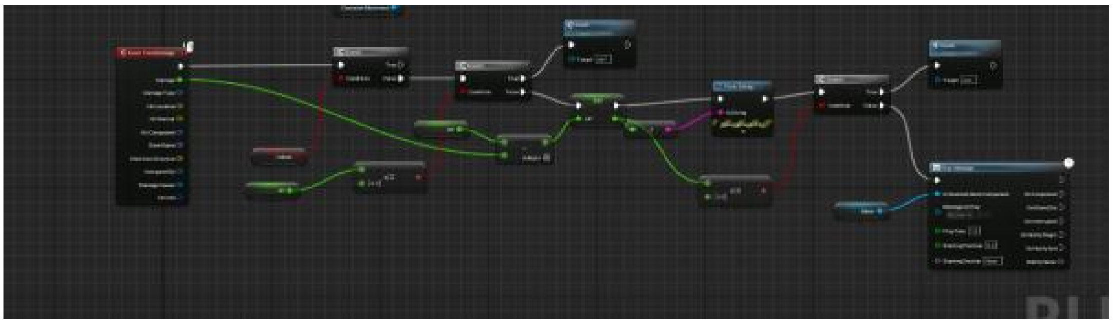
·Attack





each attack between the monster and the player is realized based on the collision detection of the HIT BOX. When there is continuous overlap, continuous attacks will occur. If they separate, it will break out of the attack loop and continue to pursue the player

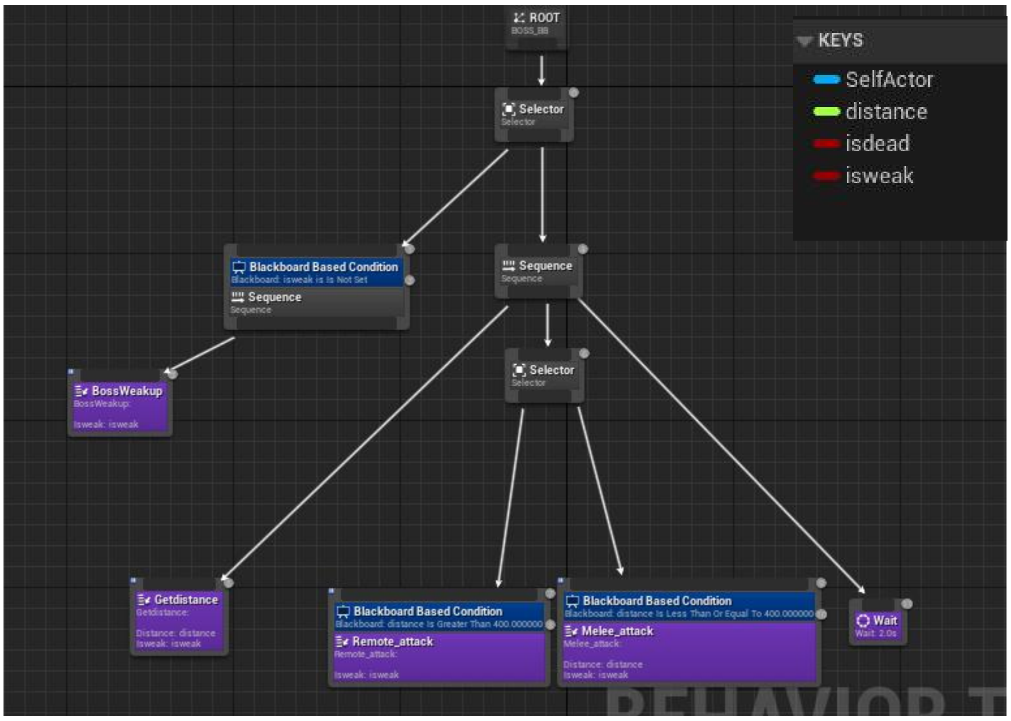
·Be attacked



-BOSS:Dragon

The boss has two attack modes: melee and ranged. When the distance between the player and the boss exceeds 400 units, the boss will remain stationary and launch fireballs at the player, causing damage. Conversely, when the distance between the player and the boss is less than 400 units, the boss will engage in melee attacks and pursue the player. If the player and the boss increase their distance again, the boss will resume its ranged attacks from its position.

1. AI Behavior Tree

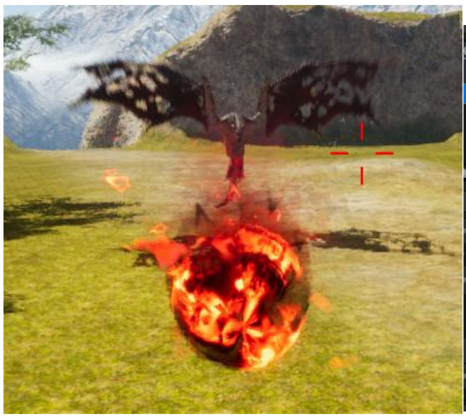
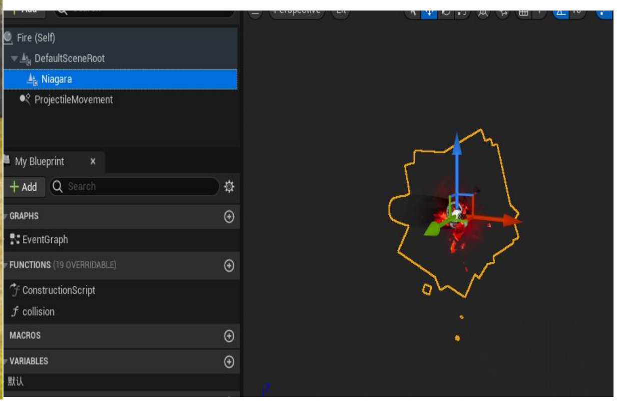


Firstly, the initial SELECTOR node sequentially executes an animation sequence for the Boss's awakening and sets *isweak* to true. Since the decorator condition for this node is that isweak is not set, the BossWeak task will only execute once. Subsequently, the topmost Selector will always choose the Sequence node for execution. It starts by determining the distance, updating the distance variable, and linking it to the blackboard's *distance* variable. This allows the second

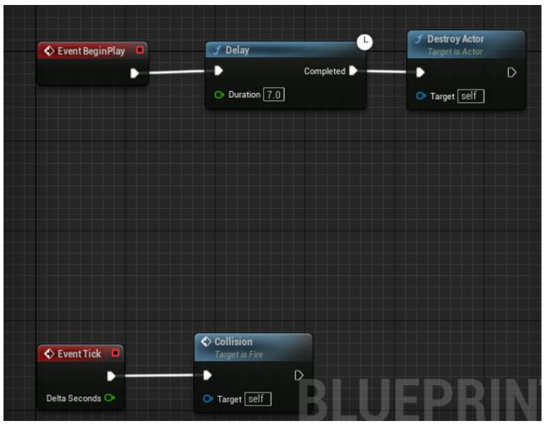
Selector node to choose the appropriate attack mode. Either *Remote_Attack* or *Melee_Attack*—as soon as one of them successfully executes, there will be a wait time of 2 seconds before making the next decision on the attack mode.

2. Remote attack

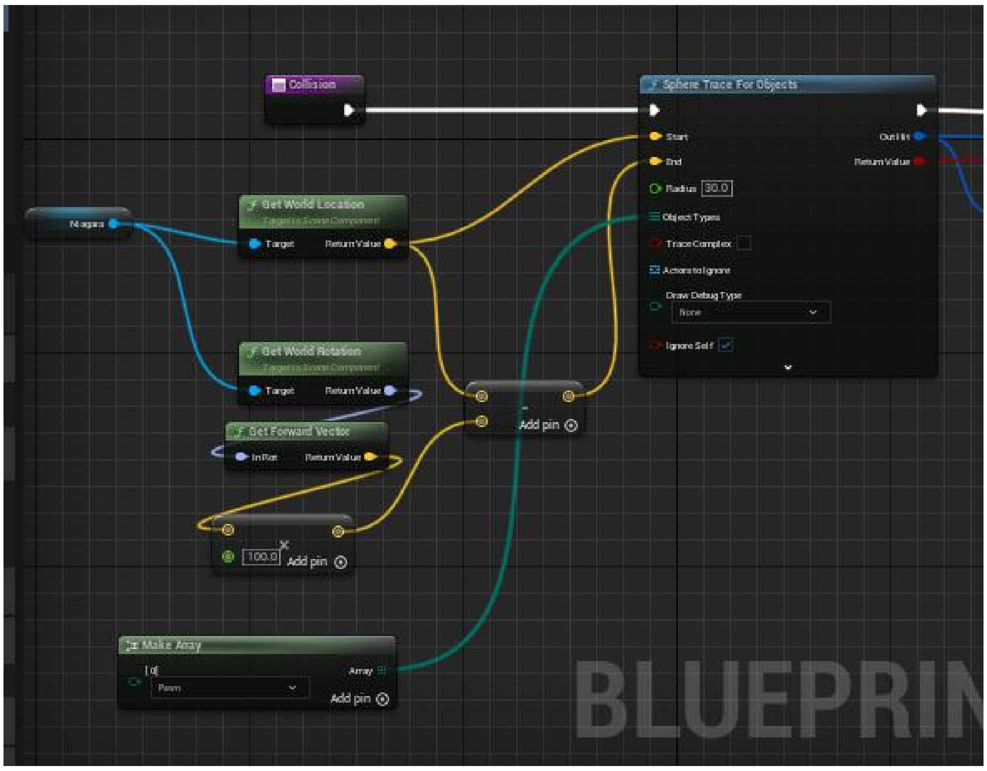
The ranged attack is implemented by attaching a collision box to the "fireball" and the boss's neck position. From this position, fireballs are launched towards the player. Collision detection with the player is carried out through the fireball's actor blueprint.

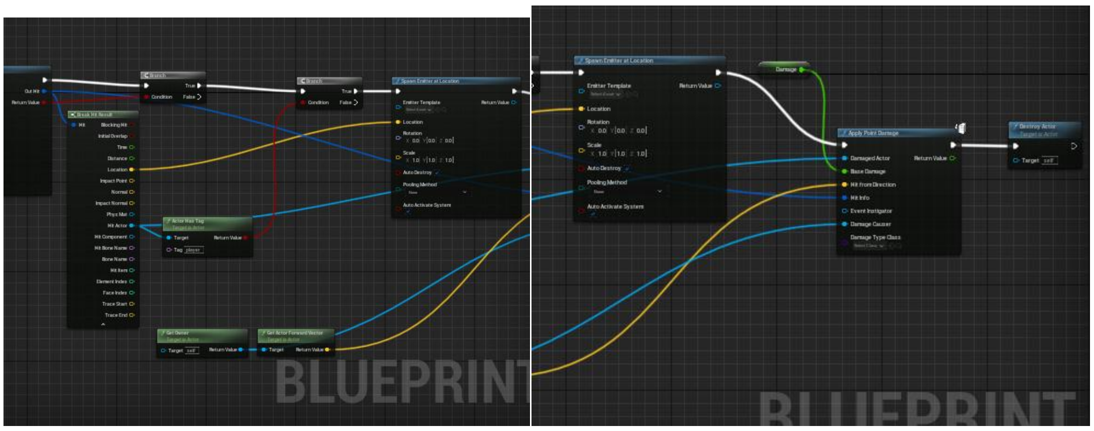
In this implementation, I used Niagara to add particle effects to the fireball



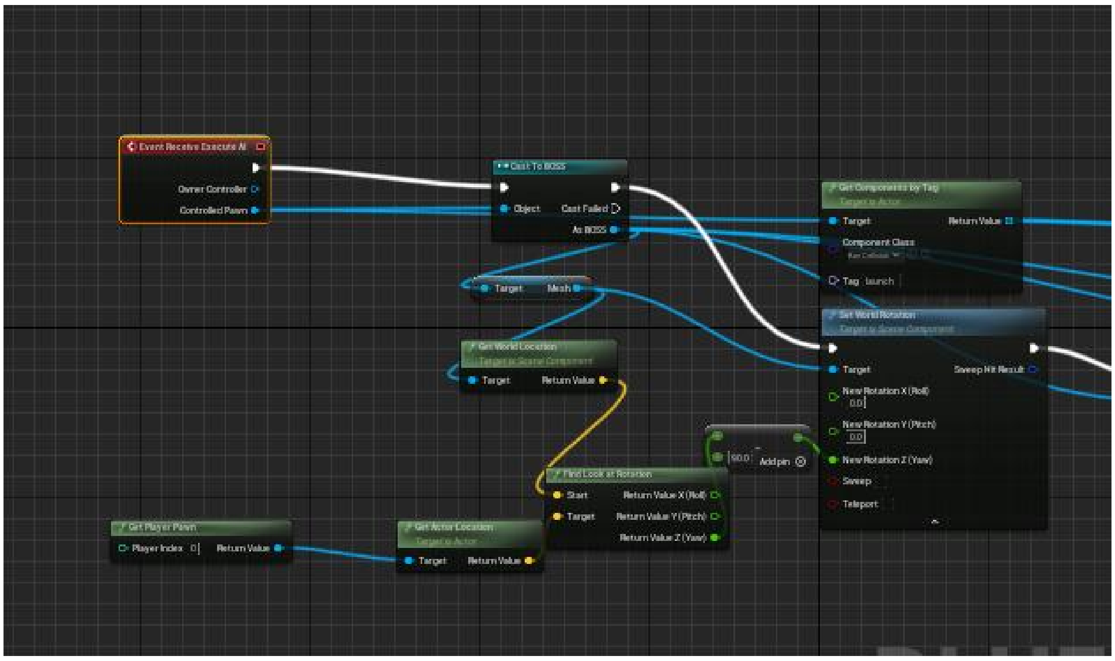
The fireball is programmed to disappear after seven seconds to avoid consuming resources. A tick timer is used to check for collisions with the player. If a collision occurs, a collision function is executed. This function operates as follows:



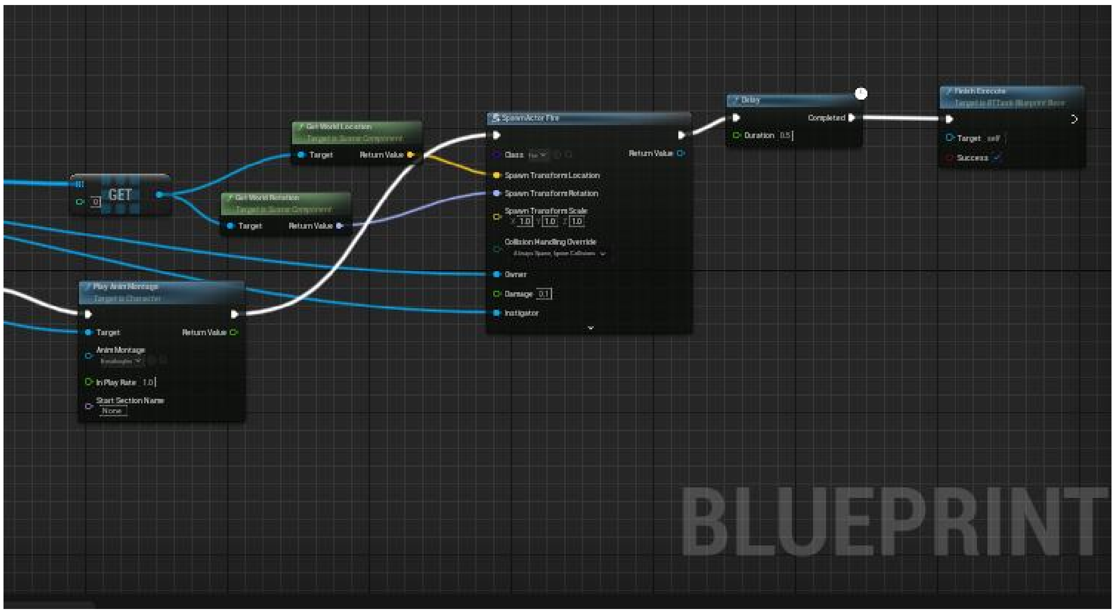
To direct the fireballs towards the player, obtain the forward vector. This vector will guide the fireballs to launch in the direction of the player.



If the attack hits the player, it will inflict damage on the player and then destroy itself.
In the implementation of Remote_Task within the AI behavior tree, the process begins by obtaining the positions of both the player and the Boss. Then, using the 'Find out location' function, it determines the necessary component. Since the Boss's mesh in the Boss blueprint is positioned at z:-90, the same offset (subtracting 90 from z) is applied here.

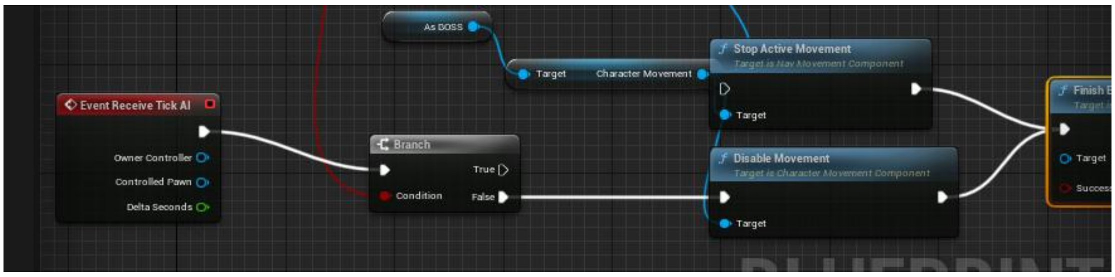


Following this, an animation montage is used to make the monster perform a fire-breathing action. Subsequently, a fireball is generated from the Boss's Collision Box, which is the point of origin. At this moment, the fireball follows the previously described logic, launching towards the player.

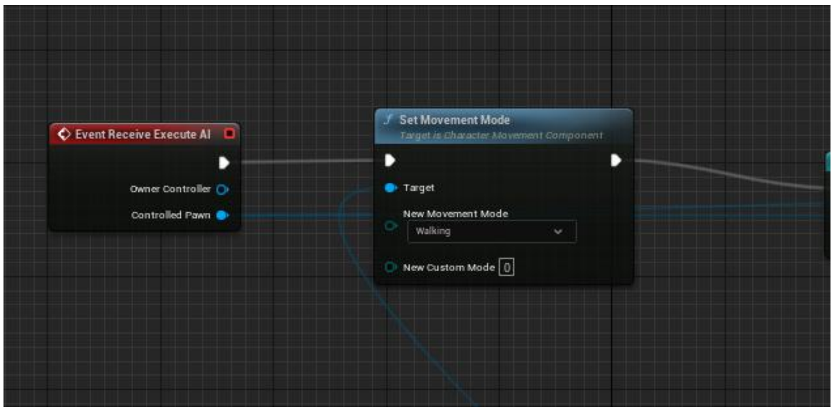


3. Melee attack

The Melee_Attack task is similar to the previous three monster behaviors, but there is a crucial aspect to consider: when the player moves beyond a 400-unit range, the Boss must cease pursuing the player. However, the AI MOVE TO action, unaffected by the completion of the AI's current task, will continue to direct the Boss towards the player's last known location. To address this, a tick event is set up. This event uses the Disable Movement function to change the movement component's mode to None, effectively stopping the Boss's pursuit when the player is out of range.



Correspondingly, it's necessary to set a new mode for movement in the AI execution event, to facilitate the possibility of the Boss resuming pursuit in close combat scenarios with the player.

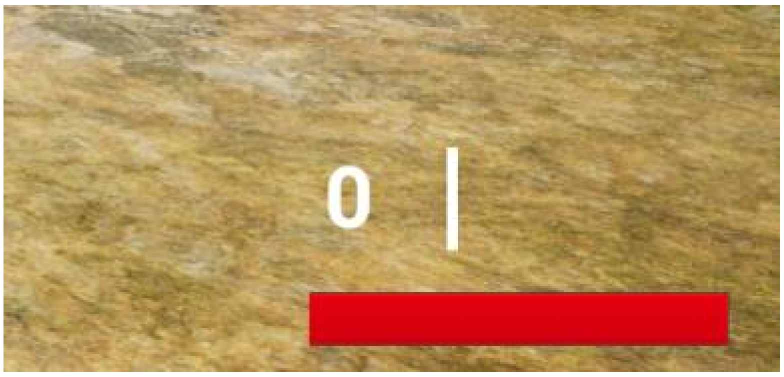


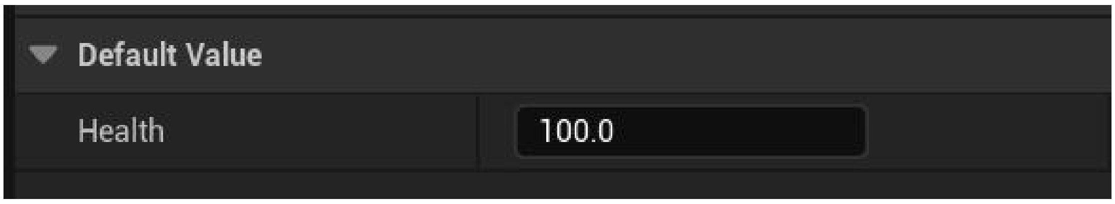
B2.5 Combat Systems

-Player death condition

There are three situations in the game that trigger a player's death:

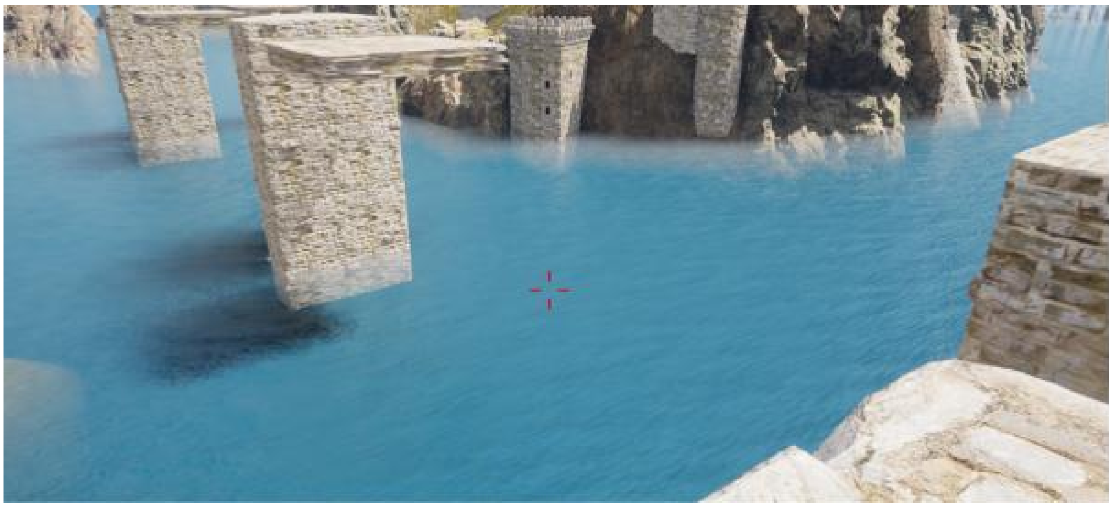
1. When the player's health level reaches zero, usually this is caused by a monster's attack.



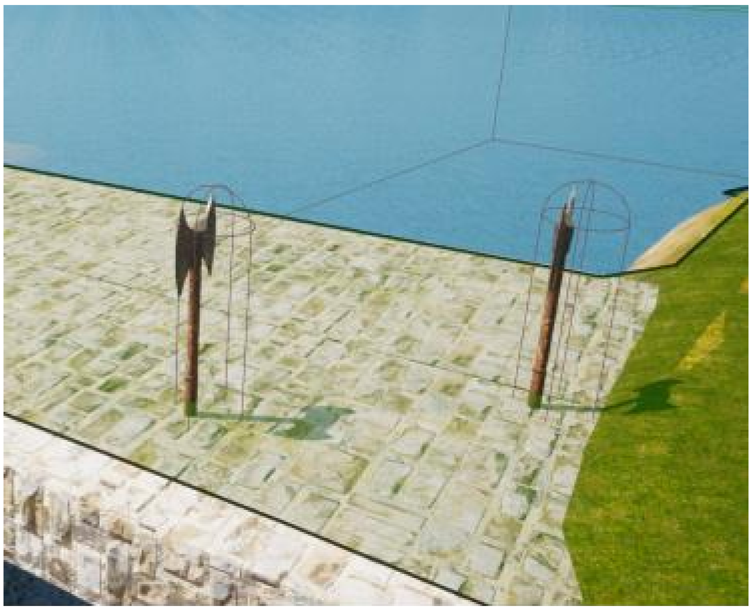

Default Value

Health 100.0

2. Players falling out of the map.

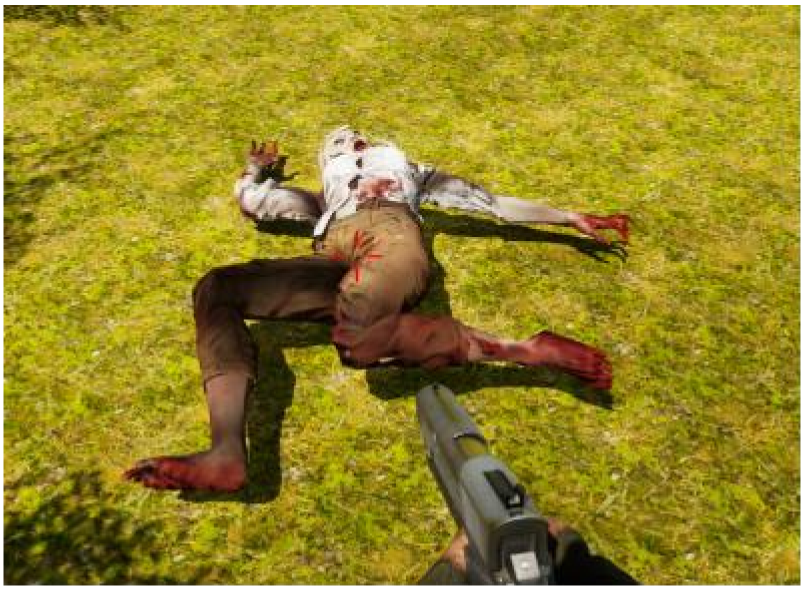


3. The player gets hit by a spinning axe trap.



-Enemy death condition

The death of an enemy in the game can only be triggered by the player attacking it until it reaches zero health.



Advanced

Default Value

Health 100.0

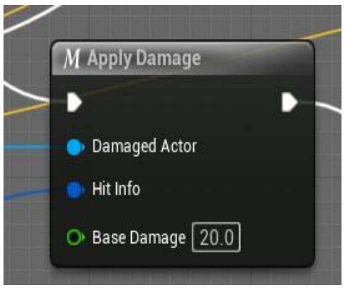
-Damage of different guns

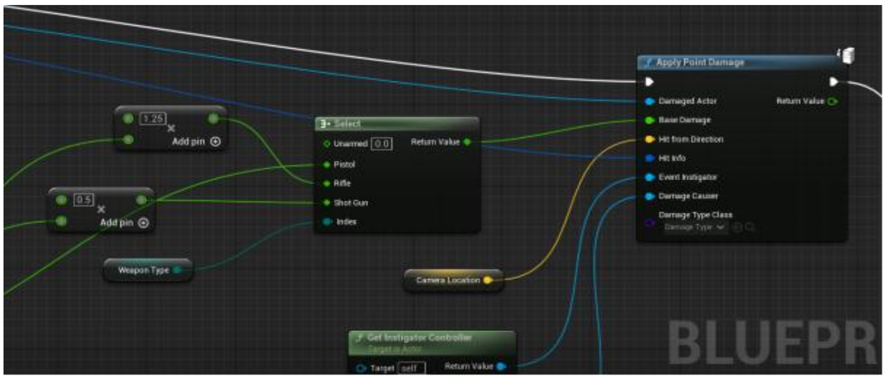
In addition to the speed and range of guns in the game, the damage they can do is different as well.

Pistols can only deal 20 points of base damage per shot.

AR can deal 20 base damage multiplied by a factor of 1.25, or 25 points per round.

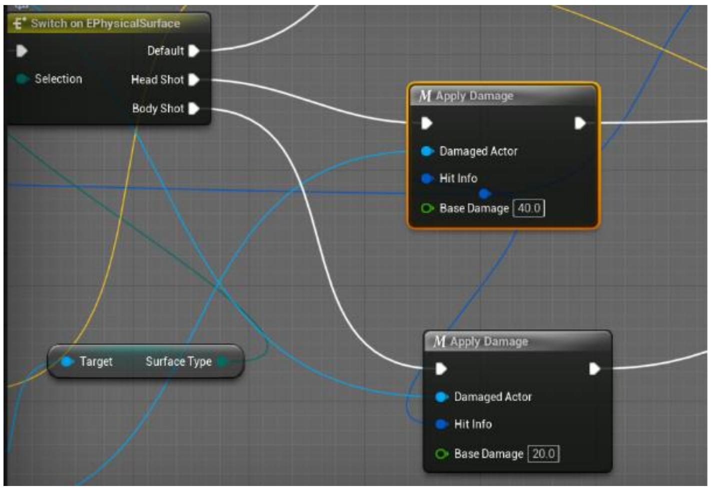
Shotguns deal 20 base damage per round times 0.5, or 10 points. This may seem far less than what the other two firearms can do, but in reality, the shotgun is capable of shooting multiple bullets at the same time with each shot. Assuming that ten bullets are fired in a single shot, the theoretical total damage dealt is 100 points.





-Headshot

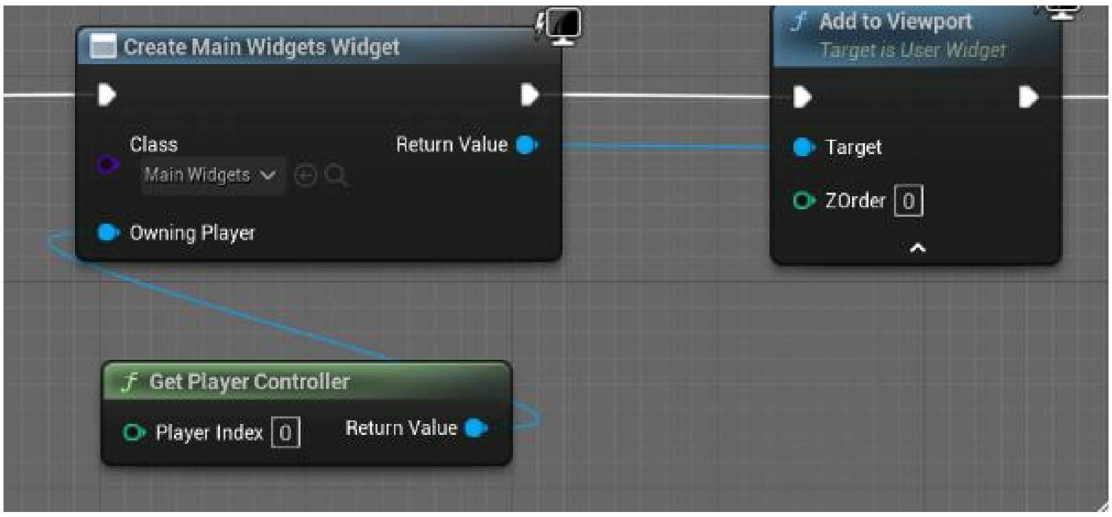
Enemies in the game have head determination, and when they are shot in the head, the base damage 20 dealt by the gun will increase to 40 points.



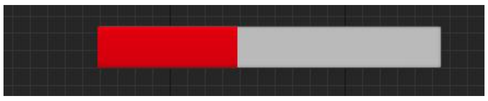
B2.6 UI

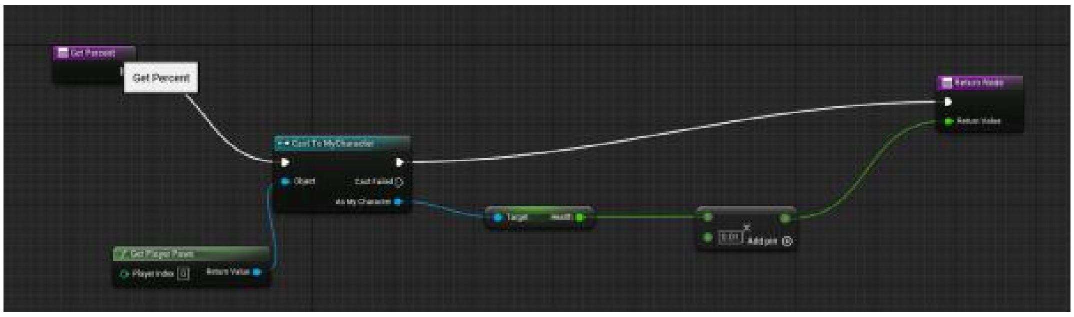
-Crosshair

There will be a crosshair in the center of the player's screen, but this doesn't represent where the bullet will hit, it's just an indication of the center of the shot's dispersion.



-Health bar



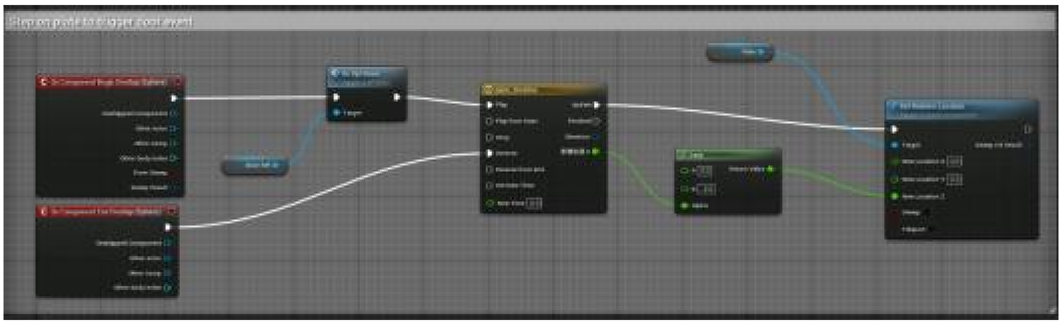


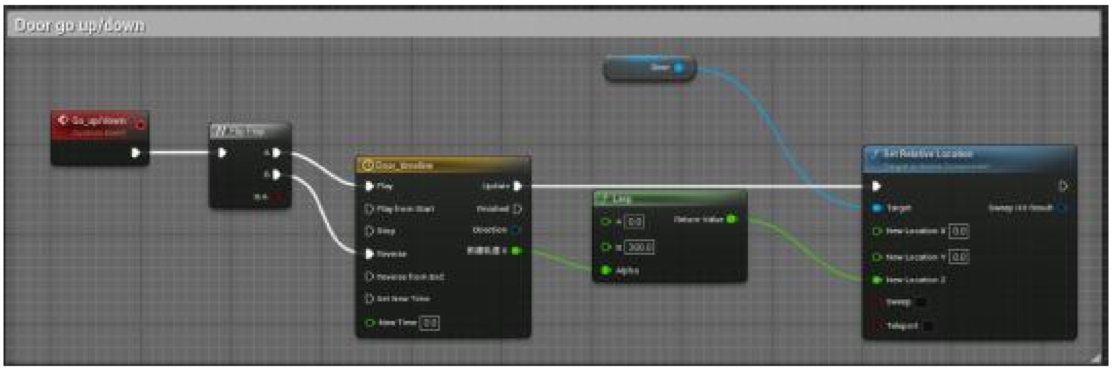
The health bar is used to indicate the player's current health, the red part is directly related to the percentage of the player's remaining health.

B2.7 Puzzle and level design

-Mechanisms and Traps

We designed a simple trap door mechanism where the door rises when a pressure plate is pressed and goes back to its original position when pressed again.





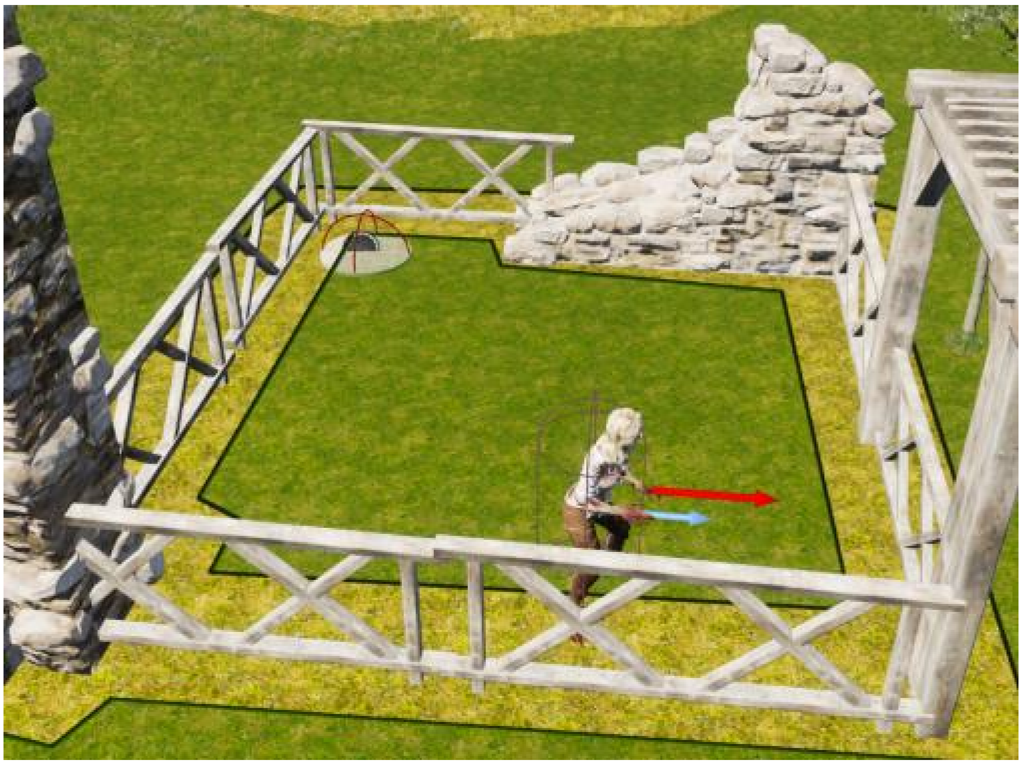
Pressure plate and door

-Puzzle and level design

Based on this simple mechanism, we introduced puzzle elements in the game. At the beginning of the game, players will encounter the first puzzle mechanism at their spawn point. This serves as a straightforward interactive reward where stepping on the pressure plate causes the gate to rise, and the player receives their first weapon as a reward. We placed red flags at weapon hiding spots to indicate to players that areas with red flags offer rewards. At this point, players also learn the concept that applying pressure to the pressure plate can open gates.

First puzzle

After the player uses the first weapon to shoot some monsters, they will move to the location of the second puzzle. Similar to the first puzzle, there are red flags here, and the player should now understand that there is a reward waiting for them. However, unlike the first puzzle, this time the pressure plate is enclosed by a fence, and the player cannot step on it because the fence is in the way. Instead, there is a zombie inside the fence. Here, players need to expand their thinking: the player does not need to step on the pressure plate themselves. As zombies chase the player, players can lure the zombie inside the fence onto the pressure plate to open the gate for them.



Zombie and plate inside fence

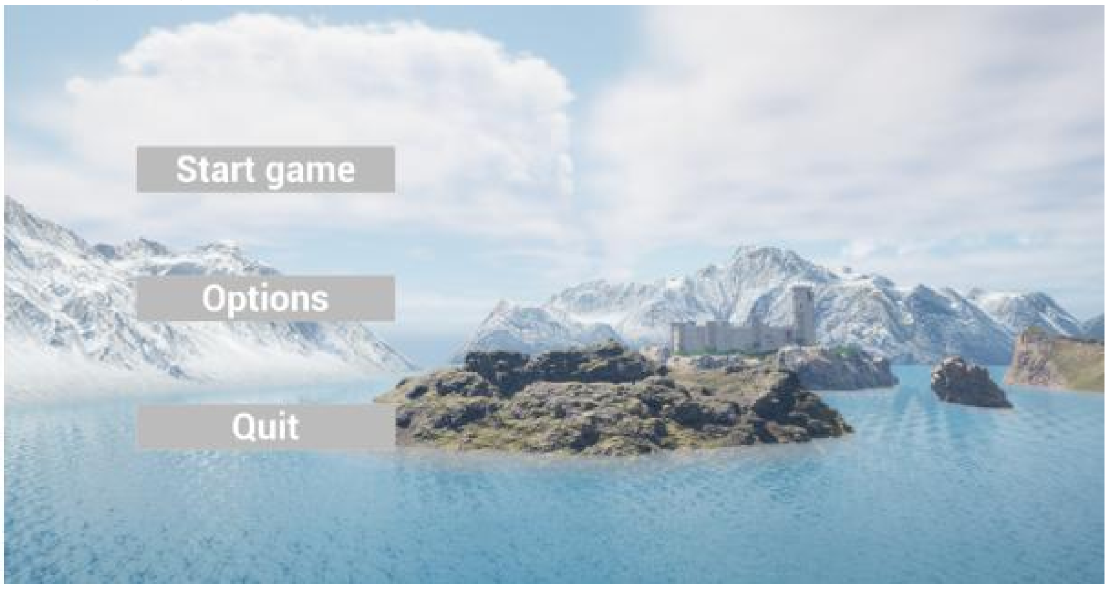
In the final stage, players will encounter a situation similar to the second puzzle. However, this time, inside the fence is not a zombie but a fox that the player needs to rescue. The fox, like the zombie, will follow the player. Based on the experience of solving the second puzzle, players should be able to easily open the final gate by guiding the fox onto the pressure plate. Finally, after passing through the gate, players escape the island together with the fox, at the end of the game.

Fox and plate behind fence

As the levels progress, players face increasingly powerful monsters. In the first stage, players face zombies, and they are equipped with a handgun. It has a low magazine capacity and low firing rate but is enough for dealing with zombies. In the second stage, monsters transform into high-health tree creatures, players will receive a powerful shotgun, suitable for large, high-health creatures. In the final castle stage, where players need to defeat a large variety of monsters, they are provided with a highly powerful assault rifle, which has a high magazine capacity to give players a higher margin of error.

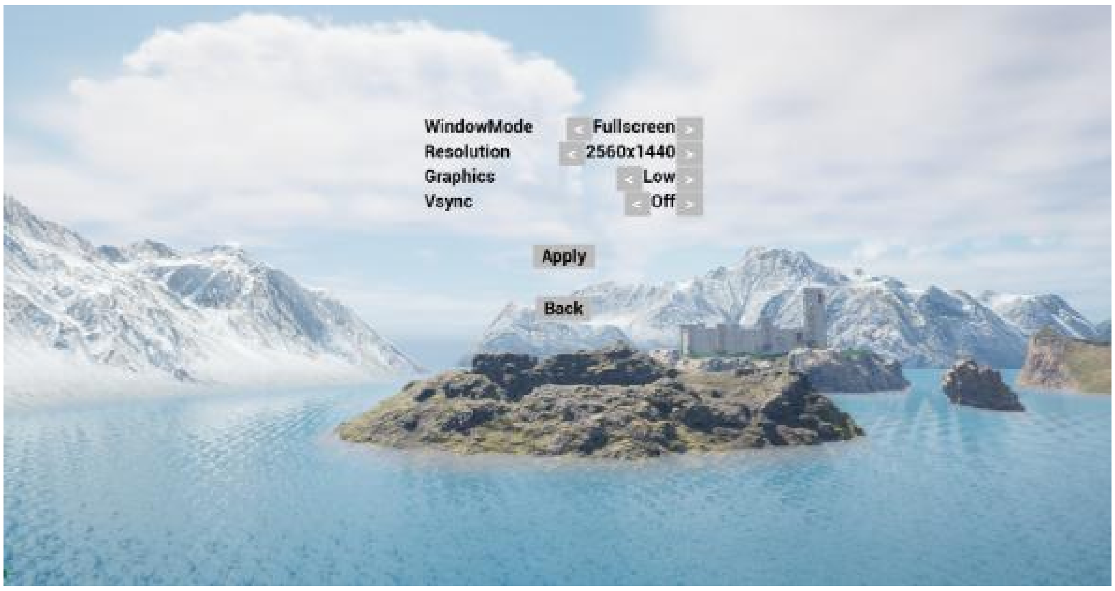
C.User Manual

Entering the system, users will see the main menu:



There are three buttons for the user to click on.
Start game: Start gameplay.
Options: Adjust game settings.
Quit: Quit game.

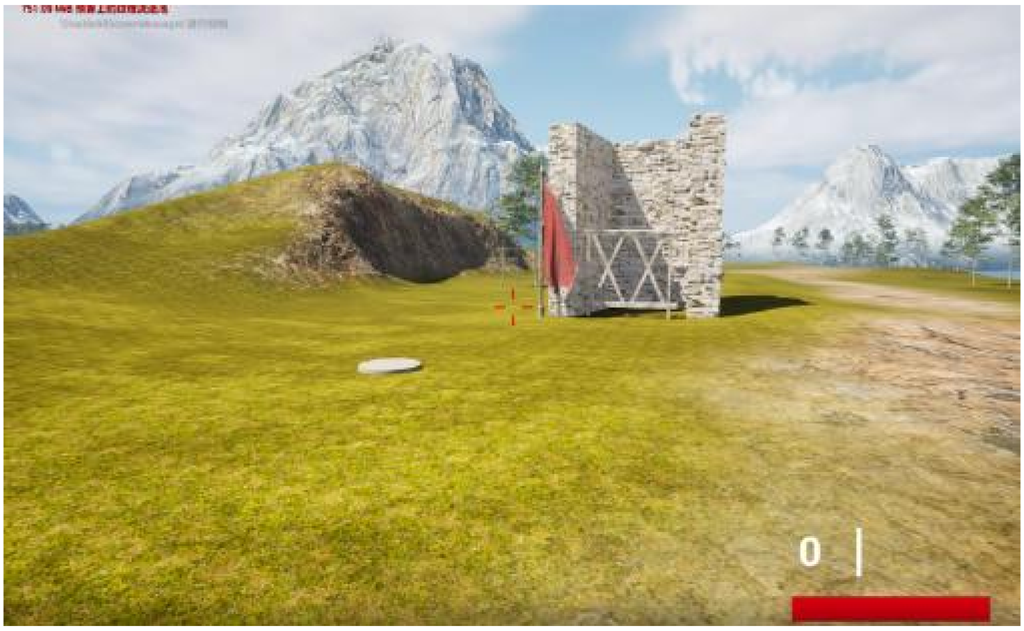
If the user clicks on [Options] they will go to the settings menu:



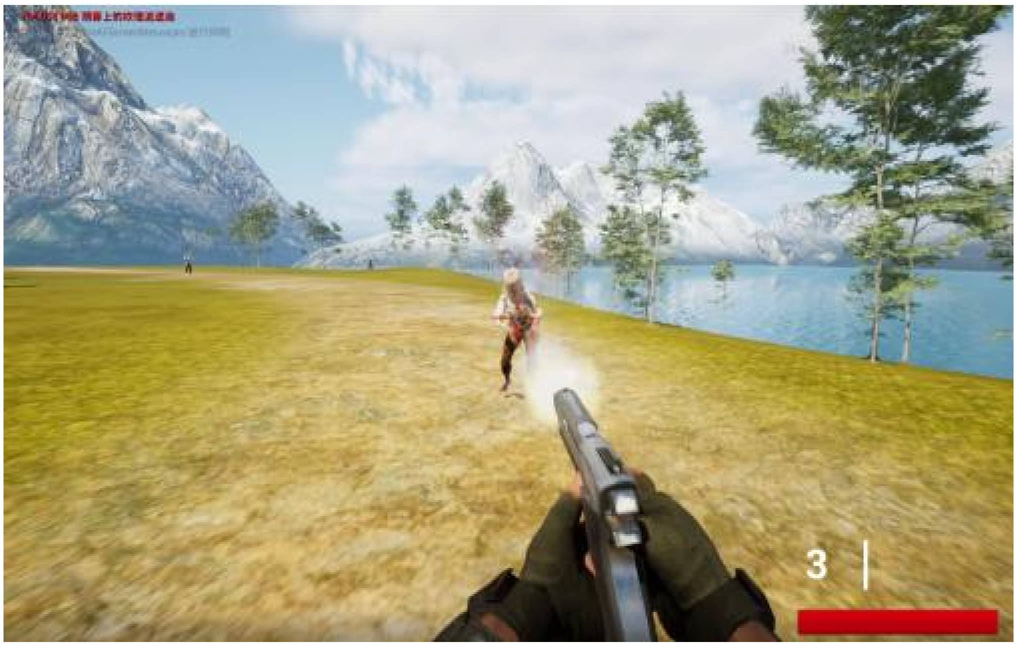
WindowMode < Fullscreen >
Resolution < 2560x1440 >
Graphics < Low >
Vsync < Off >

There are four settings available for users to adjust: window mode, resolution, graphics and Vsync. Users can adjust each setting using the arrows on either side. After making adjustments, they can press the [Apply] to synchronize the settings within the game, and then press [Back] to return to the main menu.

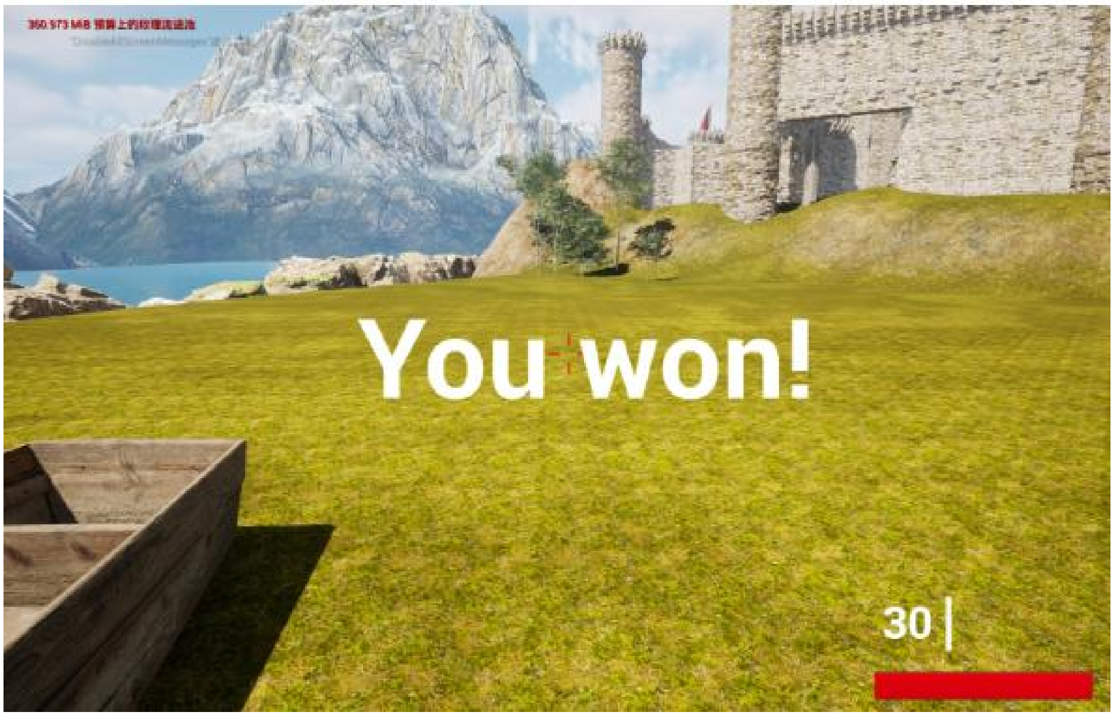
Entering actual gameplay, the player's screen will feature a crosshair in the center to aid aiming, and the bottom right corner will display the health bar and ammunition count. At this point, players can use the mouse to adjust their field of view, utilize WASD for movement, press Shift to enter sprint mode, and press Shift again to exit sprint mode.



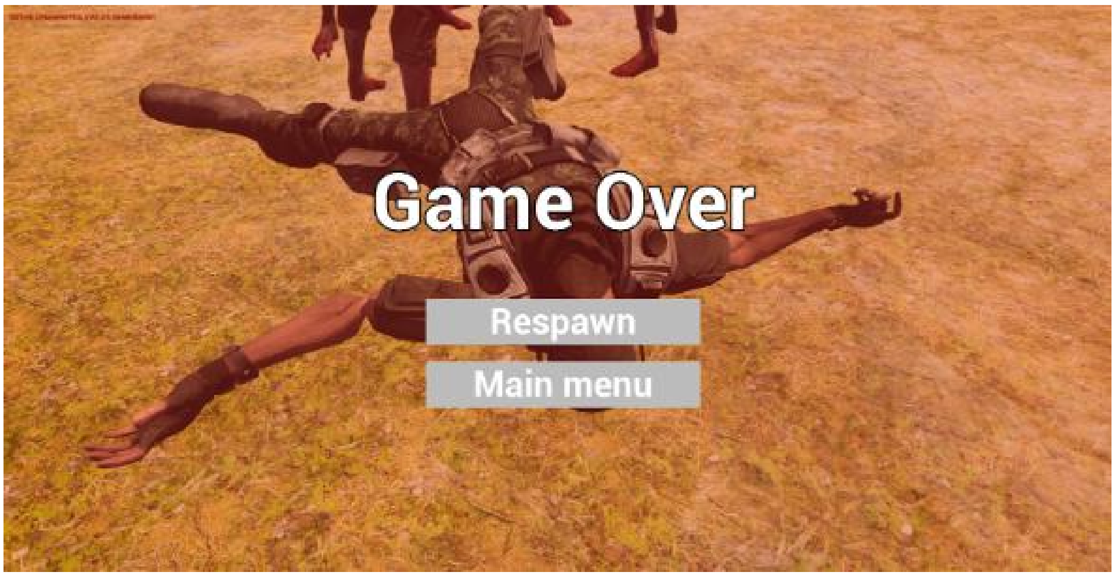
After gaining a weapon, players can shoot by pressing the left mouse button. Shooting is not allowed while in sprint mode. Pressing R allows the player to reload the weapon, and shooting is disabled during the reloading process.



To successfully complete the game, players need to go through all the stages and reach the small boat at the end of the island. This allows players to finish the game. Players are not required to defeat all the monsters to complete the game, but most monsters will obstruct the essential path for players to win the game.



If the player's health goes to zero due to monster attacks or if the player touches lethal terrain (like water), the game will be over, displaying the game_over interface. At this point, the player can choose to [Respawn], which means restarting the level or going back to the [Main menu].



D. Project file download links

D.1 packaged file

https://unsw-my.sharepoint.com/:u:/g/personal/z5445039_ad_unsw_edu_au/EYx9AF5AyVpDtjQ_9PLPgpABL9StUte33yZK6g8R34kYoA?e=ni9ubx

D.2 project file

https://unsw-my.sharepoint.com/:u:/g/personal/z5445039_ad_unsw_edu_au/Ef8QpM-OxQNMqRIWtaynpT4BT1mTQEI4nw3LtmBmGCKB_w?e=vgWF2N

Comment:
Due to limitations in compression speed, the project file and the packaging file are not the same version. It is important to note, however, that the packaged file operates effectively. Nonetheless, there are notable issues in theversion of this project file that require attention:

First,the 'player' tag was inadvertently omitted during the integration process. This oversight results in a critical gameplay issue where the player character does not lose health upon collision with fireballs.

Second,the fox's movement range was not adjusted. This misconfiguration may lead to unpredictable behavior, including the fox wandering to unintended areas or potentially falling off the game map.