Assignment 3a - Technical Design Document

Beat'Em Up

Wei Xin Soh
33351597

https://docs.google.com/document/d/10-4u7cz92vmaOrkXHZ3Y4CTin0KdxwrsRLdb5OB3zWw/edit?usp=sharing

Project Overview

The core concept of this Beat'Em Up game revolves around a thrilling rescue mission where the player pictures themselves as a valiant hero who is responsible to fight against a tide of enemies to rescue the victims. Every level presents an impressive challenge as enemies relentlessly attempt to thwart the player from success in an increasing hardness. After vanquishing a set number of enemies, the boss level can be unlocked and defeating this boss completes the rescue mission.

Game Mechanics Overview

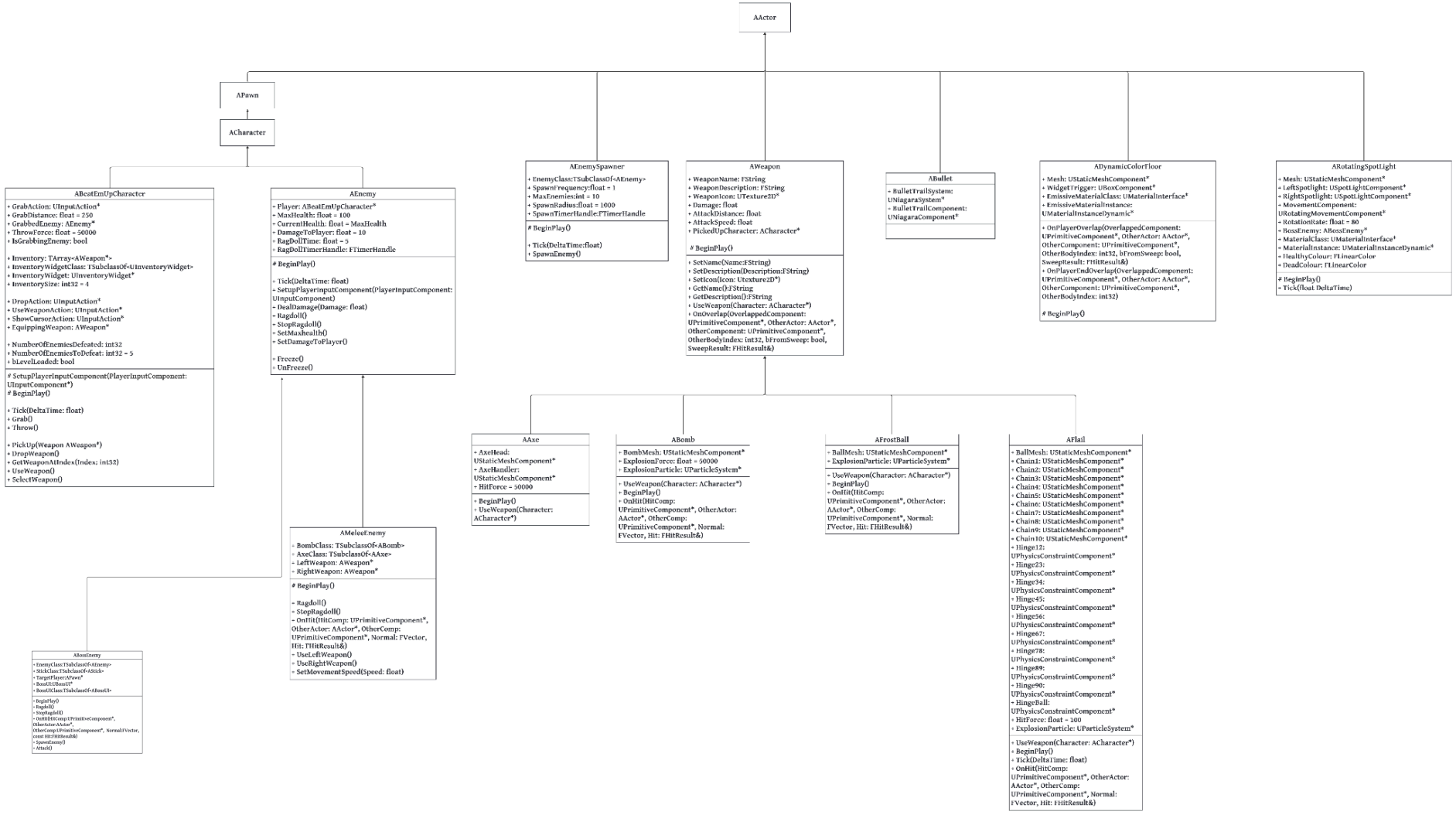
The core mechanics of the game revolve around an engaging combat system and strategic weapon selection from the inventory. The player character is not only armed with his own strength and skill but also with diverse weapons scattered throughout the game world. Each weapon boasts its own unique attributes which apply different effects on the enemies. By discovering and collecting these weapons into the inventory, the player can strategically choose any of them based on the combat situation. To keep the gameplay engaging and challenging, after battling through waves of regular enemies, the player stumbles upon a mysterious pathway. Walking along, they enter the Boss Level. Here, the boss enemy emerges in a show of power, pushing the player to their limits with relentless attacks. Utilising the weapons collected in the previous level is crucial for surviving the intense combat. The player must strategically use the available attacks to defeat the formidable boss enemy and achieve victory.

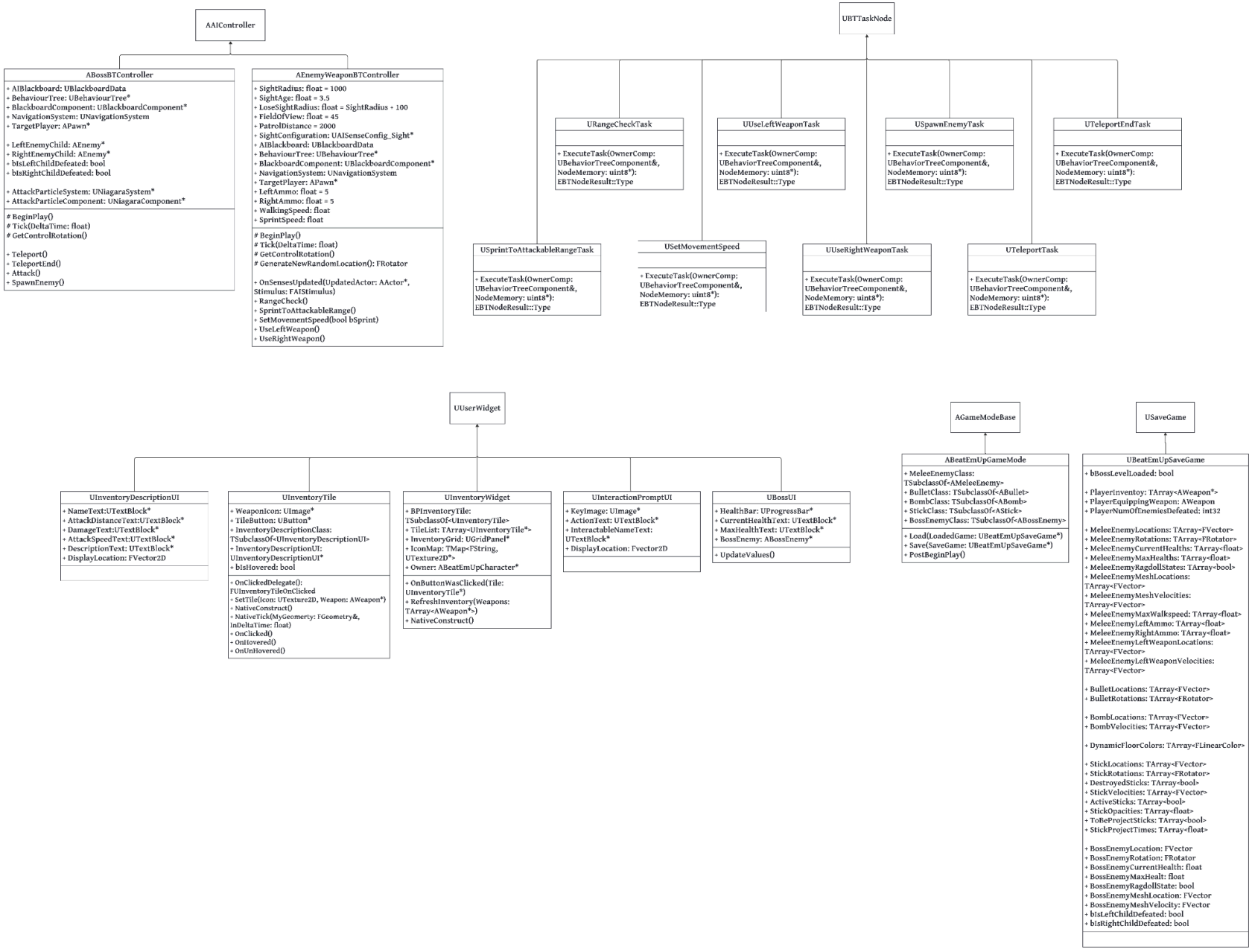
Target Platform

Based on the gameplay mechanics involved in this game, the prototype is ideally suited for the PC platform. Firstly, the intricate combat system which involves combo moves and weapon selection requires the precision of mouse and keyboard controls so that the player can execute complex attack actions easily which in turn maximises the player's effectiveness in battle. Secondly, PC platforms offer superior graphics capabilities and performance compared to mobile and consoles. This allows for stunning visual effects and detailed character animations in the game battle. Lastly, it is simple to leverage customization on graphic settings and key bindings to provide players with a personalised experience.

Game Mechanics

UML Diagram https://drive.google.com/drive/u/1/folders/1_GWlp2OMpxEknfNXTDDHM7NbOfnKQ8Un




AAIController
ABossBTController
+ AIBlackboard: UBlackboardData
+ BehaviourTree: UBehaviourTree*
+ BlackboardComponent: UBlackboardComponent*
+ NavigationSystem: UNavigationSystem
+ TargetPlayer: APawn*
+ LeftEnemyChild: AEnemy*
+ RightEnemyChild: AEnemy*
+ bIsLeftChildDefeated: bool
+ bIsRightChildDefeated: bool
+ AttackParticleSystem: UNiagaraSystem*
+ AttackParticleComponent: UNiagaraComponent*
# BeginPlay()
# Tick(DeltaTime: float)
# GetControlRotation()
+ Teleport()
+ TeleportEnd()
+ Attack()
+ SpawnEnemy()
AEnemyWeaponBTController
+ SightRadius: float = 1000
+ SightAge: float = 3.5
+ LoseSightRadius: float = SightRadius + 100
+ FieldOfView: float = 45
+ PatrolDistance = 2000
+ SightConfiguration: UAISenseConfig_Sight*
+ AIBlackboard: UBlackboardData
+ BehaviourTree: UBehaviourTree*
+ BlackboardComponent: UBlackboardComponent*
+ NavigationSystem: UNavigationSystem
+ TargetPlayer: APawn*
+ LeftAmmo: float = 5
+ RightAmmo: float = 5
+ WalkingSpeed: float
+ SprintSpeed: float
# BeginPlay()
# Tick(DeltaTime: float)
# GetControlRotation()
# GenerateNewRandomLocation(): FRotator
+ OnSensesUpdated(UpdatedActor: AActor*, Stimulus: FAIStimulus)
+ RangeCheck()
+ SprintToAttackableRange()
+ SetMovementSpeed(bool bSprint)
+ UseLeftWeapon()
+ UseRightWeapon()
UBTTaskNode
URangeCheckTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
UUseLeftWeaponTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
USpawnEnemyTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
UTeleportEndTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
USprintToAttackableRangeTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
USetMovementSpeed
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
UUseRightWeaponTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
UTeleportTask
+ ExecuteTask(OwnerComp: UBehaviorTreeComponent&, NodeMemory: uint8*): EBTNodeResult::Type
UUserWidget
UInventoryDescriptionUI
+ NameText:UTextBlock*
+ AttackDistanceText:UTextBlock*
+ DamageText:UTextBlock*
+ AttackSpeedText:UTextBlock*
+ DescriptionText: UTextBlock*
+ DisplayLocation: FVector2D
UInventoryTile
+ WeaponIcon: UImage*
+ TileButton: UButton*
+ InventoryDescriptionClass: TSubclassOf<UInventoryDescriptionUI>
+ InventoryDescriptionUI: UInventoryDescriptionUI*
+ bIsHovered: bool
+ OnClickedDelegate(): FUInventoryTileOnClicked
+ SetTile(Icon: UTexture2D, Weapon: AWeapon*)
+ NativeConstruct()
+ NativeTick(MyGeometry: FGeometry&, InDeltaTime: float)
+ OnClicked()
+ OnHovered()
+ OnUnHovered()
UInventoryWidget
+ BPInventoryTile: TSubclassOf<UInventoryTile>
+ TileList: TArray<UInventoryTile*>
+ InventoryGrid: UGridPanel*
+ IconMap: TMap<FString, UTexture2D*>
+ Owner: ABeatEmUpCharacter*
+ OnButtonWasClicked(Tile: UInventoryTile*)
+ RefreshInventory(Weapons: TArray<AWeapon*>)
+ NativeConstruct()
UInteractionPromptUI
+ KeyImage: UImage*
+ ActionText: UTextBlock*
+ InteractableNameText: UTextBlock*
+ DisplayLocation: Fvector2D
UBossUI
+ HealthBar: UProgressBar*
+ CurrentHealthText: UTextBlock*
+ MaxHealthText: UTextBlock*
+ BossEnemy: ABossEnemy*
+ UpdateValues()
AGameModeBase
ABeatEmUpGameMode
+ MeleeEnemyClass: TSubclassOf<AMeleeEnemy>
+ BulletClass: TSubclassOf<ABullet>
+ BombClass: TSubclassOf<ABomb>
+ StickClass: TSubclassOf<AStick>
+ BossEnemyClass: TSubclassOf<ABossEnemy>
+ Load(LoadedGame: UBeatEmUpSaveGame*)
+ Save(SaveGame: UBeatEmUpSaveGame*)
+ PostBeginPlay()
USaveGame
UBeatEmUpSaveGame
+ bBossLevelLoaded: bool
+ PlayerInventoy: TArray<AWeapon*>
+ PlayerEquippingWeapon: AWeapon
+ PlayerNumOfEnemiesDefeated: int32
+ MeleeEnemyLocations: TArray<FVector>
+ MeleeEnemyRotations: TArray<FRotator>
+ MeleeEnemyCurrentHealths: TArray<float>
+ MeleeEnemyMaxHealths: TArray<float>
+ MeleeEnemyRagdollStates: TArray<bool>
+ MeleeEnemyMeshLocations: TArray<FVector>
+ MeleeEnemyMeshVelocities: TArray<FVector>
+ MeleeEnemyMaxWalkspeed: TArray<float>
+ MeleeEnemyLeftAmmo: TArray<float>
+ MeleeEnemyRightAmmo: TArray<float>
+ MeleeEnemyLeftWeaponLocations: TArray<FVector>
+ MeleeEnemyLeftWeaponVelocities: TArray<FVector>
+ BulletLocations: TArray<FVector>
+ BulletRotations: TArray<FRotator>
+ BombLocations: TArray<FVector>
+ BombVelocities: TArray<FVector>
+ DynamicFloorColors: TArray<FLinearColor>
+ StickLocations: TArray<FVector>
+ StickRotations: TArray<FRotator>
+ DestroyedSticks: TArray<bool>
+ StickVelocities: TArray<FVector>
+ ActiveSticks: TArray<bool>
+ StickOpacities: TArray<float>
+ ToBeProjectSticks: TArray<bool>
+ StickProjectTimes: TArray<float>
+ BossEnemyLocation: FVector
+ BossEnemyRotation: FRotator
+ BossEnemyCurrentHealth: float
+ BossEnemyMaxHealt: float
+ BossEnemyRagdollState: bool
+ BossEnemyMeshLocation: FVector
+ BossEnemyMeshVelocity: FVector
+ bIsLeftChildDefeated: bool
+ bIsRightChildDefeated: bool

Movement Mechanics

There are two types of movements which are walking and jumping. These movements are driven by input from the player using keyboard keys (WASD) for walking and spacebar for jumping. The player character has a character controller component which translates these input into movement by interacting with the physics system to apply forces or impulses to move the character. This involves interactions with physics simulation where forces like gravity and friction affect the player character's movement. For example, physics simulation applies gravity to bring the player character back down to the ground after reaching the peak of the jump.

There are also limitations on the player character's movement. The game world is populated with static meshes representing the obstacles and other environment elements. When the player character collides with these meshes, the collision detection system prevents the character from moving through them. Furthermore, the player has a maximum walking speed of 500units/s and an initial jumping velocity of 700units/s. Upon jumping, the player's walking speed will slow down 0.85 times.

Controls

In this game, players have a variety of actions available to them. This includes movement, interaction and combat actions. The movement actions allow for precise control over the player character's position and navigation through the game world. Besides that, the interactive actions such as picking up and dropping items, grabbing enemies as well as using gravity well and trapdoor allow for interaction with the objects and enemies with designated keys. Lastly, the combat actions which include punching and attacking with weapons allow players to engage in intense battles.

Mapping	Action	Description	Keybind
IA_Move	Move Forward	Used to move the character forward	W
	Move Backwards	Used to move the character backwards	S
	Move Left	Used to move the character to the left	A
	Move Right	Used to move the character to the right	D
IA_Jump	Jump	Used to make the character jump	Spacebar
IA_Look	Look	Used to look the character's surroundings	Mouse-XY 2D-Axis
IA_Punch	Punch	Used to perform punching by character on enemy	Mouse Left Click

IA_Use	Use	Used to use the gravity well	Mouse Right Click
IA_Grab	Grab And Throw	Used to grab an enemy and throw it to the midair	G
IA_SelectWeapon	SwitchWeapon	Used to switch weapon in the inventory	Mouse Wheel Down
IA_UseWeapon	Use Weapon	Used to attack with the equipping weapon	R
IA_Drop	Drop	Used to drop weapon item	Q
IA_Pause	Pause	Used to pause the game	P

Additional Gameplay Mechanic 1

Mechanic Overview
The player character is able to collect various weapons scattered through the game map in his inventory and select one of these to attack the enemies.

Mechanic Description / Functionality
When the player character overlaps with a weapon pickup item, the pickup event is triggered. The game checks if the player's inventory is full; if not, the weapon is hidden from the game world and added to the player's inventory. To select a weapon for battle, the player can wheel down their mouse to navigate through the inventory and the focusing weapon is equipped. To drop the equipping weapon, the player can press key Q and the weapon will be removed from the inventory and added back to the game world at the player's current position.

There are four types of weapons in the game, ie. Axe, Frost ball, Bomb, and Flail. Each weapon has associated attributes such as damage, attack speed, range and some special effects. When the player attacks with a weapon, the damage inflicted is calculated based on the weapon's attributes and effects.

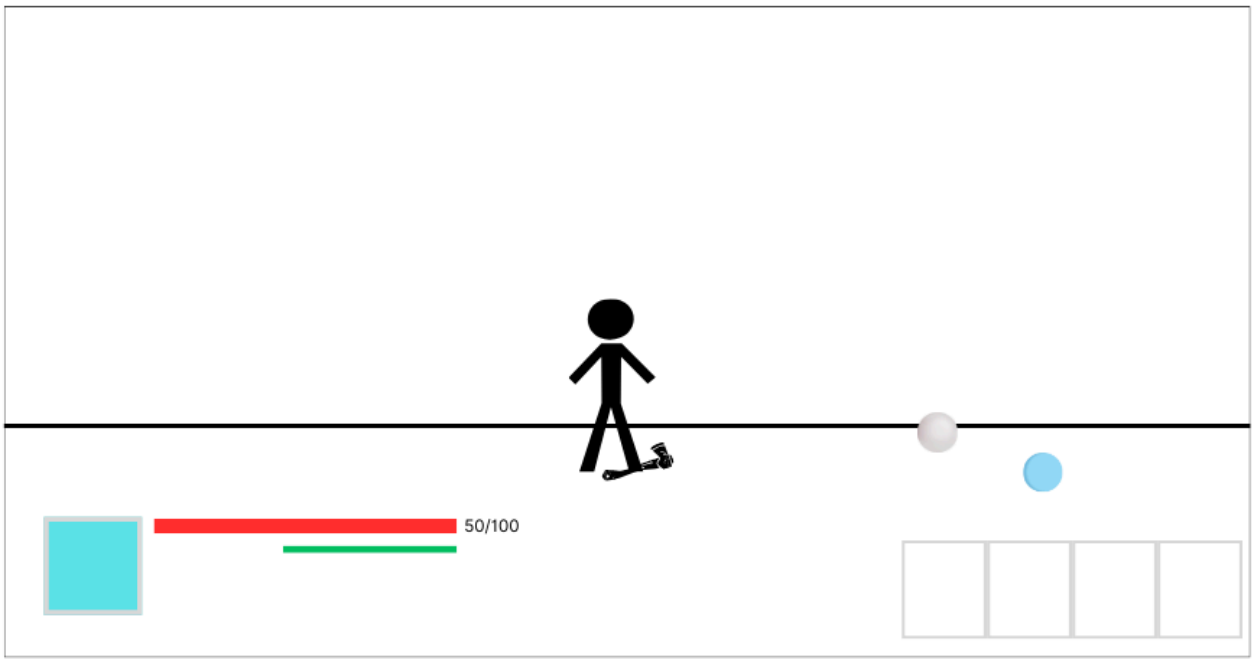
1. Axe
    a. It is a melee weapon which excels in close-range combat and deals massive damage in a single strike.
    b. Damage(20): This reflects its role as a heavy-hitting melee weapon which is typically associated with raw power and brute force.
    c. AttackDistance(400): This emphasises its strength in close-range combat.
    d. AttackSpeed(2000): It is slower compared to lighter weapons. This balanced its high damage output with a slightly slower swing speed.
    e. Hit Force(50000): This reflects its heavy knocking on the enemies.

2. Frost ball
    a. It is a frost-based weapon which is used to freeze enemies upon being hit to make them vulnerable to follow-up attacks temporarily. For example, the player can use this advantage to position freeze the enemy at the range of the gravity well or trap door.
    b. Damage(0): This reflects that it is focusing on freezing enemies in place rather than raw damage.
    c. FreezeRange(800): This large range allows players to immobilise multiple enemies simultaneously.
    d. AttackSpeed(200): It is faster as it is light enough to be thrown.
3. Bomb
    a. It is a ranged explosive weapon with moderate damage and attack speed. It can deal area-of-effect damage to multiple enemies at once.
    b. Damage(30): Moderate damage value ensures it remains effective against groups of enemies without being overly powerful.
    c. AttackDistance(800): This large range allows bombs to affect a wider area to clear out clusters of enemies.
    d. AttackSpeed(100): It is faster as it is light enough to be thrown.
    e. Explosion Force(80000): This reflects its powerful explosions to push back the enemies.
4. Flail

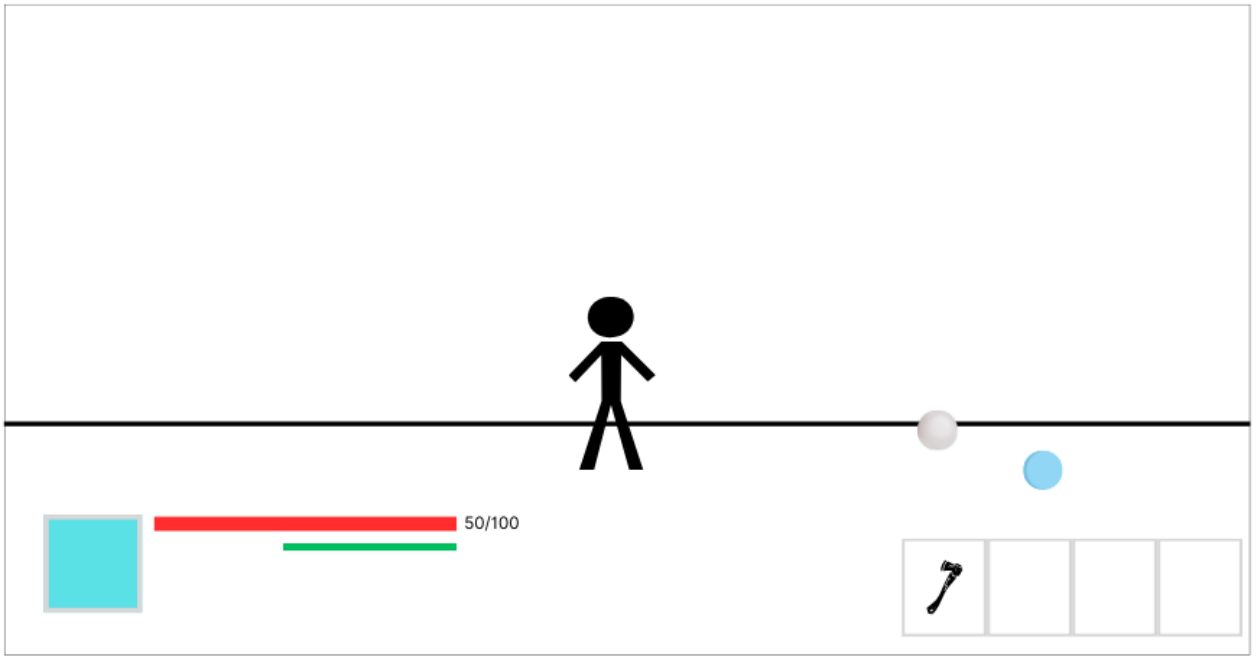
a. It is a melee weapon which excels in intermediate-range combat and deals massive damage in a single strike.
b. Damage(25): This reflects its role as a heavy-hitting melee weapon which is typically associated with raw power and brute force.
c. AttackDistance(500): This emphasises its strength in close-range combat.
d. AttackSpeed(100): It is slower compared to lighter weapons. This balanced its high damage output and combat-range with a slightly slower swing speed.
e. Hit Force(50000): This reflects its heavy knocking on the enemies.

Mockup

Player character overlapped with the weapon mesh



Weapon added to inventory



Properties & Values

[AAxe]

Variable Name	Type	Purpose
Damage = 20	float	Used to represent the damage inflicted.
AttackDistance = 400	float	Used to represent the distance of effect.
AttackSpeed = 50	float	Used to represent the speed at which the axe can be swung
HitForce = 50000	float	Used to represent the force implied to the hit target.

[AFrostBall]

Variable Name	Type	Purpose
Damage = 0	float	Used to represent the damage inflicted.
FreezeRange = 800	float	Used to represent the area of effect.
AttackSpeed = 200	float	Used to represent the speed at which the ice can be thrown.

[ABomb]

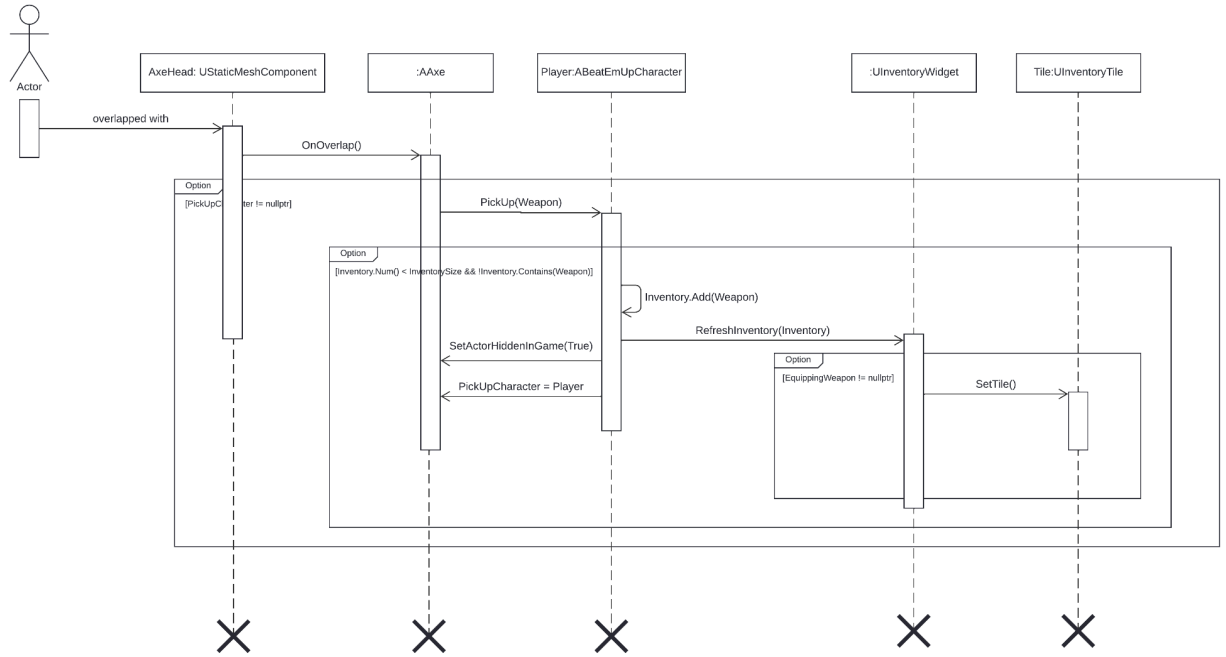
Variable Name	Type	Purpose
Damage = 30	float	Used to represent the damage inflicted.
AttackDistance = 800	float	Used to represent the area of effect.
AttackSpeed = 100	float	Used to represent the speed at which the bomb can be thrown.
ExplosionForce = 80000	float	Used to represent the force implied on enemies by the explosion to cause a knockback.

[AFlail]

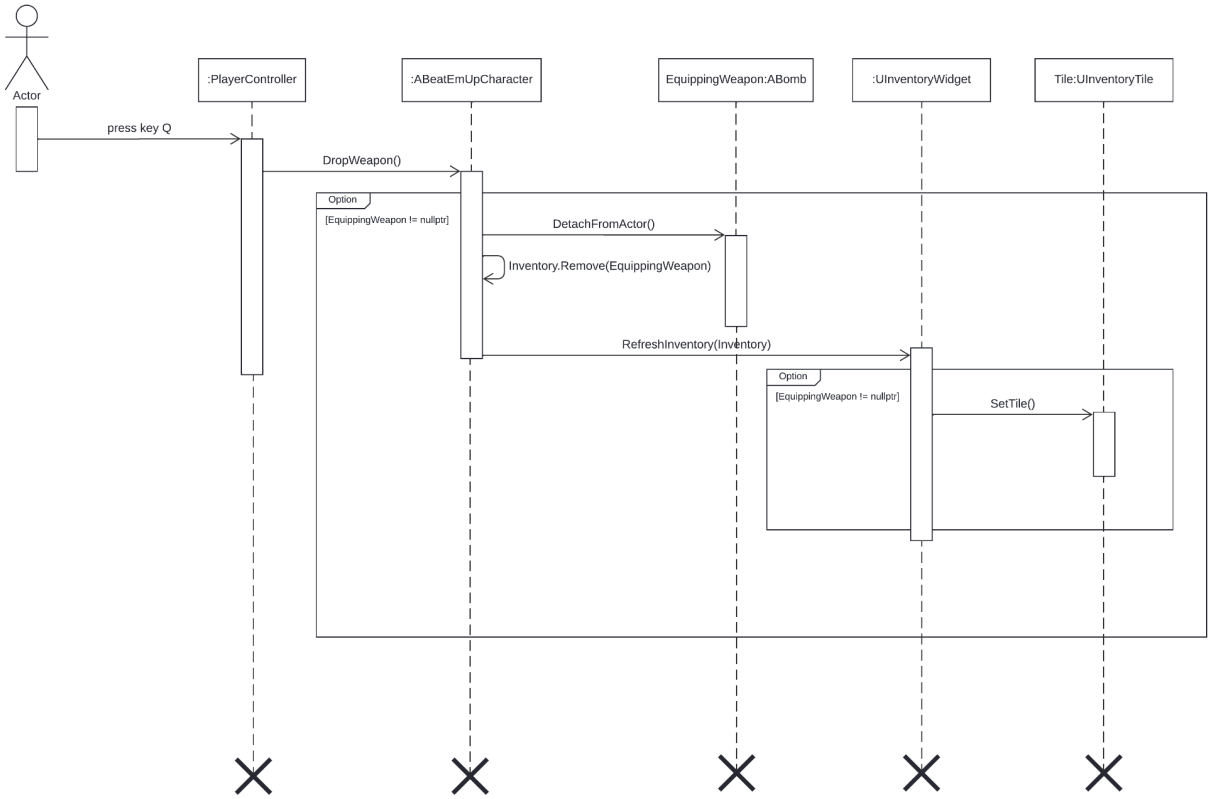
Variable Name	Type	Purpose
Damage = 10	float	Used to represent the damage inflicted.
AttackDistance = 500	float	Used to represent the distance of effect.
AttackSpeed = 100	float	Used to represent the speed at which the bomb can be thrown.
HitForce= 50000	float	Used to represent the force implied to the hit target.

Sequence Diagram

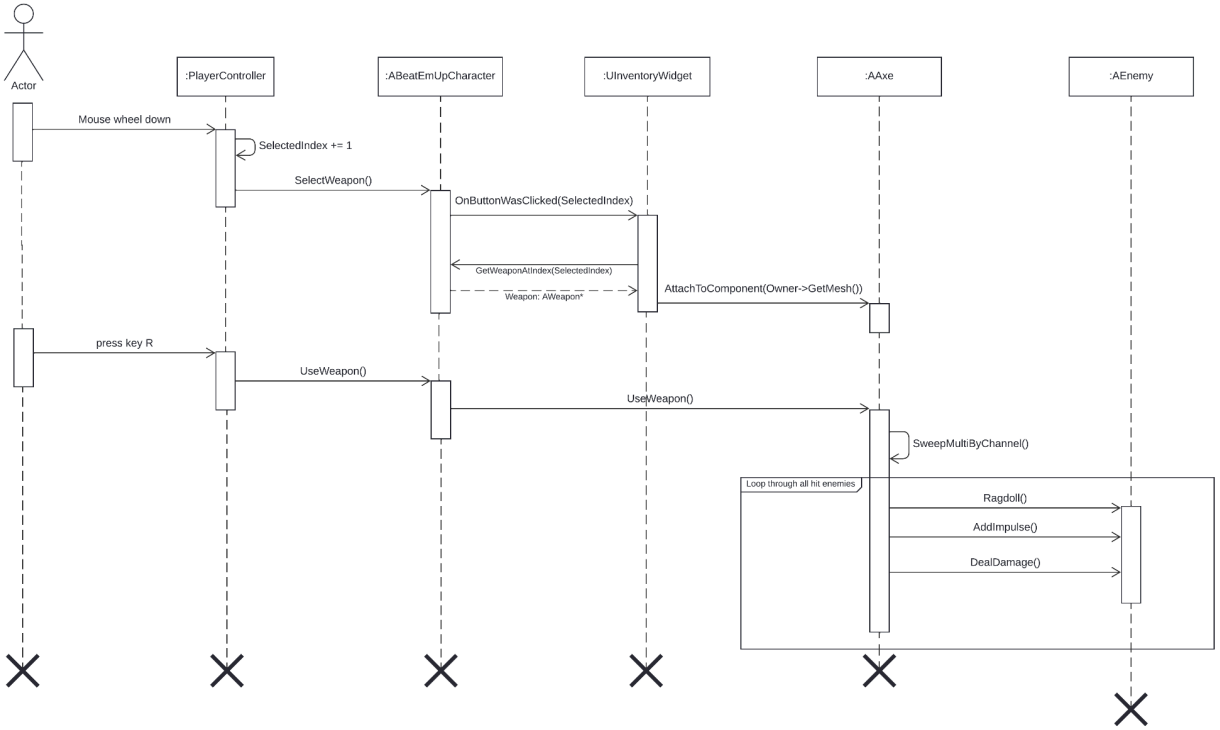
Pick Up Weapon



Drop Weapon



Select Weapon and Attack with Weapon



Additional Gameplay Mechanic 2

Mechanic Overview

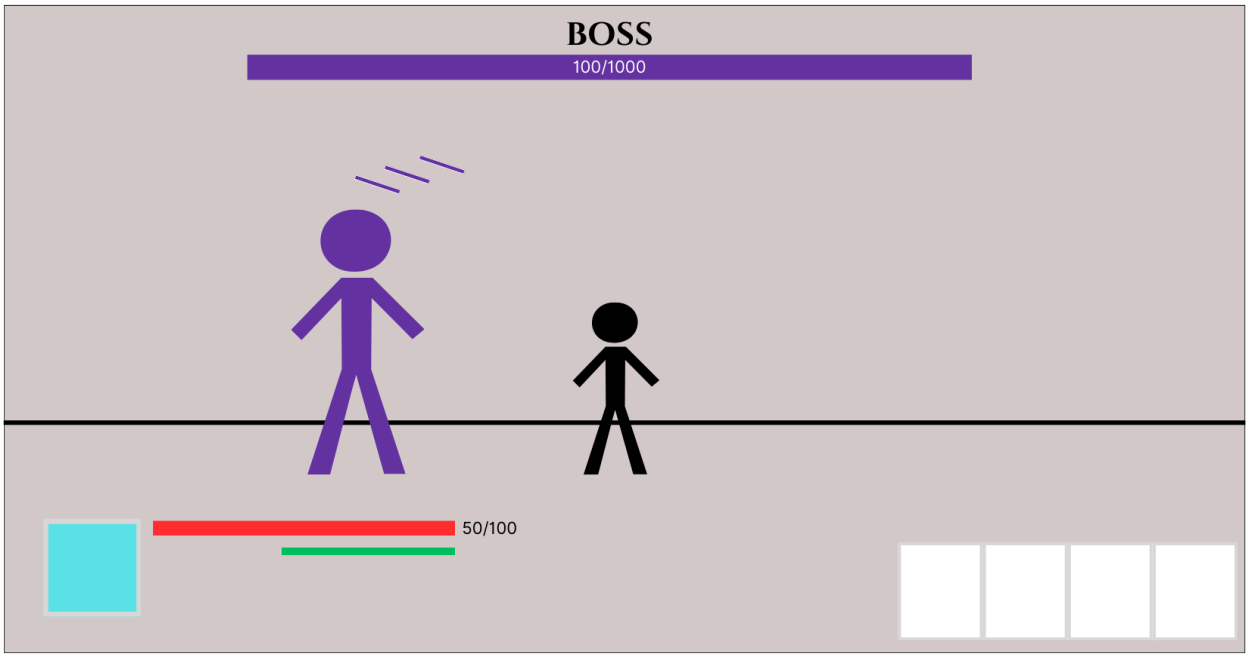
Upon defeating a specified number of regular enemies, the level trigger is activated, leading the player to the Boss Level. In this level, the boss enemy is spawned with stronger abilities, including increased health and attack power. The boss uses an advanced AI controller to perform aggressive attack patterns and teleportation which creates a significant challenge to the player to achieve victory.

Mechanic Description / Functionality

As the player progresses through a level, they encounter and defeat regular enemies scattered throughout the map. A counter is created to track the number of defeated enemies and when it reaches a predetermined threshold, the level trigger will be activated to start the level streaming for the boss level when the player overlaps with it.

In the Boss Level, the boss enemy will be spawned. The boss enemy is significantly stronger and more resilient than the regular enemy with some attack behaviours added for it. It is spawned in a larger size, with more health which makes it able to withstand more damage, and continuously teleport around the player to make its location harder to be targeted. It is controlled by the BossEnemyBTController to sense, think and act based on its behaviour tree. It will prioritise the child enemies spawning and attack while teleport whenever it is processing the cooldown for the previous task. The powerful attack that can be performed by the boss enemy is spawning powerful projectiles above itself which it then launches at the player. These projectiles can deal significant damage upon impact. When these projectiles hit the ground, they create a collision sphere to detect any nearby player within the sweep to deal damage to it, adding an additional layer of challenge and strategy to the boss encounter. Additionally, to enhance the visual impact and intimidation of the boss enemy, a Niagara particle effect is created. This effect surrounds the boss with swirling purple flames, giving it a fearful appearance and signalling to the player that they are facing a formidable enemy.

Mockup



Properties & Values

[ABeatEmUpCharacter]

Variable Name	Type	Purpose
NumOfEnemiesToDefeat = 5	float	Used to represent the number of enemies required to defeat to unlock the boss level.

[ABossEnemy]

Variable Name	Type	Purpose
MaxHealth = 1000	float	Used to represent the boss enemy's maximum health, it is set to a high value, significantly increasing the difficulty level to emphasise the challenge of the boss level.

[ALevelTrigger]

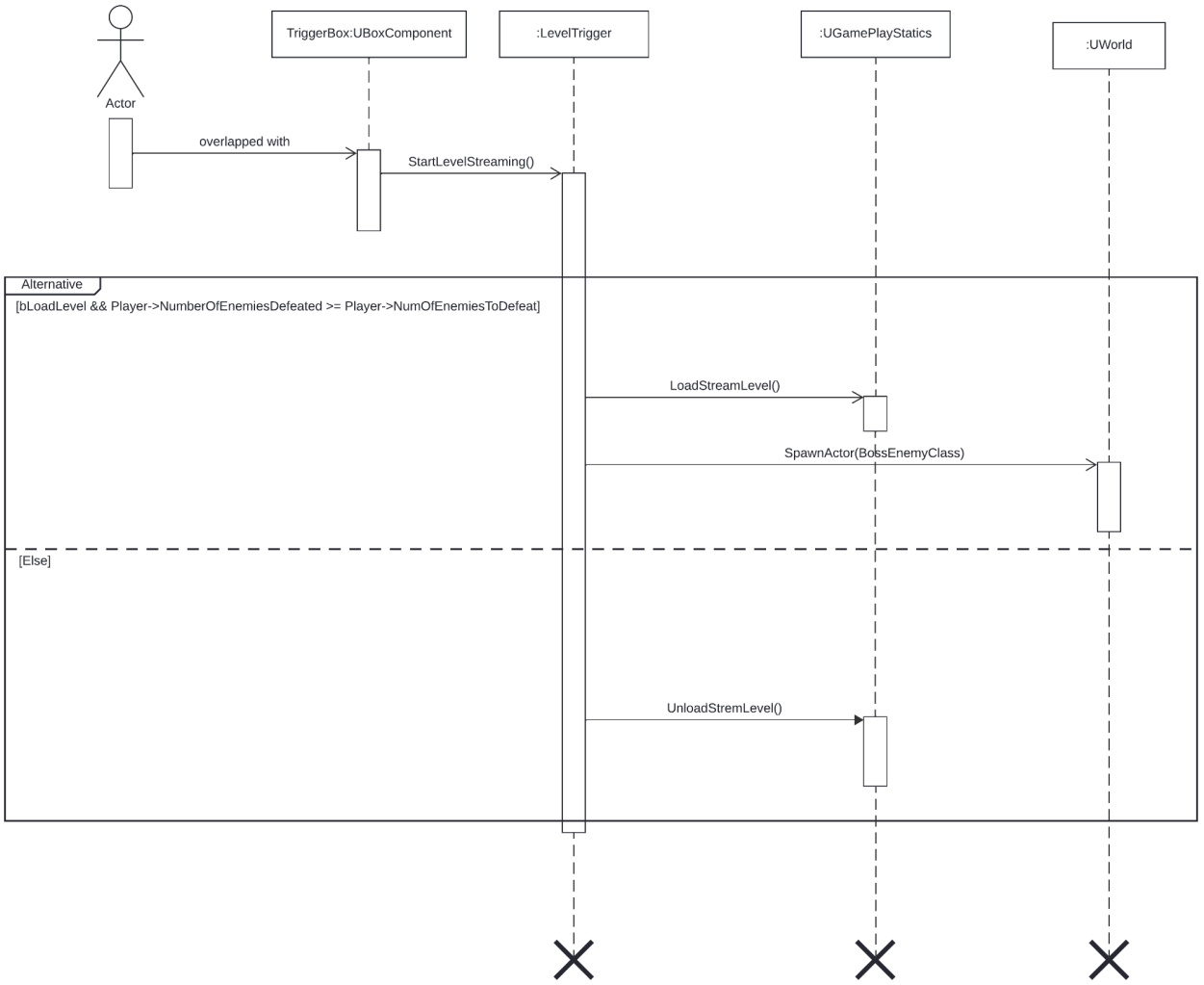
Variable Name	Type	Purpose
LevelToLoad = LevelToStream	FName	Used to load the streaming level.

[AStick]

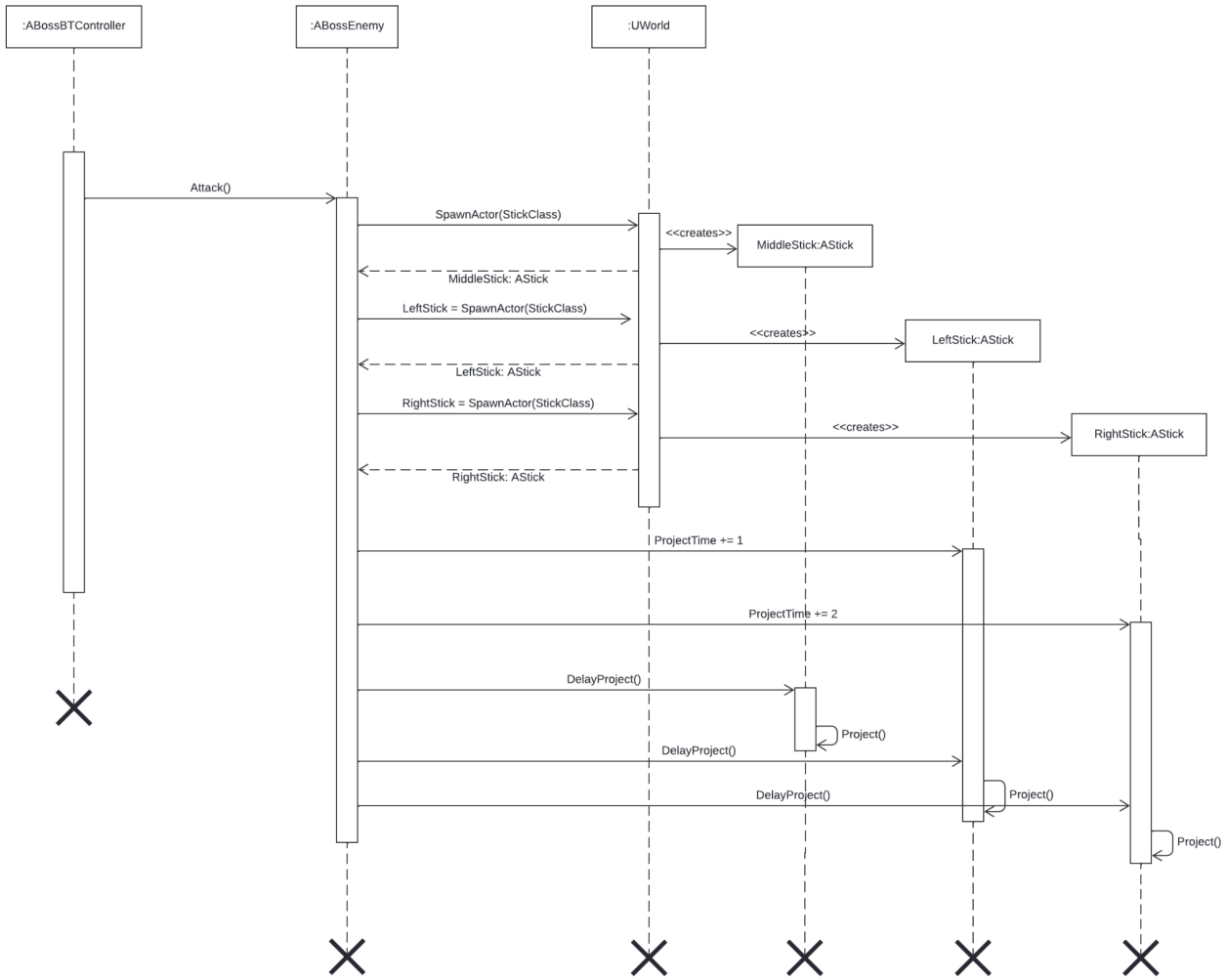
Variable Name	Type	Purpose
ProjectTime = 1	float	It allows the demonstration of the boss enemy applying a powerful attack by giving a delay for the projecting after spawning instead of launching all the sticks at the same time. The value gives a perfect duration for delay without being too slow for the player to have excessive time to respond to the attack.
CurrentOpacity = 1	float	It allows the fade out effect of the stick once it hits something. The opacity starts from 1 which is a solid state to fade out.

Sequence Diagram

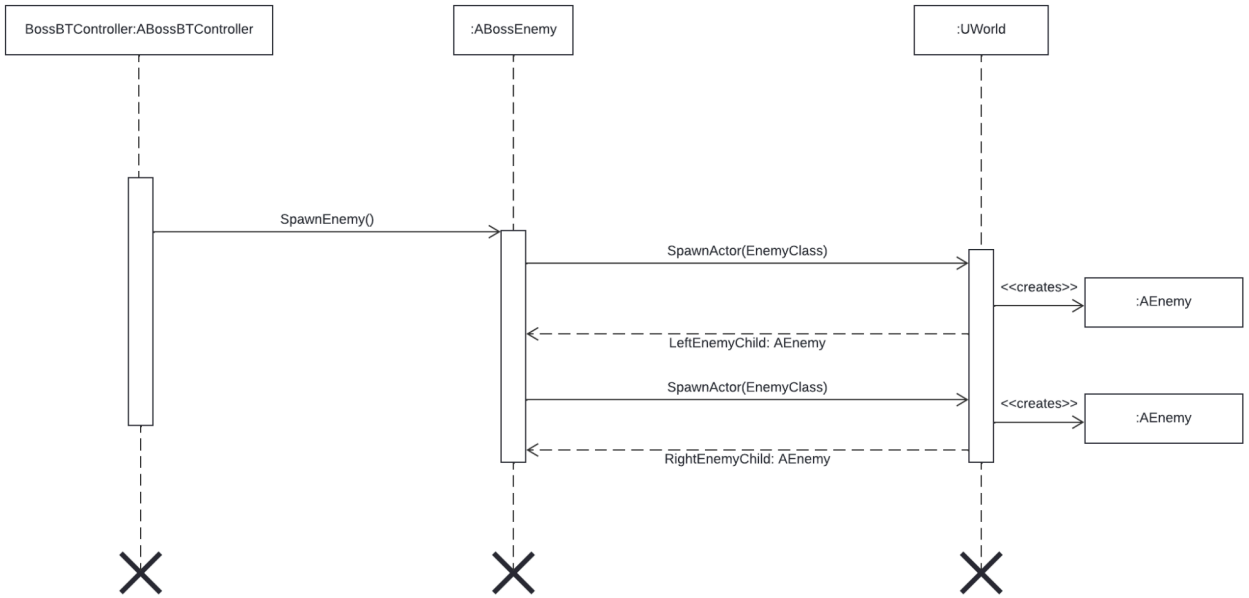
Load Boss Level



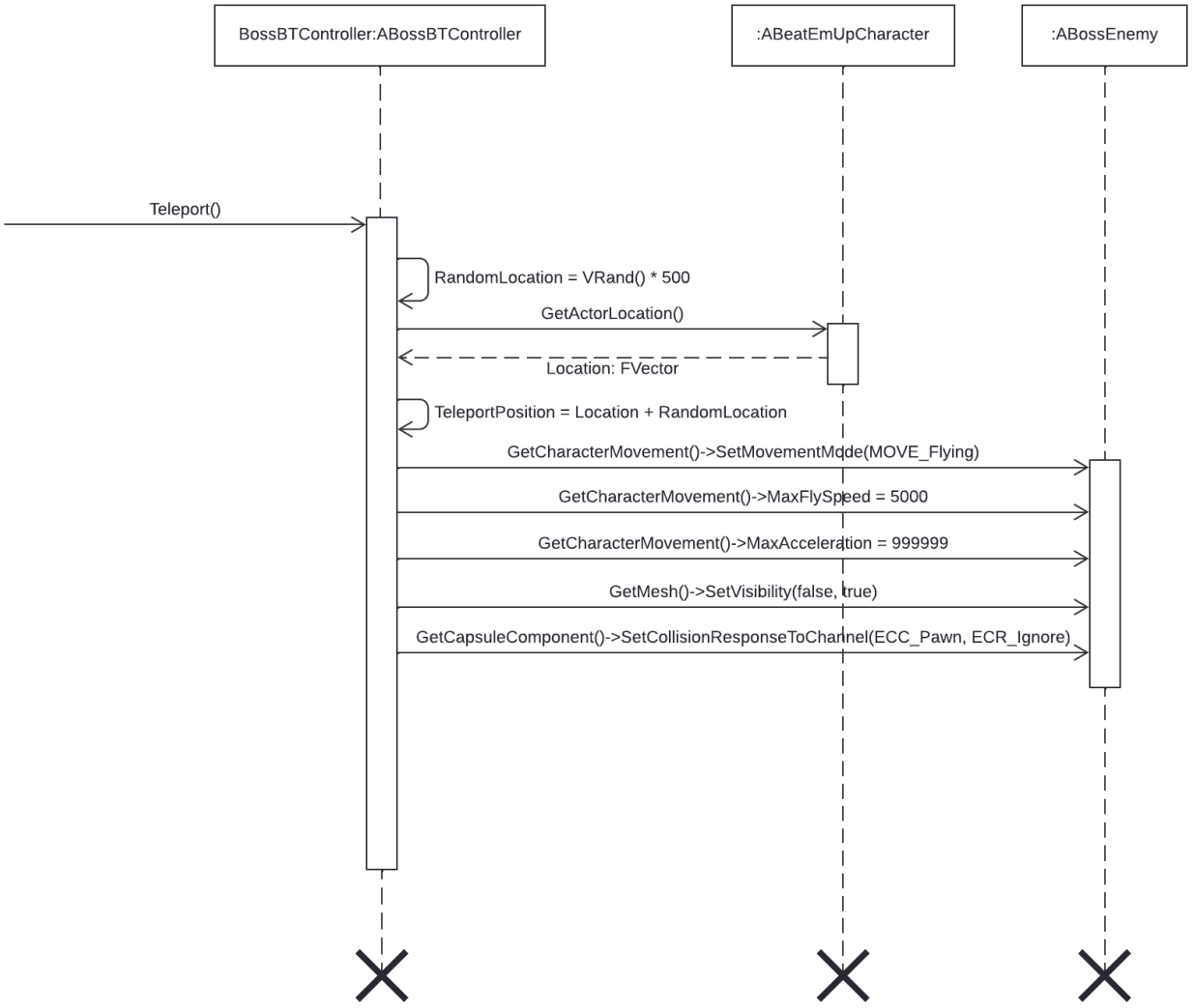
Boss Enemy Attack Target Player



Boss Enemy Spawn Child Enemies



Boss Enemy Teleport



Physics Interaction

Overview of Interaction

A physics mechanic is implemented in the game. The player character is able to grab the nearest enemy he is facing by pressing and holding the key G and throw the grabbing enemy to the midair by releasing the key.

Interaction Description

How the Interaction Works

When the user presses and holds G, the Grab() function is invoked to apply the grabbing mechanic. This mechanic uses raycasting to trace the nearest enemy that the player character is facing within the grabbing distance (ie. 250). The raycast starts from the player character's location and ends in the forward direction of the player character. If the first blocking hit of the ray is an Enemy Actor, we set the GrabbedEnemy to reference the Enemy Actor. Then, we set the boolean IsGrabbingEnemy to true which indicates the player character is grabbing an enemy.

On every tick, if the player character is grabbing an enemy, we update the location of the grabbed enemy to maintain its relative position to the player character (ie. always in front of the player character). In addition, its Z position is increased to a certain level to simulate the lifting effect so that it looks like the player character is holding the enemy in midair.

Once the user releases the key G, the Throw() function is invoked to apply the throwing mechanic. This mechanic applies the throwing effect on the grabbed enemy by adding impulse on its skeletal mesh. The impulse is applied in the direction of the forward vector of the player character's follow camera with a little straight up and the amount is multiplied by the ThrowForce variable (ie. 50000). In order to be affected by the throwing impulse, the grabbed enemy is set to ragdolling before adding the impulse on it so that it simulates physics and starts flying in midair.

Classes & Variables

[ABeatEmUpCharacter]

Variable Name	Type	Purpose
GrabDistance	GrabDistance	Used to represent the distance that the character can grab the enemy.
GrabbedEnemy	AEnemy*	Used to reference the Enemy Actor who is being grabbed by the player character.
ThrowForce	float	Used to represent the force applied to throw the grabbed enemy.
IsGrabbingEnemy	boolean	Used to indicate the player character is grabbing an Enemy Actor.
Function Name	**Return Type**	**Purpose**

Grab()	void	Used to apply the grabbing mechanic on the nearest Enemy Actor the player character is facing.
Tick()	void	Used to update the location of the Enemy Actor being grabbed by the player character to maintain its relative position to the player character while the player character is moving.
Throw()	void	Used to apply the throwing mechanic on the EnemyActor being grabbed by the player character.

[AEnemy]

Variable Name	Type	Purpose
RagdollTime	float	Used to represent the amount of time responsible for stopping enemies from ragdolling after a specific period of time
RagdollTimerHandle	FTimerHandle	Timer used to handle the responsibility for stopping enemies from ragdolling after a specific period of time.
Function Name	**Return Type**	**Purpose**
Ragdoll()	void	Used to set the enemies to ragdolling.
StopRagdoll()	void	Used to stop the enemies from ragdolling.

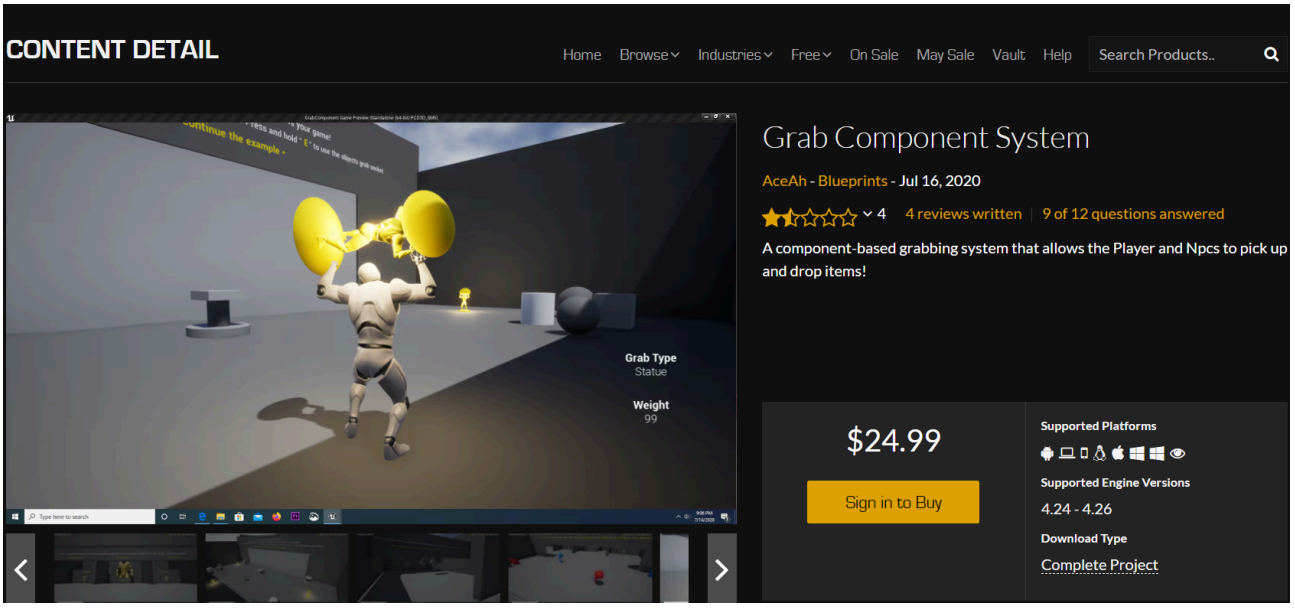
Properties and Values

Property	Description of Purpose	Value
GrabDistance	The distance that the character can grab the enemy.	250
ThrowForce	The force applied to throw the grabbed enemy.	50000

Inspiration / Reference Images

Grab Component System

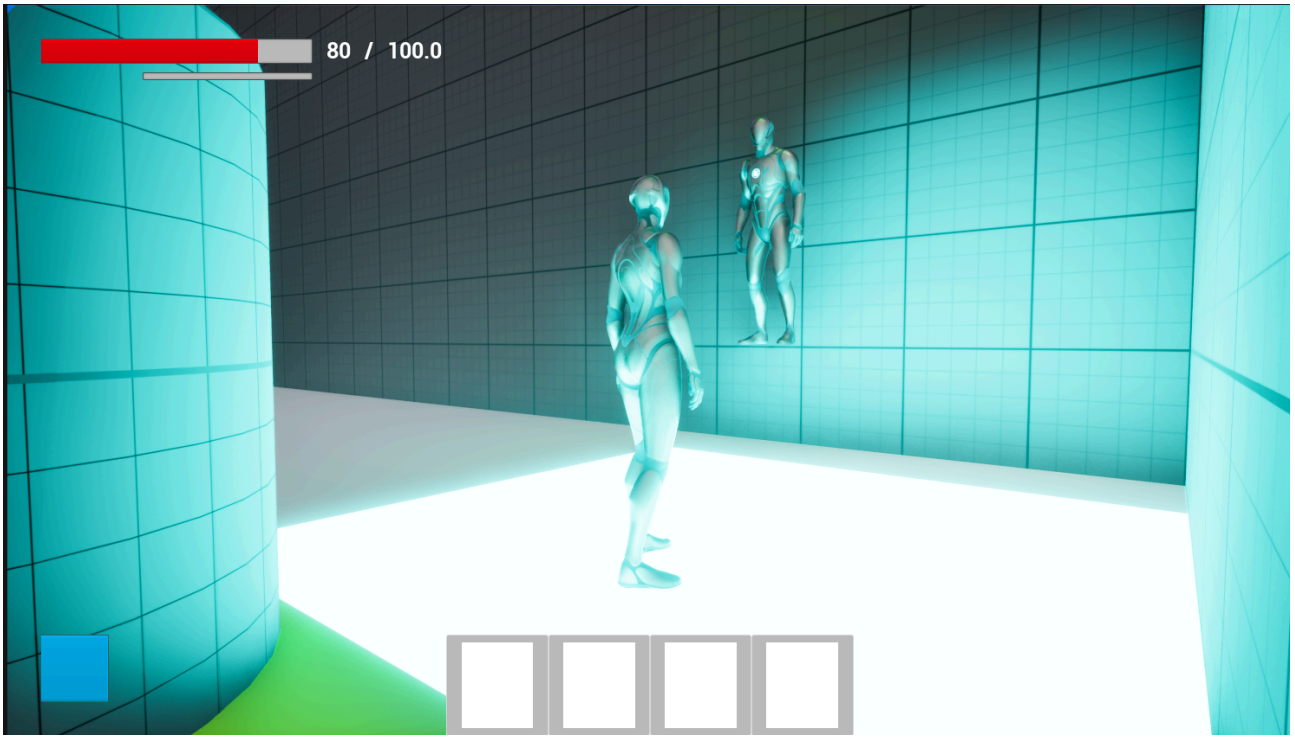
https://www.unrealengine.com/marketplace/en-US/product/grab-component-system



This physics mechanic adds depth to the combat system where the player is able to seize the grabbing enemy by throwing them towards other interactable objects such as trap doors to deal damage to them. This adds an extra layer of strategy to the combat.

In-Engine Screenshots

The player character is grabbing the enemy when the player presses and holds key G.



The grabbing enemy is thrown away by the player when the key G is released.

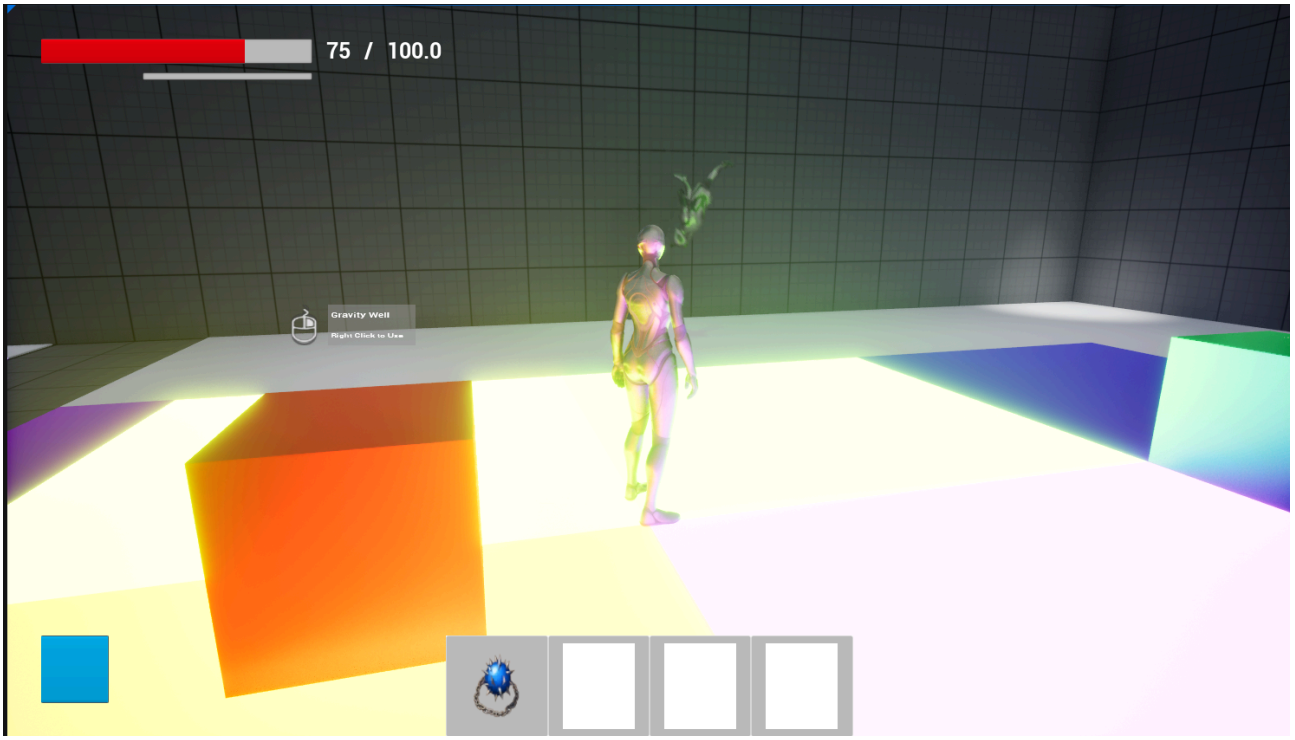
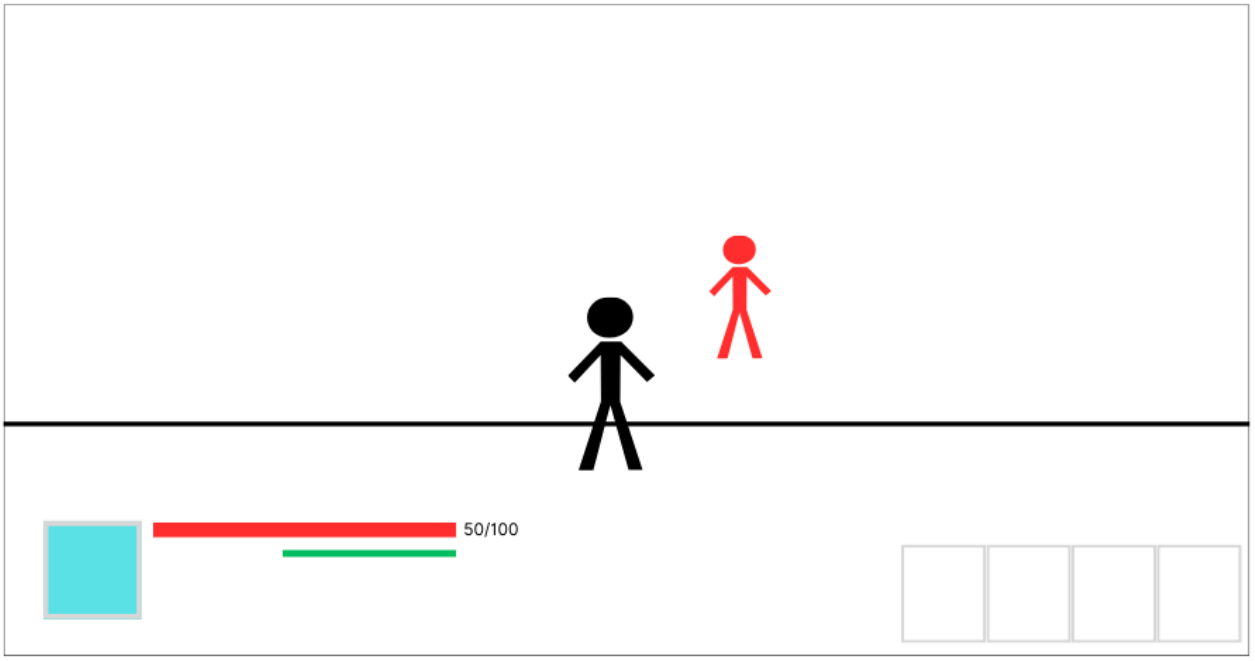
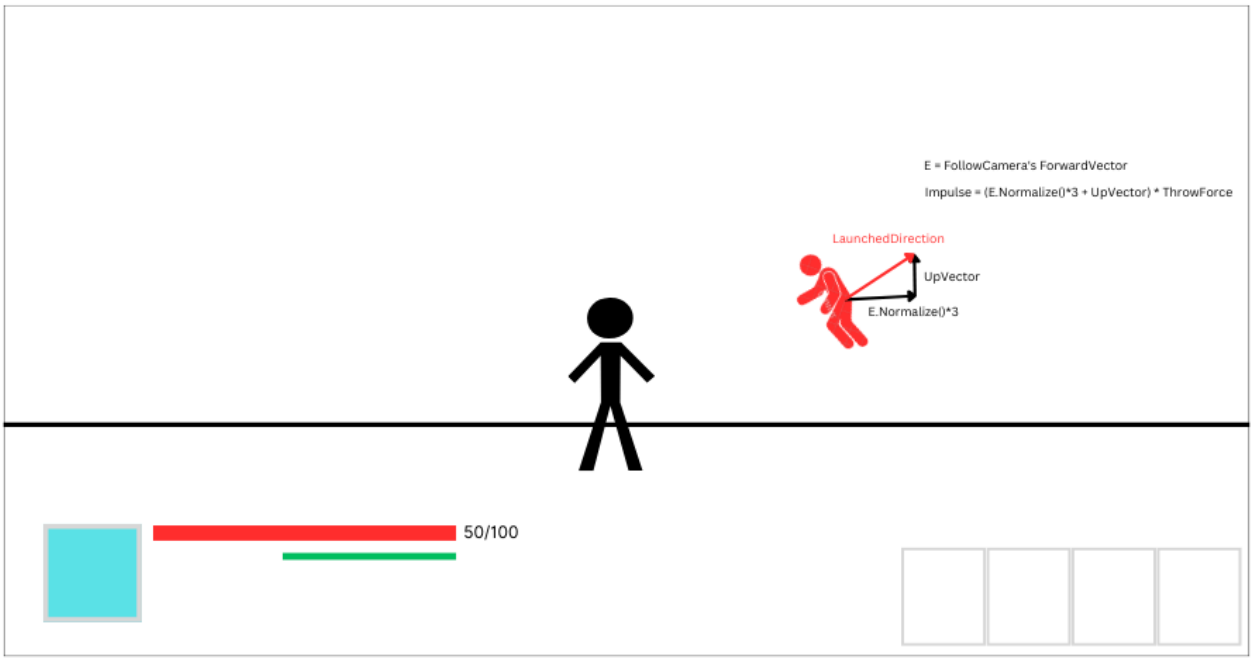
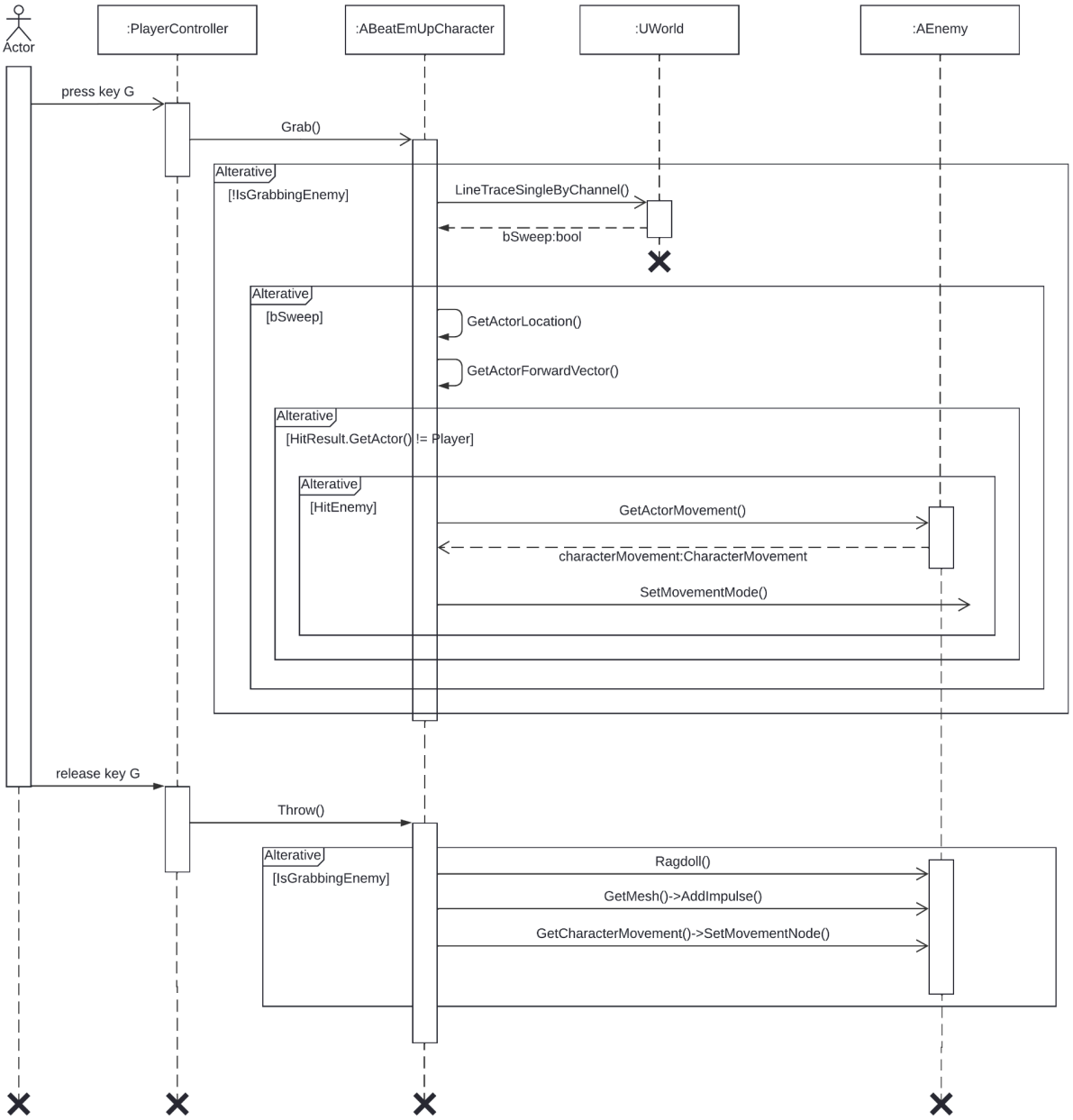


Diagram of Interaction






Actor
:PlayerController
:ABeatEmUpCharacter
:UWorld
:AEnemy
press key G
Grab()
Alterative
[!IsGrabbingEnemy]
LineTraceSingleByChannel()
bSweep:bool
Alterative
[bSweep]
GetActorLocation()
GetActorForwardVector()
Alterative
[HitResult.GetActor() != Player]
Alterative
[HitEnemy]
GetActorMovement()
characterMovement:CharacterMovement
SetMovementMode()
release key G
Throw()
Alterative
[IsGrabbingEnemy]
Ragdoll()
GetMesh()->AddImpulse()
GetCharacterMovement()->SetMovementNode()

Physics Constraint

Overview of Interaction

One of the types of the weapon that can be picked up by the player is a flail. It is made from a chain connected to a metal ball using physics constraint components as the hinges. Its physics-driven movement adds an element of realism to combat encounters, as players must anticipate and control the swinging motion of the spike ball to effectively strike enemies.

Interaction Description

How the Interaction Works

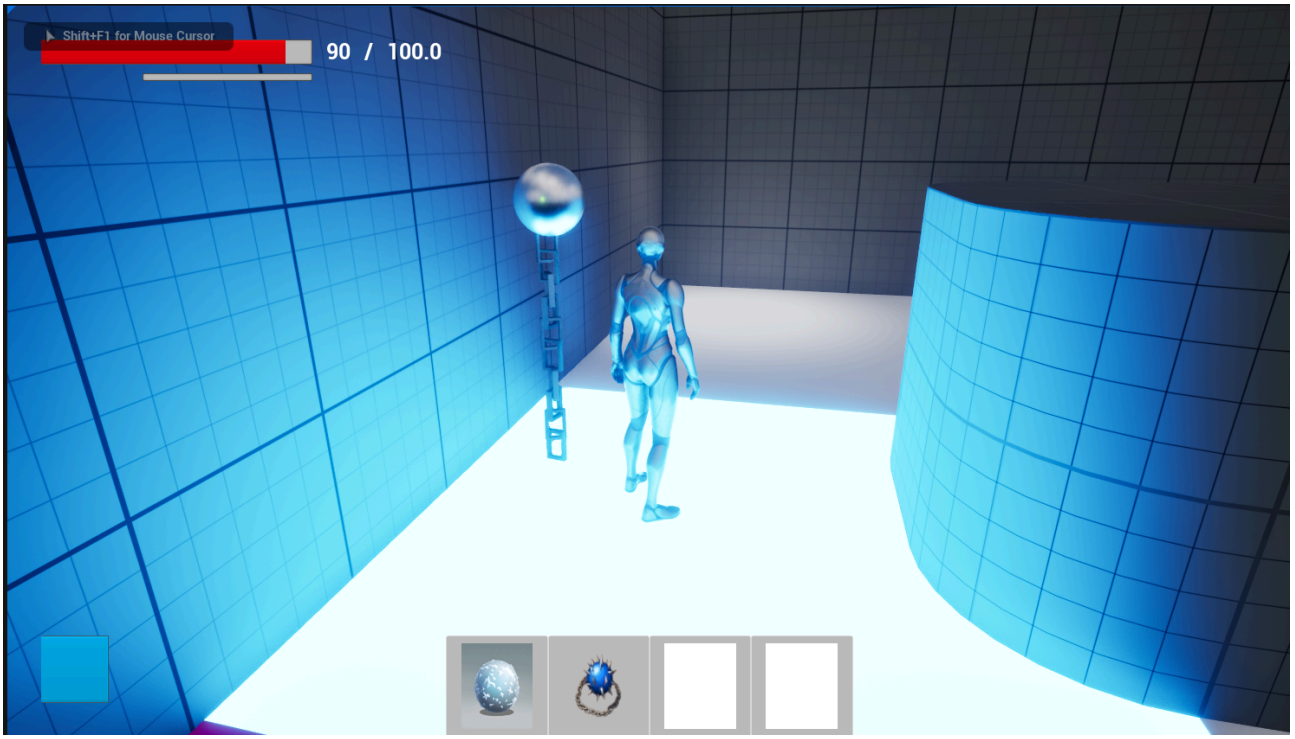
The physics constraint component is a fundamental aspect of the flail weapon's functionality in the game. When the player equips the flail, the chain part is attached to the socket located at the player character's left hand while the ball is attached to the right hand. This attachment is facilitated by the physics constraint which simulates a physical connection between the two objects. When the player presses key R to initiate an attack with the flail, the ball mesh is thrown forward by adding an impulse to the ball mesh in the direction of the player character's facing with a slight upward vector applied to simulate a throwing trajectory. The physics constraint ensures that the chain remains connected to the player character's hand throughout the attack action. This is achieved by imposing linear and angular constraints on the movement of the chain, preventing it from detaching or deviating excessively from its intended path. As a result, players can wield the flail with precision and control, making it stand out from other weapons.

Inspiration / Reference Images

Ball and Chain Trooper
https://zeldawiki.wiki/wiki/Ball_and_Chain_Trooper

A flail exhibits complex and dynamic movements such as swinging effectively showcases the effects of physics constraints.

In-Engine Screenshots

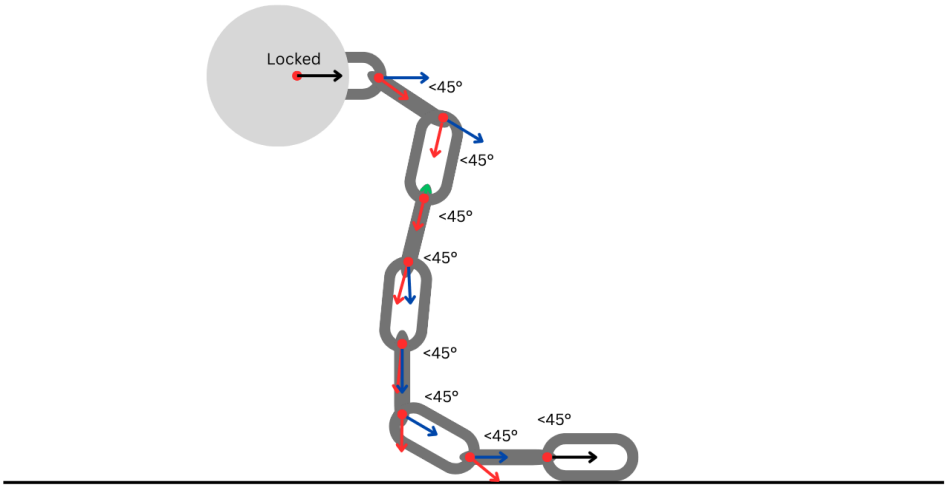


Properties and Values

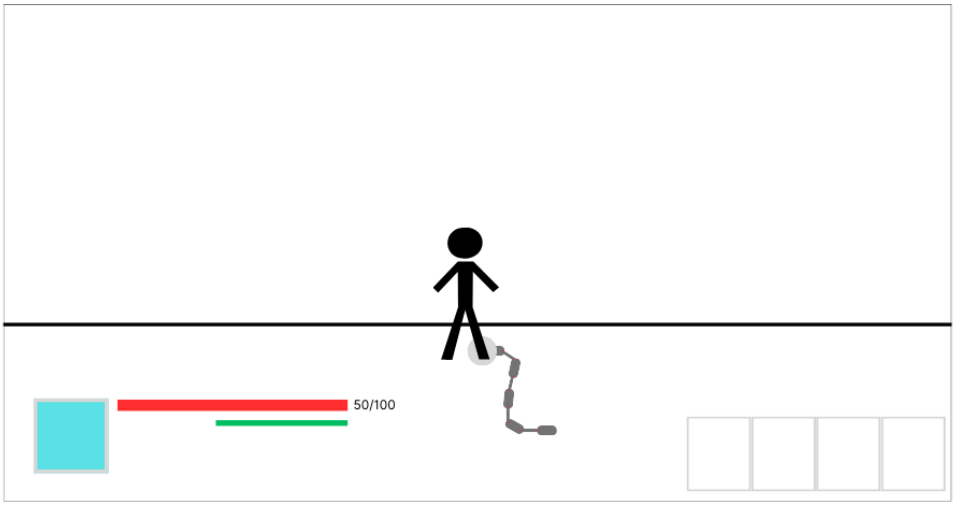
Property	Description of Purpose	Value
Angular Limit - Swing1 Limit	The limit on amount of rotation that can be applied on XY-plane	45 Degrees
Angular Limit - Swing2 Limit	The limit on amount of rotation that can be applied on XZ-plane	45 Degrees
Angular Limit - Twist Limit	The limit on symmetric amount of roll along the X-axis	45 Degrees
Linear Limit - X motion	To restrict the movement of the chain along the X-axis	Locked
Linear Limit - Y motion	To restrict the movement of the chain along the Y-axis	Locked
Linear Limit - Z motion	To restrict the movement of the chain along the X-axis	Locked

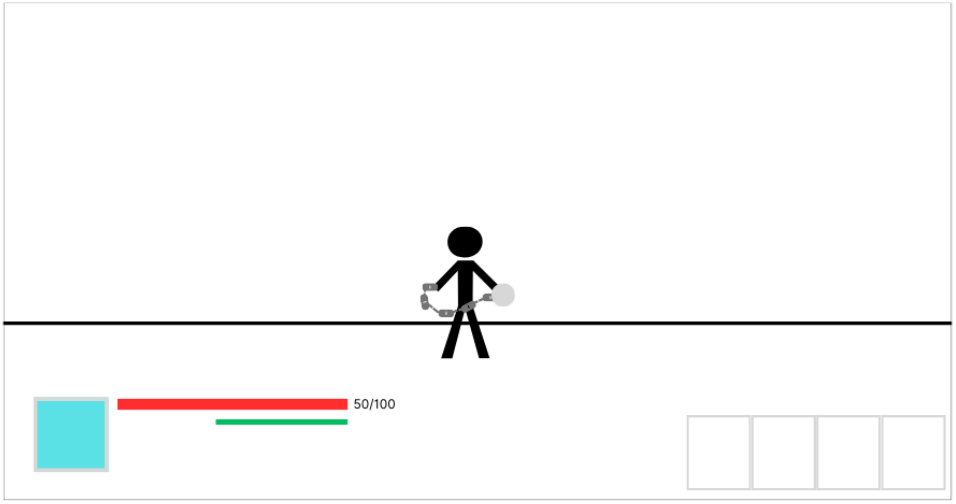
Diagram of Interaction

Physics constraint components used and placed at the red dots position



Player overlaps with the flail. After picking up, the end of the chain is attached to the player's left hand socket, and the ball mesh is attached to the player's right hand socket.





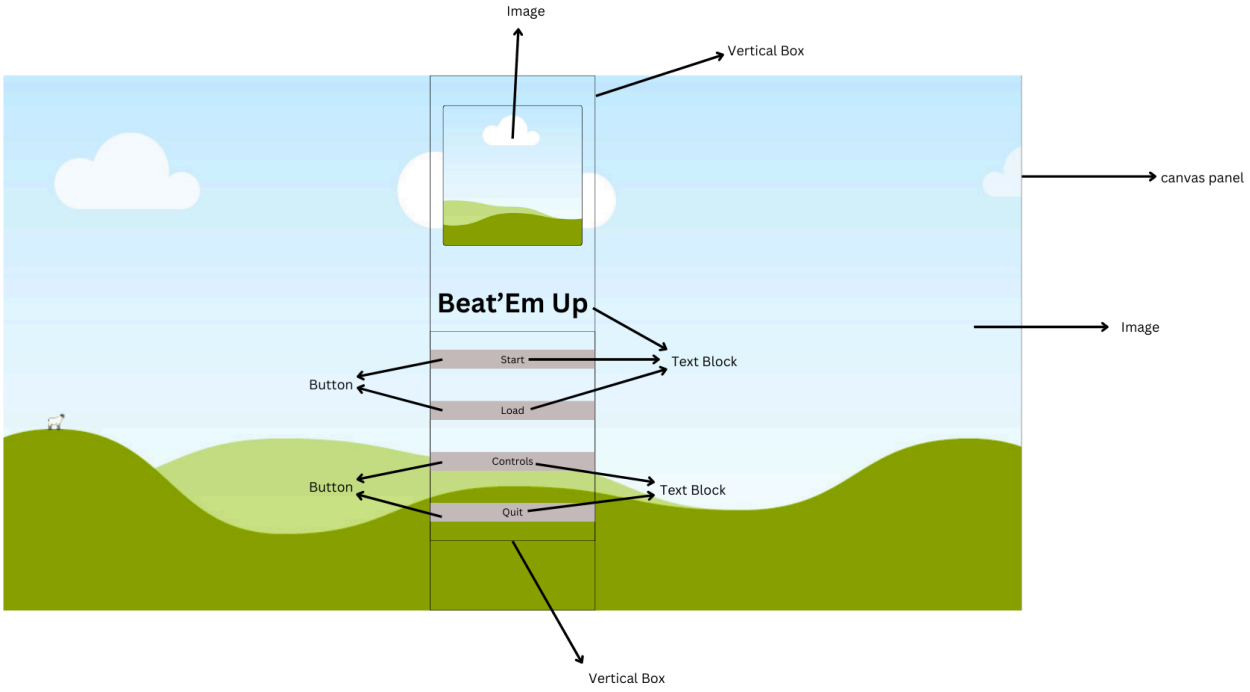
UI Design

Main Menu UI

UI Overview
The main menu is itself a level that will be shown at the beginning of the game.

UI Description / Functionality
The main menu contains the title of the game, a Start button, a Load button, a Control button and a Quit button where each has the respective functionality as their name. When the Start button is clicked, the OnClick event listener is triggered and the engine will open the ThirdPersonMap level to start the game. If the Quit button is clicked, the OnClick event listener is triggered and the engine will just simply quit the game.

UI Wireframe



In-Game UI

UI Overview
The interaction prompts are implemented for all of the Interactable (eg. Gravity Well and Trap door) to provide players a clear guidance on what can be done on those interactable items.

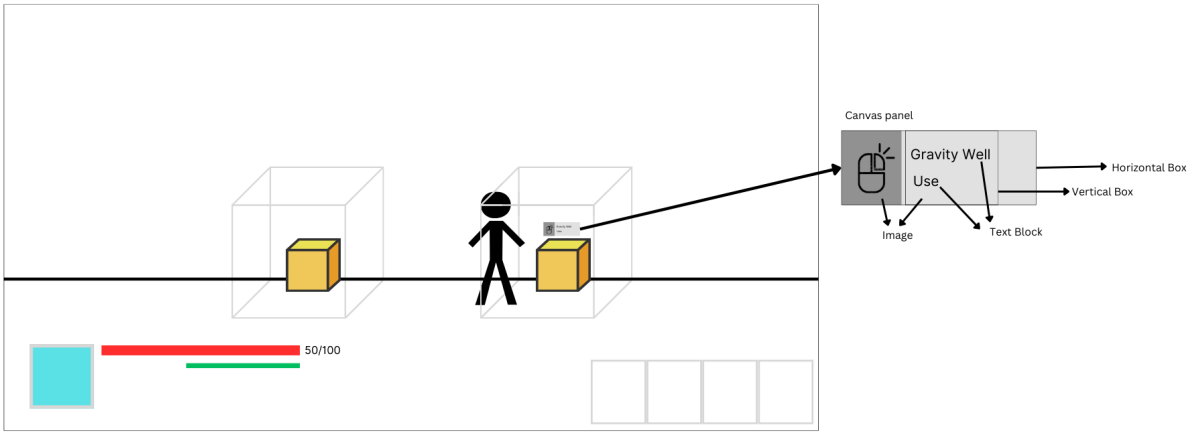
UI Description / Functionality
For each Gravity Well and Trap Door, a Widget Component is created to add the Interaction Prompt UI widget to the interactable Actor. When the game starts, the Interaction Prompt UI widget is created and the texts for the text block are set. This includes the name of the interactable objects, the instruction to interact with the objects and also a simple key icon for the instruction. The User Interface Space is set to Screen to allow the interaction prompt UI to always face towards the player's camera. Furthermore, a Box Component attached to the Root Component to show and hide

the interaction prompt whenever the overlap events are triggered. When the player character begins overlapping with the box component (ie. OnComponentBeginOverlap), it invokes the OnPlayerOverlap() function which makes the widget component visible in the game world by using SetVisibility() method available for the WidgetComponent. Conversely, when the player ends overlap with the box component (ie. OnComponentEndOverlap), it will invoke the OnPlayerEndOverlap() function to hide the widget component again. This ensures the interaction prompt UI only be displayed when the player character is near the interactable object.

C++ Properties & Purpose

Property	Description of Purpose
KeyImage	Image holder used to show the image of the key prompt to interact.
ActionText	Text Block used to show the prompt text.
InteractableNameText	Text Block used to show the label name of the interactable actor (eg. Gravity Well).
DisplayLocation	FVector2d used to store the 2D vector of the position to display the UI widget.

UI Wireframe



Artificial Intelligence

Overview of AI

The melee enemy AI is equipped with weapons in both its left and right hands. This advanced version of the basic enemy AI introduces variety into the combat system of the game. It is able smartly selects which weapon to used based on conditions such as ammo availabillity and distance to the target player. This enables it to perfor ranged attacks when advantageous, making itself to gain a tactical edge in combat.

AI Description

AI Abilities

Key abilities:

1. Patrol around
2. Chase the player
3. Attack the player using left weapon
4. Attack the player using right weapon
5. Sprint to attackable range if it is too far away from the player to perform attack
6. Walk towards player to deal damage to player while destroying itself if the ammo is used up.

Inputs & Senses

AI Senses

Sense Name	Property	Value
Sight	Sight Radius	1000
	Lost Sight Radius	1100
	Sight Age	3.5
	Field of View	45

Blackboard Values

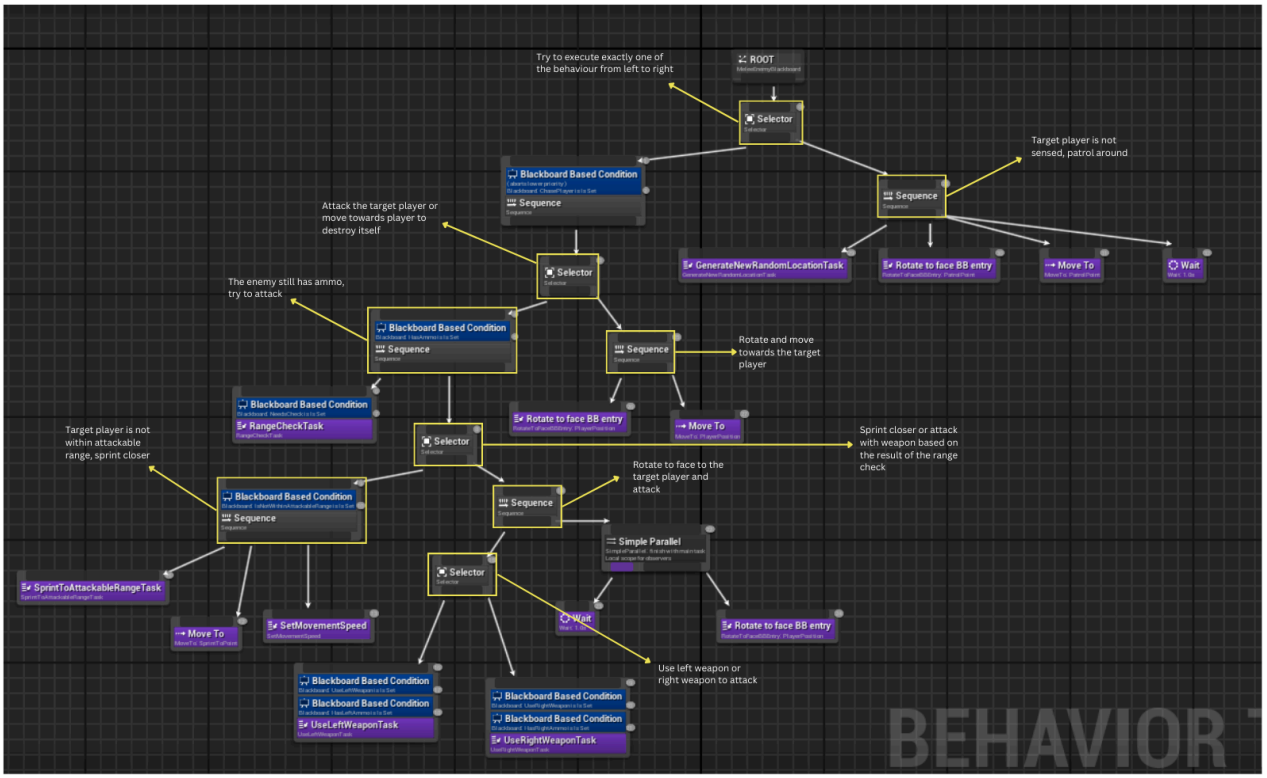
Property	Description	Related Actions
PlayerPosition	Integer variable containing the location of the player. Updated whenever the target player is sensed.	Chase the player or perform attack if within attackable range.
PatrolPoint	Vector variable containing the location to patrol to. Generated when target player	Randomly patrolling within a specified ranged.

Property	Description	Related Actions
	is not sensed.	
ChasePlayer	Boolean variable indicating it needs to chase the player. Updated whenever the target player is sensed.	Chase the player.
NeedsCheck	Boolean variable indicating a range check is needed. Updated when the melee enemy sensed the target player, sprint nearer to the target player or after performing an attack.	Check the target player is within the melee enemy's attackable range.
IsNotWithinAttackableRange	Boolean variable indicating the target player is not within the attackable range. Updated after the range check.	Sprint closer to the target player.
SprintToPoint	Vector variable containing the new location to sprint to. Calculated for the melee enemy to move nearer to the target player.	Sprint closer to the target player.
HasAmmo	Boolean variable indicating any of the weapons still has ammo. Updated after the melee enemy performed an attack.	Attack player or chase to hit the player while destroying itself.
HasLeftAmmo	Boolean variable indicating the left weapon still has ammo. Updated after the melee enemy performed an attack using its left weapon.	Attack player using left weapon.
HasRightAmmo	Boolean variable indicating the right weapon still has ammo. Updated after the melee enemy performed an attack using its right weapon.	Attack player using right weapon.
UseLeftWeapon	Boolean variable indicating using left weapon to attack is	Attack player using left weapon.

Property	Description	Related Actions
	a better strategy. Updated after the range check.	
UseRightWeapon	Boolean variable indicating using the right weapon to attack is a better strategy. Updated after the range check.	Attack player using right weapon.

When the player is not sensed, a random location is generated and stored in the PatrolPoint variable to prompt the melee enemy AI to move to this point simulating patrolling. When the player is sensed within the Sight Radius and Field of View, the player's position is located and updated in the PlayerPosition variable, along with setting the ChasePlayer as true to prompt the melee enemy AI to switch from patrolliing to chasing the player. Then, the range check is done to determine the best action to be performed to the sensed player. If the left weapon out of ammo and the player is within the attackable range of the right weapon, the UseRightWeapon variable will be set to true to prompt the melee enemy AI to attack the player with right weapon. Similar approach is applied to left weapon by setting the UseLeftWeapon variable to true when right weapon is used up. If both weapons are still available in terms of ammo condition and range, the long range weapon will be prioritised. Else, it will use the weapon available in the attack range. If both weapons are not available in the attack range, the IsWithinAttackableRange variable is set to false and a closer location is calculated and store in the SprintToPoint variable to tell the melee enemy AI to sprint closer to the player and then a new check is performed. After losing direct sight on the player but remaining within the Lost Sight Radius, the melee enemy AI can remember the player's position for a period of time indicated by the Sight Age. During this time, it relies on the last known PlayerPosition to continue its pursuit. If the Sight Age expires without reacquiring the player, it will abandon the chase and resume patrolling.

Behaviour Tree Graph



When the player is not sensed, “ChasePlayer” is set to false so the first selector fails the left behaviour and tries the right behaviour to patrol around. To patrol, GenerateRandomLocationTask is executed to generate random location, then the enemy rotates to the direction and moves to the patrol point.

When the player is sensed, “ChasePlayer” is set to true so the first selector does the left behaviour to attack the target player.
If the enemy still has ammo, it does the RangeCheckTask to check whether the target player is within the attackable range. This task determines the Blackboard values “IsNotWithinAttackableRange”, “UseLeftWeapon” and “UseRightWeapon” based on the current condition of the enemy. If the target player is not within the range, the selector does the left behaviour to sprint closer. The SprintToAttackableRangeTask calculates the exact point to sprint to and assign to the SprinToPoint Blackboard value. Then, it changes the movement speed of the enemy to simulate sprinting effect. After moving to the SprintToPoint, SetMovementSpeedTask is execute to set back the previous movement state of the enemy. Else, the selector does the right behaviour which will attack the target player using left or right weapon based on the result value from the RangeCheckTask while rotating to face the target player.
Else, the enemy uses up all the ammo, it rotates to the target player's direction and moves towards it.

Save Data

Class	Property	Purpose
ABeatEmUpCharacter	ActorLocation	Current location of player. Enables loading to set player to previous location
	ActorRotation	Current rotation of actor. Enables loading to set player to previous rotation.
	CurrentHealth	Current health of player. Enables loading to set player's health to previous health.
	MaxHealth	Maximum health of player. Enables loading to set player's maximum health to previous maximum health as the maximum health changes when player level up.
	CurrentEXP	Current EXP of player. Enables loading to set player's EXP to previous EXP.
	EXPToLevel	Current level of the player. Enables loading to set the level of the player to the previous level.
	PunchDamage	Current damage dealt by punching. Enables loading to set player's punching damage to previous damage value as the value changes when player level up.
	Inventory	Current inventory list of the player. Enables loading to add back the items to the player's inventory.
	EquippingWeapon	Current weapon the player's equipped with. Enables loading to set the player to equip the previously equipping weapon.
	NumOfEnemiesDefeated	Current number of enemies

		defeated by the player. Enables loading to set the number of enemies defeated to the previous number.
	bLevelLoaded	Boolean indicating the streaming boss level is loaded.
AEnemy	ActorLocation	Current location of enemy. Enables loading to set enemy to previous location
	ActorRotation	Current rotation of enemy. Enables loading to set enemy to previous rotation.
	CurrentHealth	Current health of enemy. Enables loading to set enemy's health to previous health.
	MaxHealth	Maximum health of enemy. Enables loading to set enemy's maximum health to previous maximum health
	RagdollState	Current state of the enemy. Enables loading to set the enemy to ragdoll state if it is ragdolling in the previous saved game.
	MeshLocation	Current position of the enemy's mesh. Enables loading to set the mesh to the previous location as the mesh might be applied impulse and hit away.
	MeshVelocity	Current velocity of the enemy's mesh. Enables loading to set the mesh to the previous velocity as the mesh might be simulating physics when applying impulse on it.
AMeleeEnemy	ActorLocation	Current location of enemy. Enables loading to set enemy to previous location
	ActorRotation	Current rotation of enemy. Enables loading to set enemy

		to previous rotation.
	CurrentHealth	Current health of enemy. Enables loading to set enemy's health to previous health.
	MaxHealth	Maximum health of enemy. Enables loading to set enemy's maximum health to previous maximum health
	RagdollState	Current state of the enemy. Enables loading to set the enemy to ragdoll state if it is ragdolling in the previous saved game.
	MeshLocation	Current position of the enemy's mesh. Enables loading to set the mesh to the previous location as the mesh might be applied impulse and hit away.
	MeshVelocity	Current velocity of the enemy's mesh. Enables loading to set the mesh to the previous velocity as the mesh might be simulating physics when applying impulse on it.
	MaxWalkSpeed	Current maximum walk speed of the enemy. Enables loading to set the enemy's maximum walk speed to the previous speed as the enemy might sprinting in the previous saved game.
	LeftAmmo	Current amount of ammo left for the left weapon. Enables loading to set the amount of ammo of the left weapon to the previous amount.
	RightAmmo	Current amount of ammo left fot the right weapon. Enables loading to set the amount od ammo of the right weapon to the previous amount.

	LeftWeaponLocation	Current position of the left weapon. Enables loading to set the left weapon to the previous position as the left weapon is a bomb which might be projected in midair in the previous saved game.
	LeftWeaponVelocity	Current velocity of the left weapon. Enables loading to set the left weapon to the previous velocity as the left weapon is a bomb which might simulate physics when being projected in the midair in the previous saved game.
ABullet	ActorLocation	Current location of bullet shot by the enemy. Enables loading to set the bullet to the previous location.
	ActorRotation	Current rotation of bullet shoot by the enemy. Enables loading to set the bullet to previous rotation.
ADynamicColorFloor	DynamicFloorColor	Current colour of the floor. Enables loading to set the floor to the previous colour as the colour might change when the player overlaps with it.
AStick	StickLocations	Current location of the stick. Enables loading to set the stick to the previous location.
	StickRotations	Current rotation of the stick. Enables loading to set the stick to the previous rotation.
	DestroyedSticks	Check the stick should be destroyed by fading out.
	StickVelocities	Current velocity of the stick launched by the boss enemy. Enables loading to set the stick to the previous velocity.
	ActiveSticks	Check the active state of the movement component of the

		stick. Enables loading to activate or deactivate the movement component.
	StickOpacities	Current opacity of the material applied to the stick. Enables loading to set it to the previous opacity as it might change when the stick is fading out.
	ToBeProjectSticks	Check the stick is waiting to be launched. Enables loading to set the waiting back.
	StickProjectTimes	Current time to be waited to launch the stick. Enables loading to set the waiting to the previous time.
BossEnemy	BossEnemyLocation	Current location of the boss enemy. Enables loading to set the boss enemy to the previous location.
	BossEnemyRotation	Current rotation of the boss enemy. Enables loading to set the boss enemy to the previous location.
	BossEnemyCurrentHealth	Current health of the boss enemy. Enables loading to set the boss enemy to the previous health.
	BossEnemyMaxHealth	Current maximum health of the boss enemy. Enables loading to set the boss enemy to the previous maximum health.
	BossEnemyRagdollState	Current state of the boss enemy. Enables loading to set the boss enemy to ragdoll state if it is ragdolling in the previous saved game.
	BossEnemyMeshLocation	Current position of the enemy's mesh. Enables loading to set the mesh to the previous location as the mesh

		might be applied impulse and hit away.
	BossEnemyMeshVelocity	Current velocity of the enemy's mesh. Enables loading to set the mesh to the previous velocity as the mesh might be simulating physics when applying impulse on it.
	bIsLeftChildDefeated	Indicator keeping track the consciousness of the left child. Enables loading to set the boss enemy to spawn or not spawn child enemies.
	bIsRightChildDefeated	Indicator keeping track the consciousness of the right child. Enables loading to set the boss enemy to spawn or not spawn child enemies.

Dynamic Material

DynamicColorFloor

Overview of Effect

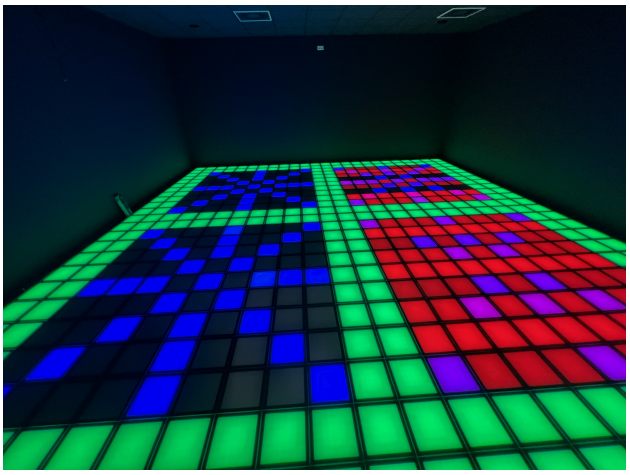
The DynamicColorFloor is an interactive plane applied with the DynamicColor material that enhances the gameplay experience by responding to player interactions. This material is designed to dynamically change to a random colour whenever the player overlaps with it. During the overlap, the floor also glows which also improves visibility in the dark mode game settings, guiding players and making the environment more navigable. As a result, players can create a colourful trail that adds a unique and visually stimulating element to each playthrough.

Effect Description

A BoxComponent is created for the DynamicColorFloor to trigger the dynamic colour change when an overlap event occurs. This BoxComponent serves as a trigger volume, detecting when the player enters or exits its boundaries. When the player begins to overlap with the box component, it set bIsOverlapping to true. Every tick, if the player is overlapping, it will set the material colour to interpolate between two colours in the Colors array using LerpUsingHSV() method based on the time and the colour change speed. This new colour is set to the scalar parameter value “BaseColor” to make the floor show a new colour. Simultaneously, to achieve the glowing effect, the scalar parameter value "EmissiveStrength" is set to 50f which increases the emissive strength of the material, causing it to emit light and glow in the dark environment. Once the player ends the overlap, bIsOverlapping is set to false to stop colour changing so that the floor retains the new colour. By setting the scalar parameter value “EmmissiveStrength” back to 0f, the glowing effect is effectively turned off while maintaining the colour changed, marking the player's path through the environment.

Inspiration / Reference Images:

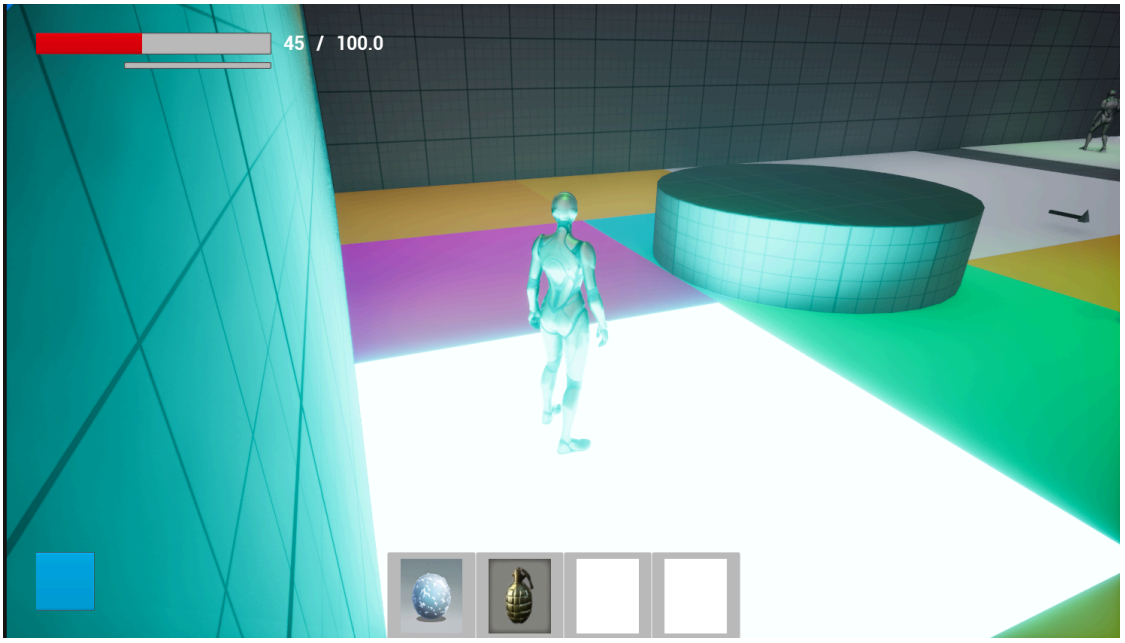
GameNight Kerpen - Interactive LED Floor
https://streetcommunication.com/interactive-led-floor-screen-game-night/



When integrated with user interaction (ie. stepping on different parts of the floor changes the colours and emissivity), it creates a playful environment and energises the space.

In-Engine Screenshots:

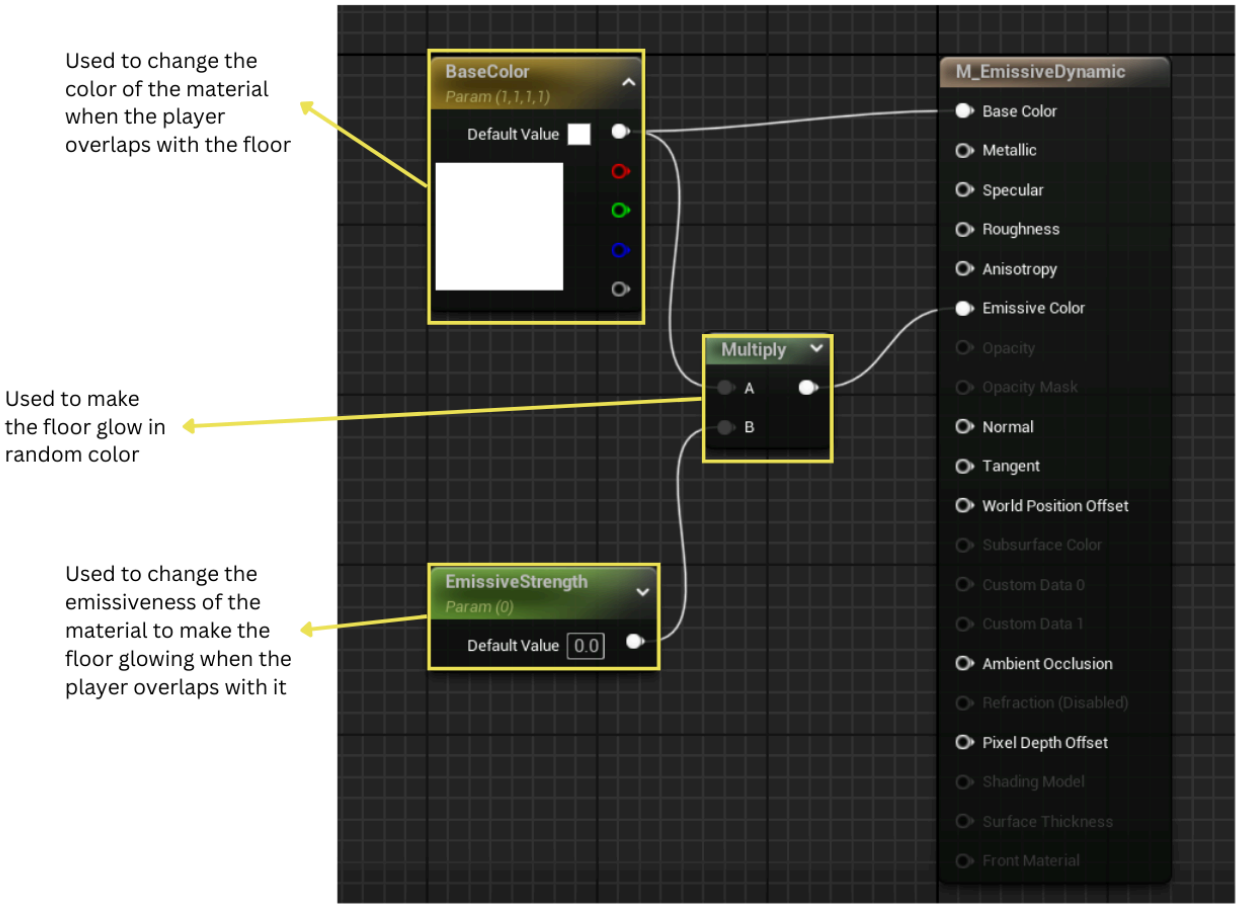
The player character overlaps with the Box Component attached to the floor, resulting in the rainbow colour change and glowing effect of the floor.



Properties and Values

Property	Description of Purpose	Value
EmissiveGlowStrength	Used to control the emissive strength of material dynamically.	float 50f
Colors	Used to store the colours to interpolate between.	TArray<FLinearColor> Red - HSV(359, 1, 1) Orange - HSV(25, 1, 1) Yellow - HSV(62, 1, 1) Green - HSV(122, 1, 1) Blue - HSV(239, 1, 1) Purple - HSV(304, 1, 1) Violet - HSV(277, 1, 1)
ColorChangeSpeed	Used to control the speed of changing the material colour.	int32 2

Node Graph



C++ Breakdown

Variable Name	Type	Purpose
BaseColor	VectorParameter	Used to control colour of material dynamically.
EmmissiveStrength	ScalarParameter	Used to control the emmissive strength of material dynamically.

Niagara Particle Effect

Niagara Particle Effect - Name of Effect

Overview of Effect

The bullet shot by the enemy has a trail effect consists of two distinct types of Niagara particle system working together to create a dynamic and immersive trail for the enemy's attacks, enhancing the visual spectacle of the gameplay experience.

Effect Description

The first particle emitter contributes to the primary trail effect, representing the trajectory of the enemy's bullet as it travels through the environment. This trail system follows the path of the bullet and features glowing streaks using jitter position. This jitter position adds a small amount of variation to the moving direction of the particles, creating a blazing visual effect. Complementing this primary trail effect, the second particle emitter continuously emits sparks along the bullet's path. These sparks serve to further highlight the bullet's movement with their rapid and energetic emission, creating an impression of immense speed and kinetic energy to the overall effect. The sparks also create a visual effect emphasising the bullet is moving at a very high speed.

Once the bullet is shot, the spark size and trail length increases to simulate the visual effect of power accumulation for the bullet. When the bullet hit something (eg. wall, floor), the length of the trail decreases every tick by decreasing the maximum life time of the particles. Also, the velocity added to the spark particles once being spawned decreases every tick to simulate the loss of energy for the bullet. When it reaches 0, the bullet and the emitter will be destroyed.

Inspiration / Reference Images:

Projectile Trail in UE5 Niagara
https://cghow.com/projectile-trail-in-ue5-niagara-tutorial-2/

Trails can convey a sense of speed and motion. Seeing the path of the bullet can make the shooting mechanics feel more dynamic and engaging, giving a better sense of the fast-moving.

In-Engine Screenshots:

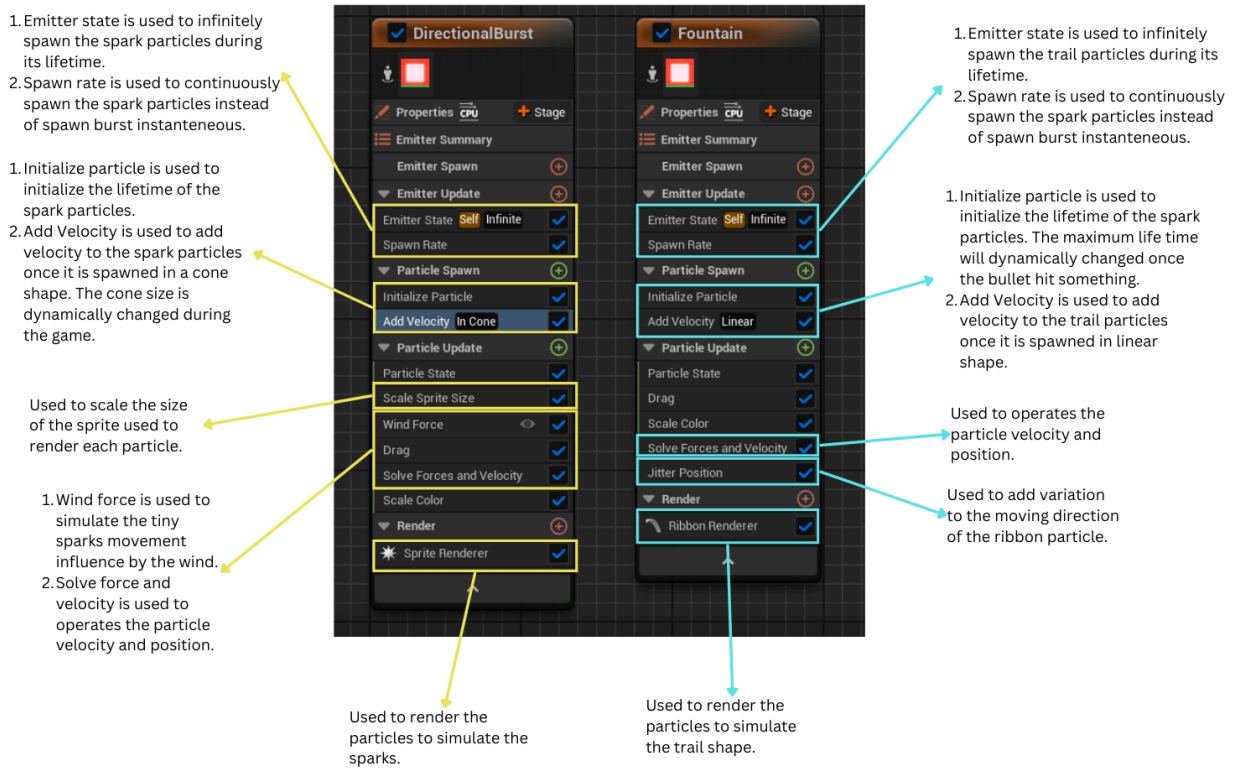
The bullet is shot with a trail following it.

Properties and Values

Property	Description of Purpose	Value
Wind Force	Used to mimic the influence of wind on the bullet's path, adding realism and dynamic movement to the trail particles.	Strength: 5000f
Spawn Rate	Used to adjust the spawning rate of the particles.	Spark - 5000 Trail - 500
Jitter Position - Jitter Amount	Used to add a small variation to the moving direction of the particle	7.0
ChangeSpeed	Used to determine the increasing/decreasing rate of the particles once the bullet shot/hit something.	float 2
ParticleMaxVelocity	Used to determine the upper bound of the maximum velocity of the spark particles.	float 800
ParticleMinVelocity	Used to determine the lower bound of the maximum velocity of the spark particles.	float 0
ParticleMaxLifeTime	Used to determine the upper bound of the maximum velocity of the trail	float 1

	particles.	
ParticleMinLifeTime	Used to determine the lower bound of the maximum velocity of the trail particles.	float 0

Niagara System / Emitters Breakdown



C++ Parameters Breakdown

Variable Name	Type	Purpose
Max Velocity	float	Used to adjust the maximum velocity of the AddVelocity under spark particle spawn. This allows the spark of the bullet trail to spawn from a smaller cone size velocity to a larger cone size and becomes small back when it hits something.
Life Time	float	Used to adjust the maximum lifetime of the Initial Particle under trail particle spawn. This allows the trail of the bullet trail to spawn from a short trail to a longer trail and becomes short back when it hits something.

Lighting Effect

Overview of Effect

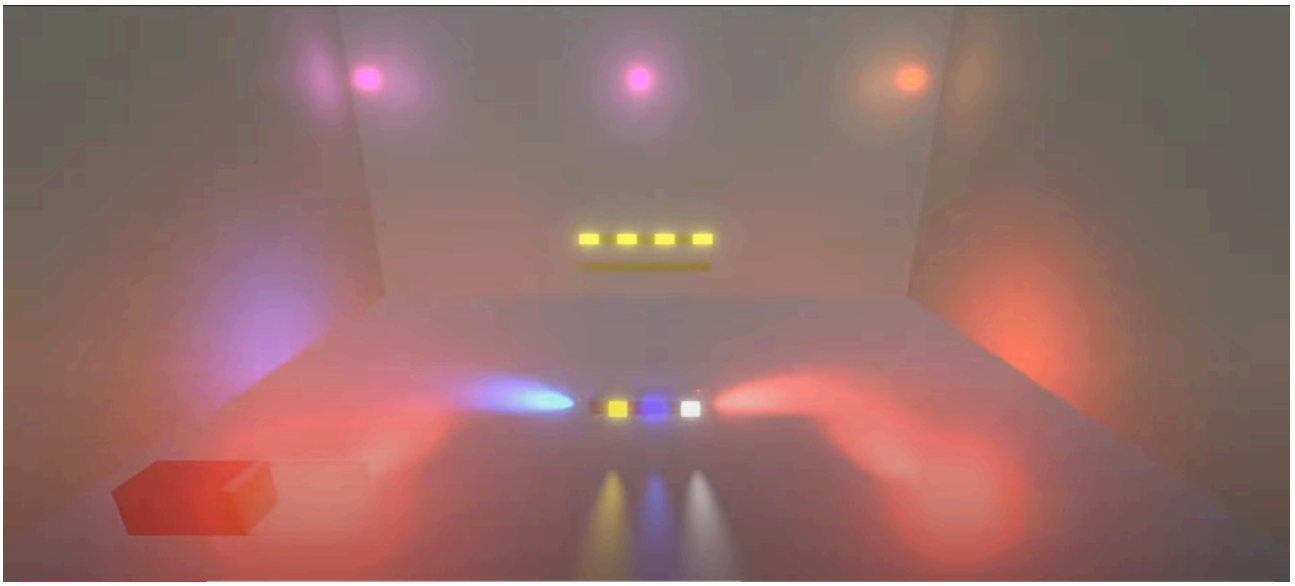
In the Boss Level, the light effect is created through the use of spotlights to emit a striking red hue, which serves to evoke a sense of urgency and danger. The two spotlights are positioned to an emitter that continuously rotates to illustrate the beams of light sweeping across the vacant space. To intensify the tense atmosphere, a flickering effect is applied to the material used by the spotlight to mimic the nervous energy of the player. This dynamic lighting not only draws the player's attention but also contributes to the immersive nature of the boss battle.

Effect Description

A flickering material is firstly created to be used as the spotlight's light function material. This material uses the Time input parameter along with sine and fractional to create a continuous flickering effect. By leveraging the Time input, the material dynamically adjusts its properties over time, causing the intensity of the light to fluctuate. This heightens the game's atmosphere by changing the frequency of the light's fluctuations based on the current health of the boss enemy. As its health decreases, the frequency increases, making it progressively harder for the player to see clearly and accurately pinpoint the boss enemy's location, thereby escalating the challenge. Meanwhile, the spotlight's colour smoothly transitions from purple to red, mirroring the rising tension and urgency of the situation. Additionally, to achieve the rotating spotlight effect, a Rotation Movement Component is created in the ARotatingSpotLight which is responsible for continuously rotating the mesh component and also the spotlights attached. Upon spawning, the RotatingSpotLight's mesh component begins to rotate at a specified rate, causing the attached spotlights to sweep across the vacant space in a circular motion.

Inspiration / Reference Images:

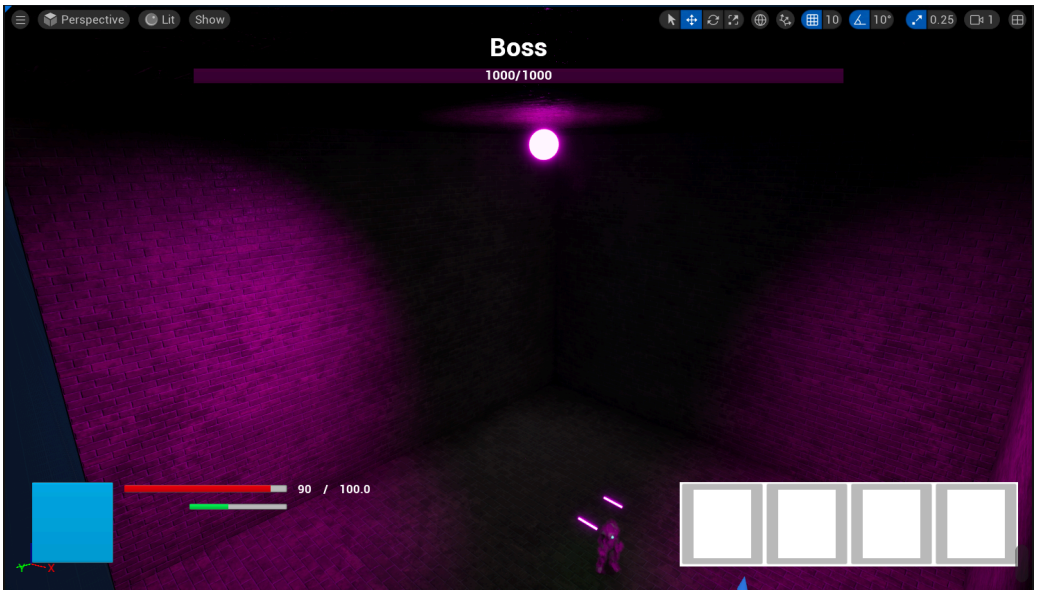
Rotating and Flashing Light in Unity (Easy Emergency Lights/Siren) https://www.youtube.com/watch?v=W7P227uLMOk

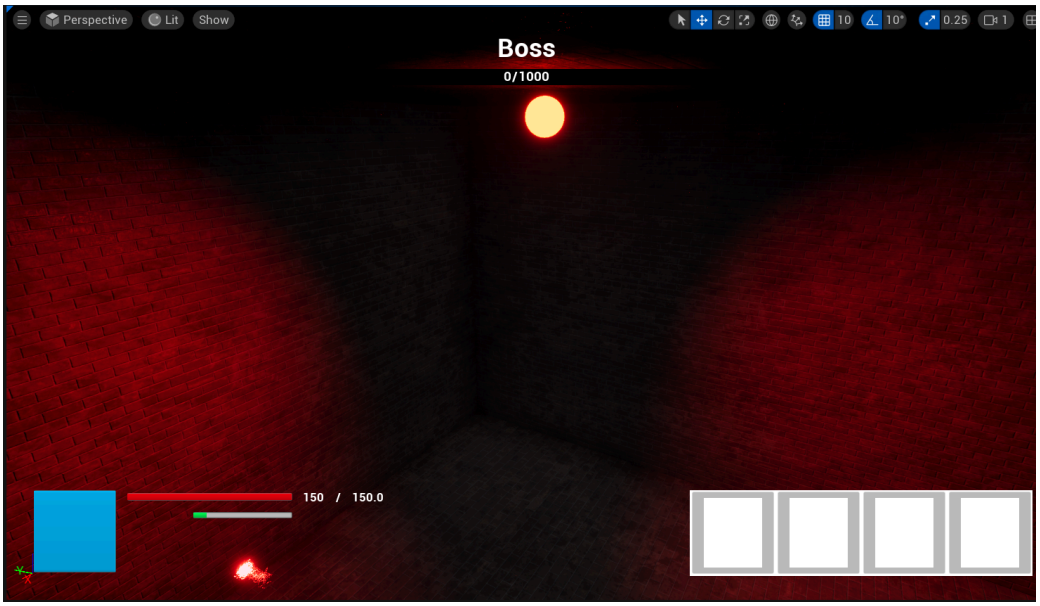


The rotating spotlight creates a sense of urgency and tension. Its sweeping motion across the darkened environment amplifies the player's sense of danger and the stakes of the encounter. Combined with the teleport behaviour of the boss enemy, it also creates a visual disturbance for the player to locate the boss enemy accurately.

In-Engine Screenshots:

The rotating spotlight is placed at the roof top of the boss level. It keeps rotating once the level start streaming. The colour and flickering frequency changes based on the current health of the boss enemy.





Properties and Values

Property	Description of Purpose	Value
Intensity	Used to set the intensity of the spotlight.	10000000
HealthyColour	Used to set the start colour of the light	HEX Linear 49004DFF
DeadColour	Used to set the end colour of the light	HEX Linear 4B000000

C++ Breakdown

Variable Name	Type	Purpose
Light Color	FLinearColor	Smoothly transitions from HealthyColour to DeadColor based on the current health of the boss enemy.
Light Functional Material - Color	FLinearColor	Smoothly transitions from HealthyColour to DeadColor based on the current health of the boss enemy.
Light Functional Material - Frequency	float	Used to increase the frequency of fluctuation for the light based on the current health of the boss enemy.