Assignment 2c - Technical Design Document

Spirits of Mythos

https://docs.google.com/document/d/1J63J_GPE_EPLKXcVBO8CtJlzHWTpcfCYNq2AC9BkDqo/edit?usp=sharing

Tameka Lougoon

Project Overview

Spirits of Mythos is a fantasy RPG game that involves puzzle solving elements. You play as the spirit of a fae who was murdered by a ruthless, magic hating king. His hatred for magic has led him to traverse the kingdom, seeking to rid it of its magic by capturing magical creatures and caging them in the castle's dungeons, to be lost to the ages.

Core Mechanic Overview

As a spirit of a forsaken fae, you must use your possessing capabilities to release and possess the captured creatures in a quest to free you all from the king's reign. With the aid of mythical creatures and their unique abilities, explore the dungeons and solve puzzles to reclaim your freedom and take revenge.

Explore the gruelling, dark dungeon, meeting different creatures the further you traverse. Befriend them by opening their cages, and they will be forever in your debt. Ask them nicely, and they won't put up much of a fuss when you possess their minds and temporarily claim their bodies as your own.

Independence is a lonely path however, as without the help of others there's only so many risks you can take without getting hurt. While you have no life left to fear death, there is only so much pain one can take before they will ultimately give up.

Will you choose independence, and be forever doomed to a fate of darkness or accept help, and exact your revenge on the king who so terribly wronged you all?

Target Platform

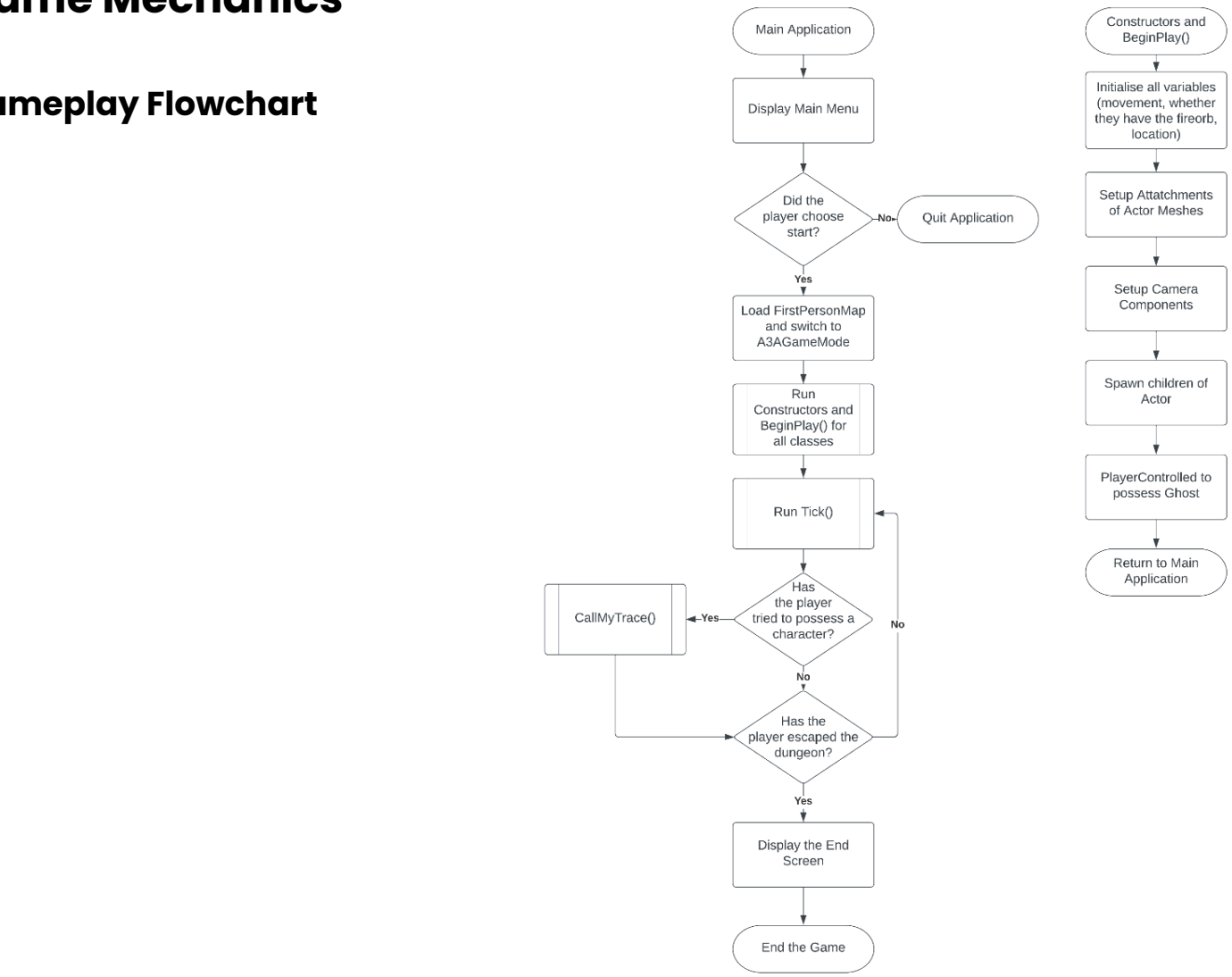
The prototype of Spirits of Mythos will be designed with a PC as the target platform. While this game can be played by anybody, it is aiming for an audience that more typically prefer more 'cozy' games that are story driven and don't feature heavy skill reliant action. Statistically, this genre is most played by people who identify as women or non-binary, and it's shown that these people most commonly game on either PC or a Nintendo switch over other consoles.
(https://newzoo.com/resources/blog/zooming-in-on-female-gamers-with-consumer-insights-data#:~:text=Women%20play%20on%20mobile%20at,and%2070%25%20for%20PC and
https://midiaresearch.com/blog/gaming-makes-headway-on-its-quest-to-gender-equal-distribution and

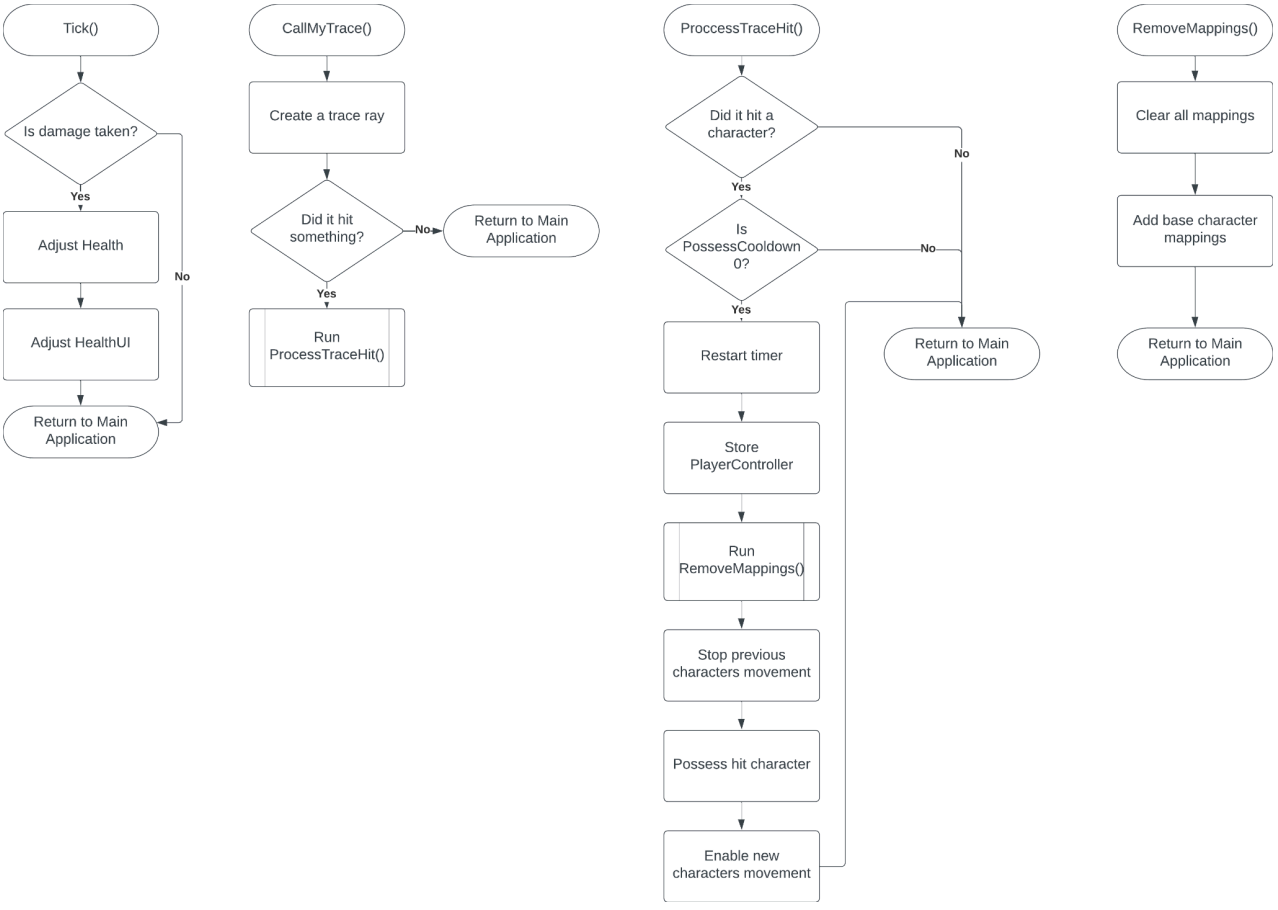
https://www.pcgamer.com/researchers-find-that-female-pc-gamers-outnumber-males/). With this knowledge of the audience, the target platform can be considered.

Spirits of Mythos relies heavily on a possess functionality, which most closely resembles a shooter in function. This is a much harder function to use on various other platforms and consoles and typically relies on aim assist features, which would take away from the user driven feel of this game; that each and every action is controlled and performed *because* of the user. Additionally, a large portion of the interaction a player has with the world is through aiming - looking, interacting, etc. With all of the above information acknowledged, it is plausible to select PC as the target platform to deliver the best experience to the targeted audience.

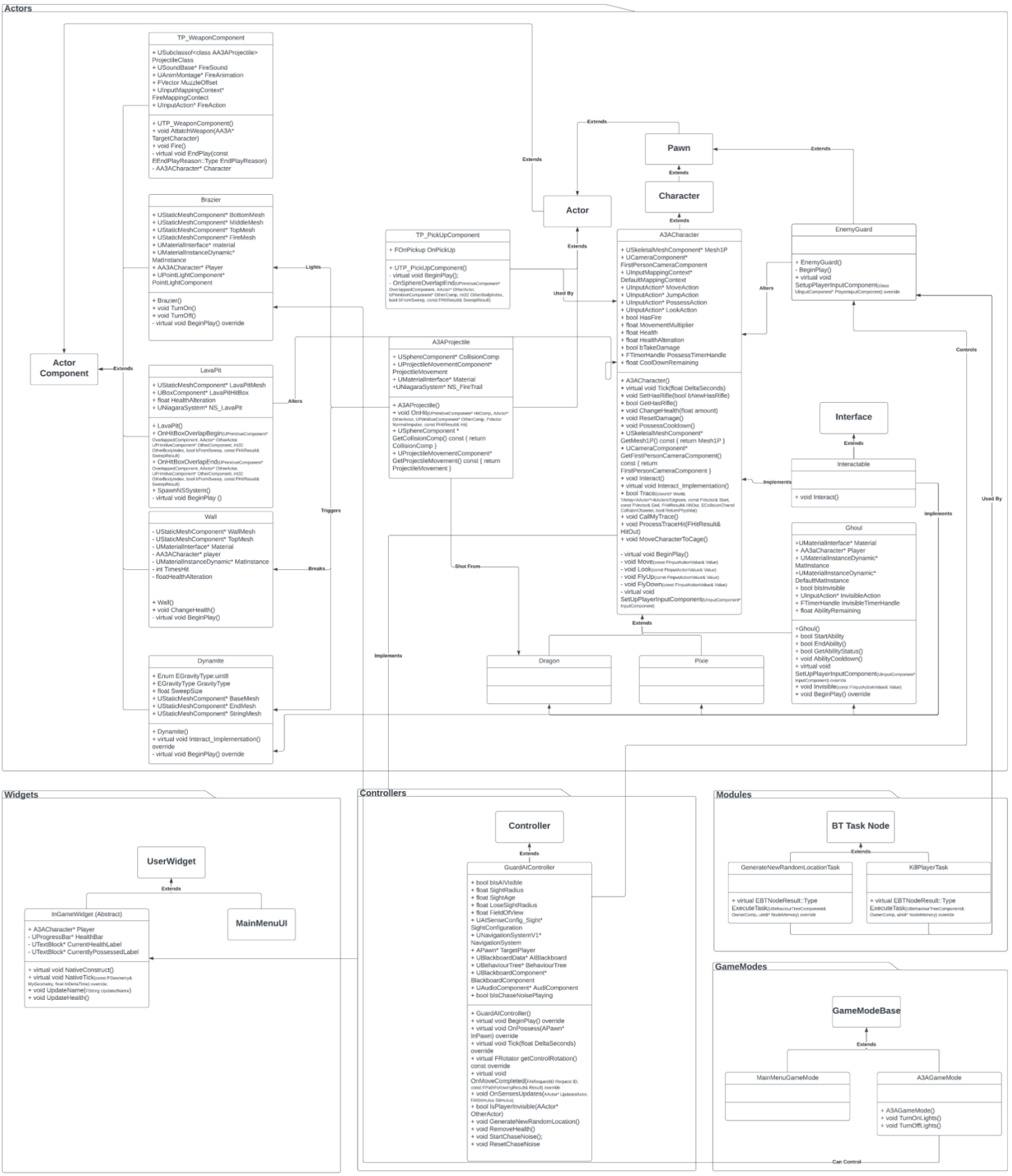
Game Mechanics

Gameplay Flowchart





UML Diagram


Actors
TP_WeaponComponent
+ USubclassof<class AA3AProjectile> ProjectileClass
+ USoundBase* FireSound
+ UAnimMontage* FireAnimation
+ FVector MuzzleOffset
+ UInputMappingContext* FireMappingContect
+ UInputAction* FireAction
+ UTP_WeaponComponent()
+ void AttatchWeapon(AA3A* TargetCharacter)
+ void Fire()
- virtual void EndPlay(const EEndPlayReason::Type EndPlayReason)
- AA3ACharacter* Character
Brazier
+ UStaticMeshComponent* BottomMesh
+ UStaticMeshComponent* MiddleMesh
+ UStaticMeshComponent* TopMesh
+ UStaticMeshComponent* FireMesh
+ UMaterialInterface* material
+ UMaterialInstanceDynamic* MatInstance
+ AA3ACharacter* Player
+ UPointLightComponent* PointLightComponent
+ Brazier()
+ void TurnOn()
+ void TurnOff()
- virtual void BeginPlay() override
Actor Component
LavaPit
+ UStaticMeshComponent* LavaPitMesh
+ UBoxComponent* LavaPitHitBox
+ float HealthAlteration
+ UNiagaraSystem* NS_LavaPit
+ LavaPit()
+ OnHitBoxOverlapBegin(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComponent, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
+ OnHitBoxOverlapEnd(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComponent, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
+ SpawnNSSystem()
- virtual void BeginPlay ()
Wall
- UStaticMeshComponent* WallMesh
- UStaticMeshComponent* TopMesh
- UMaterialInterface* Material
- AA3ACharacter* player
- UMaterialInstanceDynamic* MatInstance
- int TimesHit
- floatHealthAlteration
+ Wall()
+ void ChangeHealth()
- virtual void BeginPlay()
Dynamite
+ Enum EGravityType:uint8
+ EGravityType GravityType
+ float SweepSize
+ UStaticMeshComponent* BaseMesh
+ UStaticMeshComponent* EndMesh
+ UStaticMeshComponent* StringMesh
+ Dynamite()
+ virtual void Interact_Implementation() override
- virtual void BeginPlay() override
Actor
Pawn
Character
TP_PickUpComponent
+ FOnPickup OnPickUp
+ UTP_PickUpComponent()
- virtual void BeginPlay();
- OnSphereOverlapEnd(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult)
A3AProjectile
+ USphereComponent* CollisionComp
+ UProjectileMovementComponent* ProjectileMovement
+ UMaterialInterface* Material
+UNiagaraSystem* NS_FireTrail
+ A3AProjectile()
+ void OnHit(UPrimitiveComponent* HitComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, FVector NormalImpulse, const FHitResult& Hit)
+ USphereComponent * GetCollisionComp() const { return CollisionComp }
+ UProjectileMovementComponent* GetProjectileMovement() const { return ProjectileMovement }
A3ACharacter
+ USkeletalMeshComponent* Mesh1P
+ UCameraComponent* FirstPersonCameraComponent
+ UInputMappingContext* DefaultMappingContext
+ UInputAction* MoveAction
+ UInputAction* JumpAction
+ UInputAction* PossessAction
+ UInputAction* LookAction
+ bool HasFire
+ float MovementMultiplier
+ float Health
+ float HealthAlteration
+ bool bTakeDamage
+ FTimerHandle PossessTimerHandle
+ float CoolDownRemaining
+ A3ACharacter()
+ virtual void Tick(float DeltaSeconds)
+ void SetHasRifle(bool bNewHasRifle)
+ bool GetHasRifle()
+ void ChangeHealth(float amount)
+ void ResetDamage()
+ void PossessCooldown()
+ USkeletalMeshComponent* GetMesh1P() const { return Mesh1P }
+ UCameraComponent* GetFirstPersonCameraComponent() const { return FirstPersonCameraComponent }
+ void Interact()
+ virtual void Interact_Implementation()
+ bool Trace(UWorld* World, TArray<AActor*>*ActorsToIgnore, const FVector& Start, const FVector& End, FHitResult& HitOut, ECollisionChannel CollisionChannel, bool ReturnPhysMat)
+ void CallMyTrace()
+ void ProcessTraceHit(FHitResult& HitOut)
+ void MoveCharacterToCage()
- virtual void BeginPlay()
- void Move(const FInputActionValue& Value)
- void Look(const FInputActionValue& Value)
- void FlyUp(const FInputActionValue& Value)
- void FlyDown(const FInputActionValue& Value)
- virtual void SetUpPlayerInputComponent(UInputComponent* InputComponent)
EnemyGuard
+ EnemyGuard()
- BeginPlay()
+ virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override
Interface
Interactable
+ void Interact()
Ghoul
+UMaterialInterface* Material
+ AA3aCharacter* Player
+ UMaterialInstanceDynamic* MatInstance
+UMaterialInstanceDynamic* DefaultMatInstance
+ bool bIsInvisible
+ UInputAction* InvisibleAction
+ FTimerHandle InvisibleTimerHandle
+ float AbilityRemaining
+Ghoul()
+ bool StartAbility
+ bool EndAbility()
+ bool GetAbilityStatus()
+ void AbilityCooldown()
+ virtual void SetUpPlayerInputComponent(UInputComponent* InputComponent) override
+ void Invisible(const FInputActionValue& Value)
+ void BeginPlay() override
Dragon
Pixie
Extends
Implements
Used By
Controls
Alters
Lights
Triggers
Breaks
Shot From
Widgets
UserWidget
InGameWidget (Abstract)
+ A3ACharacter* Player
- UProgressBar* HealthBar
- UTextBlock* CurrentHealthLabel
- UTextBlock* CurrentlyPossessedLabel
+ virtual void NativeConstruct()
+ virtual void NativeTick(const FGeometry& MyGeometry, float InDeltaTime) override;
+ void UpdateName(FString UpdatedName)
+ void UpdateHealth()
MainMenuUI
Controllers
Controller
GuardAIController
+ bool bIsAiVisible
+ float SightRadius
+ float SightAge
+ float LoseSightRadius
+ float FieldOfView
+ UAISenseConfig_Sight* SightConfiguration
+ UNavigationSystemV1* NavigationSystem
+ APawn* TargetPlayer
+ UBlackboardData* AIBlackboard
+ UBehaviourTree* BehaviourTree
+ UBlackboardComponent* BlackboardComponent
+ UAudioComponent* AudiComponent
+ bool bIsChaseNoisePlaying
+ GuardAIController()
+ virtual void BeginPlay() override
+ virtual void OnPossess(APawn* InPawn) override
+ virtual void Tick(float DeltaSeconds) override
+ virtual FRotator getControlRotation() const override
+ virtual void OnMoveCompleted(FAIRequestID Request ID, const FPathFollowingResult& Result) override
+ void OnSensesUpdates(AActor* UpdatedActor, FAIStimulus Stimulus)
+ bool IsPlayerInvisible(AActor* OtherActor)
+ void GenerateNewRandomLocation()
+ void RemoveHealth()
+ void StartChaseNoise();
+ void ResetChaseNoise
Modules
BT Task Node
GenerateNewRandomLocationTask
+ virtual EBTNodeResult::Type ExecuteTask(UBehaviourTreeComponent& OwnerComp, uint8* NodeMemory) override
KillPlayerTask
+ virtual EBTNodeResult::Type ExecuteTask(UBehaviourTreeComponent& OwnerComp, uint8* NodeMemory) override
GameModes
GameModeBase
MainMenuGameMode
A3AGameMode
+ A3AGameMode()
+ void TurnOnLights()
+ void TurnOffLights()
Can Control

The first person template in unreal was used as the starting place for all of my code. It's noted that some of the class names and function/variable names have not been adjusted to match the new functions and these need to be updated accordingly eg. projectile-> fire, rifle -> fireorb, bool HasRifle -> bHasFire. It's also been noted some bools have not been named correctly with b before the variable name. This will be adjusted in future work on the game. There are also some remnants of a previous implementation attempt at adding some functions such as flying that need to be cleaned up.

Movement Mechanics

While there are currently only three additional types of characters beside the main character, it is planned that there will be more added with future development of the game. These will have some additional movement mechanics which have not yet been created, such as swimming, etc.

All characters in the game can move using the default mapping input, with the movement vector multiplied by a variable called MovementMultiplier, set to 5.0f.

In the final version of the game, the speeds of each of the characters will be adjusted. The dragon will be slowed down to represent the size of the creature and its weight, and the pixie will be made faster than the other classes as it is small and can zip around. These will be done by creating a movement multiplier in each of the child classes that overrides the main movement speed multiplier. The friction for the pixie will also be reduced, so it regains this feeling of flying and the flowy-ness that comes with it. Additionally, a small amount of gravity may be added to the pixie so that it 'sinks' without user input and needs the player to continuously press the 'up' button, to better simulate the flapping of wings.

While all of the characters use the same default mapping, only the pixie is currently able to use the Fly Up and Fly Down actions. This is achieved by turning gravity off for only the pixie class.

Likewise, only the Ghost character can jump, as it is the only character that has had the jump action assigned in the blueprint.

All players have collision preset type custom to allow trace rays to hit them, which is how the possess function has been implemented. The characters can collide with each other intentionally as they are spirits, and they all collide with the world around them, which means the floor, the walls and the cubes located around the level at the current moment. They cannot push or move the cubes, as they are what make up the 'cages' that they are kept within, so they cannot just push their way out of these instead of doing the correct action.

Controls

Below, the controls have been explained within a table. This includes all information about the control such as the name of the mapping, the action itself, the description of what the action does, the keybind and the modifiers applied to it. This table has also been separated into which specific characters the controls are for. All characters are able to use the basic Move, Look and Possess functions. This allows them to move around the map with a keyboard, look around with a mouse and possess characters that are in their crosshair/aim on right click (when within distance).

The ghost has a jumping ability, which means that they can leave the ground temporarily before falling back down. The pixie specifically has a flying ability, which means that it can move up and down on the Y-axis using the keyboard. The dragon has a fire ability, which means that after picking up the fire orb it is able to fire projectiles with the mouse. The ghoul has the ability to go invisible, which means that they can change between invisible to visible with the mouse.

The possession ability has been set to triggered instead of started so that the trace line will only call one, instead of immediately as a new character is possessed.

Mapping	Action	Description	Keybind	Modifiers
All Characters				
MoveMapping	Move Forward	Used to move all characters forward	W	N/A
	Move Backwards	Used to move all characters backwards	S	Negate
	Move Left	Used to move all characters left	A	SwizzleInputAxisValues (YXZ), Negate
	Move Right	Used to move all characters right	D	SwizzleInputAxisValues (YXZ)
LookMapping	Look Around	Used to look around the screen	Mouse XY 2D-Axis	Negate
PossessMapping	Possess Character	Used to possess another character on aim	Left Mouse Button	N/A
Ghost				

JumpMapping	Jump	Used to make the character jump	Spacebar	N/A
Pixie				
MoveMapping	Fly up	Used to fly the pixie up	E	SwizzleInputAxisValues (ZYX)
	Fly Down	Used to fly the pixie down	Q	SwizzleInputAxisValues (ZYX), Negate
Dragon				
FireMapping	Shoot Fire	Used to shoot fire after fireorb has been picked up	Right Mouse Button	N/A
Ghoul				
GhoulMapping	Turn Invisible	Used to switch between invisible and visible	Right Mouse Button	N/A

Additional mappings have been removed from the code, as well as the remnants of the old fly mapping.

Core Gameplay Mechanic

Mechanic Overview

The core gameplay mechanic for Spirits of Mythos is the idea of possessing another character that has a unique ability, of which each of these abilities needs to be used in order to escape the dungeon. In the final game there will be a number of different characters and abilities, but for this prototype just the ghost, ghoul, dragon and the pixie have been created.

The player will need to approach their new character to possess which in future development will be placed within a cage that the player has to free them from. Currently, the player must blow up a stick of dynamite placed in front of the cage door with the dragon's flame. Once the cage is opened, the user is then able to use the creature's abilities to progress them through the next stage of the puzzle. The player will need to work out which creatures they need when, and how to get to them to free them.

Mockups/wireframes have been created to demonstrate what this would look like in the game.

Third person view of possess function and line trace.

First person view of possess function and line trace.

Mechanic Description / Functionality

The creatures are all child classes of A3ACharacter so that they all have the same core functionality and possessing ability, so that the player can go back to the ghost or to another character. The possessing ability heavily relies on the LineTrace functionality in Unreal.

To possess another character, the player clicks the left button mouse. A line trace is fired from the character, and will travel in the direction that they are looking at the time. This will travel from the camera component of the character plus their forward vector multiplied by 1256. This value was selected at the time due to the ease of testing and being able to reach a character without much movement. Additionally, this is used so that the player won't get frustrated with walking backwards and forwards too much to swap between characters. Because it is planned to add cages for each of the creatures in the game, this will stop the player being able to swap into characters too early instead of completing the problems and skipping sections.

Once the trace is created, it outputs the name of the actor it hits and then processes the data. If the trace hits nothing, it exits the function. On hitting an actor, it then enters the ProcessTraceHit function, which will check the class of the actor hit and ensure that it is an interactable character. If not, it will output that the actor is not interactable and exit the function. On hitting an interactable character, it will check if the possess cool down is = 0. If not it will also exit and display the time left in the cooldown, but if it is it will then reset the time to the possess cool down limit. In this instance it has been set to 30, as it doesn't make the player wait too long to swap back but also encourages precision and thinking rather than meaningless switching. Once this has been done, the player controller is stored and the current controller is unpossessed. Then, the current character's movement is set to none. The remove mappings function is called, which removes all 'specialty' mappings from each of the characters. After this, the actor that has been hit is stored and then possessed, enabling movement and adding any additional mappings that are applicable to that character. It's proposed that in the final version of the game, the cool down will be turned into a meter visible on the in-game UI, like the health bar.

Now that the new character has been possessed, they are free to move as previously and interact with the map in ways that depend on their unique ability.

Specific details including variables and mappings for the Pixie and Ghost have been discussed under the movement mechanics section of the document, as their abilities are movement based.

The dragon is the only character that can interact with the 'Fire Orb', which they need to pick up in order to shoot fire projectiles. This is done by adding a line of code in the TP_PickUpComponent.Cpp file that uses the OnSphereOverlapBegin function that checks if the character is of class Dragon. If they are, the fire orb can be picked up and then the bool HasFire is then set to true so it can be used by the character to fire projectiles (fireballs). If the character is not of class dragon, then the function is exited and the weapon cannot be picked up.

The ghoul has the special ability of invisibility, which means that the AI Guard will not chase them when they're invisible. This is done by creating a bool variable called bIsInvisible, with the use of functions of return type EndAbility() and StartAbility() to edit this. In the constructor, bIsInvisible is set to false. This means that when the character is spawned, their ability is not currently active.

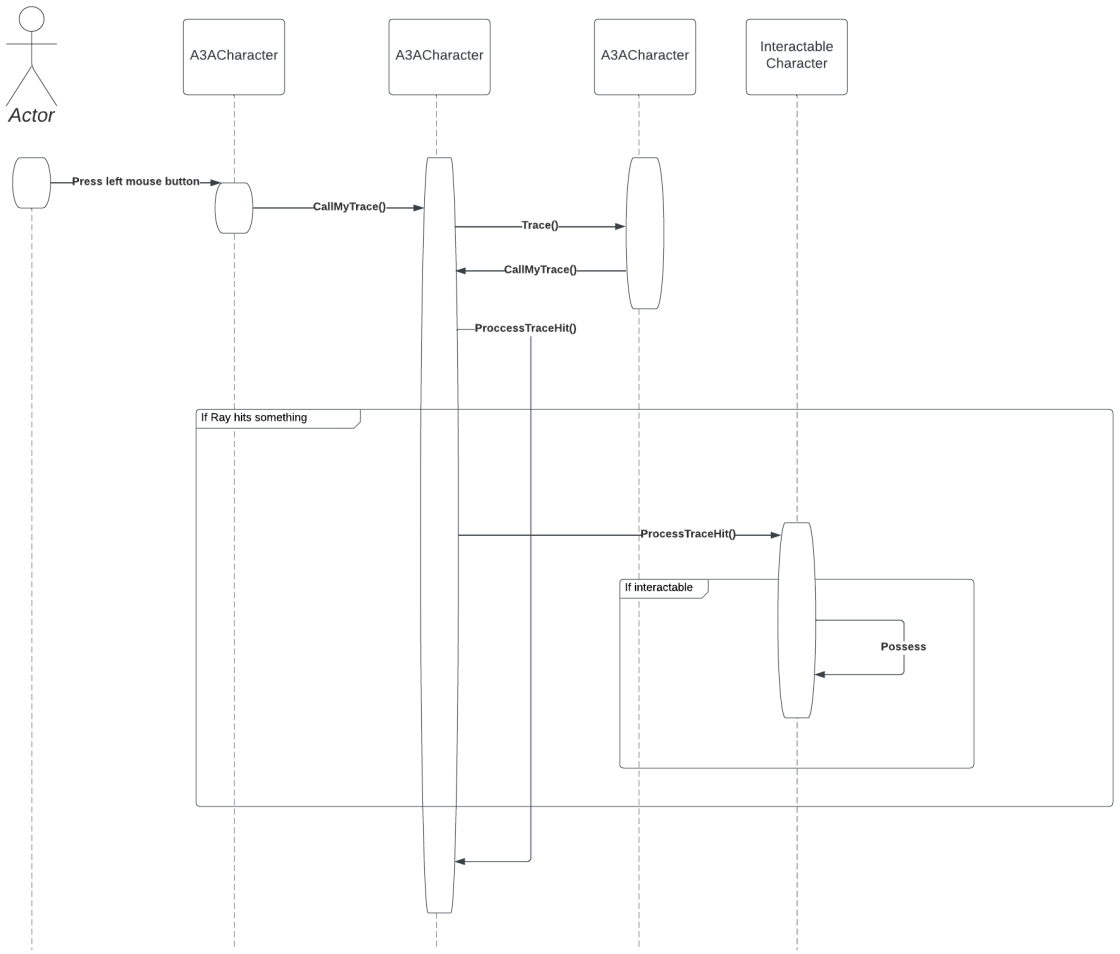
Additionally, there is a function of return type void called Invisible(). This is called when the user presses the keybind, in this case the right mouse button. It checks to see if the character is currently invisible and if it is, calls the EndAbility() function to set bIsInvisible to false and clears the invisibility timer. If the character is not currently invisible, it calls the StartAbility() function. This function resets the cooldown for 15 seconds, which at the end of the player will turn visible again.

These start and end ability functions also control the material of the character, which is explained in further detail in the dynamic material section.

When an AI Guards senses are updated, they check if the character is a ghoul and if so, if they're currently invisible. This is done by calling the GetAbilityStatus function of type bool from the Ghoul class, which will return the current value of bIsInvisible. If this is true, the AI Guard will not chase them and if it's false, it will chase.

Sequence Diagram

A sequence diagram has been created for the possess functionality for easier understanding.



UI Design

Main Menu UI

UI Overview

The main menu UI is the first thing the player sees when they open the game. It introduces the mood of the game and is also a standard opening point for any game. The main menu for Spirits of Mythos currently features ‘Play’ and ‘Quit’ buttons, which the player can then choose to interact with to either start the game, or quit the game.

It has been decided that the game would benefit from having additional options added to this screen in the future, such as an options button.

UI Description / Functionality

The menu shows a static image, with animated rotating ‘ spirit orbs’ between the main menu text and the buttons. Currently, the only interactable elements are the play and the quit options. Clicking play will begin the game, while clicking quit will exit the game.

The MainMenuUI was created by creating an Advanced Asset of type Widget Blueprint, which contained a number of different UMG Widgets. The hierarchy of these has been shown below.

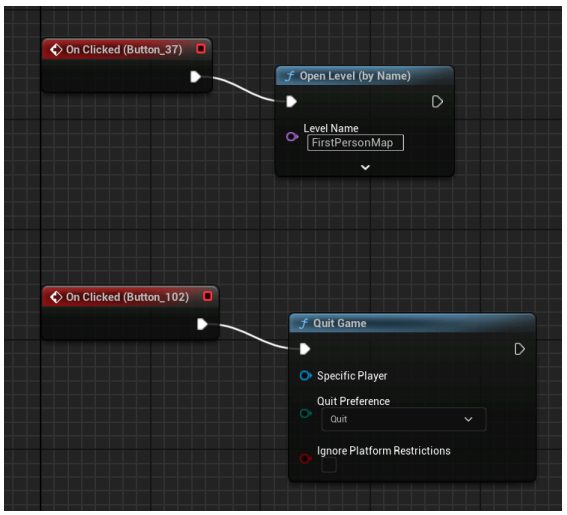
[Main Menu UI]
- [Canvas Panel]
  - Image
  - [Horizontal Box]
    - [Vertical Box]
      - Image
      - Text
      - Image
      - Text
  - [Horizontal Box]
    - [Vertical Box]
      - Text
      - Text
      - Circular Throbber
      - [Vertical Box]
        - Button
          - Text
        - Button
          - Text

Health UI			
Name	**Type**	**Purpose**	**Customised Properties**
Canvas Panel	Canvas Panel	Used to contain all elements on the screen	Screen size - > 21.5-24" Monitor
Image_38	Image	Main Menu Image Background	Anchors - 0,0,1,1 Image - MainMenuImage Draw As - Image
Button Images	Horizontal Box	Used to contain all the elements of the buttons (visual)	Anchors - 0.5,0,0.5,1 Position X - -360 Size X - 732.7
Buttons_1	Vertical Box	Used to contain all the 'Backgrounds' of the buttons	Size - Fill Padding - 40,320,40,40
Image_126	Image	Background for the play button	Size - Fill Padding - 0,0,0,40 Image - mainmenubutton Image Size - 544,170
Play_1	Text	Used to keep the same spacing as the other Button Box	Padding - 0,0,0,40 Text - ' '
Image	Image	Background for the quit button	Size - Fill Padding - 0,0,0,40 Image - mainmenubutton Image Size - 544,170
Quit_1	Text	Used to keep the same spacing as the other Button Box	Padding - 0,0,0,40 Text - ' '
Horizontal Box	Horizontal Box	Contains all the menu text, animated orbs and buttons	Anchors - 0.5,0,0.5,1 Position - -360.9 Size X - 736.6
Titles	Vertical Box	Contains all the menu text,	Size - Fill

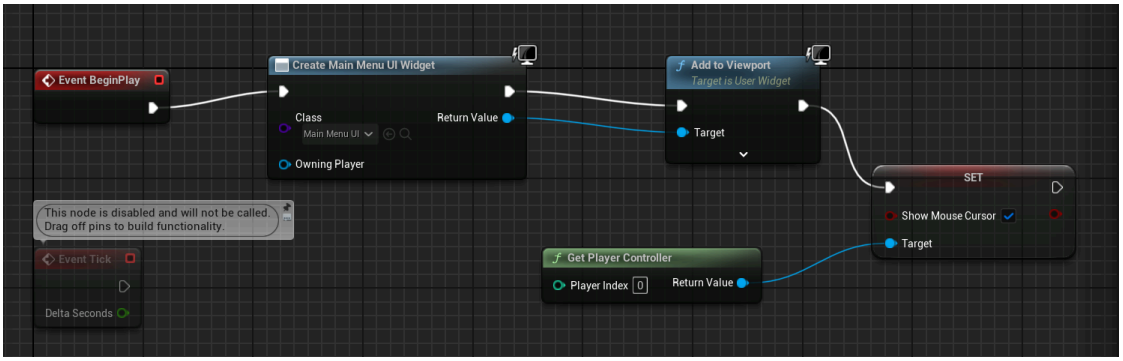
		animated orbs and buttons	
Game Title	Text	Shows the name of the game	Padding - 0,0,0,40 Text - 'Spirits of Mythos' Colour and Opacity - FFFFFFFF Font Family - Roboto Typeface - Italic Size - 71
Main Menu	Text	Tells what menu is currently showing	Padding - 0,0,0,40 Text - 'Main Menu' Colour and Opacity - 0A0C0EFF Font Family - Roboto Typeface - Regular Size - 36
CircularThrobber_0	Circular Throbber	Decorative element between text and buttons	Number of Pieces - 6 Period - 10 Radius - 16 Image Size - 20,20 Tint - 0A0C0EFF Draw as - Image
Buttons	Vertical Box	Contains all of the buttons (functional)	Padding - 40 Size - Fill
Button_37	Button	Interactable Play Button	Padding - 0,0,0,40 Size - Fill Background Colour - FFFFFF00, Alpha 0
Play	Text	Text for the play button	Padding - 0,60,0,40 Horizontal Alignment - Center Align Text - 'Play' Colour - 000000FF Font Family - Roboto Typeface - Bold Italic Size - 41
Button_102	Button	Interactable Quit Button	Padding - 0,0,0,40 Size - Fill Background Colour - FFFFFF00, Alpha 0
Quit	Text	Text for the quit button	Padding - 0,60,0,40 Horizontal Alignment - Center Align Text - 'Play'

			Colour - 000000FF Font Family - Roboto Typeface - Bold Italic Size - 41

For adding functionality to the MainMenu, it was decided that because it was just the HUD and performance was not a concern, it could be created with blueprints, as taught within the FIT2096 Unit. The level blueprint was constructed as shown below. This means that when the top button is clicked (Play) it will open the level of the game. The second button (Quit), when clicked will exit the entire game.



A new level was created which was called MainMenu, as it would be used to contain all Main Menu UI elements. This means that it would be less resource intensive, as it is not also loading in an entire level behind the interface that we can see. The following blueprint was constructed to add the widget to the HUD. This means that upon clicking play initially (starting the game), the main menu UI Widget is created and added to the players viewport. Then, the player's input (Mouse) is collected and shown on top of the UI widget so that they can see their mouse to click buttons and interact with the menu. This is shown below.



With all of this finished, a new GameMode for MainMenu could be created. This was created as a public C++ class, with a parent class of GameModeBase. No alterations were made to these C++ files in rider. In Project Settings, MainMenu was selected to be the default starting map for both the Startup Map and the Editor Map. To allow the player to be able to appear and be controlled, an

element was added to the array called GameModeMapPrefixes. The name was changed to be FirstPersonMap (the name of the level) and the GameMode was set to A3AGameMode (the default gamemode).

UI Wireframe

I knew that I wanted a very specific style of buttons for my game, which couldn't be achieved with what existed in unreal. I also knew that AI would not be able to create the exact thing that I wanted.I wanted to use buttons and health meters that were contained in a frame reminiscent of fantasy game sword hilts, such as skyrim and the witcher. So, I took to Adobe Illustrator.

I began with creating a random, almost fluid gradient with the main blue-grey colours of my generated background. Once I was happy with this base, I began to make the frames. While I didn't look at any particular reference images for this, I drew on my many years of playing Skyrim and the witcher and other dark fantasy esc cames to create tendril-like black lines that extended past the ends. I trialled a few different designs before I finally settled on one that I was happy with, which I've included below.

With this completed, I could then create my UI Wireframe, which has been provided and annotated below. For future iterations of the game, it would be nice to design an image or logo that could replace the game title text, and to come up with a tag light that might replace the text that currently says main menu. It would be wise in the future to also add an options button to the main menu. The main menu image was generated using AI (NightCafe, 2023). The measurement values differ slightly as they were adjusted once they were implemented.

In-Game UI

UI Overview

The in-game UI is seen once the player selects ‘Play’ from the home menu and loads into the game. It features a health bar, which shows the players current and max health. The current health changes as the player takes damage, and the health bar changes dynamically to reflect this. A possession tag has been implemented, which shows the ability of the character that the player is currently possessing and updates as they swap players. This means that they are more easily able to understand what they are currently able to do and once the level has been built up more in the final game, it would assist in knowing what they should do next. This is so the player is more easily able to understand their current player, and what they may be able to do. This UI also features a circular cross hair to make it easier to target and hit a creature with the possessing trace ray.

For the final game it is proposed that a possession cooldown bar also be added, which would appear in a similar way to the health bar. This is so that the user can more clearly see their progress on this cooldown and don’t get frustrated when they can’t understand why it doesn’t work right away.

UI Description / Functionality

The in-game UI shows a dynamic health bar, which empties depending on the player's health. It also features a text label beside it which shows the numeric value of the characters health. It also has a possession tag, which shows the ability of the character that the player is currently possessing. There is also a circular cross hair in the centre of the players screen, which is where the trace ray fires from.

The components of the In-Game UI have been renamed from HealthBarWidget and HealthBar UI to be InGameUI and InGameWidget to be better self documenting and reflect the code that they contain.

The InGameWidget C++ class was created with a Parent class of UserWidget, then made into an Abstract class using the Abstract keyword. Then, three UProperties were added to the class and bound using the BindWidget command so that the health bar could be dynamic. These were the UProgressBar, UTextBlock and another UTextBlock - Representing the health bar and the current health label, as well as the ability of the character the player is currently possessing. In the NativeConstruct function, a reference to the player was created so it could be called in the constructor.

In the tick function, code was added to handle the editing of the UI variables. The variables specifically relevant to the HealthBar were float Health, float HealthAlteration, bool bTakeDamage,

and then two functions of type void called ChangeHealth and ResetDamage, all located in the Character Class. For the currently possessed tag, there was a function of type void called UpdateName(), and then there was a FString variable called Type located in the character class, which was updated from the Possess functionality.

The ChangeHealth function worked by changing the bool value of bTakeDamage to true, and setting theHealthAlteration to equal amount, which is defined in the class inflicting the damage/healing. The ResetHealth function worked in a very similar way, changing bTakeDamage to false and changing health alteration to equal 0.0f, so back to not changing. Code was then added to the tick function of the character to check if they were currently taking damage and to to ensure and constraint their current health so that it couldn't go above or below the provided maximum or minimum - if their health hit 0, they would not continue to take damage and if their health was 100, they could not gain any more health. It's proposed that in the final version of the game, if the health were to hit 0 then the game would end and they would be forced to restart - potentially from a save point.

The changing possession tag happened when the possess functionality was called. After possessing the new character but before exiting the possess functionality, it would check what class the possessed character was. Depending on which one it was, the Ability variable would be set accordingly and then the widget function UpdateName() would be called with Ability as the parameter, which would then update what was displayed on screen.

The only two classes that can actively edit the Characters Health are LavaPit and the Guard AI. The LavaPit class was created of type actor. It was assigned a mesh, as well as a hit box so that the player would be able to interact with it. These were created using UProperties with the EditAnywhere tag. A HealthAlteration variable was also created in this way, so that multiple different lava pits could exist in the level causing varying amounts of damage if wanted.

Two functions of type void were created, called OnHitboxOverlapBegin and OnHitBoxOverlapEnd, to manage the alteration of health of the player on overlap. When the player overlaps with the hitbox, the OnHitBoxOverlaBegin function is called, which calls the ChangeHealth function from the Character class and alters the players health accordingly. Once the player is no longer overlapping the hitbox, it calls the function OnHitBoxOverlapEnd, which calls the ResetDamage function from the character class and stops the player from taking any more damage.

As for the AI, how it changes the players health is described in detail in the AI section of this document. A brief overview of this however is that on the AI's senses updating, it determines if the player is meant to be visible and if so, calls the ChangeHealth() function from the character class.

The InGameUI was created by creating an Advanced Asset with the newly created HealthBarWidget as the parent class, which contained a number of different UMG Widgets. The hierarchy of these has been shown below.

[InGameUI]
[Canvas Panel]
[Horizontal Box]
[Size Box]
Image
Text - CurrentHealthLabel
Text - /
Text - MaxHealthLabel
[Size Box]
Text -CurrentlyPossessedLabel
[Size Box]
Progress Bar -HealthBar
[Horizontal Box]
[Size Box]
Image

Health UI			
Name	**Type**	**Purpose**	**Customised Properties**
Canvas Panel	Canvas Panel	Used to contain all elements on the screen	Screen size - > 21.5-24" Monitor
Horizontal Box	Horizontal Box	Used to contain all of the UI health Widgets	Anchors - 0,0,1,1 Offset Bottom - 981 Size X - 700 Size Y - 100
Size Box	Size Box	Used to contain 'Background' Health bar image and text elements	Size - Auto
Image_81	Image	Health Bar 'Background'	Padding - 20,20,0,0 Brush - mainmenubutton Image Size - 544, 170 Draw as - Image
Current Health Label	Text	Shows players current health	Size - Auto Padding - 20,20,0,0
Text	Text	Shows " / " (Out of)	Size - Auto Padding - 0,20,0,0
MaxHealthLabel	Text	Shows what	Size - Auto

		players health is out of/Max health	Padding - 0,20,0,0
Size Box	Size Box	Used for spacing the UI along the top	Size - Fill
Currently Possessed Label	Text	Used to show the character the player is currently possessing	Size - Auto Padding - 0,20,20,0
Size Box	Size Box	Used to contain the interactive Health bar	Size - 560, 100
HealthBar	Progress Bar	Shows a visual representation of players health	BACKGROUND IMAGE Padding - 110,20,75,0 Background Image - None Tint - FFFF00FF, Alpha 0 Bar Fill Type - Mask FILL IMAGE Tint - FF0000FF Fill colour and opacity - FF0000FF Draw as - Image
Horizontal Box	Horizontal Box	Used to contain crosshair	Anchors - 0,0,1,1
Size Box	Size Box	Used to size crosshair	Size - Fill Padding - 0,0,0,0
Image_75	Image	Shows where the players cursor is	Padding - 0,0,0,0

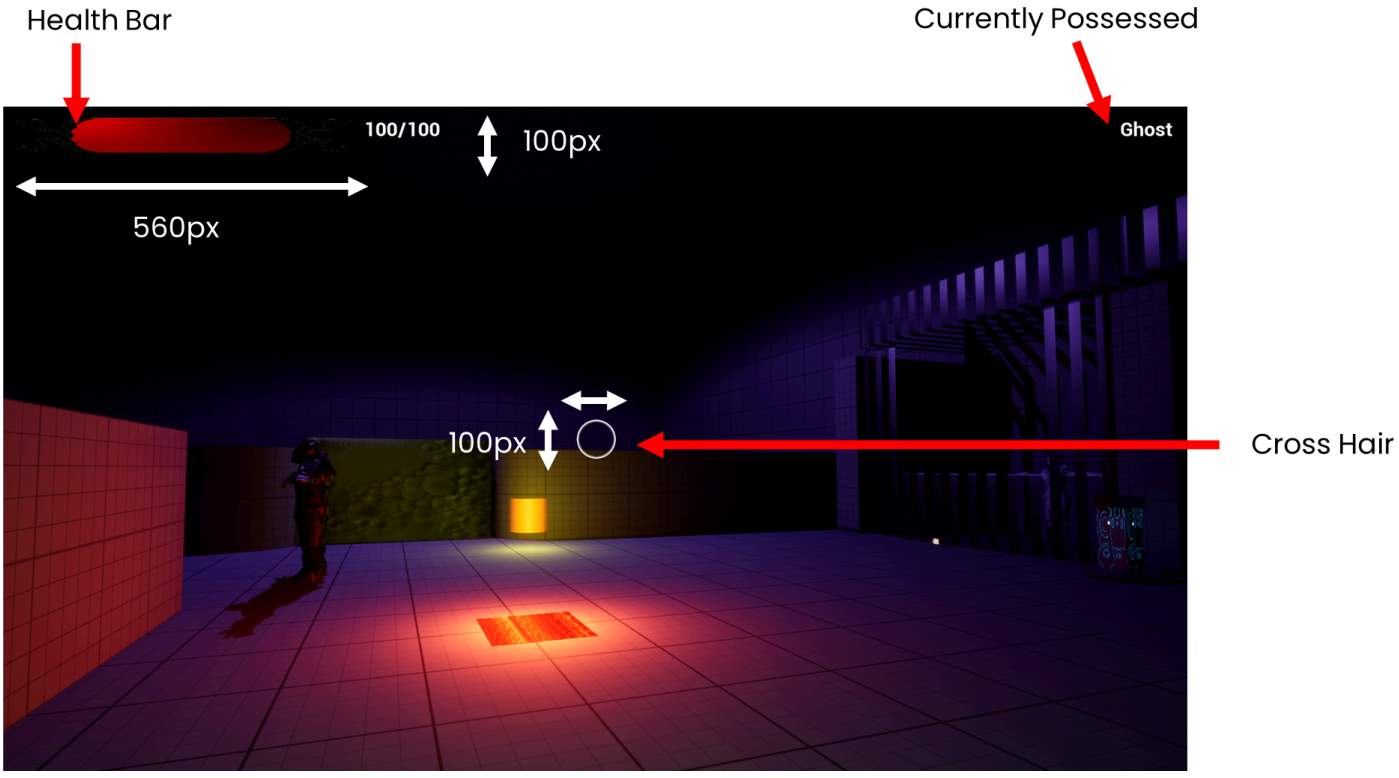
When creating the in-game UI, I knew I wanted to reuse my buttons from the home screen. However, using the inbuilt fill image tool in unreal meant that the health bar would fill from the edges of the black. While this looked ok, It didn't give an accurate representation of the players health. To fix this, I went back into Illustrator and created a cut out of my original button base, removing the gradient background where the black overlapped this. This left me with the following image.

While this meant that only the bar section of the button would fill, The image size options and the padding of the health bar were still tied to the base, which meant that this section covered up the whole health bar. To fix this and achieve the effect I wanted, I placed the original button as an image behind the 'healthbar', which I turned into the above image. I then set the sizing and padding so that the health bar only overlapped the grey part of the image underneath, and set its 'empty' colour to be transparent and fill colour to be this new image with a red tint applied. That way, when the player took damage, the healthbar was entirely contained within the black borders and more accurately represented the player's health. It also meant that the gradient effect of the original background applied to health fill, which helped with the consistency.

Once the widget had been created and the code had been implemented, it was added to the viewport in the same way that the MainMenuUI was added using the FirstPersonMap level blueprint.

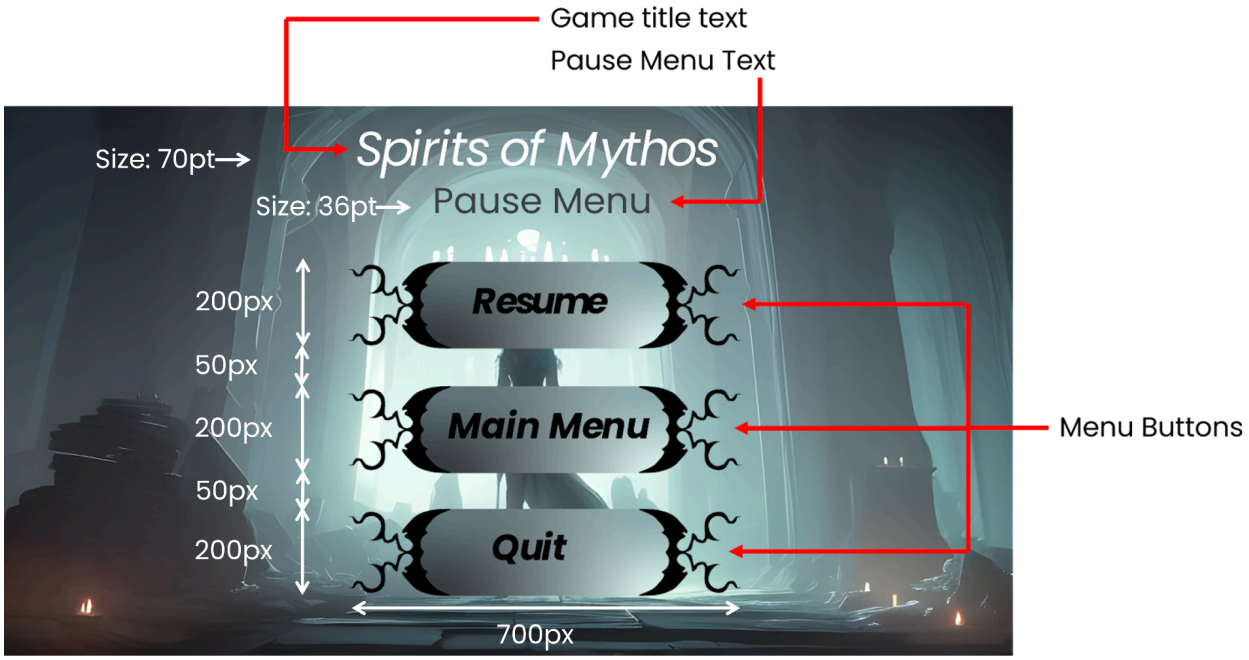
UI Wireframe

Below, a lo-fi wireframe for the in-game UI at its current stage has been completed. The measurement values differ slightly as they were adjusted once they were implemented.

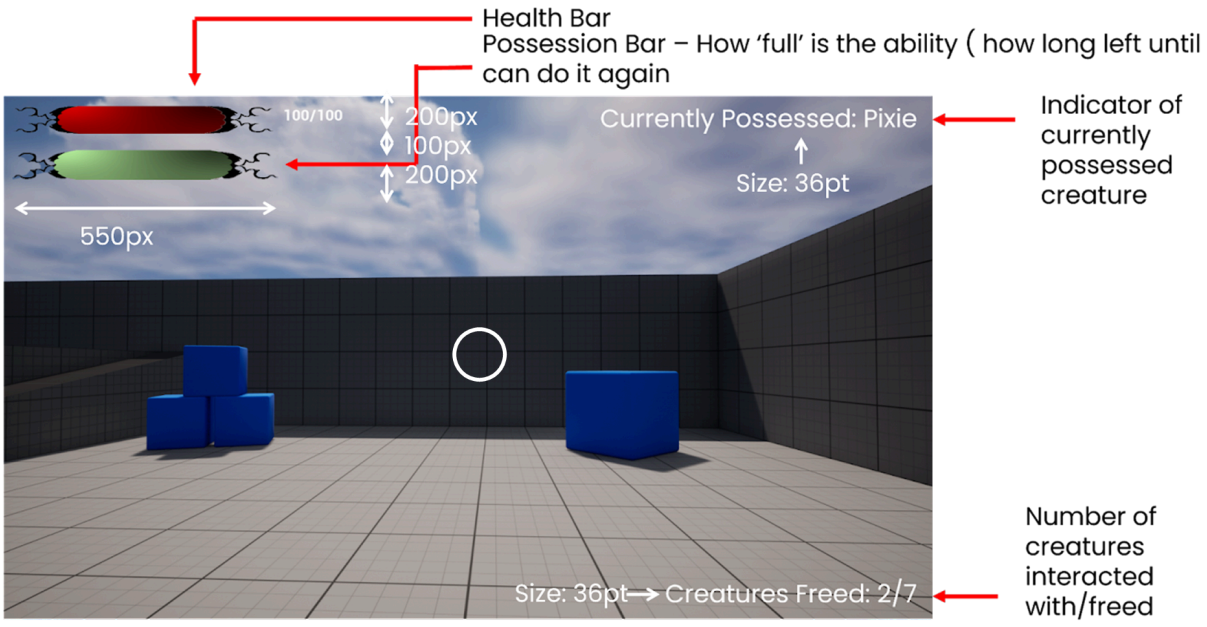


Overall, I feel that my in-game UI was lacking and I have come up with a number of plans for the final game that would improve this. The first is a pause menu, that the player would be able to

access at any time and would contain a few buttons, which have been shown in the below wireframe. The pause menu image was generated using AI (NightCafe, 2023).



I have also created a new Lo-fi wireframe with proposed additions and suggestions for the in-game UI. These involve a possession meter, which shows how long you have left to wait until you could possess another character as well as a creatures freed count, which would indicate the number of creatures interacted with/freed.



It's proposed that in further development this ability tag be replaced by icons that represent each of the unique abilities in a style that fits the rest of the UI to make the UI more visually exciting.

Dynamic Materials

Dynamic Material 1 - Invisibility (Invisible_Excited)

Overview of Effect

The invisibility effect is important, as it updates the player as to the current status of their ability. There is no supporting text or noise effect that alerts them that the ability is currently active, so having the material be dynamic and changed depending on the condition of the ability was a good way of providing this information.

Not only this, but upon entering the game, the player is unaware of any of the creature's abilities. While the player is currently informed that they are a ghoul once they possess the ghoul, they don't know what ability it has. Upon testing different keybinds, they will find the ability activated. However, if it were the same material the entire time, they would not be made aware that they had triggered the ability as it has no visible impact and they may not be near the AI Guard when trialling it.

As the dynamic material has been created so that it looks similar to the invisibility effect seen in other games, the player will recognise this and understand that the ghouls ability is invisibility. Additionally, as there are currently no indicators as to what each of the characters ability is, the player seeing this recognisable 'invisible' effect will alert them to what they can do.

Effect Description

A character being able to go invisible in games is not an uncommon feature. Amongst nearly all games that allow this effect, there seems to be a universal look for these invisible characters. Typically, it includes the character being somewhat transparents with some sort of shimmering/moving effect, oftentimes with at least a slight blue tint.

Because of this, I knew I wanted my invisibility effect to remain consistent with these so that it would be more recognizable for players what is happening when the effect activates (going invisible) as at the moment there is no supporting text telling the player what the ghouls ability is. By following these universal conventions set by other games, the player is provided necessary information about the ability.

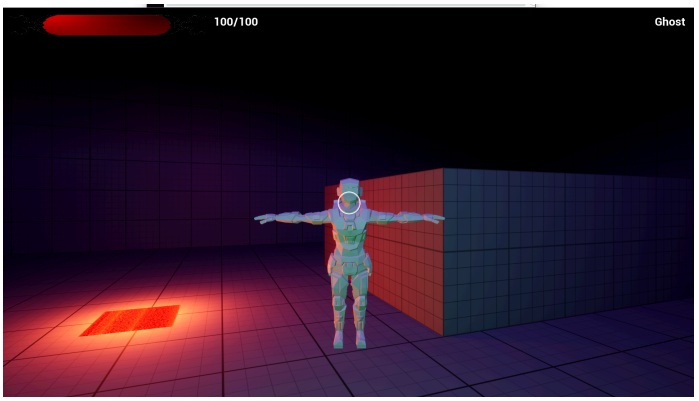
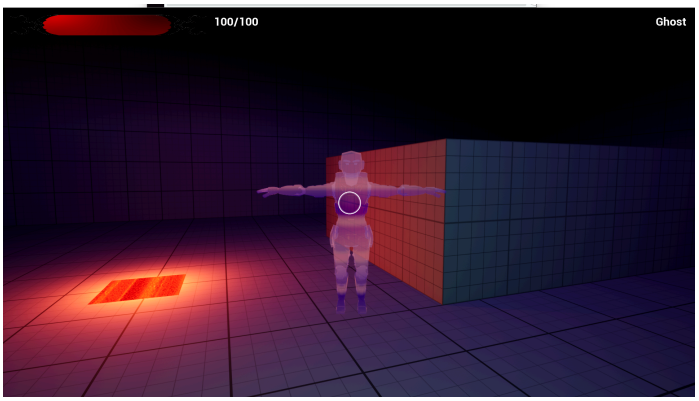
With this in mind, I created the current invisible dynamic material. It has blue and white bands that move down the ghouls body, and change in darkness and brightness and is semi transparent. Initially, the character only had this invisible effect applied to them that changes opacity when they activate the ability. However, I wasn't happy with this as I felt that it indicated that the ability was working at all times, which it wasn't.

To fix this, I changed it so that the ghoul had a rather plain grey material to begin with and upon becoming invisible, switched to this new 'invisible' material previously described. In future iterations I would like to explore how I can make this effect more blotched and appear as though it is rippling across the body, which I feel more similarly replicates the effect seen in most games.

Inspiration / Reference Images:

Skyrim - Invisibility Spell	Hogwarts Legacy - Disillusionment
	
https://imgur.com/FKLUVS7	https://media.thenerdstash.com/wp-content/uploads/2023/02/hogwarts-legacy-how-to-get-invisibility-best-ways.jpg.webp
In my opinion, skyrim really encapsulates the blue tinge that I think of when I think of an invisibility spell. It's also a very popular game that many are familiar with, so it makes sense to use it as inspiration when seeking to create a material that encourages familiarity.	Hogwarts legacy doesn't use the blue tint in their invisibility material, but it does have this dynamic sheen on the character depending on the area that they're invisible in. I chose this reference as I liked the depth it gave to the character, unlike some of the other effects that can seem quite flat.
Call of Duty - Cloaking Mission	**World Of Warcraft - Call of the Storm**
	
https://i.ytimg.com/vi/sNEeIpNy-M4/maxresdefault.jpg	https://www.wowhead.com/spell=266977/call-of-the-storm
The call of duty invisibility effect is unique from many other invisibility effects. It outlines the character in a strong blue, and then leaves the rest of them transparent. I chose this as a reference image as I liked the idea of having strong lines.	The world of warcraft invisibility material also features the strong blue I was thinking of, and has this rippling effect where the blue moves up and down the character in ripples. I chose this as a reference for this reason.

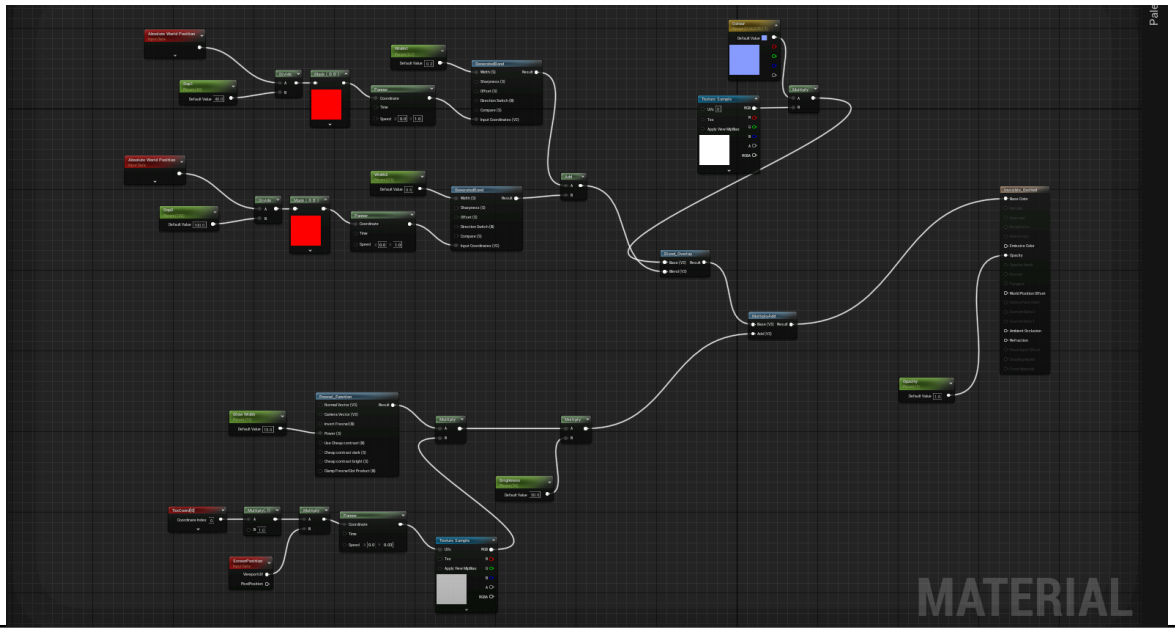
In-Engine Screenshots:

Default	Invisible
	
Before the character turns invisible, it appears as its default state, which is this plain grey.	Once the character turns invisible, its opacity turns down to 50% and the glowing banded effect begins. It will return to the default material once the player turns visible again.

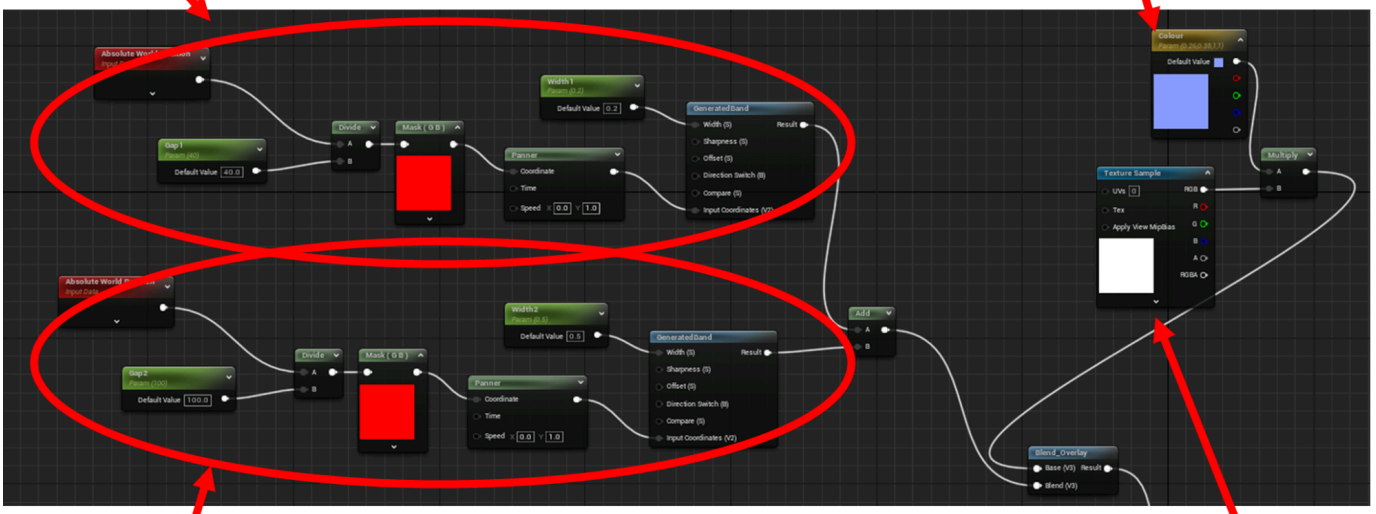
Properties and Values

Property	Description of Purpose	Value
Opacity	Used to control the opacity of the material dynamically	Float(0.5)

Node Graph

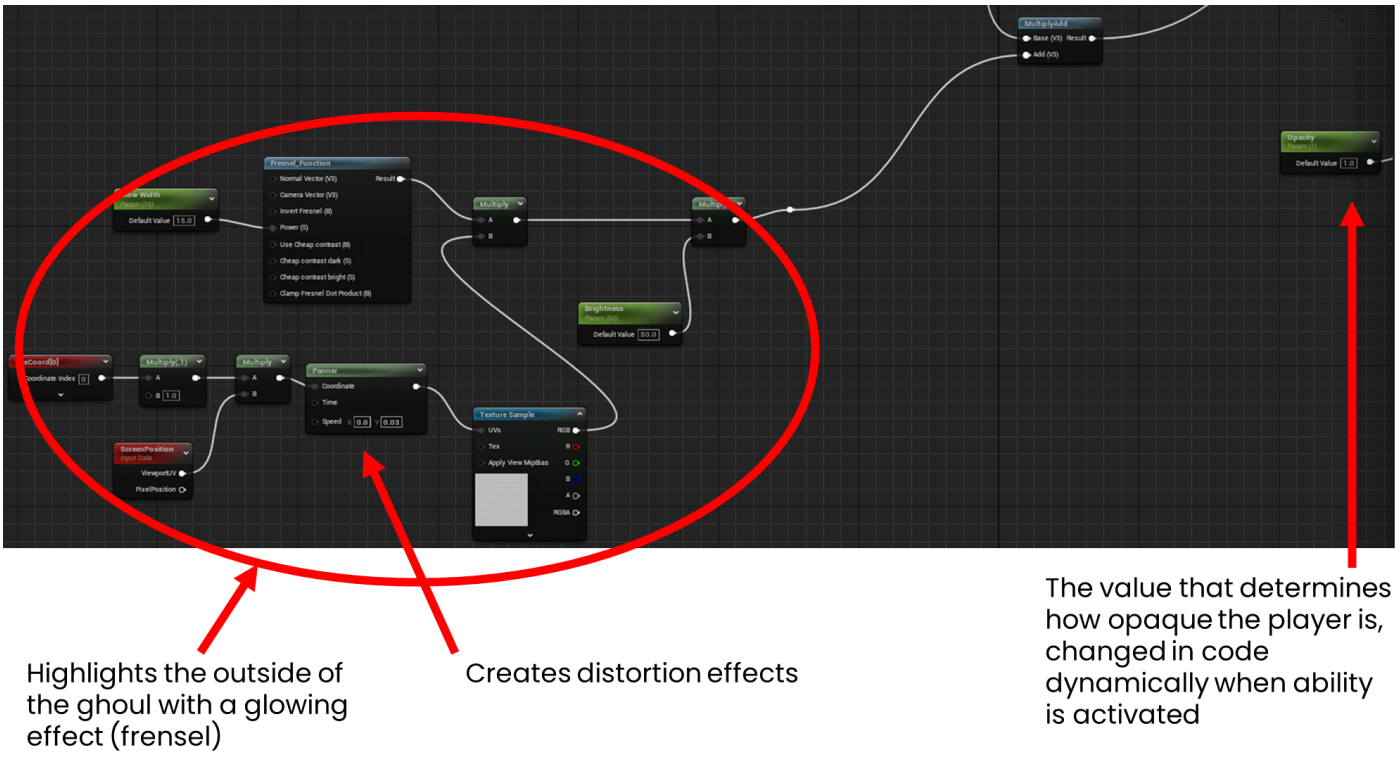
Entire Node Graph


Base Colour (Annotated)



Connected to the multiply add with the fresnel effect, which connects to the base colour node.

Fresnel Effect and Opacity (Annotated)



The circled nodes are connected to a multiply add which are joined with the panners, and then connected to the base colour node. The far right is connected to the opacity node.

Dynamic Material 2 - Wall Breaking (Wall_Excited)

Overview of Effect

Upon entering the game, there is no clear indication as to what is interactable and what is not. Typically when this happens, a player will interact with random objects until they get some sort of result, and then they will work out what to do with the newly discovered interactable object. Because of this, it was incredibly important that the wall that could be broken gave some feedback to the user that something was happening when they hit it with the fireball.

Using a dynamic effect for this material was a good way of giving the player this feedback, as the wall would visibly change the more that the player hit it. With future iterations, it is proposed that the effect be changed to replicate cracking, so that the player can tell that not only is the wall interactable, but that it will also likely break if the player were to keep hitting it.

Effect Description

On default, the wall has a stony pebbled appearance, which makes it stand out as different from the other walls in the level. This is to suggest to the player that this wall is different, and that they may be able to interact with it with a certain character.

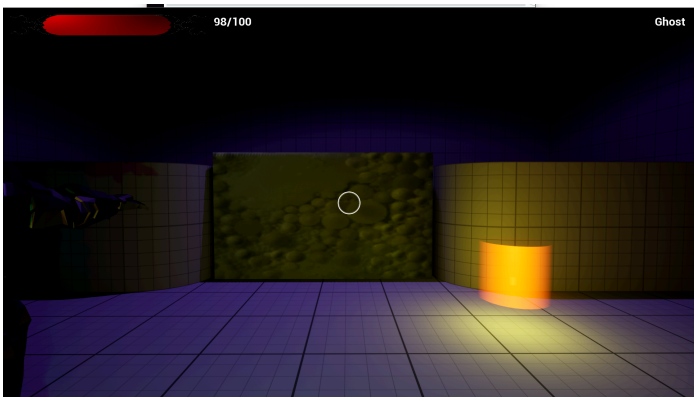
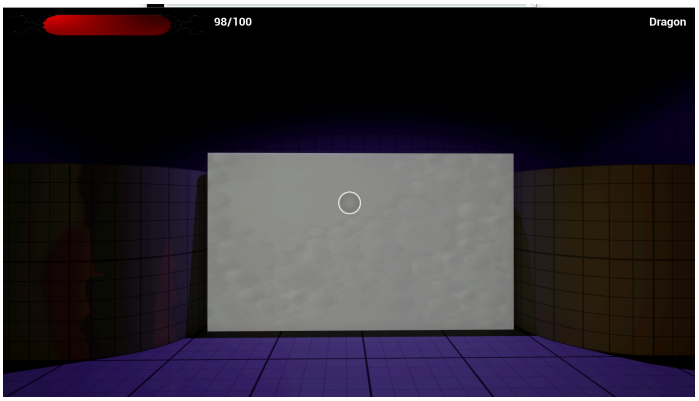
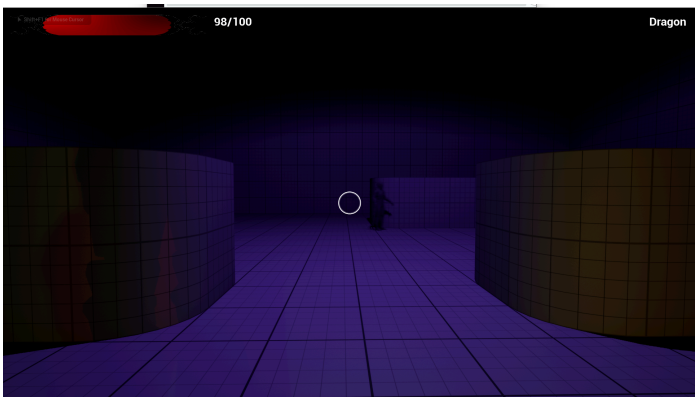
The wall becoming more emissive upon impact was largely used for testing purposes and to ensure the player could clearly see interaction with the world. However, in further development, the effect would be changed to show a cracking detail instead of an emissive appearance change which would make it feel more realistic for the game, and better fit the scene.

For this reason, most of the reference and inspiration images have been selected around what the future effect would look like.

Inspiration / Reference Images:

Spyro Reignited Trilogy - Evening Lake	
  	This wall in Spyro can be interacted with by the player to be broken through. The more the character hits it, the more it cracks open, faintly glowing in between each hit. Knowing that I wanted my wall to very clearly alert the player it was interactable and that its appearance or attributes had been updated, I particularly liked the emissive effect. Because of the very dark atmosphere in the dungeon a subtle dark crack appearing could be quite easy to miss, but having the emissive effect ensures that it'll be obvious to the player. Additionally, because the game is focused on magic, having a magic glow emit from the cracks caused by a magical creature is a nice touch.

In-Engine Screenshots:

Before Hit	One Hit
	
Before the wall is hit, it mostly just looks like a plain wall, except for the fact that its texture differs from the walls around it.	When the wall is hit for the first time, it becomes slightly emissive. In further development, this will be changed to be a small emissive crack.
Three Hits	**Four Hits (Destroyed)**
	
On the following hits, the wall becomes more emissive. In future development, this would be changed to show a crack slowly getting bigger from the centre of the wall, with a glowing emissive centre.	On the final hit, which is the fourth hit, the will is destroyed and disappears. In further development a change in the material might occur for a brief moment before the wall disappears, such as it flooding with the magical glow.

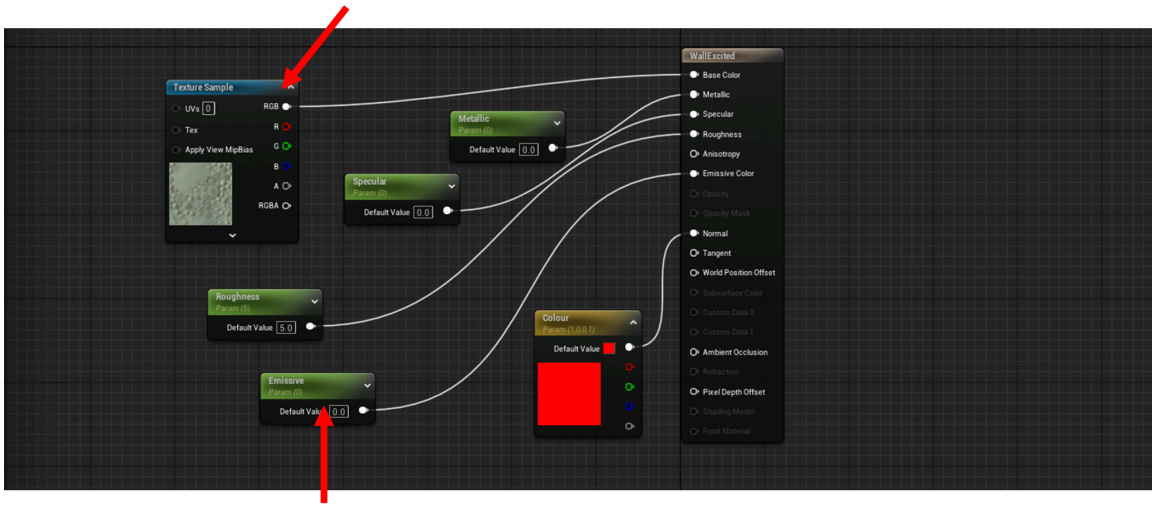
Properties and Values

Property	Description of Purpose	Value
Colour (case 1)	Used to control the colour of the material dynamically	FLinearColor(0.25, 0 , 0)
Emissive (case 1)	Used to control the emissive value of the material dynamically	Float(0.1)
Colour (case 2)	Used to control the colour of the material dynamically	FLinearColor(0.5, 0 , 0)
Emissive (case 1)	Used to control the emissive value of the material dynamically	Float(0.2)
Colour (case 3)	Used to control the colour of the material dynamically	FLinearColor(0.75, 0 , 0)
Emissive (case 1)	Used to control the emissive value of the material dynamically	Float(0.3)
Colour (case 4)	Used to control the colour of the material dynamically	FLinearColor1, 0 , 0)
Emissive (case 1)	Used to control the emissive value of the material dynamically	Float(0.4)

Node Graph

Entire Node Graph

The texture of the wall



The value that determines how emissive it is, changed in code dynamically on hit

Dynamic Material 3 - Brazier Lighting (Fire_Excited)

Overview of Effect

This dynamic material is the fire that is set at the top of the brazier. When the dragon hits it with a fireball it lights on fire, showing a fire sitting on top and letting off some light. It will give the player some light to make navigating the dark environment in the dungeon a bit easier.
A point light has been added to the brazier, so that upon the fireball hitting the brazier it will turn on so it will actually provide light to the area, giving the player actual reason to interact with the object.

Effect Description

Currently, when the brazier gets hit by a projectile it lights up and emits a soft red emissive glow in the pattern of the fire. The underneath layer sways slightly side to side, to give it the effect of the fire crackling and moving. As mentioned above, an orange/red pointlight has been added to the game so that it has more impact on the gameplay. This will be a very warm light, that is of medium strength to illuminate the dark dungeon and make it feel more like the dungeon has been abandoned, and that the player is slowly bringing it back to life.

Inspiration / Reference Images:

Hogwarts Legacy - Brazier Bridge Challenge	
 	In Hogwarts legacy, there are many unlit braziers around the world. The player can choose to interact with these by using Confingo or Incendio, fire spells and hitting the braziers with them. Upon being hit, they will ignite and continue to burn. As Spirits of Mythos is in a dark dungeon, the idea of braziers that can be used to add some more light to the world was quite appealing. It also allowed for some additional interaction with the world which made the game feel less plain and lacking. I liked the fact that the fire let off a soft glow as can be seen in the second image, and that it moved in the way that actual fire would and appeared as though it was burning.

In-Engine Screenshots:

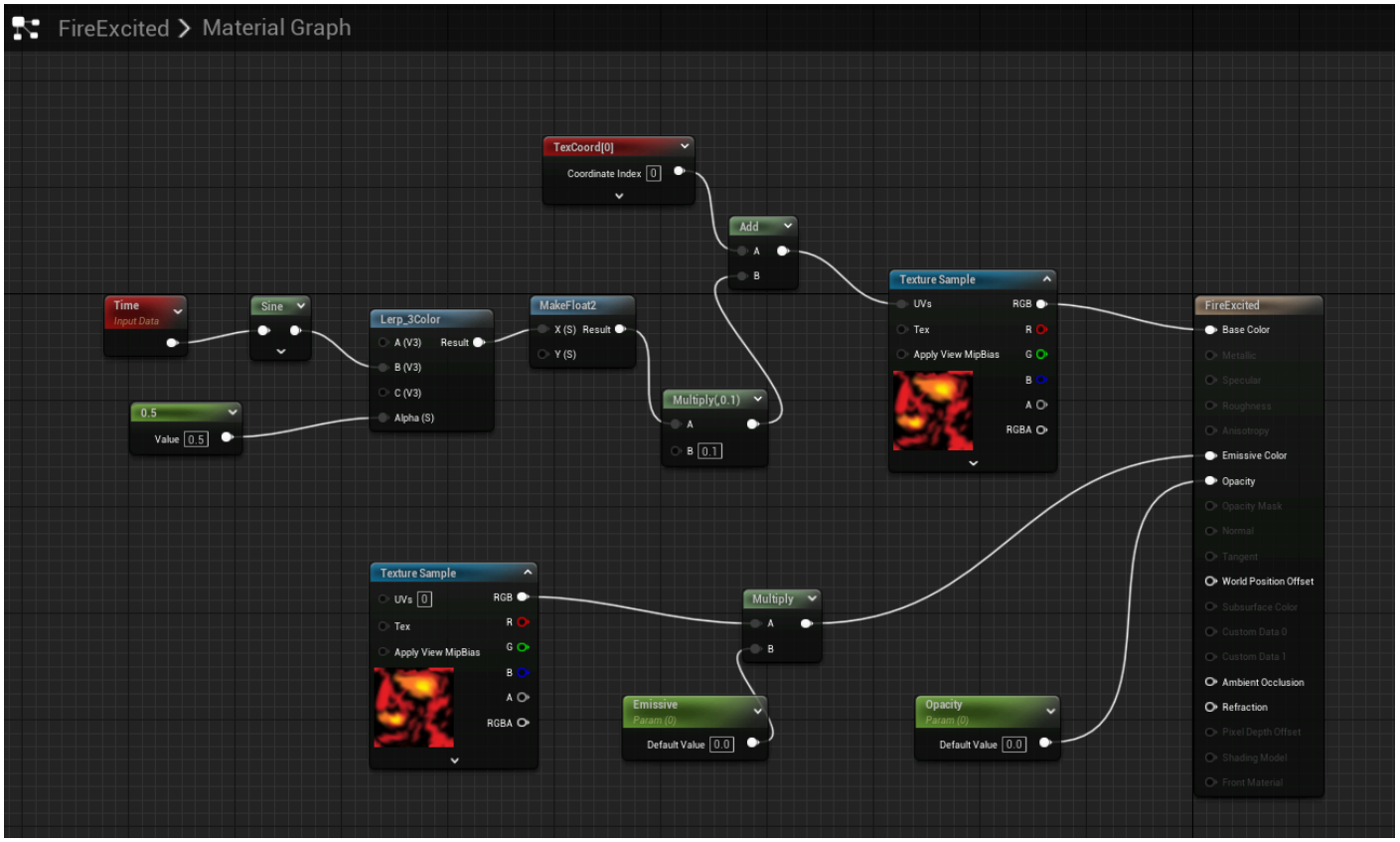
Unlit	Lit
	
Before the brazier is hit by the fireball, it's a plain pedestal.	Once the brazier has been hit by the fireball, the top fire shape appears and emits a soft red glow. It also has a soft pointlight to better light up the area.

Properties and Values

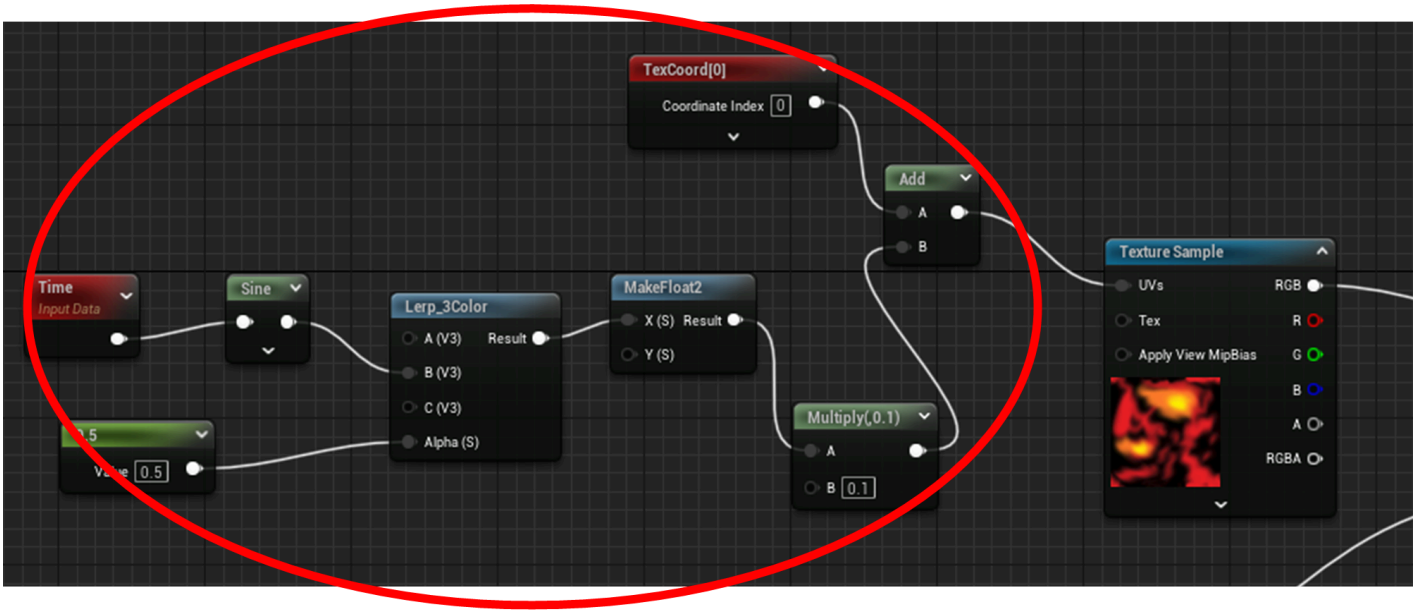
Property	Description of Purpose	Value
Opacity	Used to control the opacity of the material dynamically	Float(1)
Emissive	Used to control the emissive value of the material dynamically	Float(1)

Node Graph

Entire Node Graph



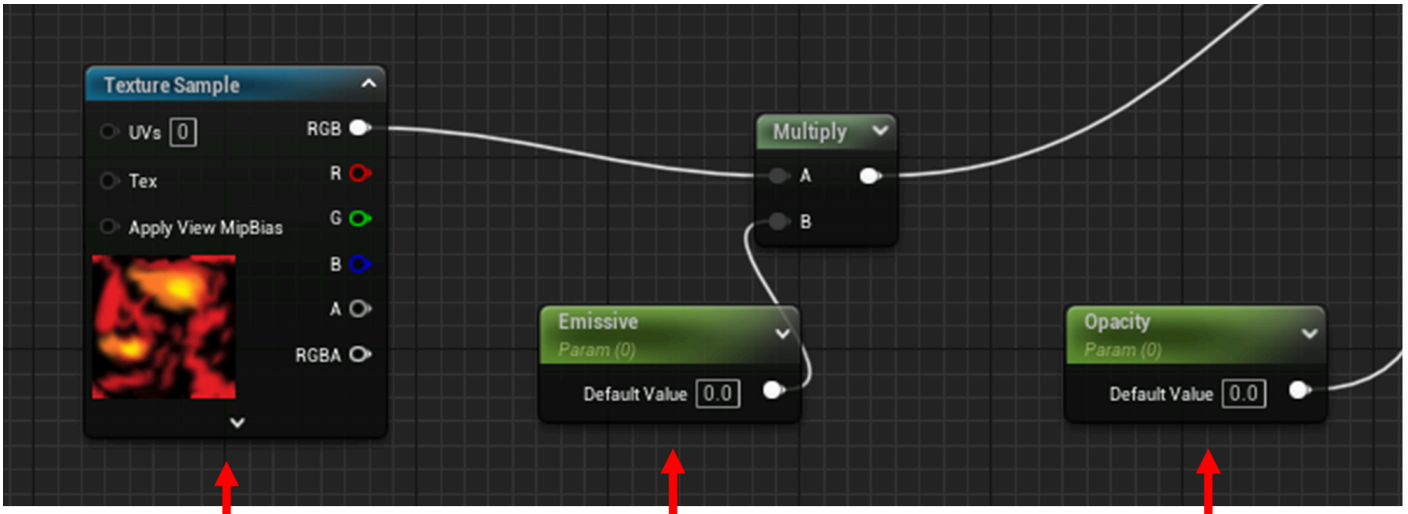
Base Colour (Annotated)



Makes the texture underneath the emissive sway left to right to produce 'alive' effect

Connected to the base colour node.

Emissive (Annotated)



Texture and colour for the emissive map

The value that determines how emissive it is, changed in code dynamically on hit

The value that determines if the fireball is visible or not (opaque or transparent), changed in code dynamically on hit

Connected to the emissive and opacity nodes.

In further development, a range of additional dynamic materials are likely to be implemented. Such an example of one for future implementation is a dynamic fireball shot from the dragon, which will decrease its size as it gets further away and also decrease emissiveness and opacity, representing fire fizzling away.

Physics

Overview of Interaction

When the player begins the game, all creatures except for the dragon are locked in cages. This is done to guide the player that the dragon is the first creature that should be interacted with. The physics interaction has been created as a way of freeing each of the additional creatures.

The current physics implementation is an explosion on the ignition of a stick of dynamite. In front of each of the cages housing the different creatures, a stick of dynamite has been created. Upon the ignition of this stick of dynamite, which is done by hitting it with a fireball, it creates an explosion which 'breaks open' the cage of the creatures. This allows the player to free them and now explore the level with them, therefore able to use their abilities.

Because this is quite simple, for future iterations of the game it is proposed that these cages will be placed in a different location, that are surrounded by some sort of magic or other interesting effect. Once the explosion has occurred, these sections of the cage will temporarily float before being slammed down by this second physics implementation, before they explode and disappear.

Interaction Description

How the Interaction Works

The interaction begins when the fireball hits the stick of dynamite. Upon hitting something, the fireball checks what it has hit. If it has hit a stick of dynamite, it then calls the dynamite to execute the interact implementation.

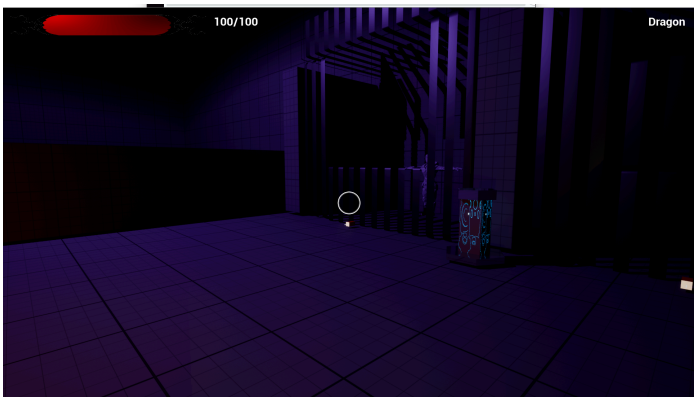
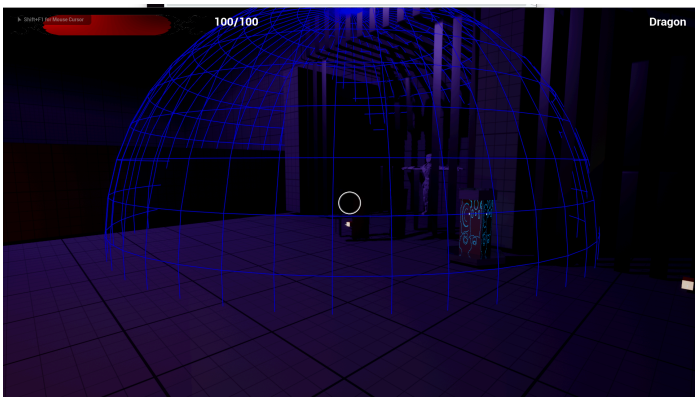
The interact implementation checks where the dynamite currently is and draws a collision sphere around it, which has been made a size of 500. This size was selected as it ensured that the explosion would hit all elements of the cage that were meant to move, but not overlap with much of the exterior world. Then, it checks all actors that have been hit within the collision sphere. If they have gravity enabled, then a radial impulse is added to them to push them away from the stick of dynamite, with fall off applied so that the effect is weaker the further away the actor is from the explosion. This is done to best replicate real world explosions. This is the gravity type push case.

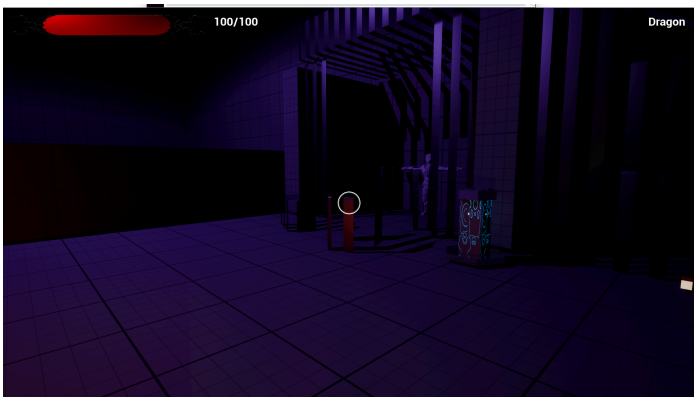
However, there is also a gravity type down case. While this currently has no function, in further development it will create the second effect that has been described above.

Inspiration / Reference Images

Minecraft - Creeper Explosion	Half-Life 2
	
Minecraft was actually a main inspiration for how the physics interaction would work in my game. I really liked that when a creeper exploded, while it did destroy the area it was on, it flung out the 'destroyed area' in the form of blocks you could pick up. I liked this idea of not completely destroying, just moving.	In a Half Life 2 Explosion, there are a couple things that happen. Some items will be destroyed and show a burning effect, while others are immune to the damage and will instead have impulse applied to them. In this example above, an explosion has taken place and the metal piping in the middle has been flung across the room. This is what I wanted to happen with the cages - they don't get destroyed as such, but they are somewhat broken apart and the door is torn off and flung.

In-Engine Screenshots

At rest (before)	During
	
Everything is at rest as the explosion has not yet been triggered.	The explosion has been triggered and the debug sphere has been drawn. It has read all of the objects in the area and for any that are interactable, applied an impulse of 2000 at the core with linear fall off. This has resulted in the cage's bottom bars being thrown backwards and upwards, due to the stick of dynamite being smaller than them and resting in front of them on the floor.

At rest (after)

The explosion has now finished, and everything has fallen back to the ground and finished interacting with each other. They'll remain at rest until another explosion.

Properties and Values

Property	Description of Purpose	Value
Origin	Where the explosive blast comes from	Location (of the dynamite)
Radius	How far the explosion travels/how far away an object can be interacted with	SweepSize (Currently set to 500)
Strength	The amount of impulse applied to the object at the centre of the explosion	2000.0f
Falloff	Allows the strength of the impulse to lower as the object is further away from the dynamite	RIF_Linear
bVelChange	Determines whether or not mass will have an effect	true

It is noted that in future iterations, bVelChange will likely be changed to false so that the dynamite can interact with different world objects as well. It is also suggested that the strength of the impulse would be decreased, as it is happening in quite a small space.

Artificial Intelligence

Overview of AI

As Spirits of Mythos is a game about being captured and hated, it felt very necessary to have a guard patrolling to ensure that none of the creatures escape the dungeon. Because of this, it was decided that an AI Guard would be created to fulfil this role.

The guard patrols the second area that the player enters, after breaking down the wall in the dungeon. It randomly generates new points to patrol within this second area, roaming the space until it spots the player. Upon their sight perception being triggered, if they are not invisible it will chase them and begin withdrawing health. This happens quite quickly as while the player is faster than the Guard, they should not just be able to run past them into a new safe area. The player will have a safe space to retreat to in the beginning section of the dungeon, as the AI cannot follow them there and their sight of the character will expire upon them crossing this point in the level.

It is proposed that in future development the perception sense of touch will be added to the guard, so that if the player approaches the guard from behind and runs into them they will be triggered, even if they do not see the player. Additionally, a hound would be created to follow the guard, with the perception element of sound, so that if the invisible character gets too close they will hear them, which will add an extra layer of difficulty to the game.

AI Description

AI Abilities

Currently, the AI Guard can only do a very limited range of tasks. It patrols the second area of the level by generating a new random location, moving to this point and then waiting until it patrols to a new destination. Currently, it can only be triggered by sight of the player. Upon sight, the AI will check if the player is currently invisible. If they are, it will continue patrolling and ignore them.

However, if the player is visible, it will save its location and begin to follow the player. At the same time, the AI will call the health alteration function from the player class to begin removing health.

While the current AI implementation is very simple, suggestions have been outlined above to how it may be changed in further development to be more complex. Due to the plan to implement these, the suggested inputs required for these have been included and highlighted blue.

Inputs & Senses

AI Senses

Sense Name	Property	Value
Guard		
Sight	Sight Radius	500
	Lost Sight Radius	Sight Radius + 30 (530)
	Sight Age	3.5
	Field of View	90
Touch	Max Age	5
Hound		
Sight	Sight Radius	700
	Lost Sight Radius	Sight Radius + 30 (830)
	Sight Age	5
	Field of View	135
Hearing	Hearing Range	800
	Hearing Age	5
Touch	Max Age	6

Blackboard Values

Property	Description	Related Actions
Guard		
PatrolPoint	Vector variable containing a randomly generated location within the nav mesh, updated when a guard is not chasing the player	Generate New Random Location, Movement
PlayerPosition	Vector variable containing the players current location, updated every tick as the player moves	Chase Player
ChasePlayer	Bool variable containing if the player should be chased, updated when stimulus is sensed	Kill Player, Chase Player
Hound		
GuardPosition	Vector variable containing the Guard AI's position + a small vector offset, updated when the Guard moves	Follow Guard
PlayerPosition	Vector variable containing the players current location, updated every tick as the player moves	Chase Player
ChasePlayer	Bool variable containing if the player should be chased, updated when stimulus is sensed	Kill Player, Chase Player

The blackboard values and AI senses work together to allow the AI to execute tasks. For the currently implemented AI guard, the sense of sight and proposed sense of touch are what determine which action the AI will take. If its senses have not been stimulated, then it will continue to patrol the game. When there is no stimulus, the chase player property is cleared and it is instructed instead to move to the patrol point, which is generated by the generate new location function. However, when the senses are updated the chase player property is set to true, and the player position is updated. The guard will now turn to the player and chase them, as well as calling the remove health function. The guard will continue to chase the player, until the player is no longer in their sight and their sight or proposed touch has expired.

The Guard has been given a field of view of 90, as humans typically have around 180 degrees of vision.

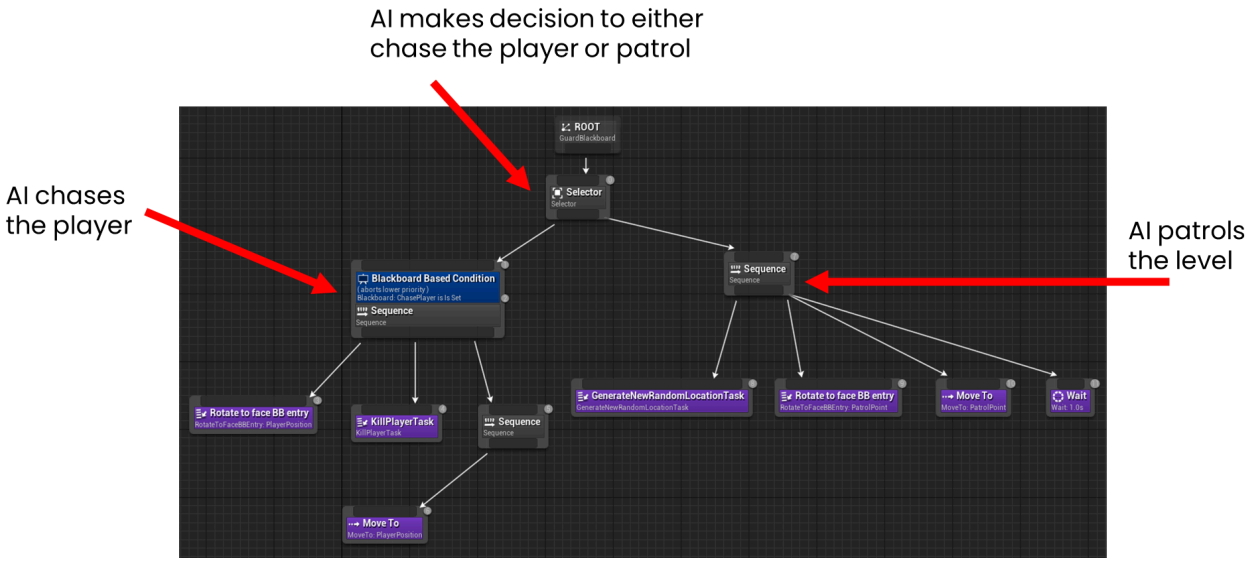
The proposed values for the hound would work in a very similar way. The hound will either be patrolling the level and following the guard, or chasing the player.

When the hound is following the guard, it will take the guard's location and add a small vector offset to it so that the hound is following at a slight distance, rather than being overlapped with the guard. When the hound is stimulated by either the sight, hearing or touch OR the guard is stimulated, the chase player will be implemented in the same way as the guard.

The hound has been given a better sense of sight than the Guard, as generally animals have better senses than humans. The hounds field of view has also been set to 135, as a dog generally has around 270 degrees of vision.

Behaviour Tree Graph

When the AI is active, it's doing one of two things: either chasing the player and draining their health, or patrolling the level. The behaviour tree below belongs to this AI.



Because the AI needs to chase the player when it sees them, rather than continuing to patrol, the blackboard condition of chase player is set is given a higher priority. This means the AI will always chase the player over other tasks if its stimulus has been triggered.

When Chase Player is set, the AI will rotate to turn the player and move incrementally towards the updated player location each tick. While the player is in sight, it will also call the kill player function, which removes 1 health each tick. When the AI has lost sight of the player or its sight has expired, it will go back to patrolling the level.

Otherwise, when the chase player is not set, the AI will patrol the level. It'll call the generate new random location task to receive a new location, then it will rotate to face this new location. Then, it'll travel to this new location and wait 1 second. This will continue to loop until the AI stimulus is triggered again.

Niagara Particles

Niagara Particle Effect 1 - Fire Trail (NS_FireTrail)

Overview of Effect

The FireTrail effect adds another layer of detail to the fireball projectile. As this is one of the most used abilities in the game, it was important that this effect was the most developed and had more complexity to it. To achieve this, a particle effect was created and added to the player's fireball projectile.

It's a stream of fire, sparks and ash that follow behind the projectile, connected at the fireball mesh. It lights up the environment like an actual fireball would, which makes the projectile feel more like a fireball. The further away the fireball travels the more it disperses, again feeling more realistic. Having the projectile look more like fire is also useful for conveying to the user what they can do and may use the ability for - such as lighting the braziers.

Effect Description

I wanted the effect to appear as a sizzling trail of sparks and embers left behind as the projectile flys through the air, as well as a little bit of ash as these burn out. Because I wanted it to appear as though the effect was a result of the fireball, I had to attach the effect to the actor and make sure it followed it's rotation to make it seem as though it was flowing the same way and as though it was being altered by the wind it was pushing against, instead of going the same way no matter what direction it was shot in.

Inspiration / Reference Images:

Spyro: A Hero's Tail - Fire Breath	Hogwarts Legacy - Incendio
	
In this example of fire, I particularly like the mix of reds and yellows that have been used. In the	I quite like the Hogwarts Legacy fire effect, from how it streams out of the wand and follows a

hogwarts legacy effect the colour is very flat, whereas this one has an interesting depth to it. I'd like to explore how different emitters can be set to different colours to achieve something similar.	trajectory of an invisible projectile, as well as how sparks shoot away from the main fire effect. I also noted the emissivity of the particles.

In-Engine Screenshots:

Projectile has been fired

The fireball has been shot, and is trailing after the projectile. Here, the player has been moved to the side slightly to better show the effect and how it follows the projectile, as well as how it is flaring out at the back and beginning to drop.

Properties and Values

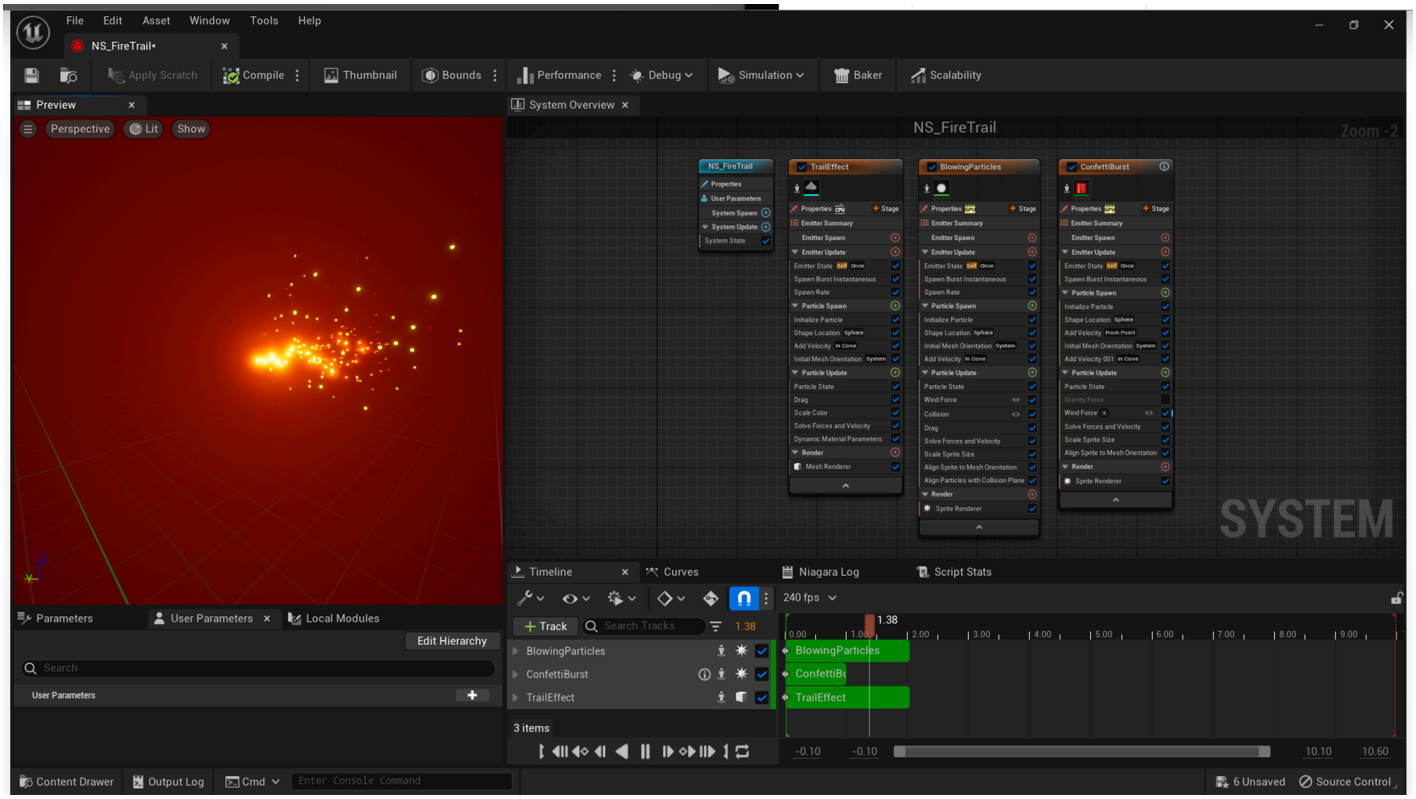
*Only values that have been changed from the templates have been included

TrailEffect		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Once
Spawn Burst Instantaneous	Spawns an Instantaneous burst of particles	Spawn Count: 1
Spawn Rate	Decides how many particles spawn	SpawnRate: 90
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Direct Set Lifetime Min: 2 Lifetime Max: 3 Colour Mode: Direct Set Colour: R1, G0.06, B0, A1 Sprite Size Mode: Random Uniform Uniform Sprite Min: 4 Uniform Sprite Max: 8
Shape Location	Location of the system	Shape Primitive: Sphere Sphere Radius: 5
Add Velocity	Add velocity to the particles	Velocity Mode: In Cone Velocity Speed Min: 500 Velocity Speed Max: 850 Cone Axis: -1,0,0 Cone Angle: 90
Mesh Renderer	Responsible for rendering the mesh	Meshes: ControlRig_Diamond_solid Enable Override Material: true Override Material: FireSparks
BlowingParticles		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Once
Spawn Burst	Spawns an Instantaneous burst of	Spawn Count: 1

Instantaneous	particles	
Spawn Rate	Decides how many particles spawn	SpawnRate: 40
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 2 Lifetime Max: 3 Colour Mode: Direct Set Colour: R1, G0.9, B0, A1 Sprite Size Mode: Random Uniform Uniform Sprite Min: 4 Uniform Sprite Max: 8
Shape Location	Location of the system	Shape Primitive: Sphere Sphere Radius: 50
Add Velocity	Add velocity to the particles	Velocity Mode: In Cone Velocity Speed Min: 500 Velocity Speed Max: 850 Cone Axis: -1,0,0 Cone Angle: 90
Wind Force	Adds wind force to the particles	Wind Speed: 200,0,0
ConfettiBurst		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Once
Spawn Burst Instantaneous	Spawns an Instantaneous burst of particles	Spawn Count: 100
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 1 Lifetime Max: 4 Colour Mode: Unset Sprite Size Mode: Random Uniform Uniform Sprite Min: 4 Uniform Sprite Max: 8
Shape Location	Location of the system	Shape Primitive: Sphere Sphere Radius: 12
Add Velocity	Add velocity to the particles	Velocity Mode: From Point

		Velocity Speed: Random Range Float Velocity Speed Min: 150 Velocity Speed Max: 600 Origin Offset: 0,0,-8
Add Velocity	Add velocity to the particles	Velocity Mode: In Cone Velocity Speed Min: 400 Velocity Speed Max: 600 Cone Axis: -1,0,0 Cone Angle: 15
Wind Force	Adds wind force to the particles	Wind Speed: 50,0,0
Sprite Renderer	Renders the sprite	Material: TranslucentLaserPointer

Niagara System / Emitters Breakdown

Entire System


Main Effect (Annotated)

Responsible for main glowy particles (the center effect)

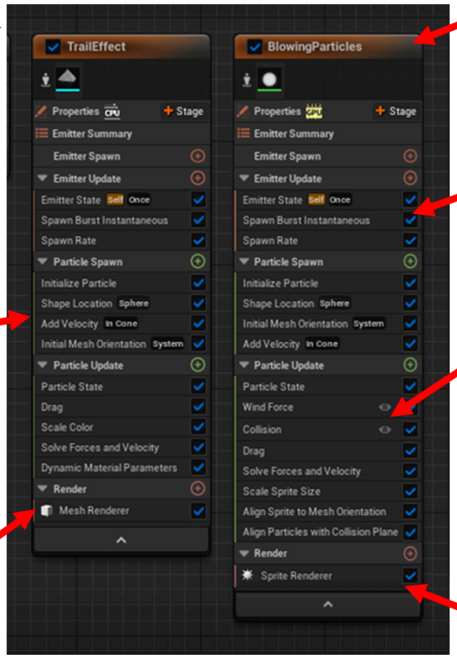
Responsible for the sparks

Spawns a group of particles instantly and only once, so the spawned trails follows the projectile

Spawns a group of particles instantly and only once, so the spawned trails follows the projectile

Adds velocity in a cone, allowing the sparks to 'fall off'

Pushes them out more, separating them from other particles



Renders the mesh

Renders the sprite

Ash (Annotated)

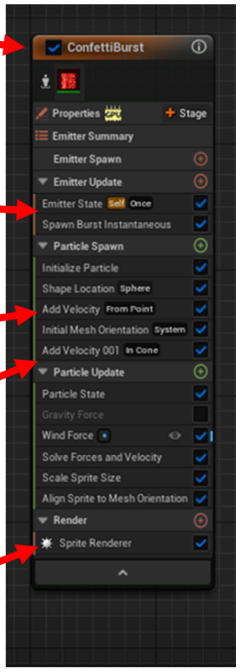
Responsible for the bursting ash

Spawns a group of particles instantly and only once, so the spawned trails follows the projectile

"pushes" them out from projectile point

Adds velocity in a cone, allowing the sparks to 'fall off'

Renders the sprite



Niagara Particle Effect 2 - Lava Pit (NS_LavaPit)

Overview of Effect

The Lava Pit effect adds more depth to the existing lava pit and makes it more obvious what it actually is and how it might be interacted with. This effect was spawned on top of the lava pit, fitting the dimensions of the lavapit component meaning it spanned the entire component hit box area, and not any further.

Having this effect further conveys the idea that the player cannot walk on this area. However, the spurting of the lava can be seen from over the wall which creates a point of interest and encourages the player over to the area, directing their play.

Effect Description

It shows a bubbling pit of lava that sits on top of the red material base. These bubbles constantly spawn and move, with sparks drifting up from this to give the effect of the heat and fire bubbling up and off of it. There is a plume of lava that intermittently bursts from the lava pit, flowing for a few seconds before falling back down.

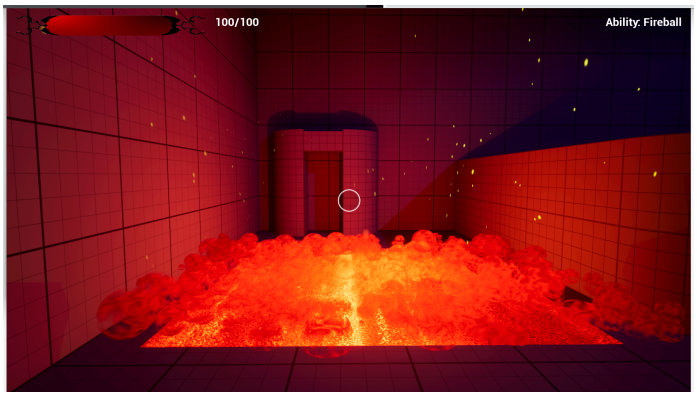
I had a very specific idea of how I wanted this lava pit to look, and I feel that the reference images fit perfectly with this vision. I wanted it to be textured and dynamic, not just flat like a lot of lava is in games. I also knew that I wanted it to have an event where it would create this lava plume.

Inspiration / Reference Images:

Hot Lava - Level 3	Wizard101 - Dragonspyre Fishing
	
I really liked the 3D effect the lava had here, and that it wasn't just a plain flat texture like it	In this instance of lava, I liked the 'fountain spurt' that shot up randomly from the lava

is in a lot of other games. I also liked the mix of yellow and red in the lava, which gave it additional depth on top of the physical depth.	pool. It added additional detail and texture to the effect which I really liked, rather than being a mostly static effect.

In-Engine Screenshots:

At rest	With flume
	
The lava bubbles with different sized bubbles that differ slightly in colour, with sparks coming up into the air.	The lava spurts upwards, creating a flume that runs for a couple of seconds before falling back down again. It then stays at the at rest stage for a couple of seconds before spurting up again.

Properties and Values

*Only values that have been changed from the templates have been included

HangingParticulates001		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Infinite
Spawn Rate	Decides how many particles spawn	SpawnRate: 40
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 5 Lifetime Max: 8 Colour Mode: Direct Set Colour: R1, G0.8, B0, A1 Mesh Scale Mode: Random

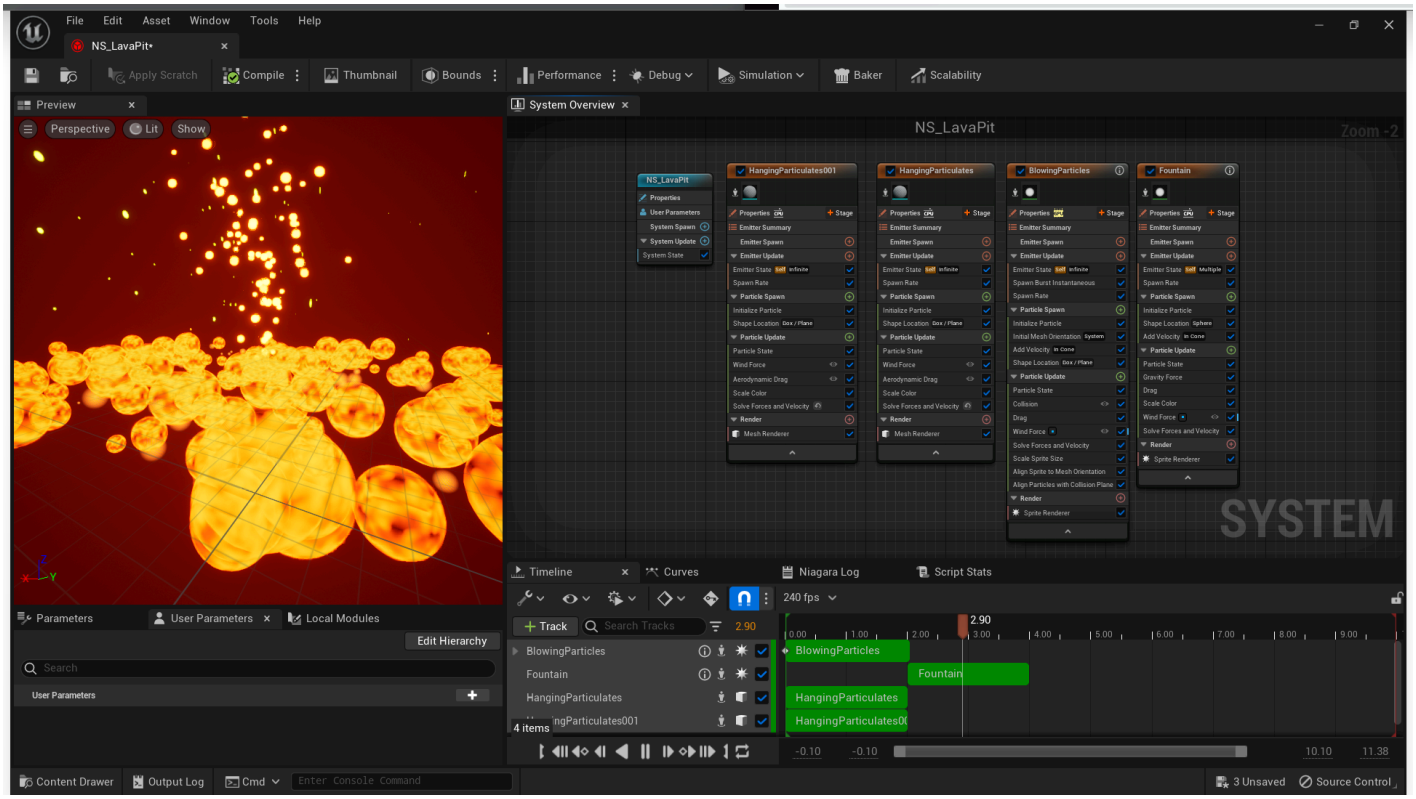
		Uniform Uniform Sprite Min: 0.02 Uniform Sprite Max: 0.2
Shape Location	Location of the system	Shape Primitive: Box/Plane Box/Plane Mode: Box Plane Size: 600,400,30
Wind Force	Adds wind force to the particles	Wind Speed: 1,0,0
Mesh Renderer	Responsible for rendering the mesh	Meshes: MaterialSphere Enable Override Material: true Override Material: M_LavaPit
HangingParticulates		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Infinite
Spawn Rate	Decides how many particles spawn	SpawnRate: 40
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 5 Lifetime Max: 8 Colour Mode: Direct Set Colour: R0.4, G0.05, B0, A0 Mesh Scale Mode: Random Uniform Uniform Sprite Min: 0.1 Uniform Sprite Max: 0.5
Shape Location	Location of the system	Shape Primitive: Box/Plane Box/Plane Mode: Box Plane Size: 600,400,30
Wind Force	Adds wind force to the particles	Wind Speed: 1,0,0
Mesh Renderer	Responsible for rendering the mesh	Meshes: MaterialSphere Enable Override Material: true Override Material: M_LavaPit
BlowingParticles		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter	Loop Behaviour: Infinite

	runs	
Spawn Burst Instantaneous	Spawns an Instantaneous burst of particles	Spawn Count: 1
Spawn Rate	Decides how many particles spawn	SpawnRate: 40
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 2 Lifetime Max: 3 Colour Mode: Direct Set Colour: R1, G0.09, B0, A1 Sprite Size Mode: Random Uniform Uniform Sprite Min: 4 Uniform Sprite Max: 8
Shape Location	Location of the system	Shape Primitive: Box/Plane Box/Plane Mode: Plane Plane Size: 600,400
Add Velocity	Add velocity to the particles	Velocity Mode: In Cone Velocity Speed Min: 200 Velocity Speed Max: 300 Cone Axis: 0,0,1 Cone Angle: 90
Wind Force	Adds wind force to the particles	Wind Speed: 0,0,-200
Fountain		
Property	**Description of Purpose**	**Value**
Emitter State	Controls how often and when the emitter runs	Loop Behaviour: Multiple Loop Count: 10 Loop Duration: 1.98 seconds Delay: 2 seconds
Spawn Rate	Decides how many particles spawn	SpawnRate: 90
Initialize Particle	Determines the settings on the initialization of the particle	Lifetime Mode: Random Lifetime Min: 1.4 Lifetime Max: 1.75 Colour Mode: Direct Set Colour: R1, G0.07, B0, A1 Sprite Size Mode: Random Uniform Uniform Sprite Min: 6

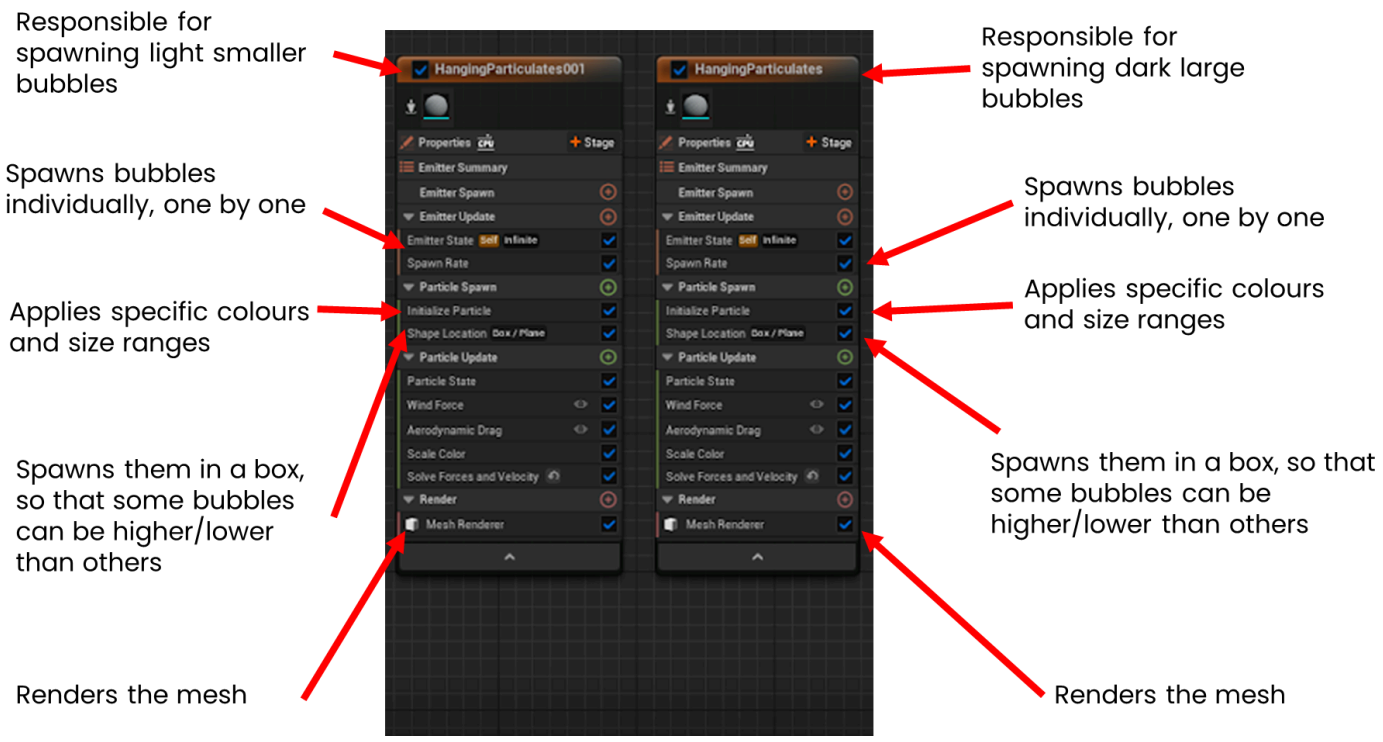
		Uniform Sprite Max: 12
Shape Location	Location of the system	Shape Primitive: Sphere Sphere Radius: 20
Add Velocity	Add velocity to the particles	Velocity Mode: In Cone Velocity Speed Min: 500 Velocity Speed Max: 850 Cone Angle: 32
Gravity Force	Applies gravity to the particles	Gravity: 0,0,-980
Wind Force	Adds wind force to the particles	Wind Speed: 0,0,300

Niagara System / Emitters Breakdown

Entire System



Lava Bubbles (Annotated)



Lava Flume and Sparks (Annotated)

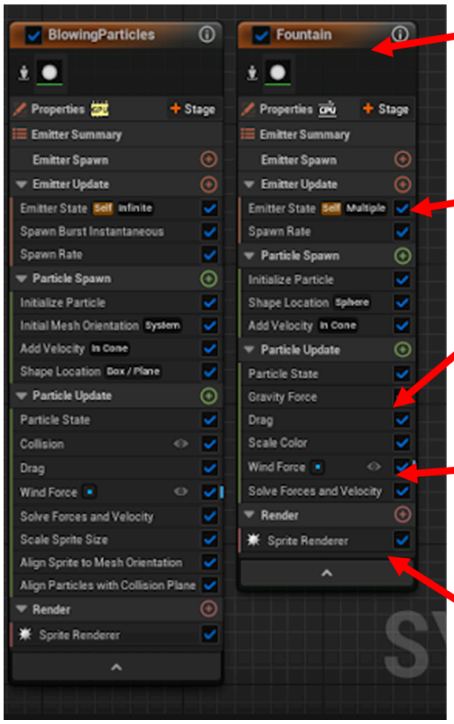
Responsible for spawning sparks

Spawns a group of particles instantly, so that they all blow up in once instead of in trickles

Spawns them on the same plane as the lava pit then adds velocity in a cone

Applies and upwards wind force so they float up

Renders the sprite



Responsible for spawning lava spurt

Makes effect happen intermittently

Causes them to fall back down after peaking

Applies and upwards wind force so they forcefully spurt upwards

Renders the sprite

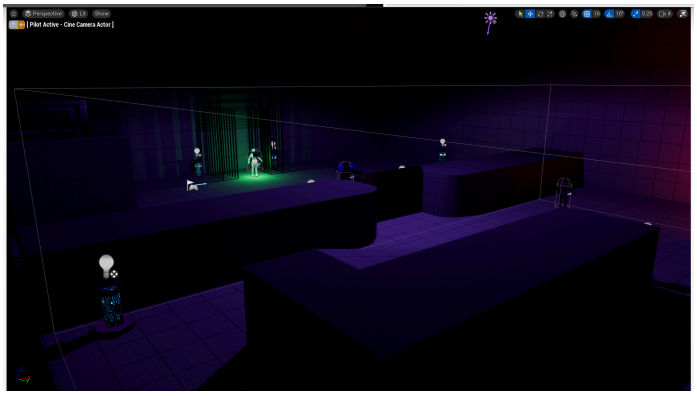
Sequencing / Cinematic

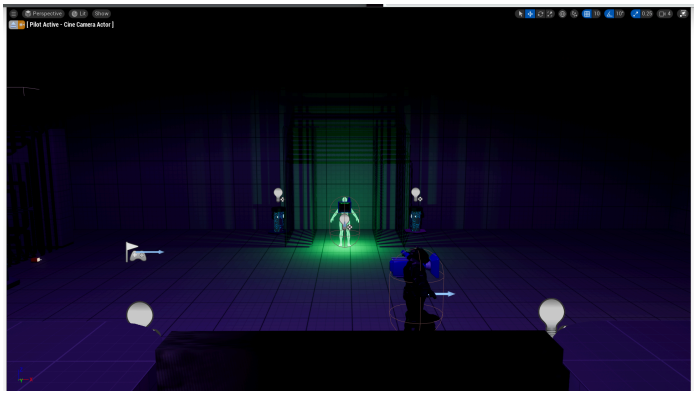
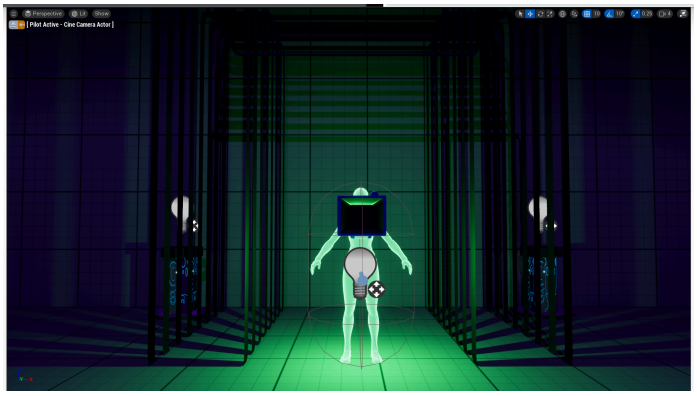
Overview of Sequence

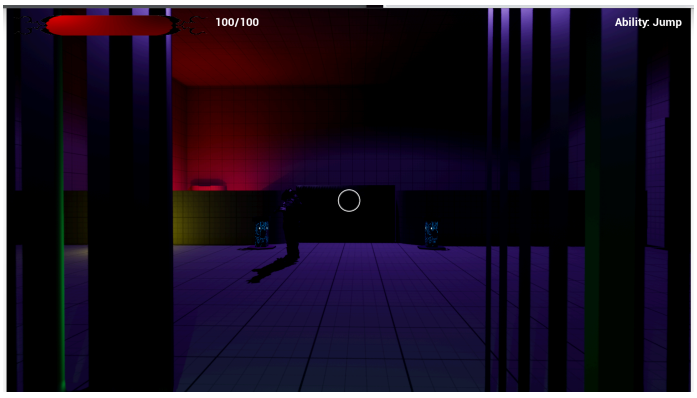
An Opening sequence cinematic has been created to introduce the user to the level, as well as the main character that they will be playing. It features a brief pan across the dungeon, before zooming in infront of the character. It pauses briefly, showing them free in the dungeon, before being slammed into the cage and closing to black. It then shows them slowly blinking and waking up, looking around the level before control is handed back to the player. In the start of the cutscene the braziers are lit and at the end they have gone out, meant to convey the time that has passed since they've been caged.

The cutscene makes the game more atmospheric and appear more polished, which greatly enhances the player's experience. It very briefly provides a glimpse of behind the wall in the game, which gives the player the information that they will need to explore past this area somehow. The cut to darkness and the braziers turning off shows that there has been a clear pass in time and change in the environment since the cutscene has taken place, and also suggests to the player that they might be able to interact with the environment and these braziers to bring light. The camera moving backwards after the blinking and the FOV widening shows the user that they are now playing the character that has been previously shown, and suggest that control has now been handed over to them. Once this has ended the player has full control to be able to play the game.

Camera Angles / Properties

OpeningSequence	
	1. Where the camera starts the cinematic from. This is where the camera begins before moving to its next location and changing angles. Location: 31, 3247, 614 Rotation: -0, -13, -51

	2. The camera has now moved in to focus better on the character, settling here in line with them before moving in to show a closer picture of them. Location: 1488, 2143, 425 Rotation: -0, -8, -89
	3. The camera has now settled on the character, and will stay paused here for a moment before triggering the movement event. Location: 1466, 927, 153 Rotation: -0, -2, -89
	4. The camera has now just slammed forwards into the wall with the character and faded to black, preparing for the eye blink scene. Current Focal Length: 25 Vignette Intensity: 100 Location: 1478, 193, 205 Rotation: -0, -1, -90
	5. The eye blinks. There are three eye blinks, and with each one the scene comes more into focus. The Vignette Intensity changes between 0 and 5 to simulate the eyes closing, with the focus distance slowly increasing from 1.5 to 4000 during the scene. Each eye blink slightly moves the camera left, then right, the back to the centre.

	6. The camera draws back to more seamlessly switch to the player camera. Current Focal Length: 20 Location: 1447, 188, 170 Rotation: -0, -0, -88

Scripted Events

There are three scripted events that occur during the sequence. The first and third event will be talked about together, as they directly relate to each other. The first occurs right as the cutscene begins, which is turning on all of the braziers. The function is executed from the sequence, casting to the currently active gamemode. From here, it runs the 'TurnOnLights()' function. This function first creates an array, which it will then fill by taking all of the actors in the world of type brazier. Then it has an iterator, in which for each of the braziers stored it will run the 'TurnOn()' function in the brazier class. This function sets the Dynamic material to be visible and emissive, as well as setting the intensity to 5000.0f for the point lights. Once this has been done, the iterator will move to the next brazier in the array until they are all turned on.

This happens right as the sequence begins, which gives the appearance that all of these are already on when it happens. This gives the appearance that the dungeon is inhabited and visited frequently and has been maintained.

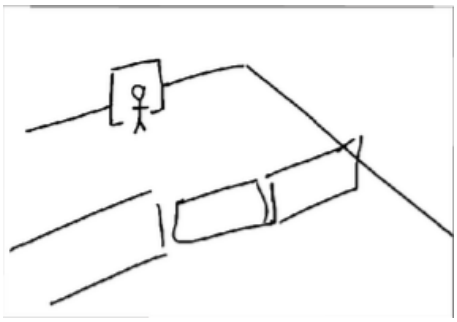
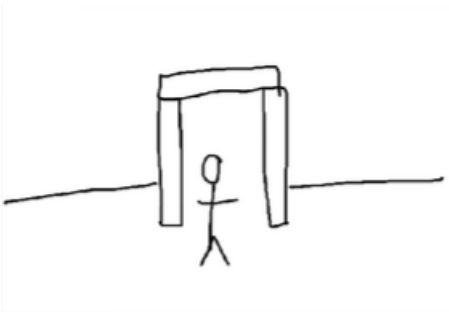
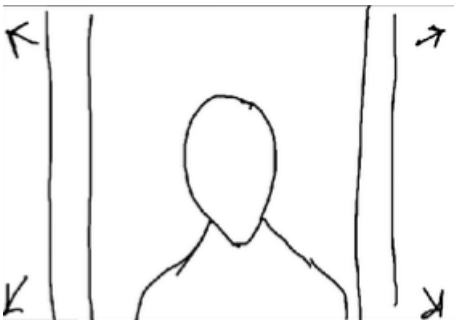
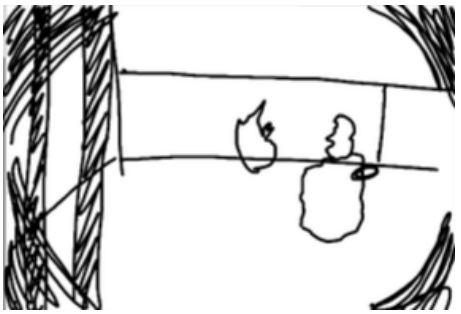
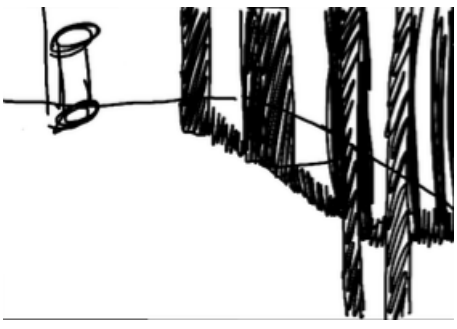
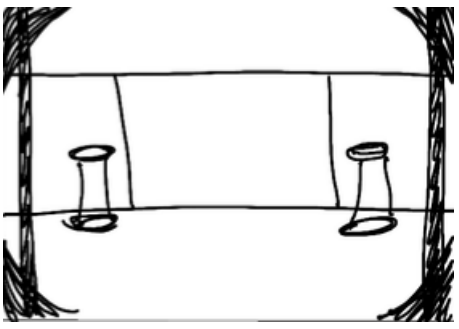
The third event is very similar to the first. It turns off all the braziers in the level and is executed in the same way as turning the lights on. Called from the sequence, the 'TurnLightsOff()' function iterates through each of the braziers in the array, calling the 'TurnOff()' function from brazier. This sets the dynamic material to be transparent and sets emissive to zero, as well as setting the intensity of the point light to be 0, therefore it appears off. It does this for each of the braziers while the character's screen is black after having hit the wall. This event is done here so that it makes the dungeon seem more cold and convey a long time skip from when the character was caged, to when they're waking back up.

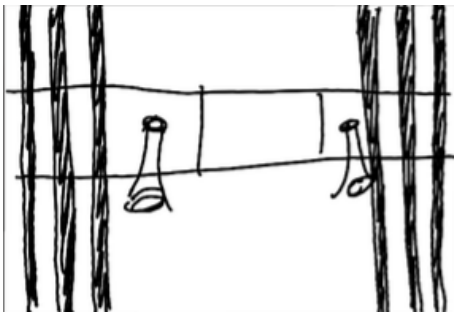
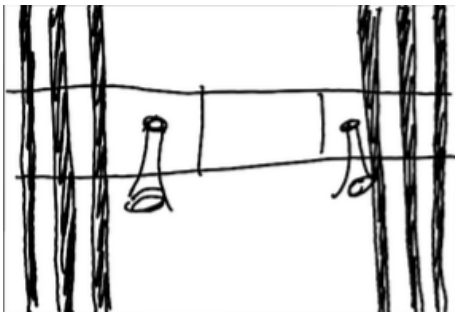
The other event that runs is the caging of the character, which is located in the character class. This function is called 'MoveCharacterToCage()' and is called in the sequence by casting to the currently played character and then running the function. This function is called LaunchCharacter, which takes the force to be applied to the character and moves them accordingly. A force of 4000

in magnitude and direction has been applied to the character in the -y direction, which gives the appearance of the character being slammed backwards into the cage and knocked out.

This allows the story of the character being caged to be told, as well as showing that time has passed while they've been asleep. This fade to black allows the camera to change from third person to first person as well in a less jarring way.

Storyboard

		
Start at the top left corner	Zoom in on the character standing in front of the cage, line up with them	Fly in quickly, zooming in as they're thrown into the back of the cage and follow the movement
		
Fade quickly to black as they pass out	Fad black away slowly in eye opening effect, blurry and looking to the left	Fade back to black, starting to move to the right
		

Fad black away slowly in eye opening effect, blurry and looking to the right	Fade back to black, starting to move to the middle	Fad black away slowly in eye opening effect, blurry and looking to the middle and focus in on everything
		
Slowly pan backwards and widen the FOV	Hand back control to the player and move to the player camera	

MetaSound

Overview of Sound Effect

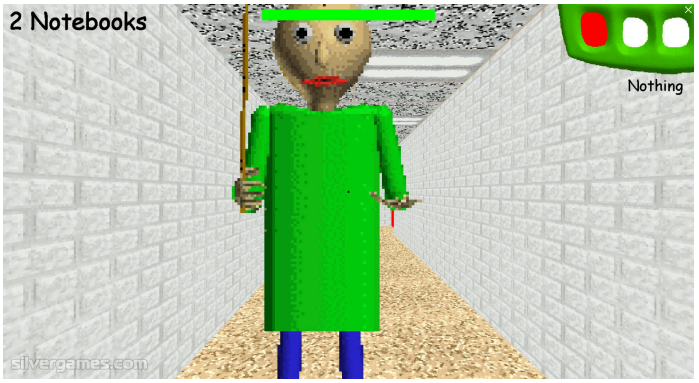
The MetaSound that has been implemented is the ChaseNoise sound effect. This sound effect is triggered upon the AI Guard spotting the character, and ends when their senses are no longer being stimulated. The sound effect has a few layers to bring the game more atmosphere and make it feel as though the AI is further interacting with the player. A low droning whistle rings out continuously, with a clanging noise that changes in speed depending on the distance from the AI. There are also panting noises added to this effect, making it feel as though the character is scared and exhausting themself trying to escape the AI.

The MetaSound has been implemented as a way of building tension while the character is chased by the AI, as well as alerting them that they are now being chased. Because of the dark nature of the dungeon, if the AI creeps up behind them they may be unaware of its presence. A sound effect such as this would alert them that they are now being chased adding to the atmosphere that the dark environment provides.

Effect Description

Upon the AI sensing the player, a droning whistle that moves up and down in pitch will play. This is a continuous drone that won't stop until the AI loses sight of the player. There will also be a sharp clanging noise that will repeat at a speed dependent on the distance of the AI guard from the player. I knew that I wanted the main goal of this sound effect to eb to contribute to the atmosphere of the game, so I looked to games that used noises to build suspense for inspiration.

Inspiration / Reference:

Baldi's Basics - Ruler Slap	
	Baldis Basics has a sound effect where Baldi regularly slaps a ruler against his hand, and the player can use the volume of this noise to determine how close or far away Baldi is from them. I really liked the idea of a continous sound changing depending on the distance to the player and the possibility of using this as a mechanic. However, I wanted to do something other than just changing the volume, which made me think of a sound that repeats at different rates depending on the distance from the player.
SirenHead: Southpoint - Chase Noise	
	When SirenHead begins to chase the player, there is a dorning radio noise (playing from its speakers, presumably why the specific noise that was used was used). While this radio noise wasn't relevant to the story or characters of Spirits of mythos, I really liked the ambience it gave to the environment and this sense of dread. This effect gave me the inspiration for the ambience I wanted to create with the sound effect, which allowed me to experiment with different sounds for this continuous effect.

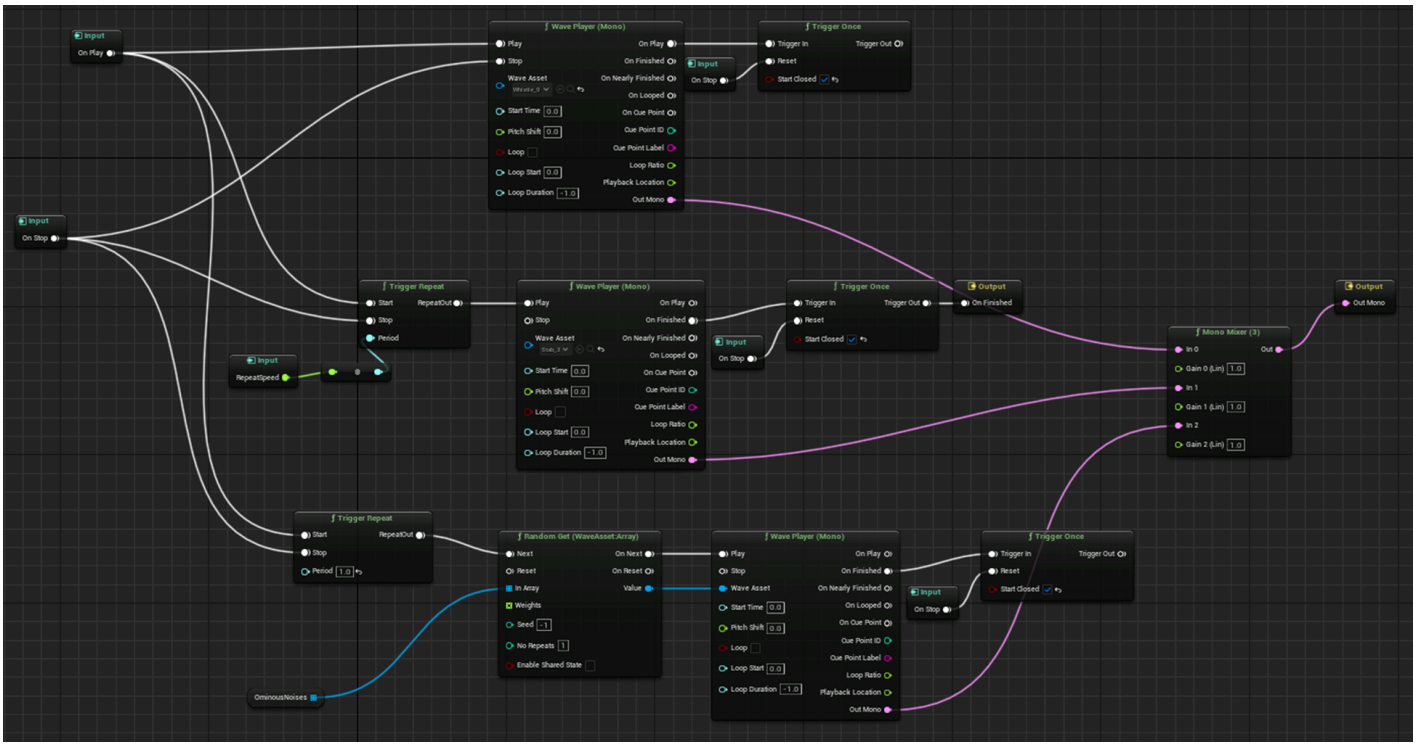
Properties and Values

Property	Description of Purpose	Value
RepeatSpeed	Value used to represent the period of the trigger repeat	(1- DistanceToPlayer / 1000)
Trigger Repeat Period	Used to determined period between repeating sounds for breathing	1.0f
OminousNoises	Array holding different breathing sounds	

ChaseNoiseAttenuation	Shows sound drop over a distance	Inner Radius: 400 Fall Off Distance: 3600

MetaSound Diagram

Entire MetaSoundDiagram



Whistle (Annotated)

Tells all nodes when to play, controlled in c++, called when AI senses player

Plays the single droning whistle declared in the wave asset section

Tells the sound effect to end

Triggers the On finished Output only one time, but can be reset anytime

Clangs (Annotated)

Tells all nodes when to stop, controlled in c++, called when AI no longer senses player

Used to repeat the sound at a determined interval, set by the RepeatSpeed in the code

Triggers the On finished Output only one time, but can be reset anytime

Notifies the code the sound effect has finished

Speed retrieved from the code, determined by the distance of the AI from the player

Plays the clang sound declared under the wave asset

Tells the sound effect to end

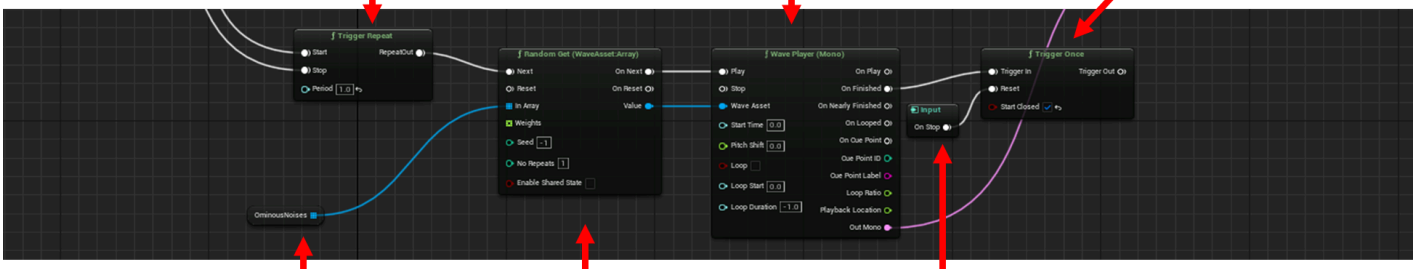
All Out Monos are attached to a mono mixer, which combines them so that they can be output

Breaths (Annotated)

Used to repeat the sound at a period of 1.0

Outputs the randomly selected sound from Random Get

Triggers the On finished Output only one time, but can be reset anytime



An array of four breathing noises to be selected from

Selects a random asset to be played from the OminousNoises array

Tells the sound effect to end

Appendix

AI Generated Assets
MainMenuImage.ui
“Woman ghost in dungeon”
Stable Diffusion, on NightCafe
SDv1.5

Sound Assets - All sourced from https://sonniss.com/gameaudiogdc
Fire Projectile Noise
"CHEMICAL ACID Sizzle, Burn, Short 02"
Magic Elements Vol 2
Articulated Sounds

Whistle
"Whistle 9"
Conspiracy Theory
344 Audio

OminousNoisesArray
"HUMAN BREATH Male_ Sleepy Yawn_E"
Human Male Breathe
Articulated Sounds

OminousNoisesArray
"HUMAN BREATH Male_Deep Mouth Inhale and Powerful Strong Exhale with Acceleration_A"
Human Male Breathe
Articulated Sounds

OminousNoisesArray
"HUMAN BREATH Male_Deep Opened Mouth Breath with S Formed Lips Long Exhale_F"
Human Male Breathe
Articulated Sounds

OminousNoisesArray
"HUMAN BREATH Male_Mouth Inhale and Exhale Like Got Frightened Intermittent _C"
Human Male Breathe
Articulated Sounds

Stab
"Stab 3"
Conspiracy Theory
344 Audio