

# Technical Design Document

**Teixos**

William John Lautama

Project Overview	5
Core Mechanic Overview	5
Target Platform	5
Game Mechanics	6
Gameplay Flowchart	6
UML Diagram	9
Movement Mechanics	10
Controls	11
Core Gameplay Mechanic	12
Mechanic Overview	12
Mechanic Description / Functionality	12
Sequence Diagram	15
UI Design	15
Main Menu UI	15
UI Overview	15
UI Description / Functionality	15
UI Wireframe	16
In-Game UI	18
UI Overview	18
UI Description / Functionality	18
UI Wireframe	19
Dynamic Materials	20
Dynamic Material 1 - NoiseFlow	20
Overview of Effect	20
Effect Description	20
Inspiration / Reference Images:	20
In-Engine Screenshots:	21
Properties and Values	22
Node Graph	23
Dynamic Material 2 - Water	26
Overview of Effect	26
Effect Description	26
Inspiration / Reference Images:	27

In-Engine Screenshots:	28
Properties and Values	29
Node Graph	30
Dynamic Material 3 - Glow	36
Overview of Effect	36
Effect Description	37
Inspiration / Reference Images:	37
In-Engine Screenshots:	38
Properties and Values	38
Node Graph	39
Dynamic Material 4 - Ground	42
Overview of Effect	42
Effect Description	42
Inspiration / Reference Images:	42
In-Engine Screenshots:	43
Properties and Values	44
Node Graph	44
Physics	47
Overview of Interaction	47
Interaction Description	47
How the Interaction Works	47
Inspiration / Reference Images	48
In-Engine Screenshots	49
Properties and Values	51
Artificial Intelligence	53
Overview of AI	53
AI Description	53
AI Abilities	53
Inputs & Senses	54
AI Senses	54
Blackboard Values	54
Behaviour Tree Graph	56
Niagara Particles	57

Niagara Particle Effect 1 - BlackHole	57
Overview of Effect	57
Effect Description	57
Inspiration / Reference Images:	57
In-Engine Screenshots:	58
Properties and Values	59
BlackHole_Core Emitter	59
Core_Particles Emitter	61
Ring_Particles Emitter	63
Swirl Emitter	65
Niagara System / Emitters Breakdown	67
BlackHole_Core	68
Core_Particles	68
Ring_Particles	69
Niagara Particle Effect 2 - RockTrail	69
Overview of Effect	69
Effect Description	70
Inspiration / Reference Images:	70
In-Engine Screenshots:	71
Properties and Values	71
Main_Rock_Trail Emitter	71
Small-Rock_Pieces Emitter	73
Niagara System / Emitters Breakdown	78
Main_Rock_Trail	79
Small_Rock_Pieces	79
Sequencing / Cinematic	79
Overview of Sequence	79
Camera Angles / Properties	80
Scripted Events	89
Storyboard	90
MetaSound	97
Overview of Sound Effect	97
Effect Description	97

Inspiration / Reference:	98
Properties and Values	98
MetaSound Diagram	99

## **Project Overview**

The game focuses on a wall running mechanic combined with high speed traversal in a parkour setting. The aim is to implement this core mechanic efficiently and make it engaging for the player. The game is being developed for PC as it allows for smoother execution of the wall running mechanic with the keyboard, and the FPS format is easier to control with a keyboard and mouse compared to consoles. This aims to provide a comfortable gaming experience for players.

## **Core Mechanic Overview**

The core mechanic of my game is the speedy wall running mechanic. A wall running mechanic is not a new concept in the games world but every game that has a wall running mechanic implements the wall run mechanic differently. A wall run mechanic is not easy to implement smoothly and be presented to the players in a good way, that's why I want to focus on implementing it with great efficiency and actually makes the player want to play the game based on the core mechanic.

The other thing that is in my core mechanic is speed. Since the basic idea of my game is a parkour game and the prompt that I chose is "A device that allows for fast traversal." This leads to the player having fast running capabilities that could lead to an engaging experience with the provided levels as well.

## **Target Platform**

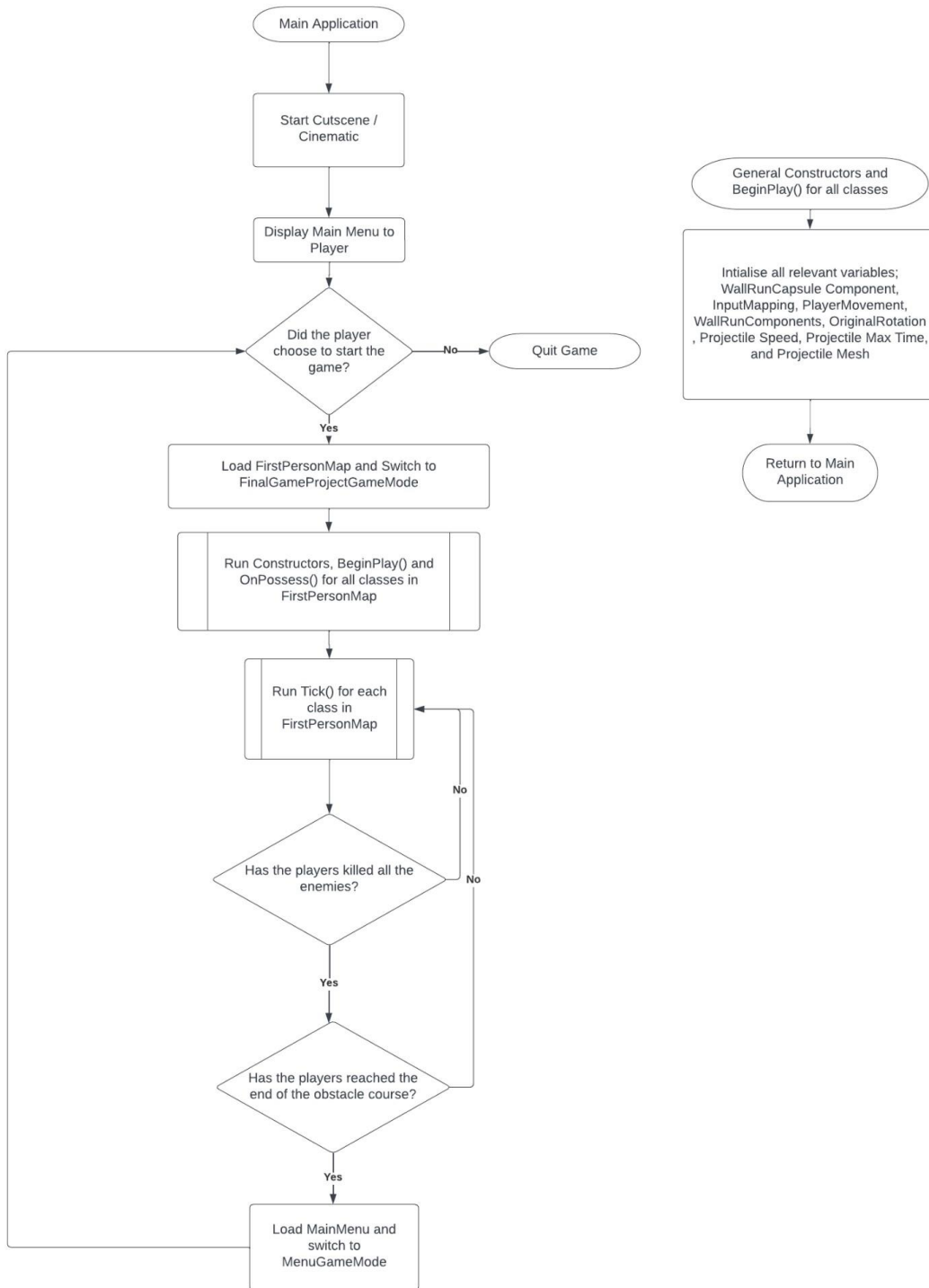
The platform that I am making this game for is PC. This is due to the fact that with pc, and the keyboard that are attached, the wall running mechanic can run smoother since it is very focused on pressing multiple keys on the keyboard at the same time. While this effect can be achieved in other consoles, it is harder to do so, especially on handheld and consoles that rely on the analog stick for movement.

Since this game is using the FPS format, it is also easier to move on PC rather than in console since the player is using a keyboard just for controls and a mouse solely for camera movement rather than the right analog stick for some consoles. This can be seen by the comparison of the amount of people who play games on the FPS genre on the PC and on consoles. This could lead to a more comfortable gaming experience for the players overall.

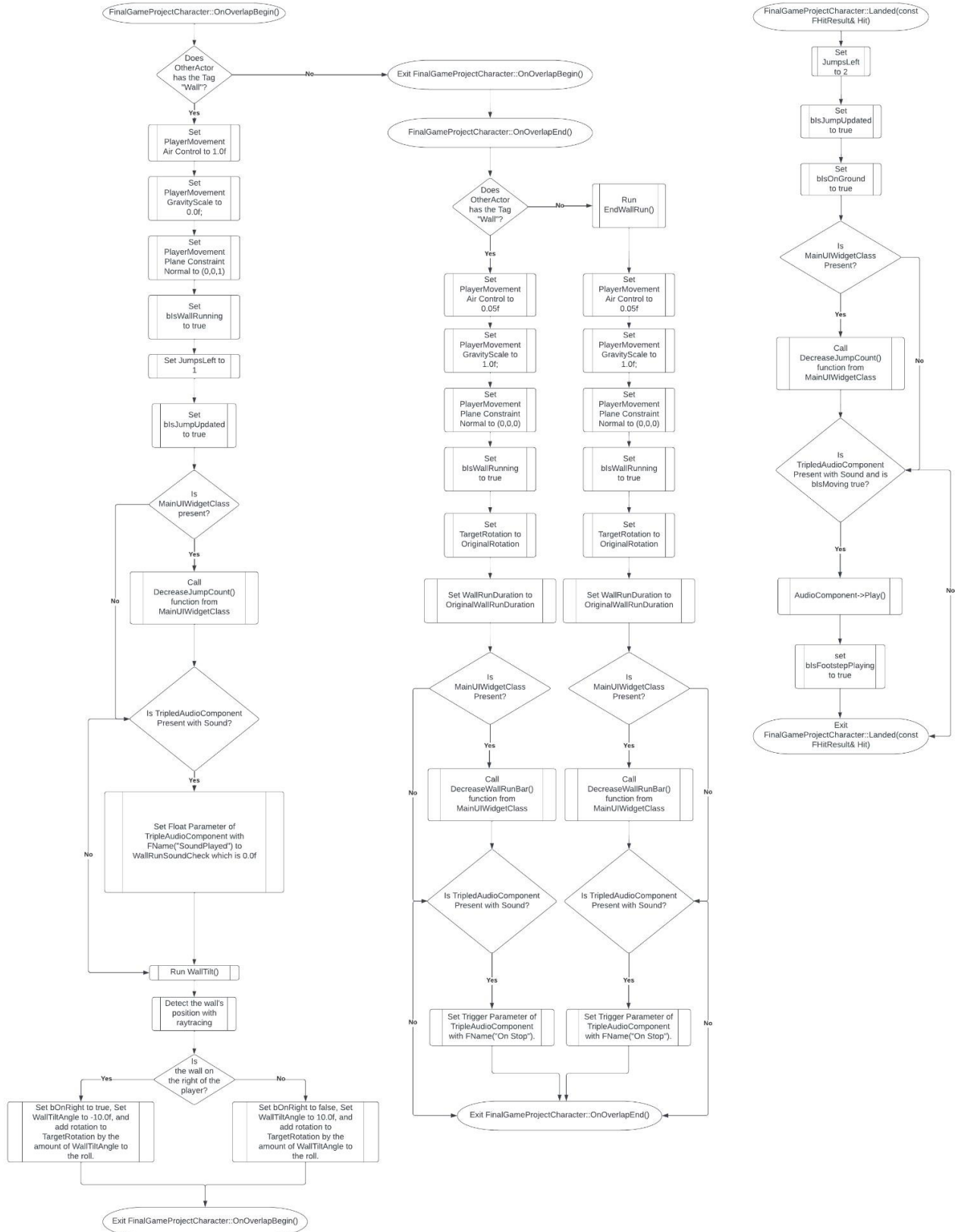
# Game Mechanics

## Gameplay Flowchart

Flow Chart 1: Overall Game Logic



Flow Chart 2: Player Collision with Walls and Landing





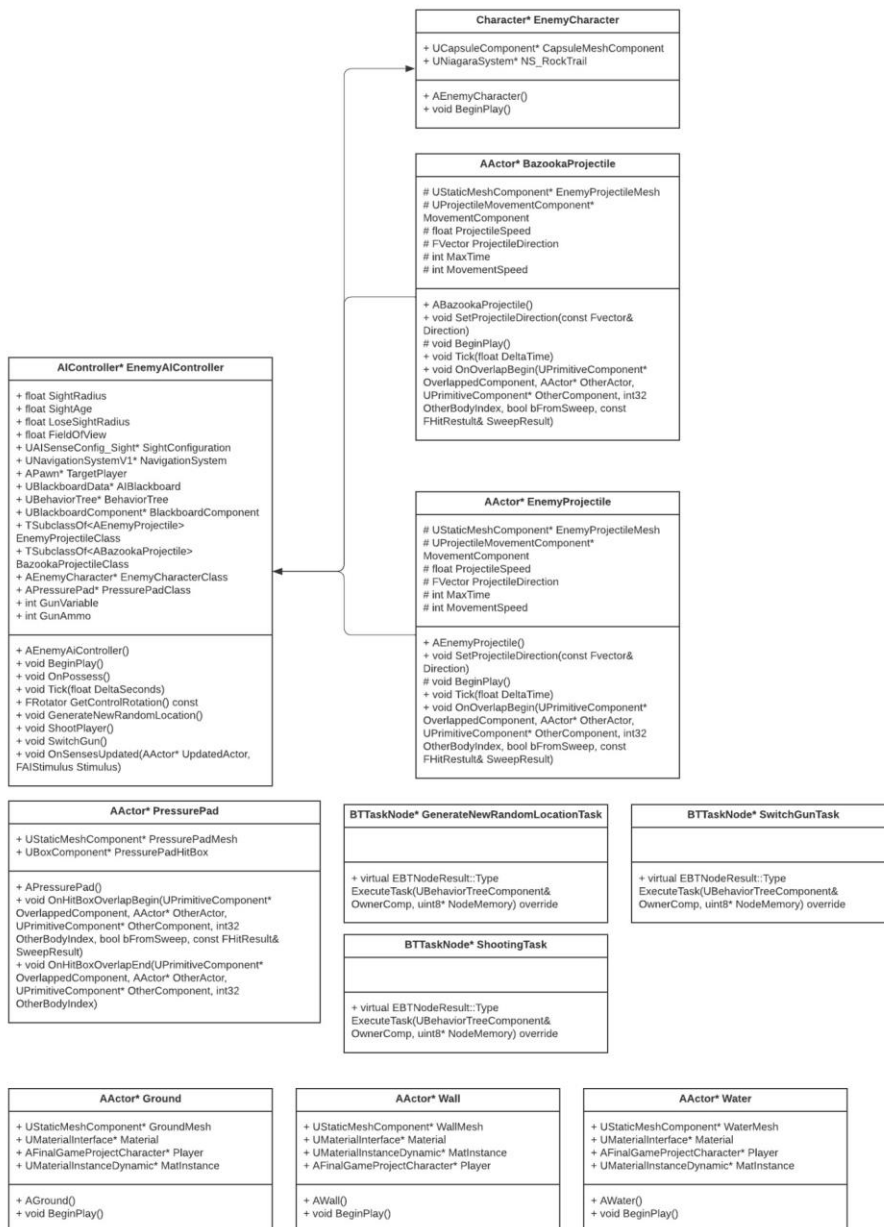
Flow Chart 3: FinalGameProjectCharacter Tick() Function



# UML Diagram

UML Class Diagram





## Movement Mechanics

The camera movement of the player follows the player's mouse cursor. This movement was provided by the fps template itself. For the movement mechanics, the player is able to do basic movements and also jump movements. The basic movements are the player moving to the right, left, forward, and backwards. This movement was done through the fps template. Even though the jump function is also implemented in the first person template, I decided to implement my own jump mechanic. The function that I used for my jump mechanic is CallJump(). I initialise the variable PlayerVector and in the CallJump() function, I set the PlayerVector based on where the walls are when the player wall runs (left of the player or right of the player). The usage of the PlayerVector can be seen from when the player actually jumps.

Before I mention the usage of PlayerVector, it is worth mentioning that there is an if statement before it using the variable JumpsLeft to limit the amount of jumps the player is able to take. There is also another if statement to determine if the player is wall running or not. If the player is wall running, rather than making

the player jump upwards, the player jumps the opposite side of the wall. For example, if the player is wall running with the wall on the right side of the player and then jumps, the player will jump towards the upward left side of the player. Then there's also an else statement in the end of the if statement that allows the player to jump normally. After all of this happens, the variable JumpsLeft then gets decreased by 1. For the JumpsLeft variable it starts with 2 and will reset to 2 every time the player lands or ends the wall run since I want the player to double jump every time.

There is also a sprint movement that the player is able to do when the player holds shift. This makes the player move faster than the normal walking method. There are some limitations on the movement, with the jumping movement, it could be better by allowing the player to jump towards anywhere the player is facing instead of the preset jump methods. The other thing that limits the movements are the collisions in the game. There are a lot of objects and walls in the game that could limit the movement. This could also be a potential gameplay feature that stops the player from continuing the level.

## Controls

Mapping	Action	Description	Keybind	Modifiers
LookMapping	Look Around	Used to move the character's camera based on the mouse cursor	Mouse Movements	Modify ControllerYaw and ControllerPitch input
WalkingMapping	Move Forward	Used to move the character forward	W	SwizzleAxisValue
WalkingMapping	Move Backwards	Used to move the character backwards	S	SwizzleAxisValue, Negate
WalkingMapping	Move to the Right	Used to move the character to the right	D	Change ActorRightVector
WalkingMapping	Move to the Left	Used to move the character to the right	A	Change ActorRightVector, Negative

JumpMapping	Jump	Used to make the player jumps depends on which state the player is in (Wall Running or Normal)	Space Bar	
SprintMapping	Sprint	Used to make the player walk faster (sprint)	Shift	
FireMapping	Shoot	Used to shoot projectiles	Left Mouse Button	
SlowDownTimeMapping	Slow Down Time	Used to slow down time	Q	
SwitchGun1	Switch To Bouncy Gun	Used to Switch To Bouncy Gun When Shooting	1	
SwitchGun2	Switch To Black Hole Gun	Used to Switch To Black Hole Gun When Shooting	2	
SwitchGun3	Switch To Dual Gun	Used to Switch To Dual Gun When Shooting	3	

## Core Gameplay Mechanic

### Mechanic Overview

My core mechanic in the game is the Wall Run Mechanic. The mechanic allows the player to wall run on every wall smoothly. This mechanic is still in the early stages of development so it could be improved. The player is also able to double jump and jump from the walls to other walls to continue wall running. The player's sprinting mechanic also affects the wall run mechanic and it can affect the pacing of the game.

### Mechanic Description / Functionality

There are a couple of ways to implement the wall run mechanic but I am going to describe the method on how I did it. So firstly, I initialised an extra capsule component and resized it so that it is bigger than the

player capsule collider. Then I made an `OnOverlapBegin()` function and an `OnOverlapEnd()` function for the Capsule

Component. What the `OnOverlapBegin()` function does is that it checks if the `OtherActor` that it is colliding with has the tag "Wall". If it does, then I modified the `PlayerMovement` in 3 different ways :

- `PlayerMovement->AirControl = 1.0f;`

This code allows the player to have full control over the movements while in the air (in this case while wall running)

- `PlayerMovement->GravityScale = 0.0f;`

This code allows the player to have no gravity while doing the wall run. This code is implemented to prevent the player from falling over when doing the wall run. - `PlayerMovement-`

`>SetPlaneConstraintNormal(FVector(0,0,1));`

This code is called to constraint the player from moving up and down. This will make it feel like the player is walking on the flat ground. - `bIsWallRunning = true;`

This code is called to change the player state to the player currently wall running. - `JumpsLeft = 1;`

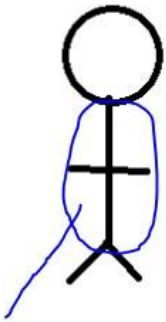
This code is called to reset the jump mechanic to just 1 because the double jump reset only happens when the player lands.

After those lines of code are called, the function `WallTilt()` is called and in that function, with raycasting, the player will be able to tell which side the wall is on when the player is running. Basically, there is a line that will be called every time that `WallTilt()` is called. The line will be located on the right side of the player and if the line is colliding with something that means, the wall is on the right side of the player otherwise if the line is not colliding then the wall is on the left side of the player. This is done to set the variable of `WallTiltAngle` to the appropriate variables to prepare the player to tilt the camera everytime the player does a wall run. From the `WallTiltAngle` variable, a variable called `TargetRotation` is also adjusted and set based on the `WallTiltAngle` variable. This then will affect the code in the `Tick()` function and using `AddControllerRollInput`, the camera will be rotated until it reaches the `TargetRotation` which is the tilted values from before.

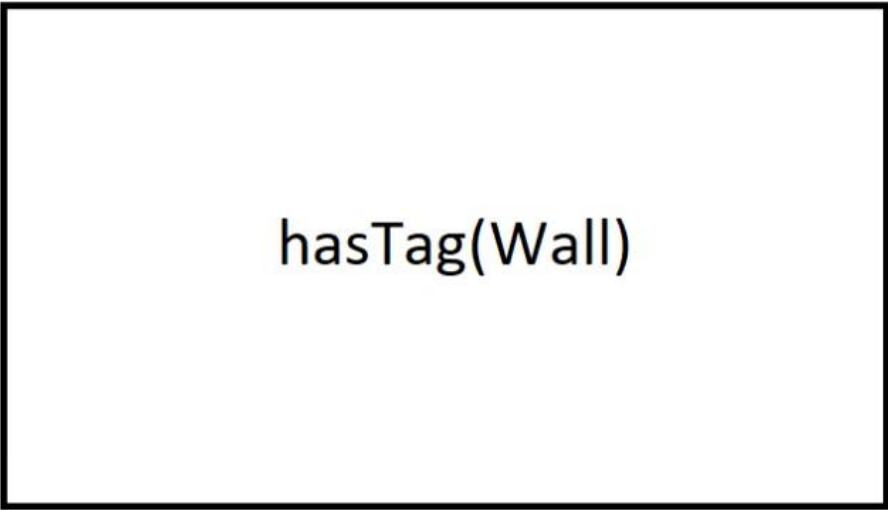
On the `OnOverlapEnd` function, the `AirControl`, `GravityScale`, and `PlaneConstraint` of the player is set to the default values since the player is not wall running anymore. The boolean variable `bIsWallRunning` is also set to false and the `TargetRotation` is set to the `OriginalRotation` variable which was called in the constructor to rotate the player's camera back to the original position using the method in the `Tick()` function.

There is also a limited wall run duration mechanic that allows the player to wall run with a limited amount of time. The variable `WallRunDuration` is called first and set as a number then when the player is wall running, the if statement in the `Tick()` function is called and decreases the value of `WallRunDuration` by 1 each tick.

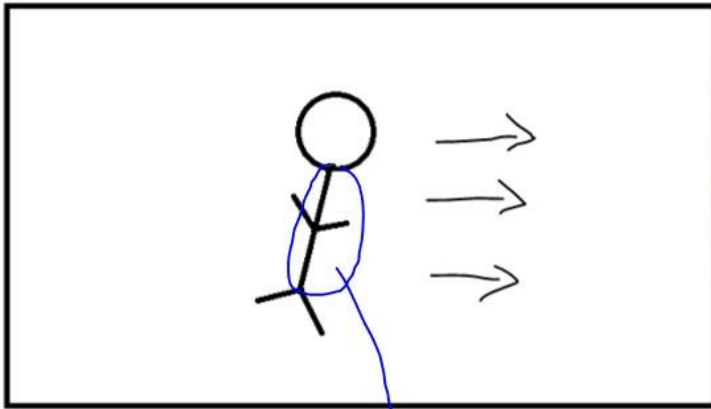
Player



Hitbox



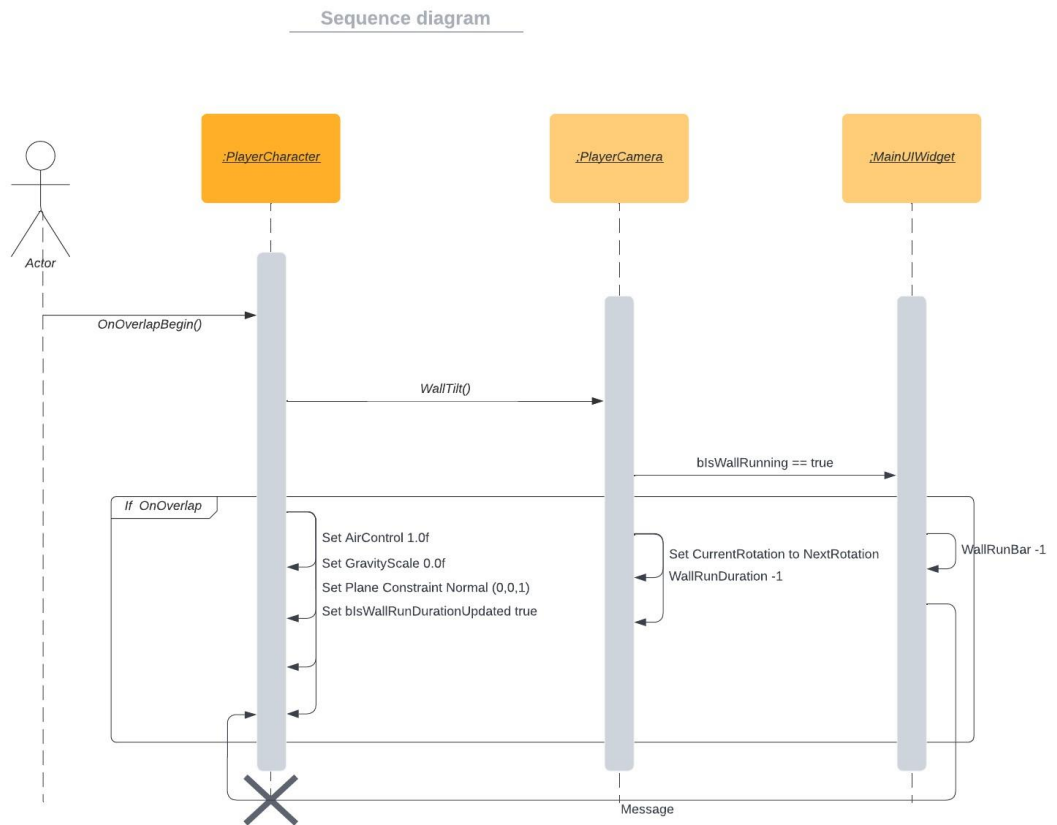
hasTag(Wall)



Overlap

```
PlayerMovement->AirControl = 1.0f  
PlayerMovement->GravityScale = 0.0f  
PlayerMovement->  
SetPlaneConstraintNormal(FVector(0,0,1))  
bIsWallRunning = true  
JumpsLeft = 2
```

# Sequence Diagram



## UI Design

### Main Menu UI

#### UI Overview

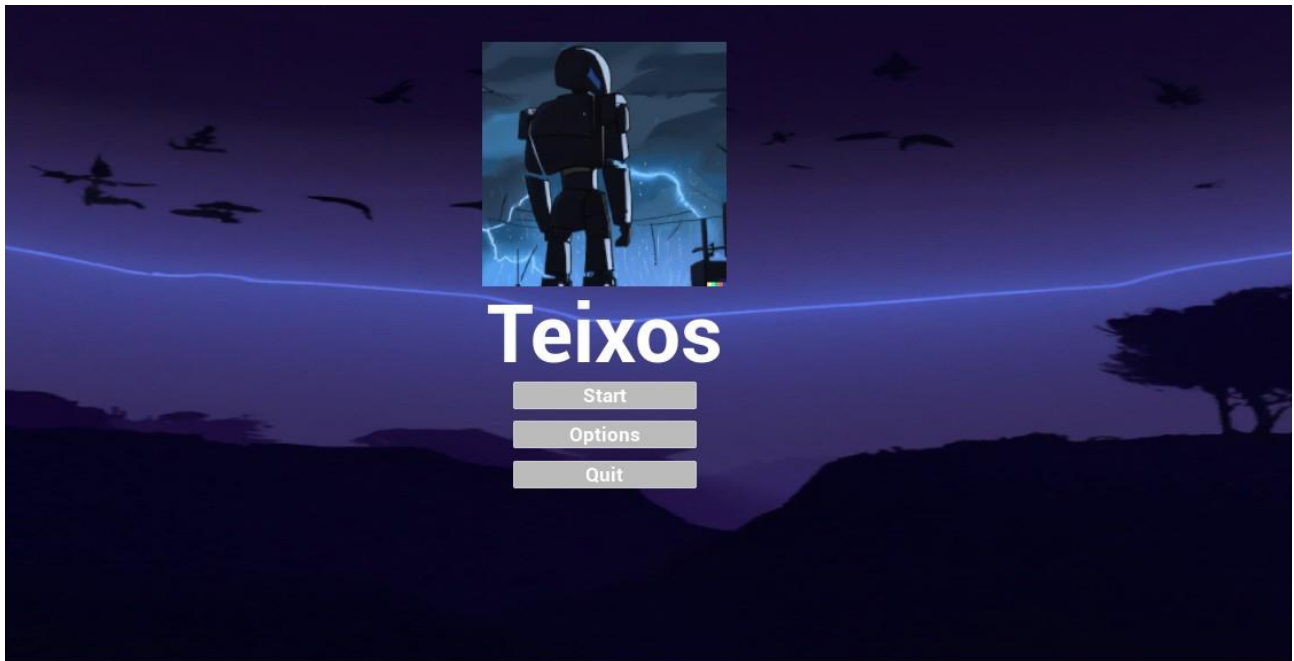
In my main menu UI, the buttons and the functionality of the menu is very basic. Main menu in a video game is very crucial since it marks the start of the playthrough of each player. A good main menu is really important to grab the player's attention and to make the player motivated to start playing the game. The interesting aspect of a main menu can be achieved by the style of the button, the background images or video, the title, etc.

#### UI Description / Functionality

The main menu that I made in my game is very basic. It has 3 buttons in total, the Start button, the Options button, and the Quit button. The options button hasn't been customised yet so it does nothing at the moment but it is planned to have a volume slider, brightness slider, and the credits of who made the game. The start button transitions the player to another scene which is the first level but in this case it's called the FirstPersonPlayerMap. The quit button quits the game when pressed. Despite being basic, these buttons and functionality serve its purposes so it doesn't matter if it's basic or not. The functionality of these buttons are made in blueprints. The title is also there with it being "Teixos."



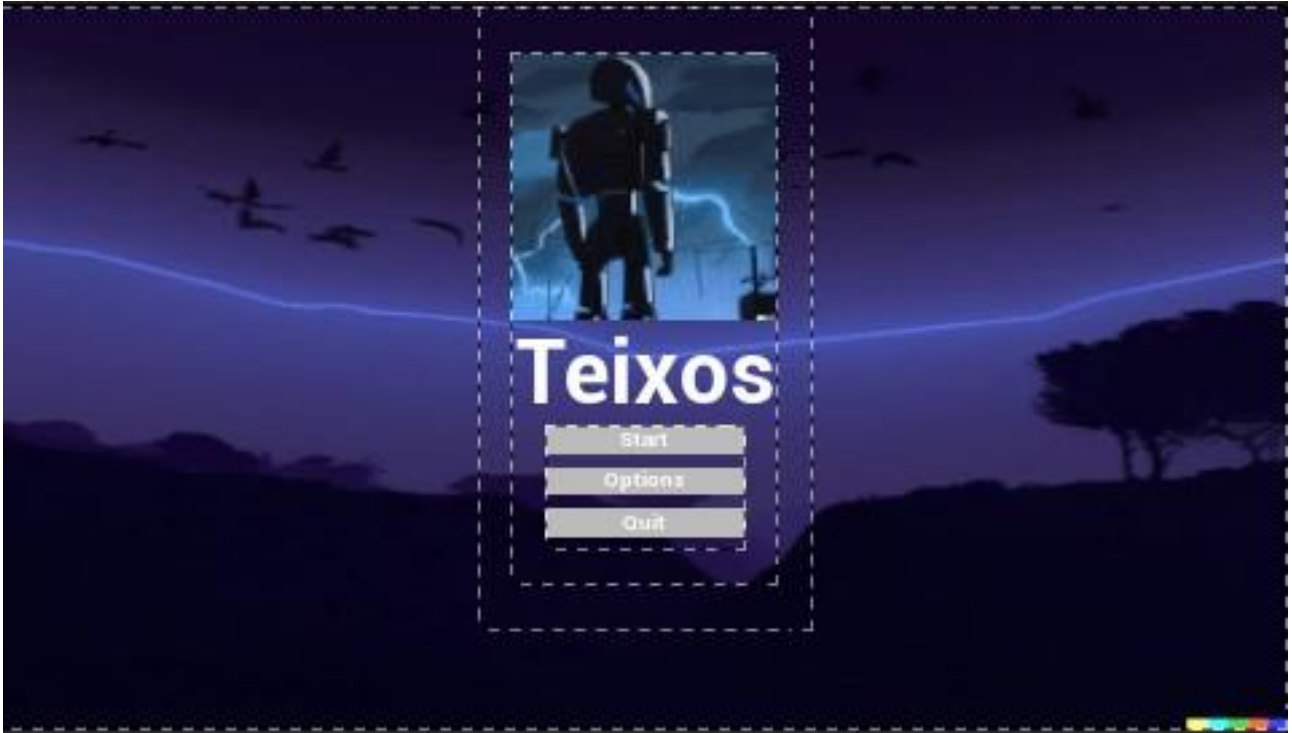
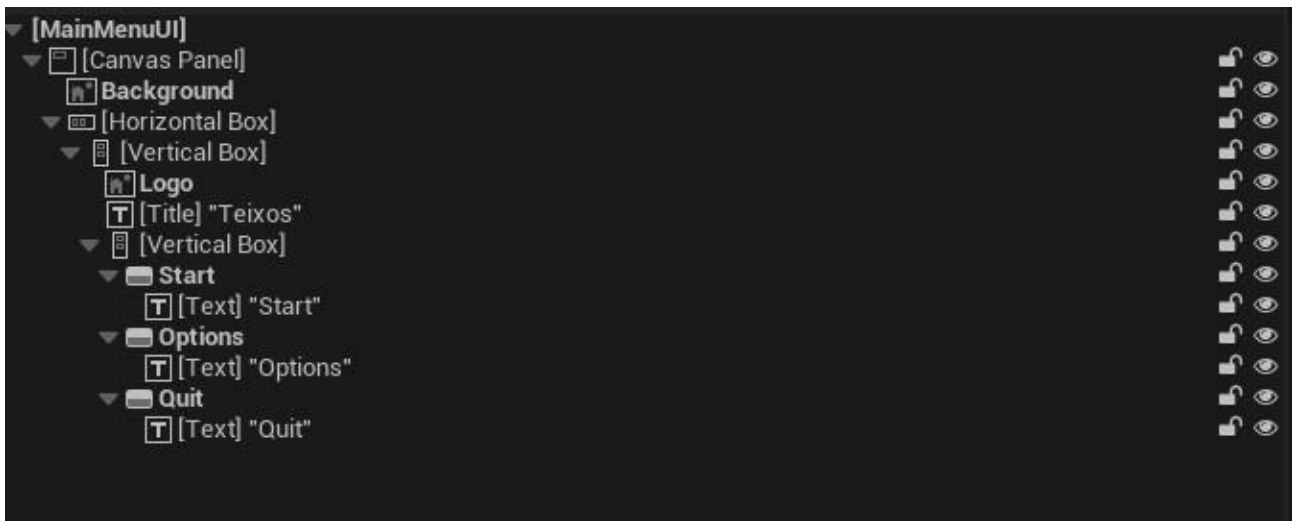
## UI Wireframe



Background Image : Generated by Dall-E Mini 2

Robot Image : Generated by Dall-E Mini 2

The buttons are made using the UI widget functionality made in Widget blueprints.



- The Background image is adjusted with the Anchor so that it fills up the whole screen.
- Horizontal Box : Middle Anchor, Position X = -250.0, Size X = 500.0, Offset Bottom = 150.0
- Vertical Box : Middle Horizontal and Vertical Alignment, Top and Bottom Padding by 40.0
- Logo Image : Fill Horizontal and Vertically
- Title Text : Center Align Horizontally and Vertically
- Vertical Box : Fill Horizontally and Vertically, Left, Right, and Bottom Padding by 50.0
- Start Button : Fill Horizontally and Vertically, Bottom Padding by 20.0
- Start Text : Center Align Horizontally and Vertically, Left and Right Padding by 4.0 and Top and Bottom Padding by 2.0
- Options Button : Fill Horizontally and Vertically, Bottom Padding by 20.0
- Options Text : Center Align Horizontally and Vertically, Left and Right Padding by 4.0 and Top and Bottom Padding by 2.0
- Quit Button : Fill Horizontally and Vertically, Bottom Padding by 20.0

- Quit Text : Center Align Horizontally and Vertically, Left and Right Padding by 4.0 and Top and Bottom Padding by 2.0

-

## **In-Game UI**

### **UI Overview**

The In-Game UI that I have in my game is pretty simple but incredibly useful to the players. It is also worth mentioning that In-Game UI needs to be minimal in size but still visible clearly to the players. This is to decrease cluttering in the player's screen and to keep the player's comfortability of camera viewing.

### **UI Description / Functionality**

I have a total of 4 in-game UI in my scene which are : Health bar, jump count, wall run bar, and a switching gun UI. The health bar is very self explanatory which what it does is gives the player a health amount and decreases when the player gets hurt. A method of the player getting hurt has not yet been implemented since there are no enemies yet but it is going to be planned in the future. For the coding side of it, I used `BindWidget` with class `UProgressBar` for the HealthBar, `UTextBlock` for the Health Label. I also used `NativeConstruct()` to initialise the widget and used the functions that I made to update the values of every tick depending on the values that were set to the HealthBar which is the value Health. After that, I made a widget blueprint with it being the child of the C++ class of the HealthBar class and adjusted the UI design there.

The jump count is a number that indicates the amount of times the player can jump. In the future, this is going to change to some picture or some icons. For example, if I have 2 jumps left that means there are 2 jump icons on the left side of my screen. The Wall run bar is a bar that tells the player how much longer they can wall run. This is very useful and adds constraint to the gameplay so that it adds extra challenge to the player and makes for an interesting gameplay experience. For the coding side of both of the UI, it is very similar to the health bar UI. I used a `UProgressBar` for the WallRunBar and `UTextBlock` for the JumpCountText. I also used `UImage` for the SwitchGunImage and the `UTexture2D` to change the images to my choice of guns. I used `NativeConstruct()` to initialise it and then used the functions that I made to update the UI with the variables which are the WallRunDuration and JumpsLeft. For when changing the gun icon, I used `SetBrushFromTexture` to do it.

## UI Wireframe



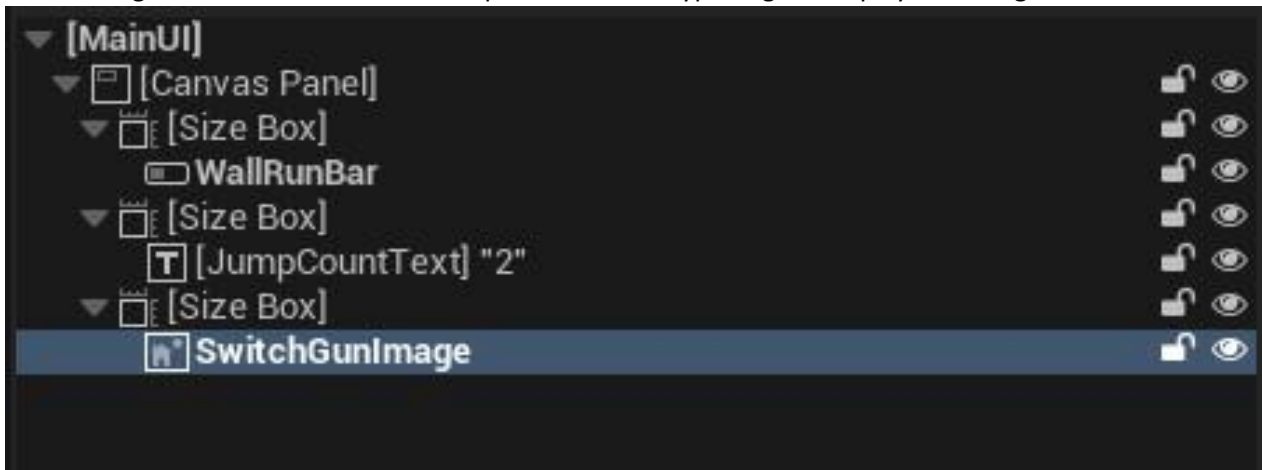
The Red Bar : The Red Bar is the Health Bar

Numbers besides the red bar : The numbers represent the amount of health the player has.

Blue bar : The Blue bar represents how long the player can wall run left. It is currently full since the player is not currently wall running.

Number above the blue bar : The numbers represent the jump counter and tells the player how much jump the player has left. It is 2 currently since the player is on land and has not jumped.

The Icon Right Next to the Number : It represents which type of gun the player is using.



- Size Box for WallRunBar : It is anchored to the bottom, has a -100.0 Position Y, Offset Right of 1000.0, and Size Y of 100.0
- Wall Run Progress Bar : Fill Horizontally and Vertically and Padding on Left and Top with 20.0
- Size Box for JumpCountText : It is anchored to the bottom left, Position X of 20.0, Position Y of -200.0, Size X and Y of 100.0
- Jump Count Text : It is Left Align Horizontally and Bottom Align Vertically.
- Size Box for Switch Gun Image : It is anchored to the bottom left, Position X of 130.0, Position Y of -200.0, Size X and Y of 100.0
- Switch Gun Image : Fill Horizontally, Bottom Align Vertically

# Dynamic Materials

## Dynamic Material 1 - NoiseFlow

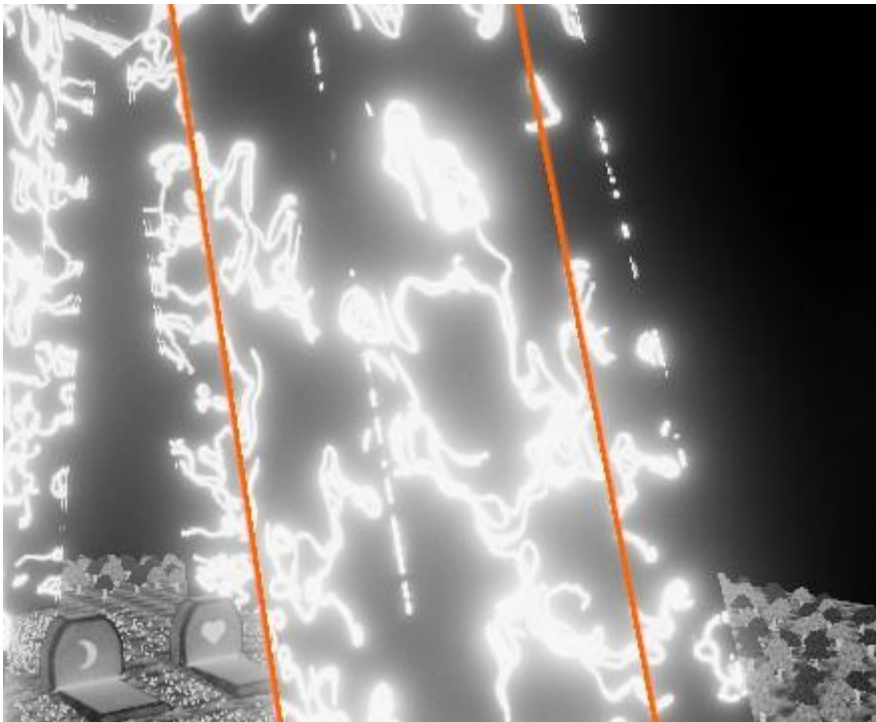
### Overview of Effect

This effect is called NoiseFlow because it is filled with the Noise Node which is just a random veins of colours throughout an object. It also flows through an object that it is implemented in. This object has 2 different colours in 2 different locations of the materials which are the colour red on the outer part of it and the colour blue on the inner part of it. This material is used for the walls that the player will be wall running on. This material wall appears as a sphere mask on the wall and on where the player specifically is. This is done to increase the emotional impact of the wall running and to give the player a sense of satisfaction and aesthetics when they are wall running. It also follows the player which adds the extra impact on it. Since the colour of the material originally is black when the player is not near it, it adds to the contrast of when the player is touching it and when the player is not.

### Effect Description

The effect is just a collection of big chunks of noise that moves in a different direction based on which layer it is. If it is the red part of it which is the outer colour, the movement speed of it is 10.0 by X and 5.0 by Y while the blue part of it which is the inner colour, the movement speed of it is 50.0 by X and 30.0 by Y. I got the inspiration for the effect from watching youtube explanations on nodes and after I found out the noise node exists, I experimented with it. I also just finished my other unit's assignments "Immersive Environment" and I just made a natural environment that consists of just emissiveness so I decided to try and make a really cool material just from emissivity. I also want to limit myself from using custom textures since I realised there's a lot of cool things I can make without using textures in this material.

Inspiration / Reference Images:



This squiggly motive is what I used for my other assignment and after I figured out I could do squiggly things with the noise node, I immediately thought of this.

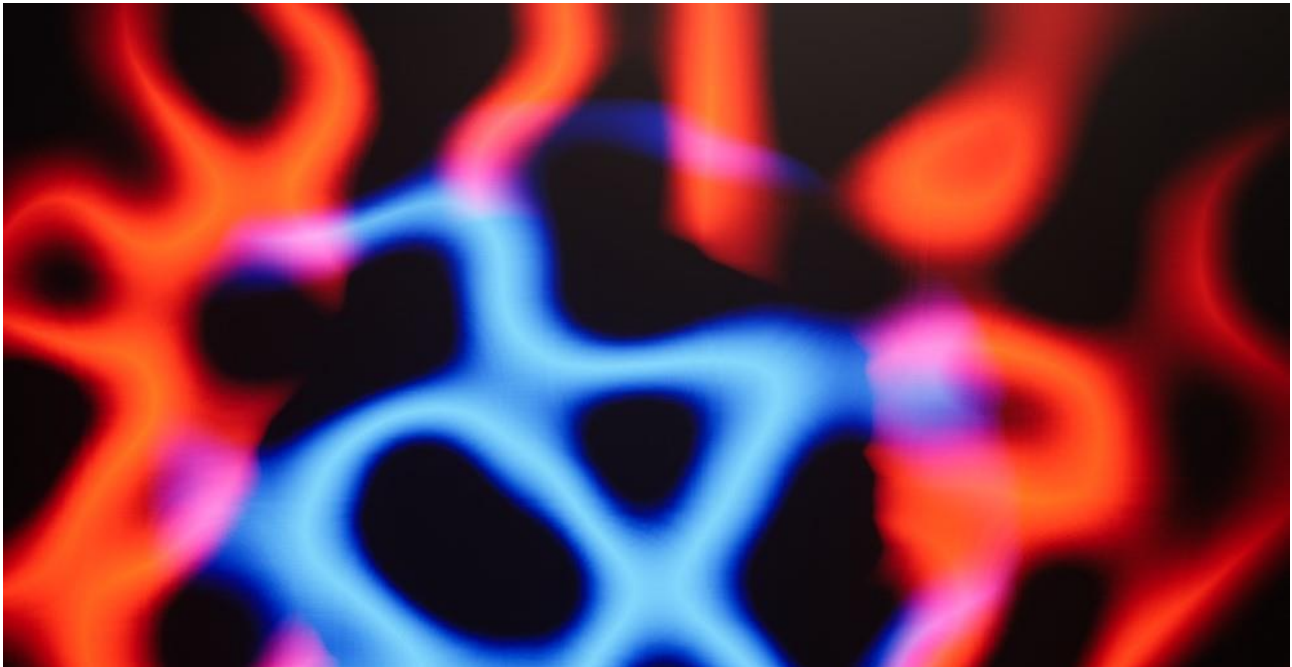


<https://youtu.be/ut80qnOtNRw>

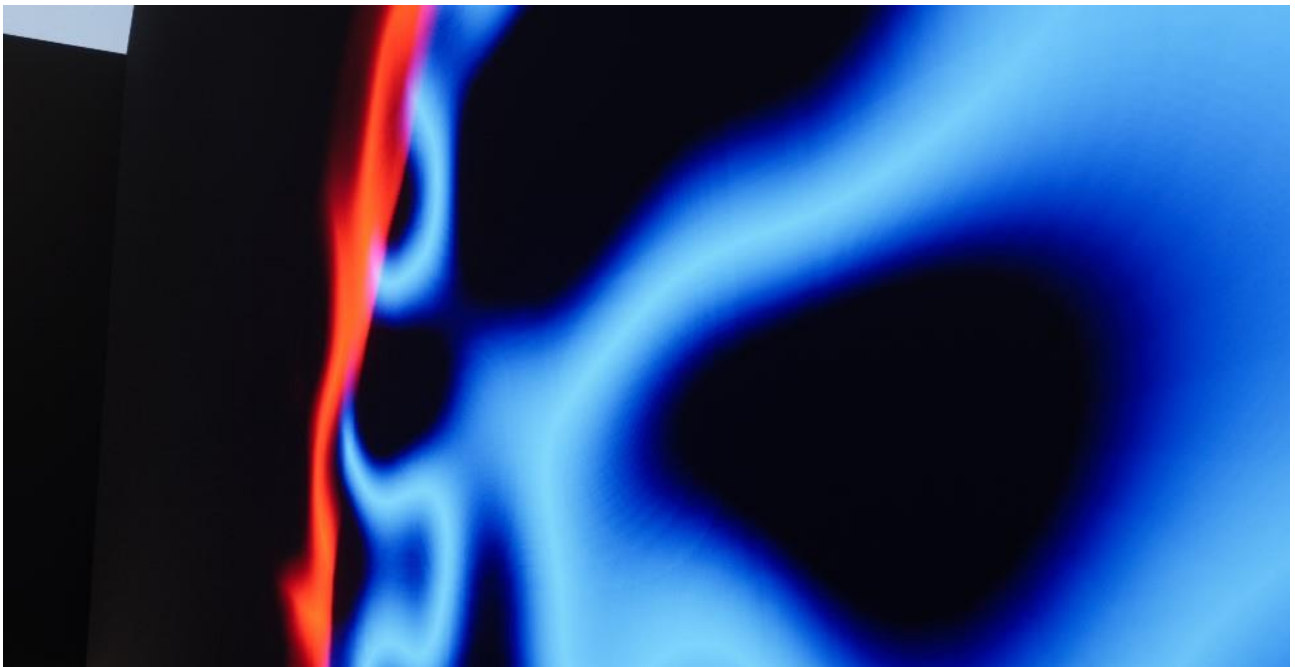
This youtube video helped me in the creation of the material of the noise flow itself.

In-Engine Screenshots:

So basically, as the player goes closer to the wall, the material change will appear. The change depends on how far the player is from the wall. If the player is not close enough, the blue part of the material will not appear clearly as it can be seen from the image below.



But when the player touches it, the blue part will fill the sphere mask near the player and the red part will only be located on the outer part of it.

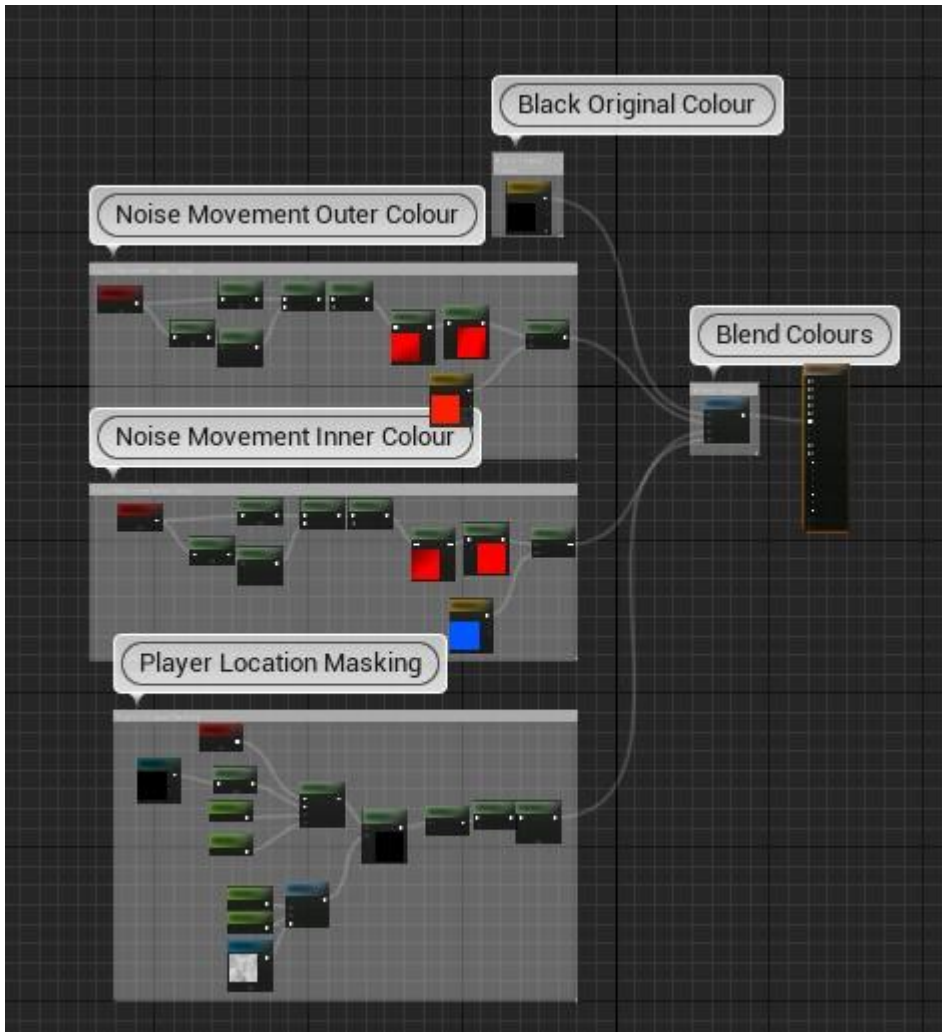


Obviously, it's really hard to showcase the whole effect of when the player is touching the wall since the game is in first person so the player is unable to see the whole sphere mask, but that from the screenshots that has been provided, it is clear enough on how it works without showing the full picture of it.

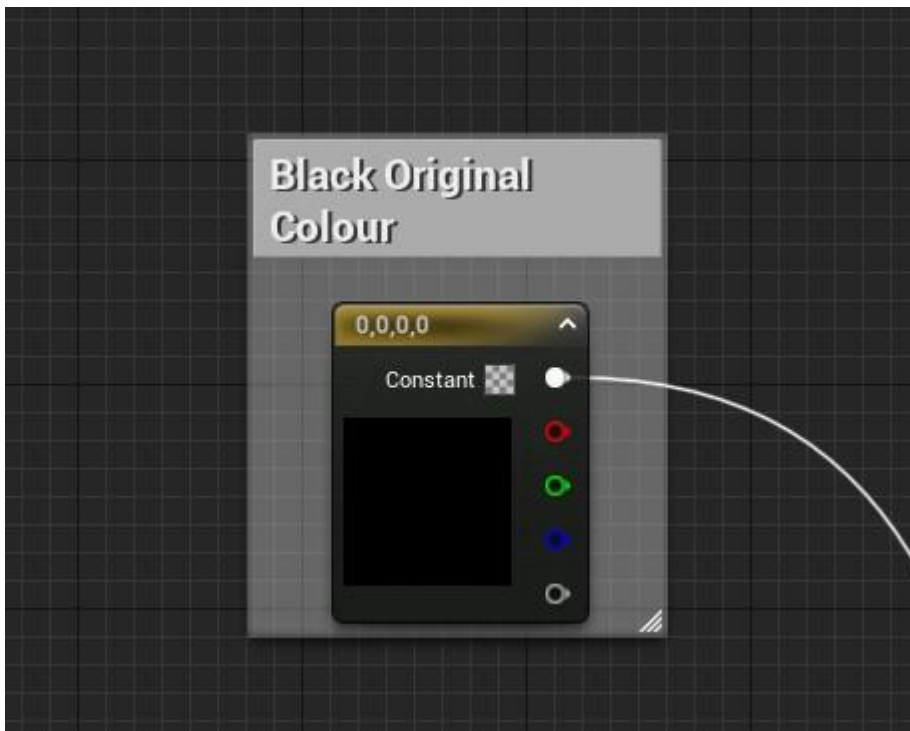
#### Properties and Values

Property	Description of Purpose	Value
MpcCollectionInstance ("Location")	The purpose of the MpcCollectionInstance is to access the Vector variable inside called "Location." It detects where the player is on the map and how close the player is to the materials that use the "Location" variable from the MpcCollectionInstance.	The value of it is dependent on the actor's location.

## Node Graph

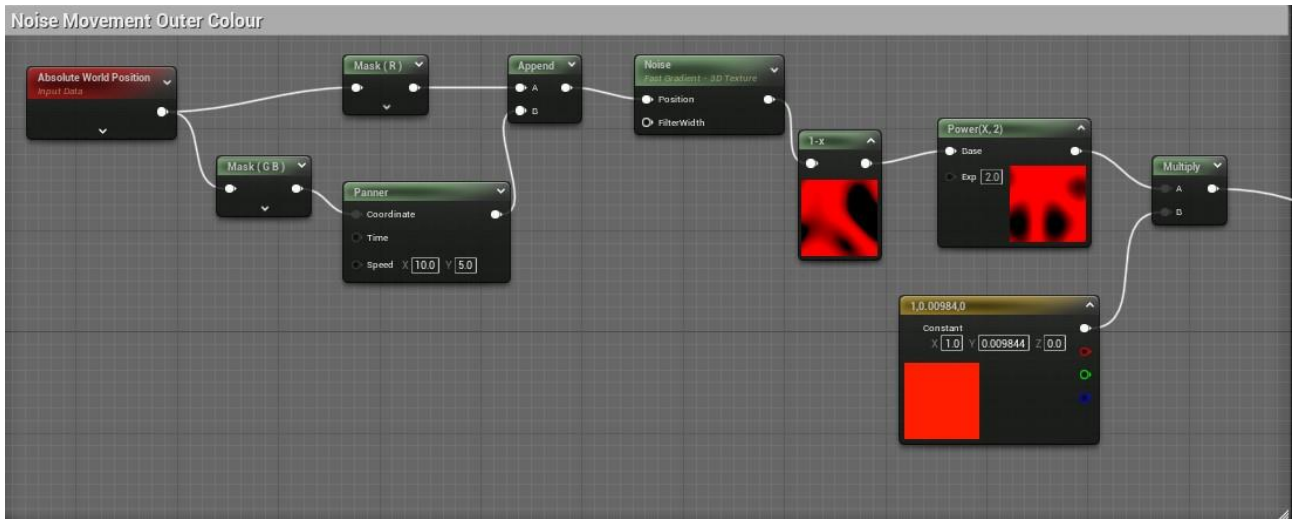


This is all of the grouped material nodes in this material. All of it will be explained individually.



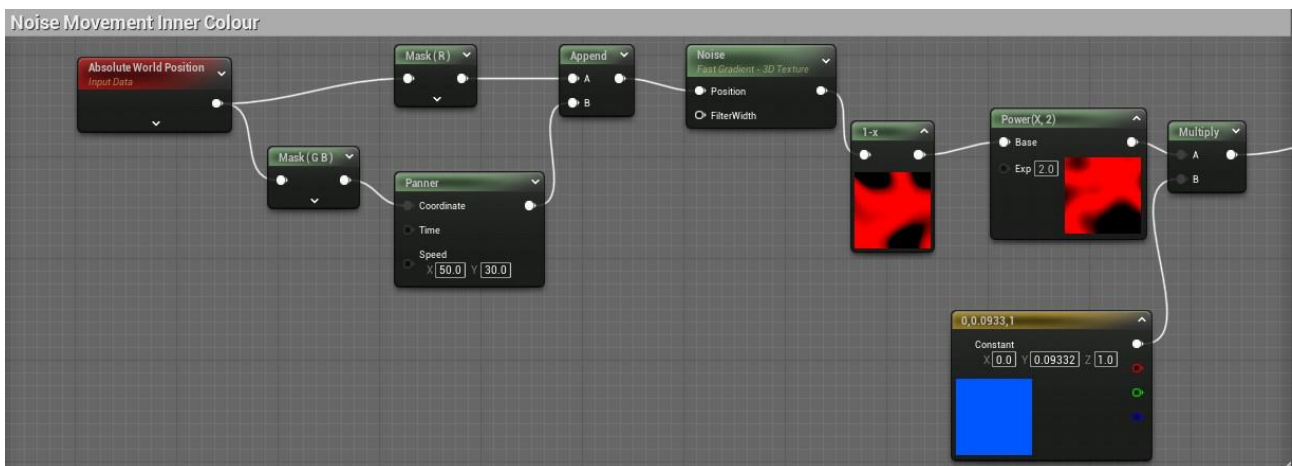
The first node is very self explanatory. It is just a black colour node for the original colour of the material before the interaction.



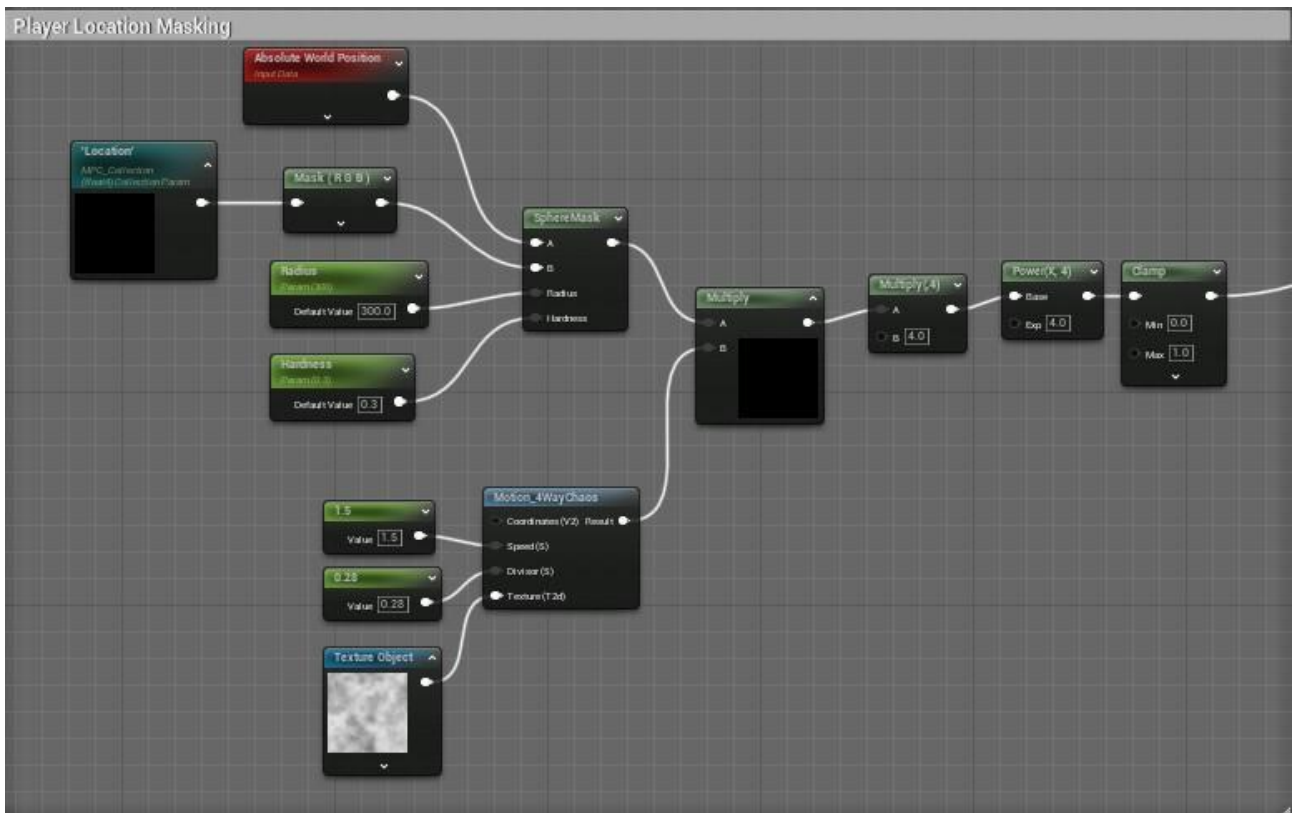


This next one is a bit more complicated. It starts with the Absolute World Position node which outputs the absolute world position of a pixel on a mesh in the scene. It is also divided into 2 RGB masks. The R mask is not adjusted so it just connects to the append node but the G and B mask is adjusted with the Panner which moves the material component with the speed of X 10.0 and Y 5.0. It is then connected to the append node.

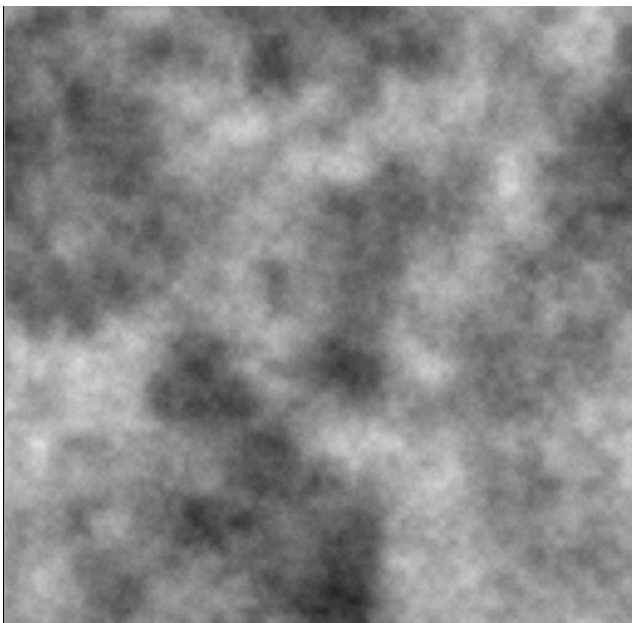
The append node then is connected to the Noise node which displays the veiny and flowy art of it. I also lowered the scale and the levels of the noise so that it looks thick and big unlike the regular small and thin noises. Then, I used the 1-x node to flip the black and the blue colour of the material and then I used the power x2 node to brighten up the colour a little bit. Then I connected it with the multiply node with the red colour which has the values (1.0, 0.009844, 0.0).



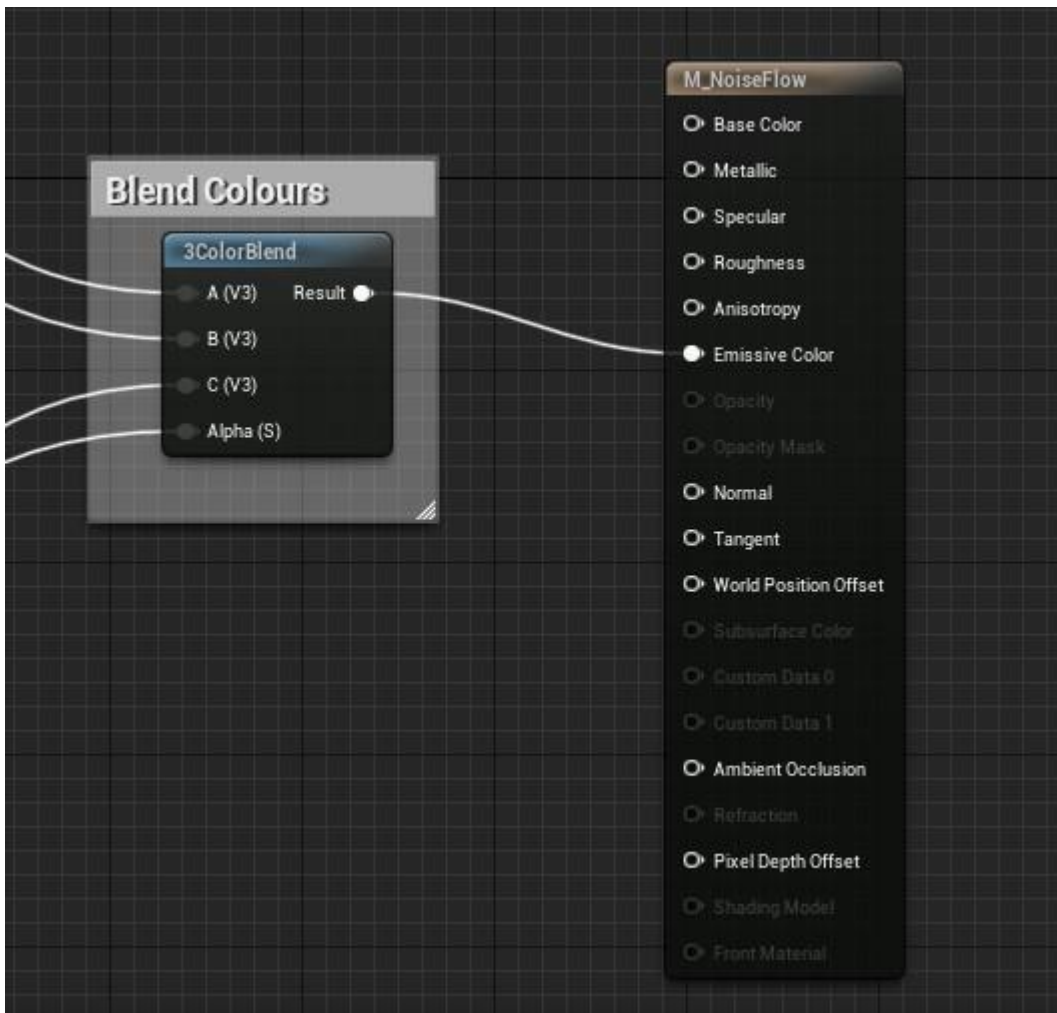
The inner colour part of the material is fairly similar with the outer part. The only difference with it is the paneer speed which is (50.0 X and 30.0 Y) and the blue colour which has the value (0.0, 0.09332, 1.0).



This part of the nodes is a bit trickier than the rest. Basically it takes the value of “Location” from the MPC\_Collection which was adjusted in the code based on the player’s location, then it connects to the RGB mask. It also has an Absolute World Partition and 2 other parameters (300.0 Radius and 0.3 Hardness). All these 4 nodes are then connected to a Sphere Mask which gives a Sphere Mask Dynamic Change to material whenever the player is near the object. The SPHERE mask then is connected to the Multiply Node. The other thing that is connected to the Multiply node is the Motion\_4WayChaos function. Basically what it does is that it makes the outer edges of the mask ripple and it adds this buzzing effect of it similar to a portal. The value of the Speed and Division of it are 1.5 and 0.28. It is also connected to a Texture Object that is provided by the engine which is called T\_Perlin\_Mask (See Image Below).



Then I multiplied it again by 4, added it to the power of 4 and then clamped it to finish off the effect.



Then, I used the 3ColorBlend function and connected the previous nodes to it. I connected the original colour to the A(V3) part, the outer red colour to the B(V3) part, the inner blue colour to the C(V3) part, and then the Player Location Masking to the Alpha(S). Then I connected the 3ColorBlend node to the Emissive Color to make the material pop.

## Dynamic Material 2 - Water

### Overview of Effect

So basically, this material is a stylized single layer water material. Having water in a game is always a cool feature and I added this to make it look cooler. In one of the levels, the player has to fight enemies that are walking on single layer water. This creates a sense of variety of terrain in the game. Ranging from regular looking floors, to water, this keeps the player engaged when playing the game. This makes the parts of the area that has a single layer water in it seem more interesting and different even though the difference is just adding water with the same ground textures. This material is the most complex material that I made since it has the most nodes connected to it. It has ripple effects, spot marks, water murkiness, etc. It also has the player location masking since the water around the player becomes clearer when the player is walking on the water.

### Effect Description

The effect of the water is basically inspired by stylized water in video games. I also stumbled upon a youtube video that explained it perfectly on how to make it and what each node does. I followed the

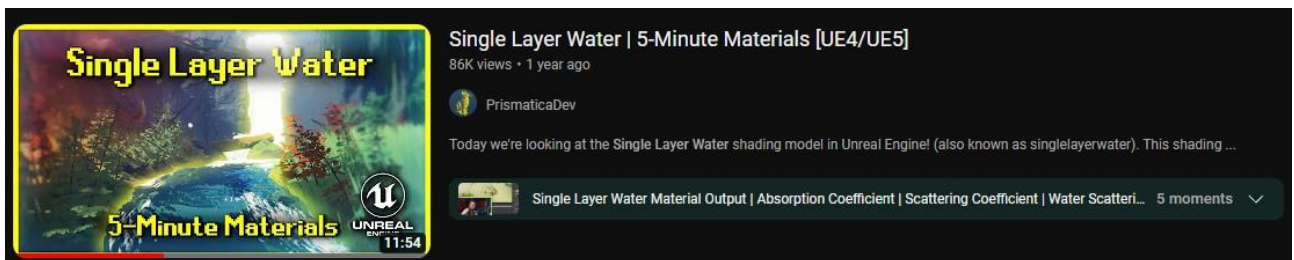
tutorial from the video and I adjusted it accordingly so that it fits better to my game. I didn't want to make the water too realistic since the aim of my game in the first place is not to be realistic. The water has a blue colour overall with a ripple effect, water murkiness, water scatter and opacity spots.

Inspiration / Reference Images:



<https://www.patreon.com/posts/stylized-water-26014793>

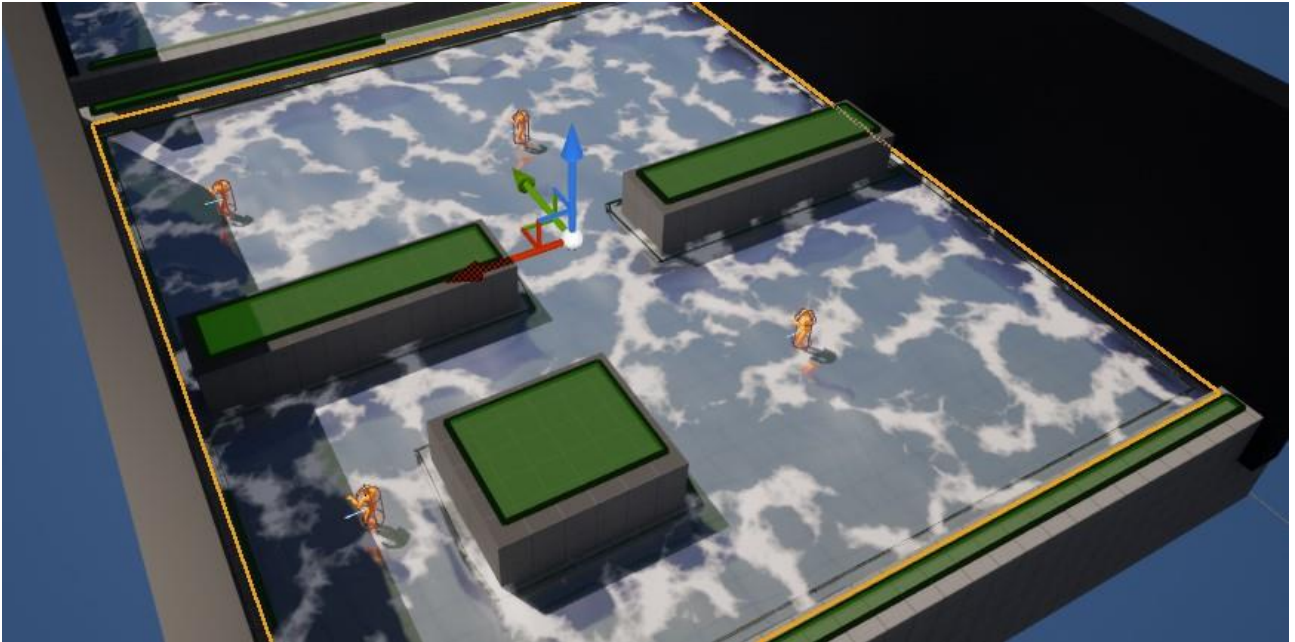
This image helped me with envisioning what a stylized water looks like.



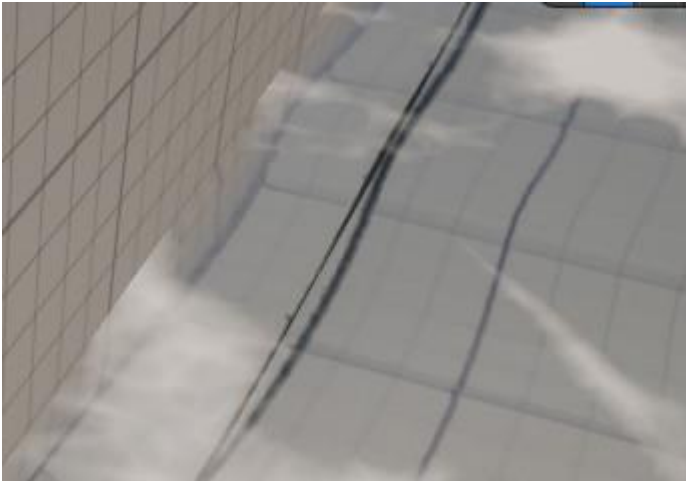
<https://youtu.be/KLI3PZeupFM>

This video helped me with making the single layer water. I adjusted the values and the nodes accordingly so it is more unique and original.

## In-Engine Screenshots:



This is the overview of the material. The material is placed on a plane static mesh since it is a single layer water material. The opacity spots can be seen from the white spots on the water.



As it can be seen, the edges of the objects are a bit squiggly. This is because I added the ripple effect so that the water can cause a ripple effect on the edges of the objects.

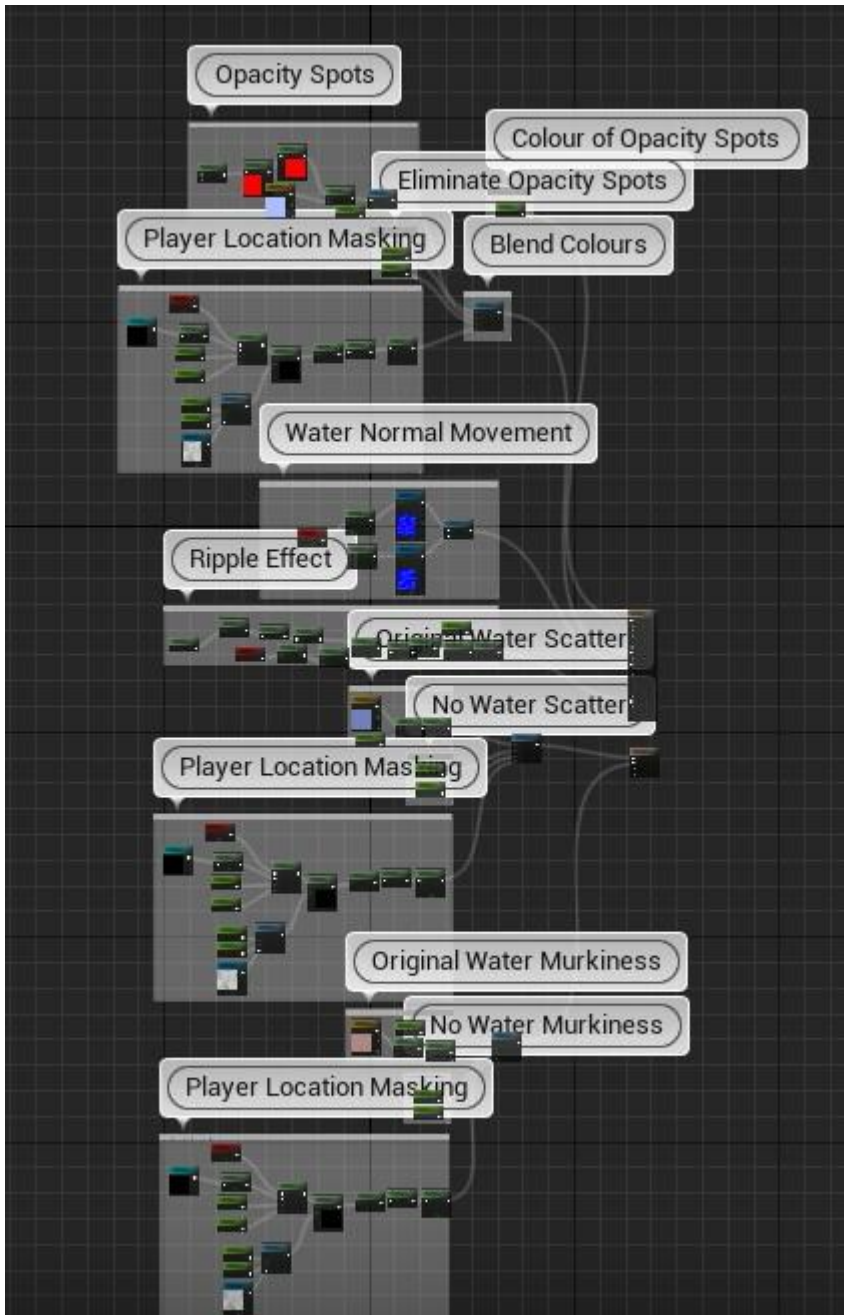


When the player is inside the water layer, it can be seen that there is a circular area around the player that makes the water clearer. It is hard to look at the circle directly but by using the image that I presented the full image can be grasped easily.

#### Properties and Values

Property	Description of Purpose	Value
MpcCollectionInstance ("Location")	The purpose of the MpcCollectionInstance is to access the Vector variable inside called "Location." It detects where the player is on the map and how close the player is to the materials that use the "Location" variable from the MpcCollectionInstance.	The value of it is dependent on the actor's location.

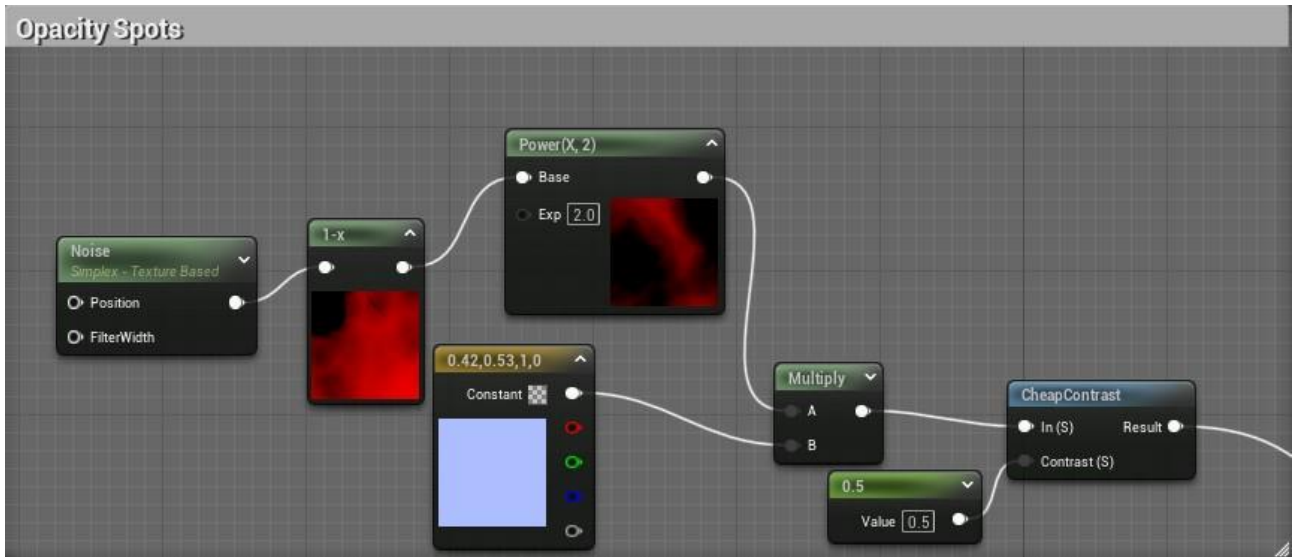
## Node Graph



This is the whole overview of the nodes. I will go through each annotated node group individually.



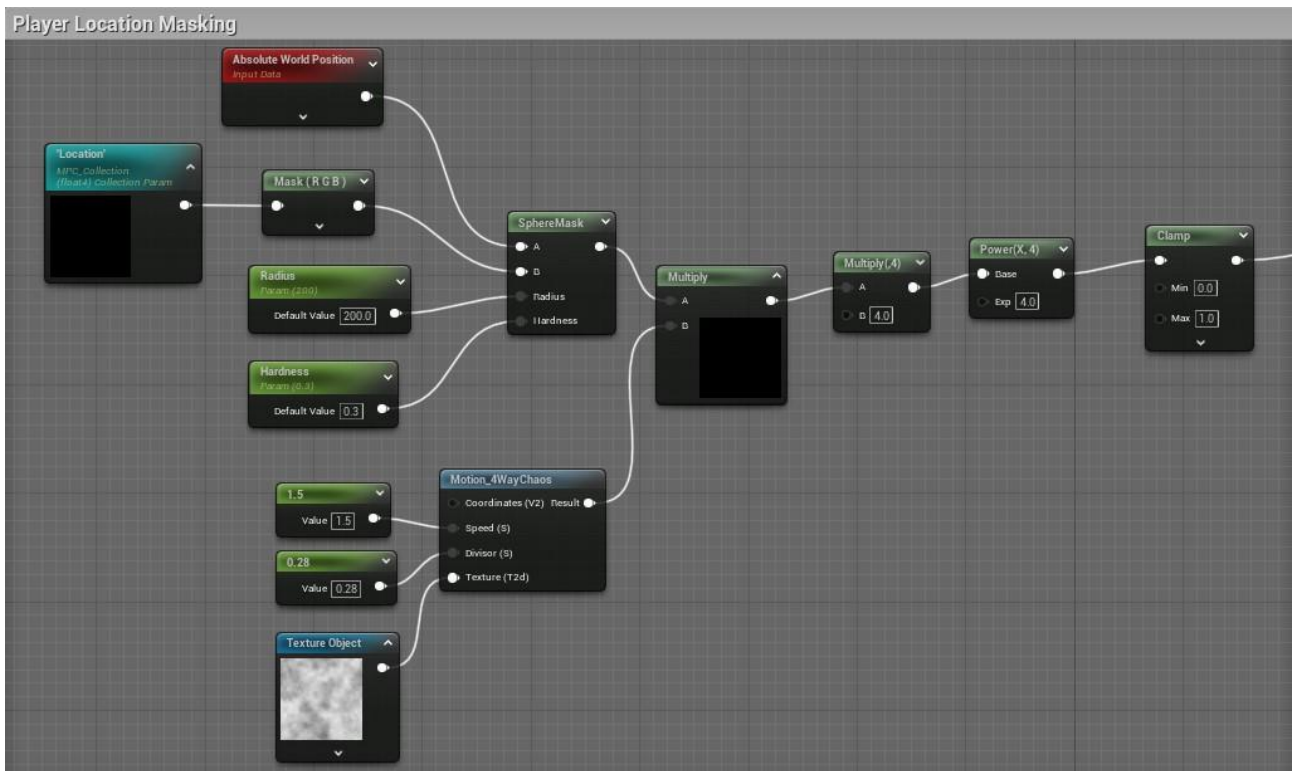
The colour of opacity spots is just a subtle grey ish colour therefore I only needed to use the 1 parameter for it and I put 0.4 on it.



I used the noise node to create the texture for the opacity spots. Then I used 1-x and powered it by 2. Then I multiply it with the colour (0.42, 0.53, 1, 0). I then added the CheapContrast material function to add the contrast for it and add the 0.5 value to the Contrast(S) connection.



I added this to eliminate the opacity spots when relevant.

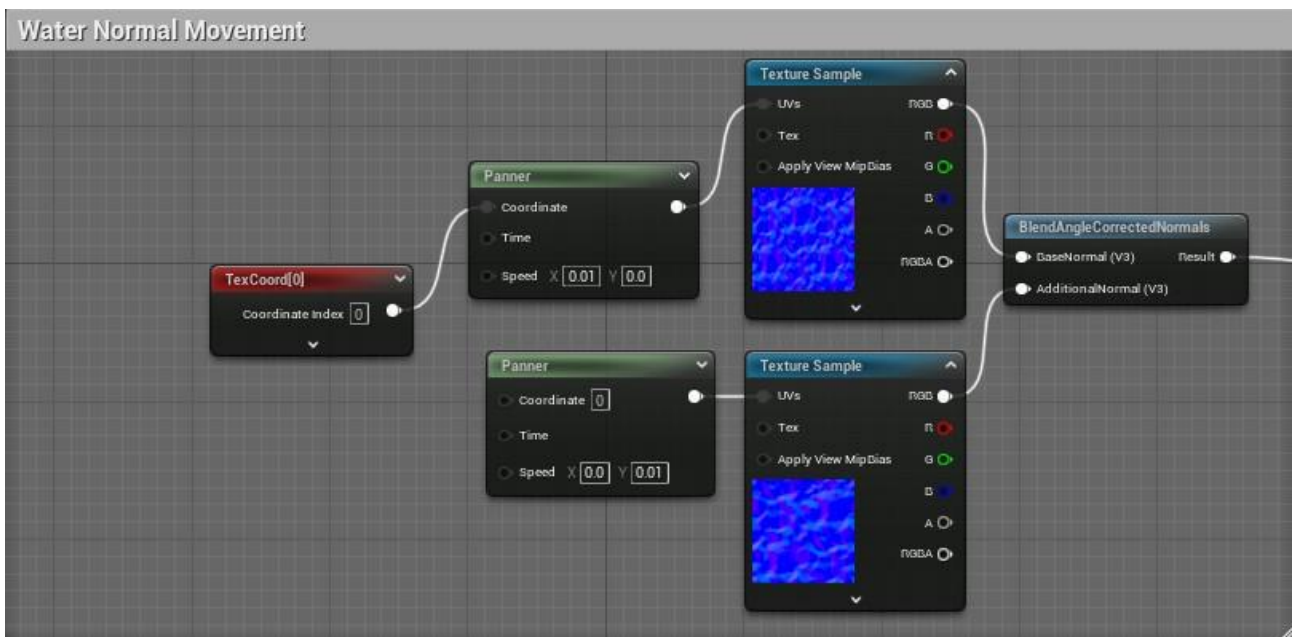




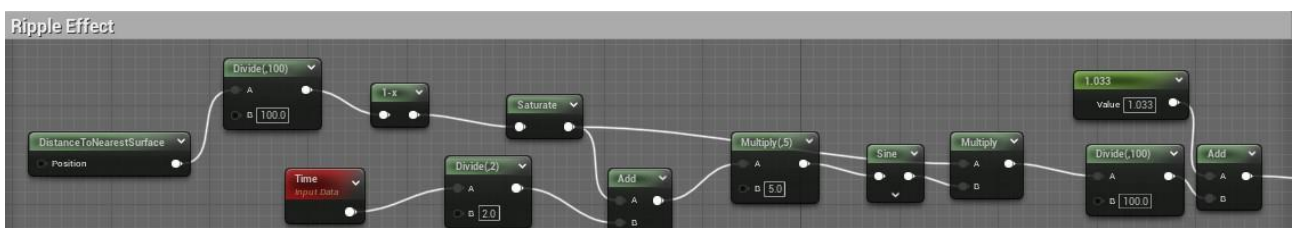
This Player Location Masking method is the same as the one explained in the first dynamic material. This is done to make the opacity spots disappear when the player is present in the water making the water clearer.



I connected the Opacity Spots to A(V3), 2 of the 0 nodes to B(V3) and C(V3) to eliminate the opacity spots when the player is present and the Player Location Masking to the Alpha(s).



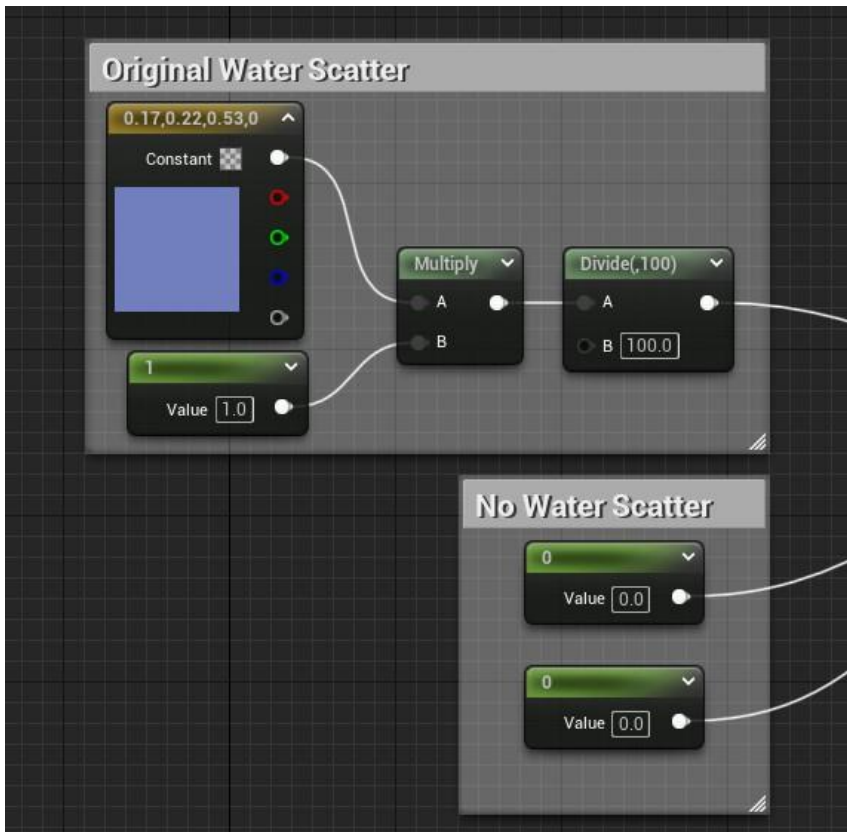
For the water movement, I used the normal texture sample called T\_Water\_N that is already in the engine. Then I added TexCoord[0] to the first panner and then I also added a second panner to the other texture sample that I duplicated. The Speed of the Panner on the top is 0.01 X and the speed of the Panner on the bottom is 0.01 Y. Then I connected both of the textures to a function called BlendAngleCorrectedNormals so I can mix 2 moving normals at the same time.



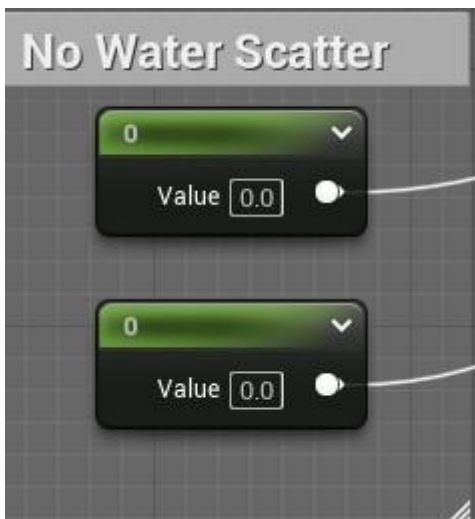
For the ripple effect, I started it with DistanceToNearestSurface which means that it takes the distance to the nearest object. Then I divide it by 100, add minus 1, and saturate it. Then I add an Add note that is connected to a Time node that is divided by 2. Then I multiply the Add Node by 5, add a Sine, Multiply it

with the earlier saturated node and then add a divide by 100. Then I connected it to the add node and added

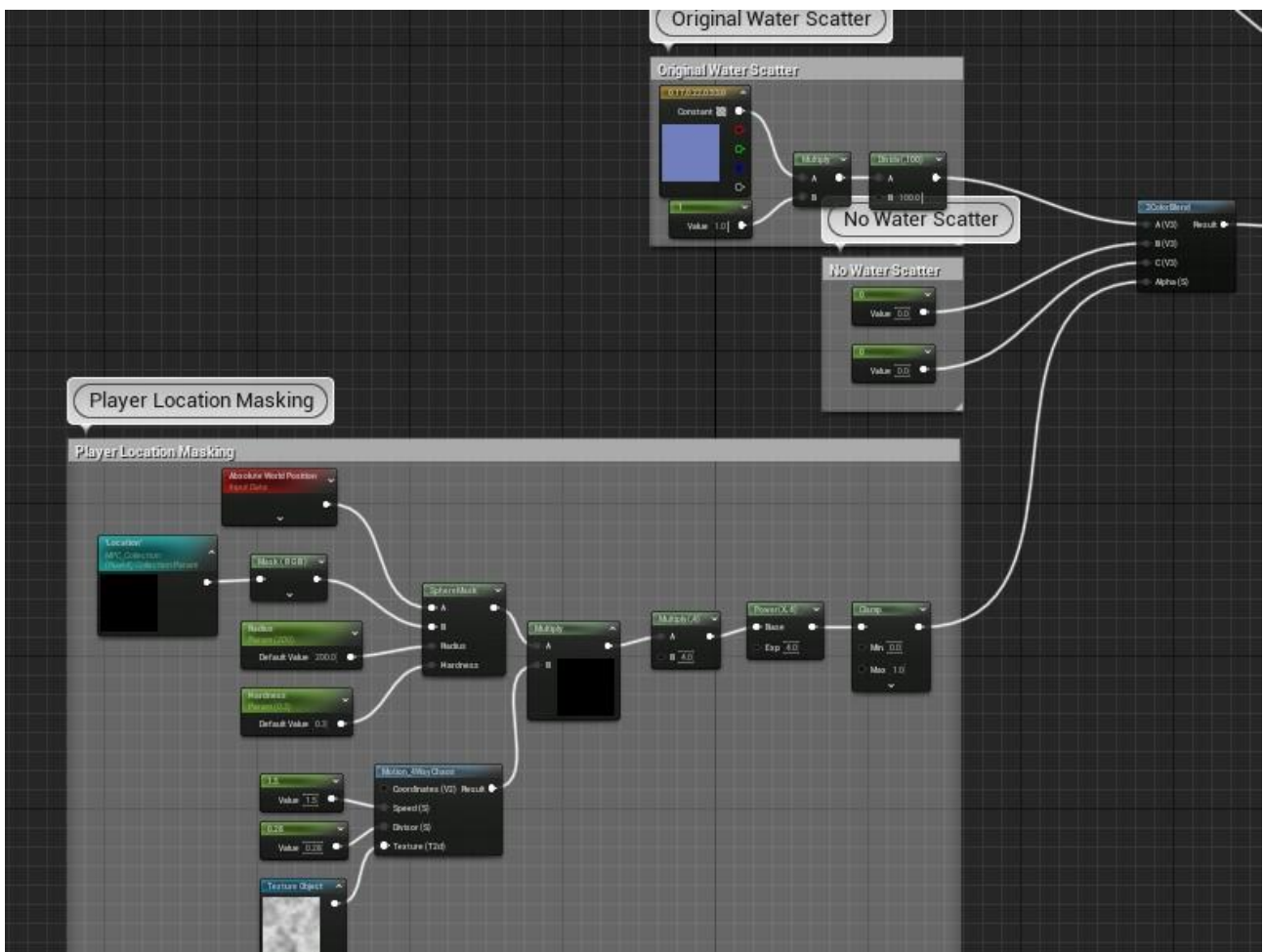
1.033 value to it. Basically what happens is, it takes an object that is present in the water, and adds a ripple effect around the object.



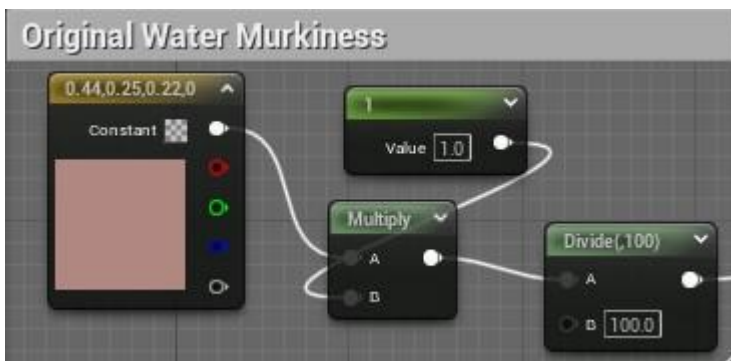
I added a colour which is (0.17, 0.22, 0.53, 0) to a multiple node with the value of 1 so I can adjust it later on, then divide it by 100. This is for the water scatter.



These 2 values represent the values of when the water doesn't have the water scatter. These values are triggered when the player's location is on the water.



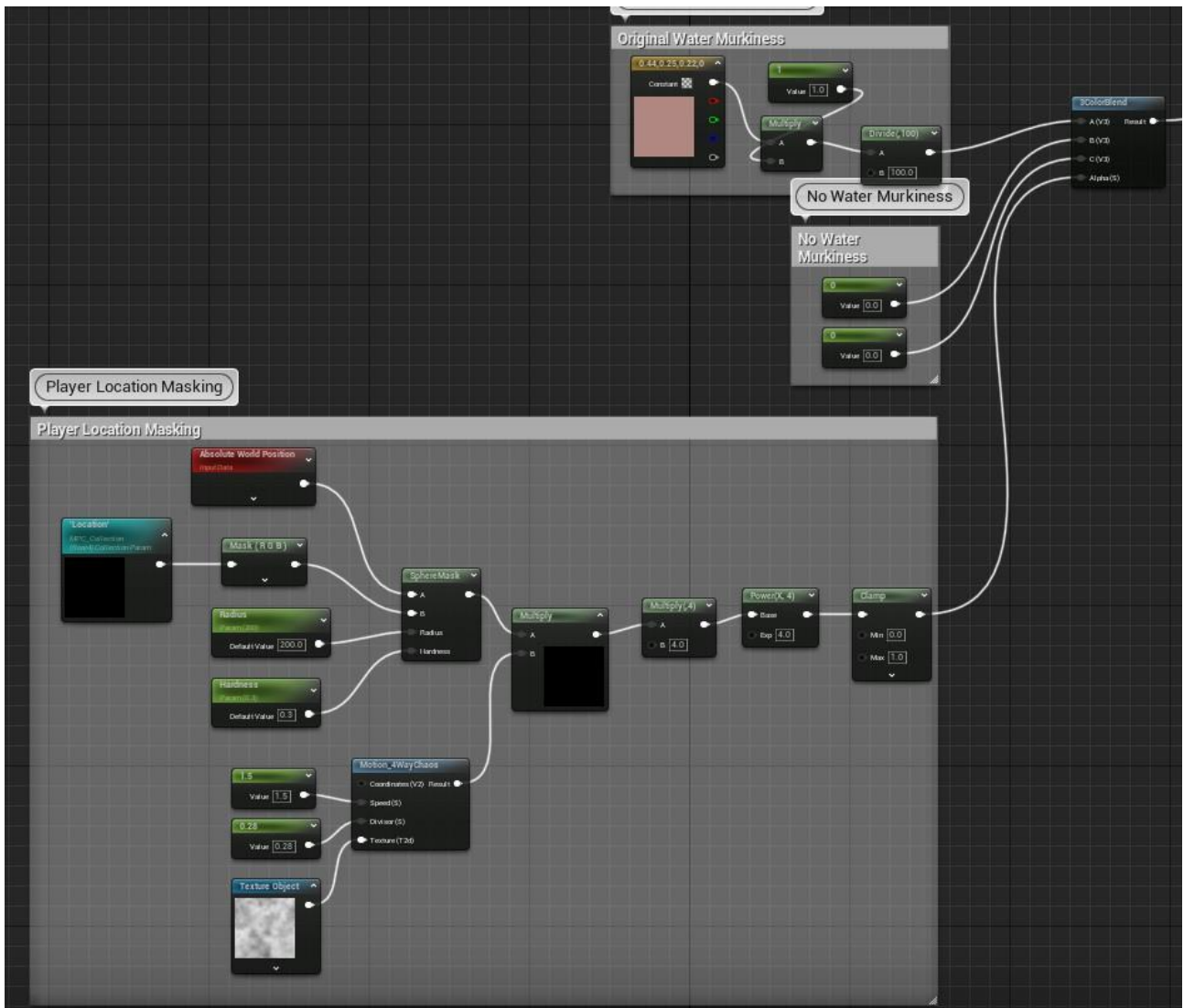
The Original Water Scatter, the no Water scatter and the Player Location masking is then connected to the 3ColorBlen function. The Original Water Scatter is connected to A(V3), No Water Scatter is connected to B(V3) and C(V3) and the Player Location Masking which is already discussed, is connected to Alpha(S).



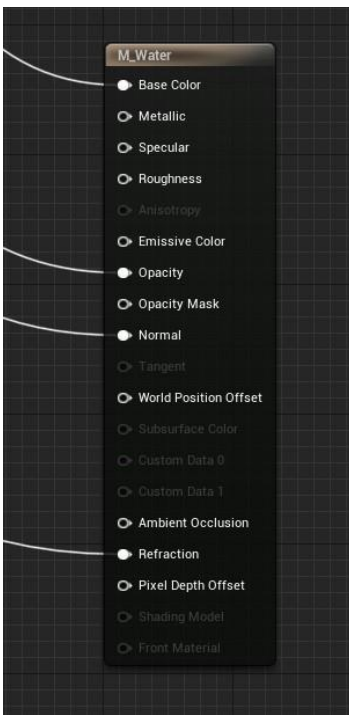
This adds to the Water Murkiness. I used the colour (0.44, 0.25, 0.22, 0) and then multiplied it by 1 and then divided it by 100.



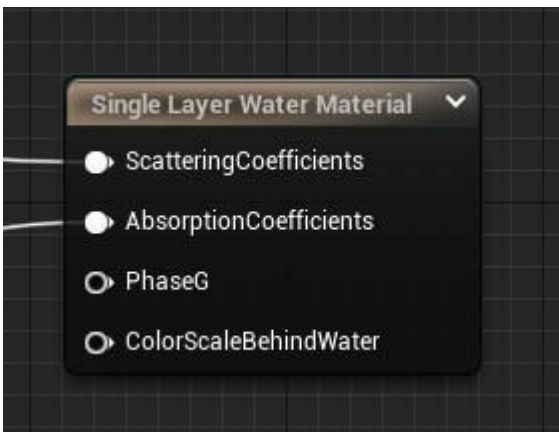
Similar to the previously discussed nodes, this is to represent the water when there is no murkiness present.



All of this is then connected to the 3ColorBlend function again with the Original Water Murkiness connected to A(V3), No water Murkiness connected to B(V3) and C(V3) and Player Location Masking connected to Alpha(s).



The connection for the main material node is pretty straightforward. The Base colour is connected to the Opacity Spots colour parameter, the Opacity is connected to the 3ColourBlend function that is connected to the Opacity Spots. The Normal is connected to the normal that I mentioned earlier and the refraction is connected to the water ripple.



There is another main material node called Single Layer Water Material that I used. This is an unreal specific node that helps me with creating stylized water with Water Scatter, Water Absorption and ColorScale Behind Water. I decided not to use the ColorScaleBehindWater because when I tested it with the Player Location Masking, it adds a colour to the mask and it doesn't make the water clear so I removed it. For the ScatteringCoefficients, it is connected to the 3ColorBlend function that is connected to the Water Scatter and the AbsorptionCoefficients is connected to the 3ColorBlend function that is connected to the Water Murkiness.

## Dynamic Material 3 - Glow

### Overview of Effect

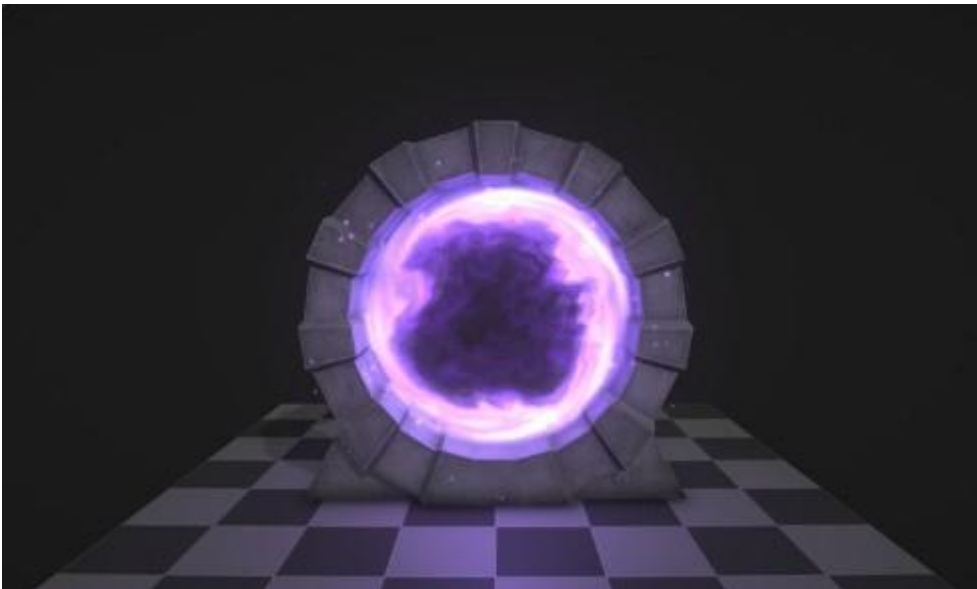
Basically this material is the simplest out of the other materials. This material starts off as a transparent material and then it gives a material change based on the player's location. It changes the emissivity with 2

colour layers which are orange and purple. It changes it based on the Sphere Mask that the material node has. The impact of the material in the gameplay is it adds a variety of level design to the game. It can act as a barrier that constrains the player from going out of their zone or as an invisible path that the player can go through as one of the puzzles. The outer layer of the material change is a subtle purple with a bright orange in the middle. It also has a small subtle purple circle in the middle of when the player is on the material. This adds aesthetic and dynamic impact to the player to keep the player engaged and increase the immersivity of the gameplay.

## Effect Description

The effect is inspired by a portal that is present in video games, especially the ones with high emissivity. With high emissivity, it allows the player to see the material change easily compared to other material changes. There was also a youtube video that helped me in the creation of this material and I changed the values and the colours of it so that it is original enough.

Inspiration / Reference Images:



<https://www.pinterest.com.au/pin/747597606874526851/>

I found this image on pinterest that resembles a portal with the difference between the inner and the outer colour of the portal.



UE5 | Electric and Magnetic Wall Effect | Material Tutorial | Unreal Engine 5

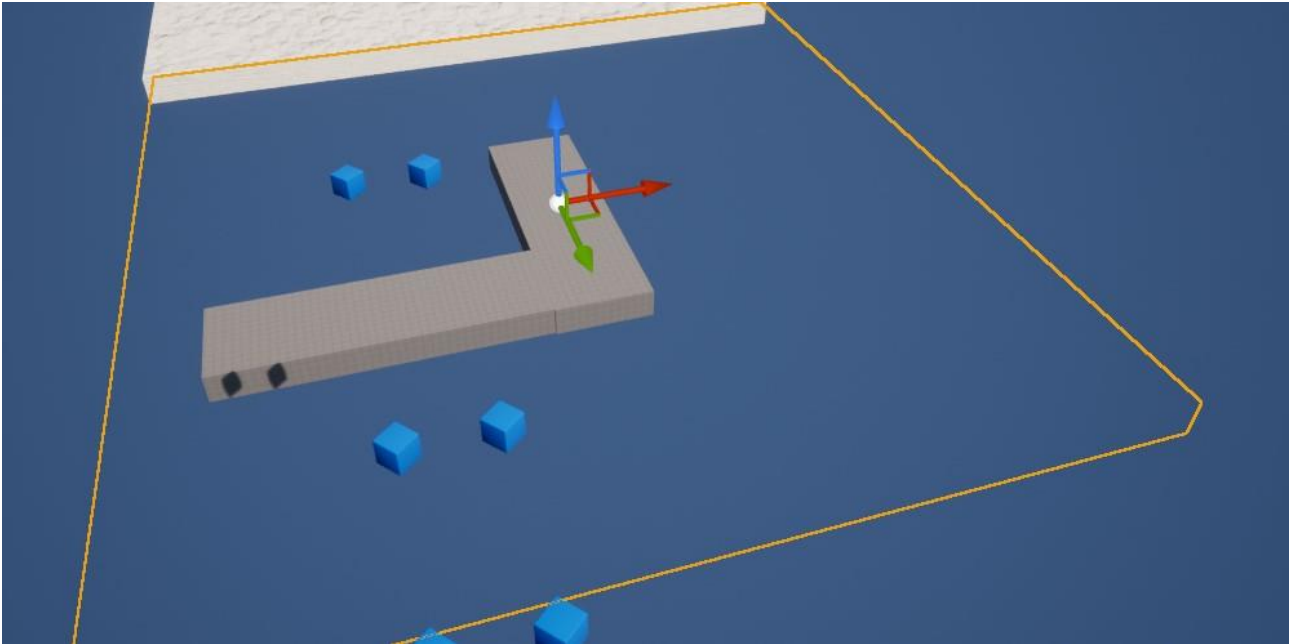
Coreb Games • 478 views

MARKETPLACE: Our Projects: <https://www.unrealengine.com/marketplace/en-US/profile/Coreb+Games?count=20&sortBy=effectiveDate&sortDir=DESC&start=0> FOLLOW US EVERYWHERE: Facebook:...

<https://youtu.be/uReAS0eolxM>

This video helped me in the making of player location masking and the portal effect of it.

### In-Engine Screenshot:



So as it can be seen, the material doesn't display anything when the player is not there making it transparent. This is because of the starting point of the material's emissivity and everything which is 0.



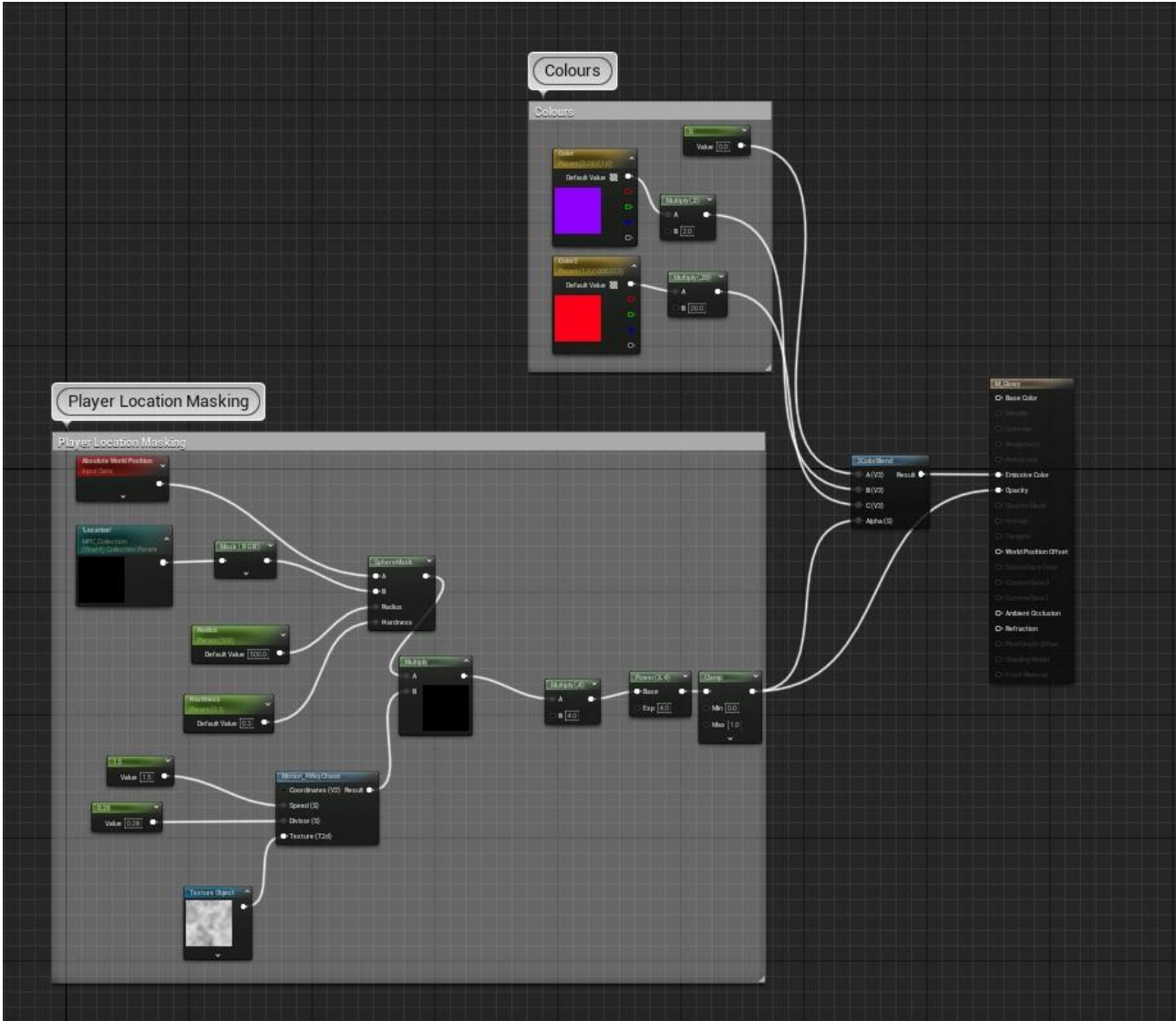
As it can be seen, when the player is on top of the invisible platform, it emits an emissive with 2 colour layers that was explained before.

### Properties and Values

Property	Description of Purpose	Value
----------	------------------------	-------

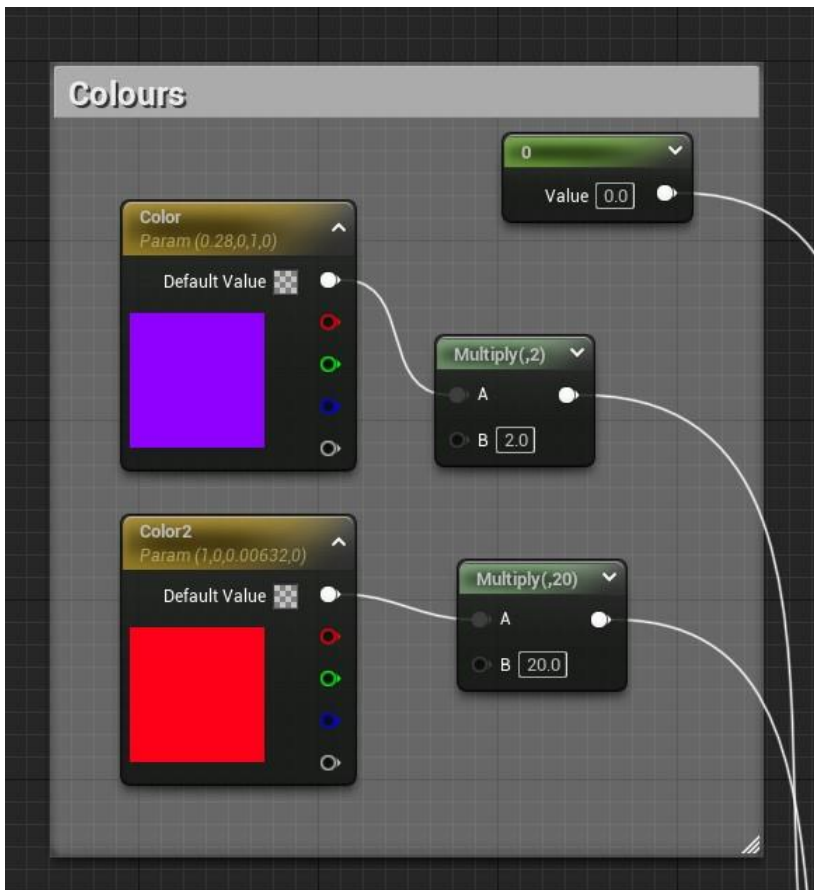
<p>MpcCollectionInstance ("Location")</p>	<p>The purpose of the MpcCollectionInstance is to access the Vector variable inside called "Location." It detects where the player is on the map and how close the player is to the materials that use the "Location" variable from the MpcCollectionInstance.</p>	<p>The value of it is dependent on the actor's location.</p>
---	--	--

**Node Graph**

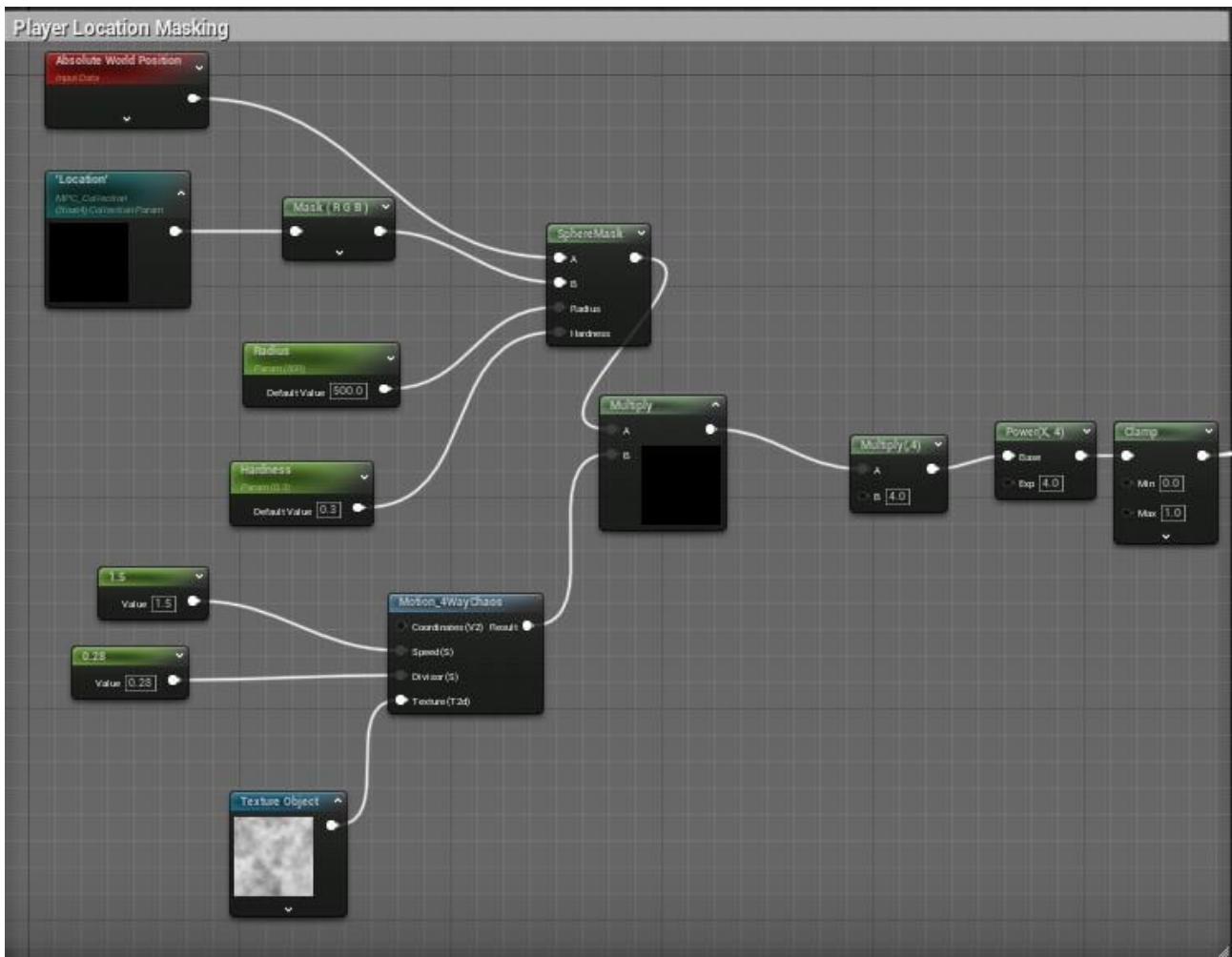


This is the overall material nodes that exist. As it can be seen, it has less nodes than the other materials but sometimes in materials, less is more.

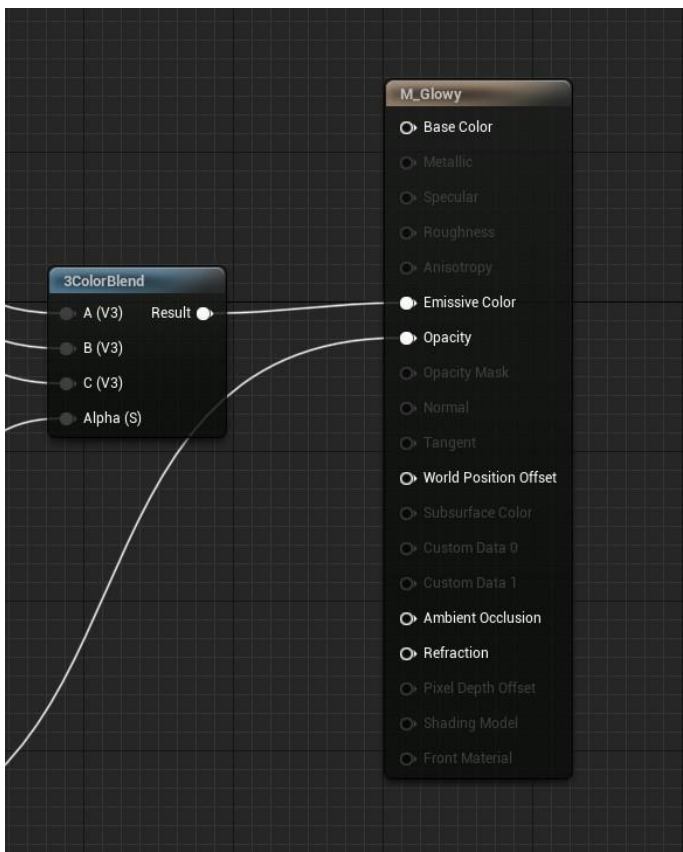




From this, it can be seen I was using 2 different colours which are red (1,0,0.00632, 0) and purple (0.28,0,1,0). The red is multiplied by 20 to add intensity and the intensity of the purple is multiplied by 2.0. It is worth mentioning that the reason the red has more intensity is to make it look like it's more orange than red and make the purple more subtle. The 0 parameter on the top represents 0 colour as the starting point since it is transparent.



This is the Player Location Masking that has been discussed in the previous 2 materials before. This is used to provide a mask of where the material changes which is a sphere based on the player's location.



The A(V3) is connected to the 0 value from earlier since it marks the starting point. The B(V3) is connected to the purple colour and the C(V3) is connected to the red value. The Alpha(s) is connected to the Player Location Mask. All of this is then connected to the Emissive Color from the 3ColorBlend function. The Opacity is also connected to the Player Location Mask so that it is only opaque when the Player's Location is close to the material.

## Dynamic Material 4 - Ground

### Overview of Effect

The effect of this material is fairly simple. It starts out as a snowy ground and then when the player gets on top of it, it becomes transparent. The reason why I added this material is to add a variety to the level design of the game. With this material, there are a couple of possibilities of a new level being added with the solid to transparent material. Certain puzzles and clues for a specific puzzle can be made to adhere to the material that could start out as an object hidden behind the objects with this material and then when the player gets close to it, the player is able to see the object.

### Effect Description

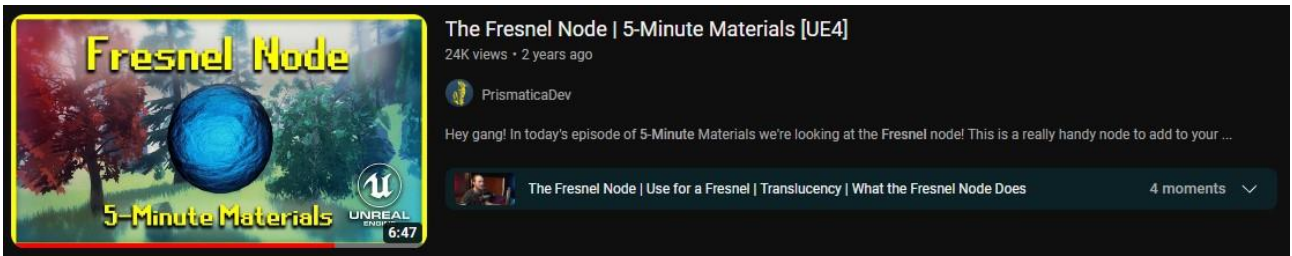
This effect is inspired by just snow in general. Snow in general is usually also linked with ice and ice is transparent. So technically, this shows the dynamics between snow and ice and how when the player goes close to the snow, it technically turns into ice since it became transparent. It is also inspired by snow that is present in famous video games that will be mentioned in the screenshots sections.

Inspiration / Reference Images:



Source : Breath of the Wild

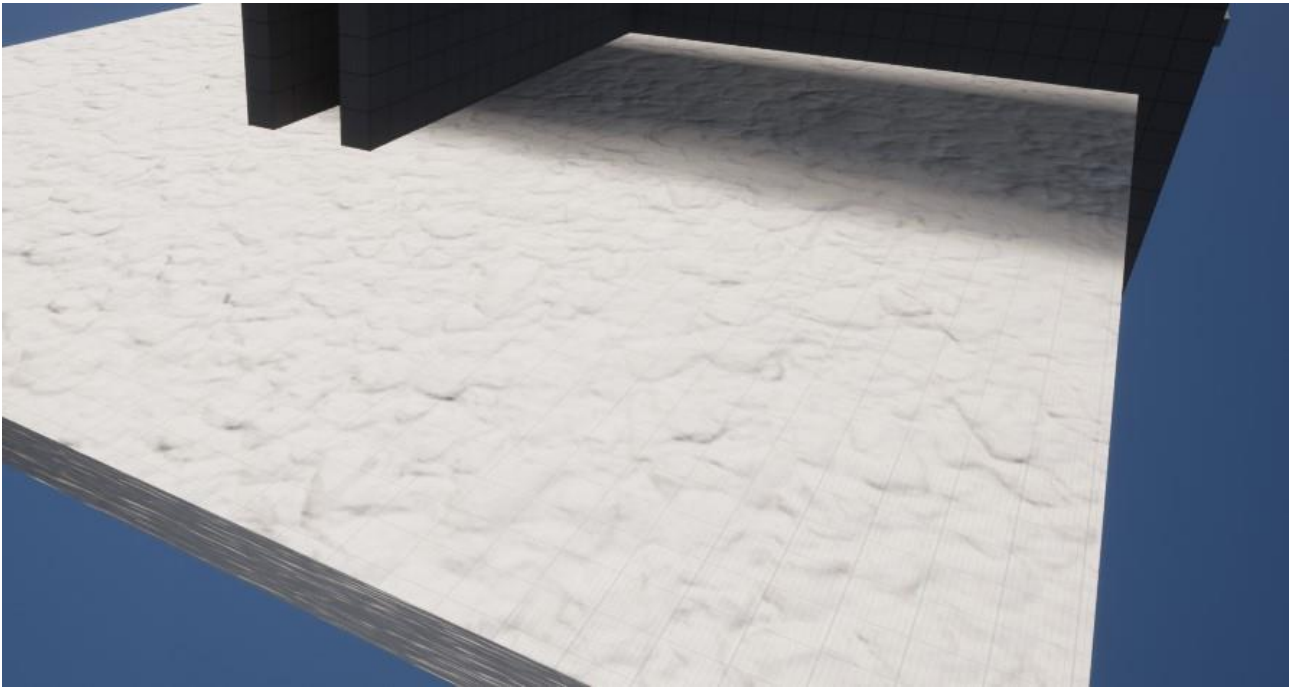
I got the snow inspiration from the game Breath of the wild. The snow in this game looks very stylized and simple to portray the cold in the area. This inspires me to make my very own snow and implement it in my game.



Source : <https://youtu.be/YCHelilKhzg>

This youtube tutorial helped me understand the concept of fresnel and it helped me in the process of making the snow effect. The fresnel effect allows me to combine 2 types of textures to form an ice effect between 2 different colours which are the white and the cyan colours.

In-Engine Screenshots:



This is the default material before the player interacts with it. It can be seen that it is very texturised with the normals and it is a mixture between the white and the cyan colour so that it creates this snowy effect.

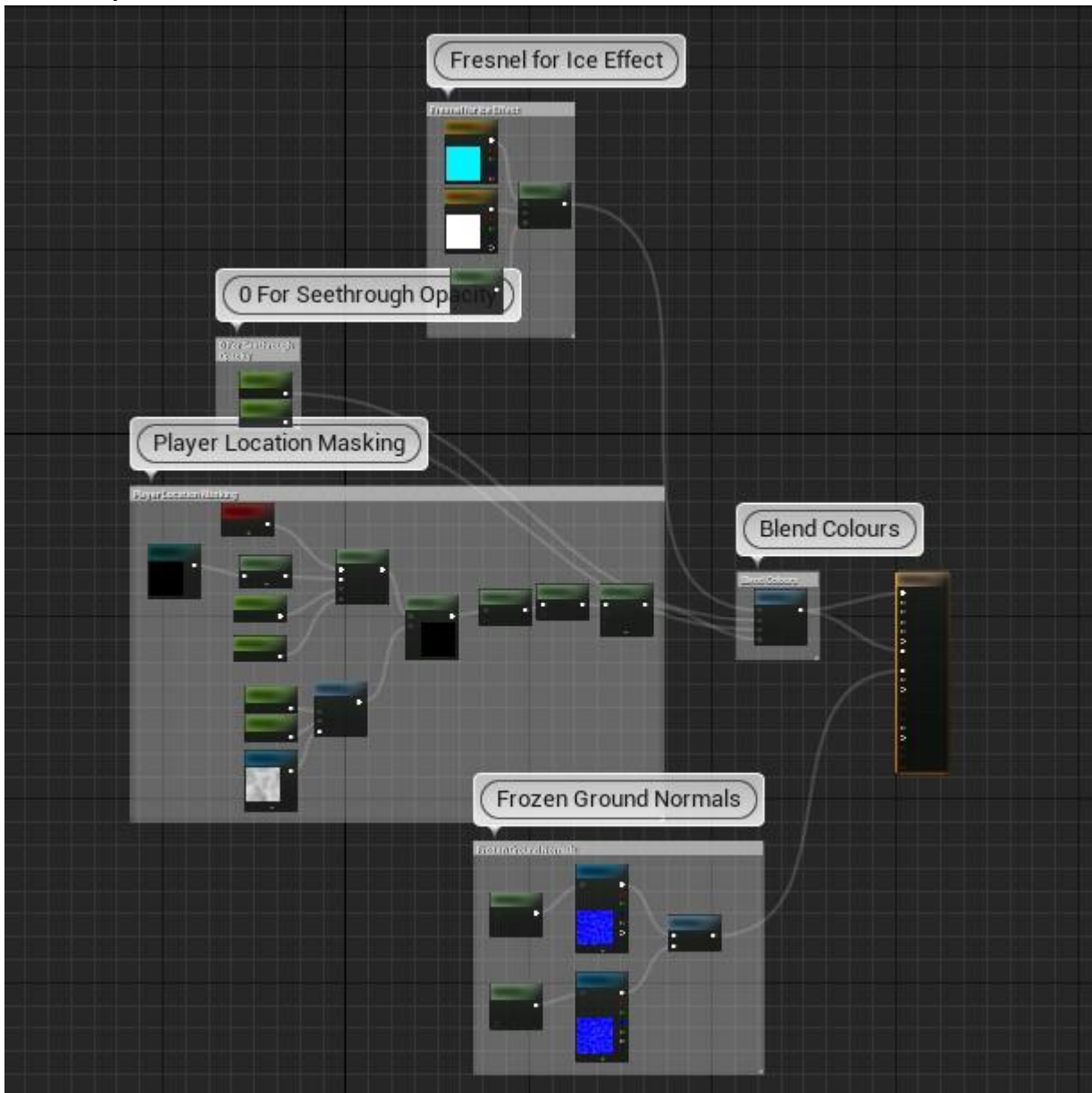


It can be seen that when the player is on top of the snow ground, it becomes transparent. It is hard to showcase the whole circle around the player appearing when the player is on the ground but from the screenshot above, the whole aspect of the material is clear enough just like the transparency effect.

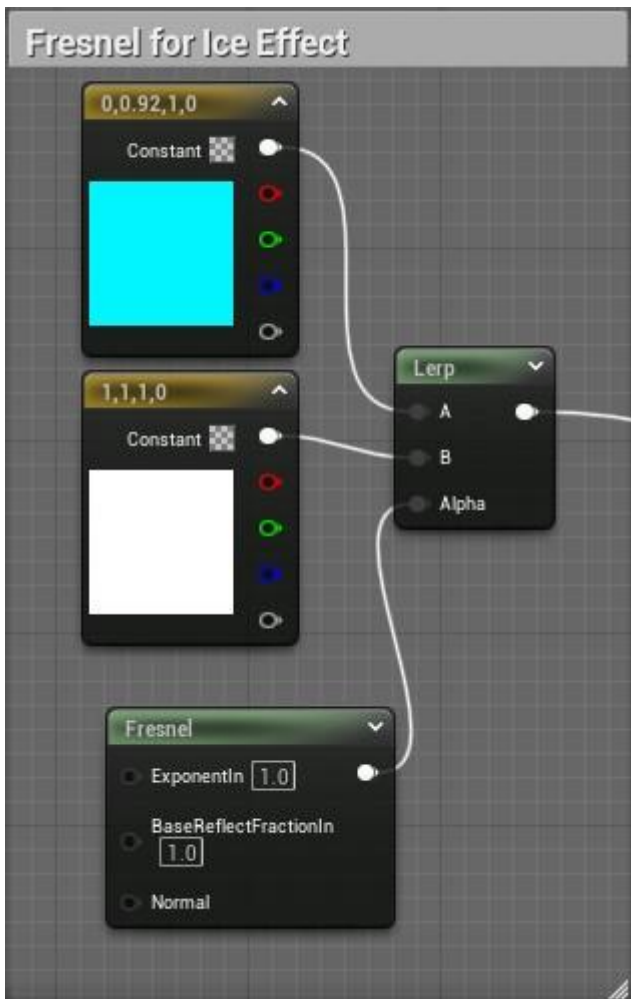
### Properties and Values

Property	Description of Purpose	Value
MpcCollectionInstance ("Location")	The purpose of the MpcCollectionInstance is to access the Vector variable inside called "Location." It detects where the player is on the map and how close the player is to the materials that use the "Location" variable from the MpcCollectionInstance.	The value of it is dependent on the actor's location.

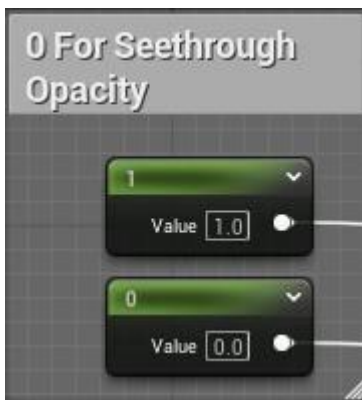
### Node Graph



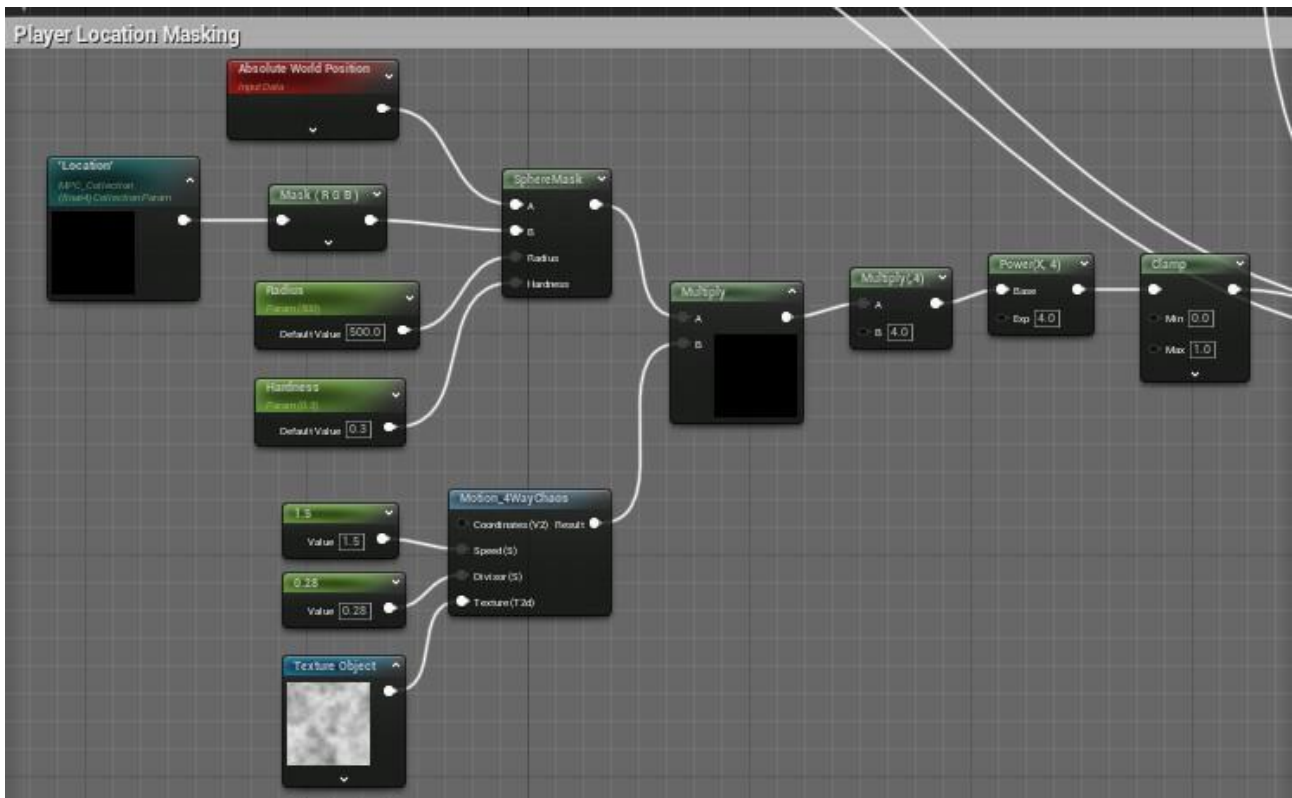
This is the overall nodes that are used to create the material. Each individual of the material nodes that are grouped will be explained one by one.



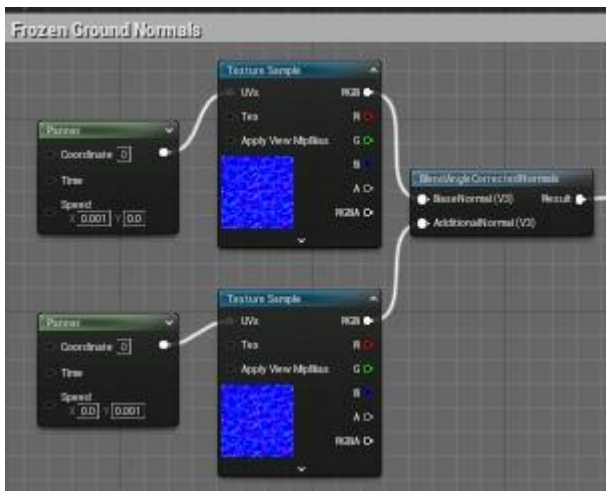
This node is combined with the Fresnel node to make the mixing effect of the snow. The two colours combined are the Cyan colour (0,0.92,1,0) and the white colour (1,1,1,0). Then this is connected to the lerp node with the Fresnel alpha to mix it thoroughly.



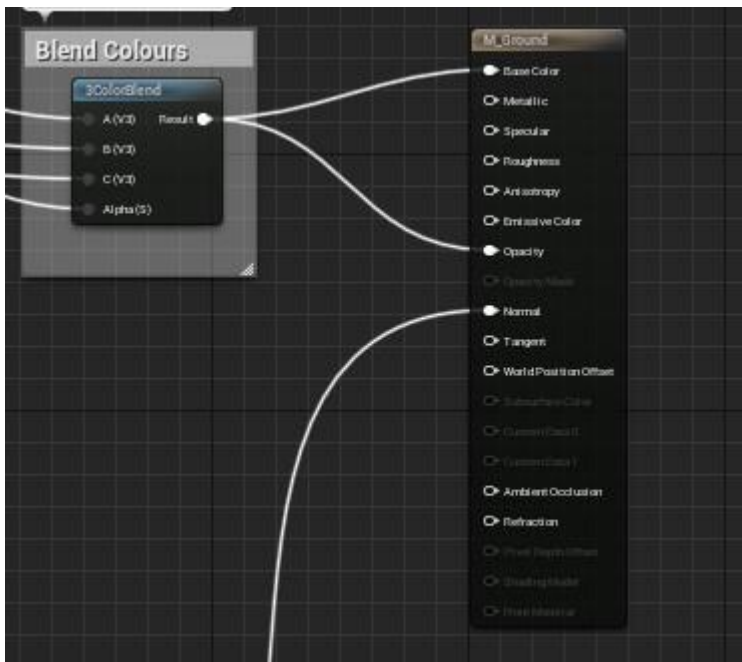
These 2 nodes are used to create the transparent effect of when the player is close to the material.



This Player Location Masking is already mentioned in the materials sections before. This grouped section basically takes the player's location and then makes a sphere mask based on the Player's location to the material.



These nodes are used to create the texturised part of the material. I used 2 panners to make it move subtly and two different normals so that it looks more texturised and I used the BlendAngleCorrectedNormals function to blend both normals together. The Normal Map that I used is the Pebbles\_029\_Normal.



I used the 3ColorBlend function as mentioned in the materials before to create the masking effect. The A(V3) is connected to the Fresnel Ice Effect, the B(V3) is connected to the 1 node and the C(V3) is connected to the 0 node to create the transparent effect. Then, the Alpha(S) is connected to the player Location Masking. Then I connected the 3ColorBlend to the Base Color and the Opacity so that it affects the colours and the opacity of the material. Then I connected the Normal to the grouped Normal Maps nodes that I made earlier.

## Physics

### Overview of Interaction

There are 3 different physics interactions that happen in the game. These are the jump mechanic, the bouncy ball projectile, and the black hole projectile. The jump mechanic launches the player based on where the player is looking at after a wall run. This gives the player the sense of control when jumping from walls for better fluidity of the core mechanic. The Bouncy Ball projectile is pretty self explanatory with it bouncing from one object to another. The Bouncy Ball projectile can be used for trickshots, puzzle solving, and other possibilities in the game. Lastly, the Black Hole Projectile sucks all objects with physics on to it and gives an explosion effect when it is gone. This projectile can be used for puzzle solving as well and could also be used for moving objects with physics easily. All of these physics interactions provide the player with more variety of mechanics that can be used while using the core mechanic to keep the player engaged and immersed in the game.

### Interaction Description

#### How the Interaction Works

So the first component which is the jump mechanic uses the LaunchCharacter() function. With this function, it allows me to launch the player wherever I want it to. There are 2 versions of the jumps which are the "Jump when Player is Wall Running." or



“Regular Jmp.” The Regular Jump just Launches the Character with the LaunchCharacter function by FVector(0, 0, 500) and sets the bXYOverride false and the bZOverride to false as well. When the player is in the middle of wall running and the player jumps, it also launches the character but with different values. Basically, before launching the character, I added an FVector variable called JumpBoost which gets the ActorForwardVector of X and Y from the CameraComponent and 1.5 as the Z value. Then on the launch character, I put JumpBoost.GetSafeNormal()\*1000 as the value of the LaunchVelocity since I want the values to be from where the player is looking at times by 1000 since the value by itself is really small. Then for the bXYOverride and the bZOverride I set it both to false.

The bouncy ball mechanic is fairly simple. There is no value that I adjusted to adhere to the physics component but instead I put a specific line of code called MovementComponent->bShouldBounce = true. This basically takes the MovementComponent function and then enables the boolean that is already inside. This enables the projectile to bounce after hitting every object with collision enabled.

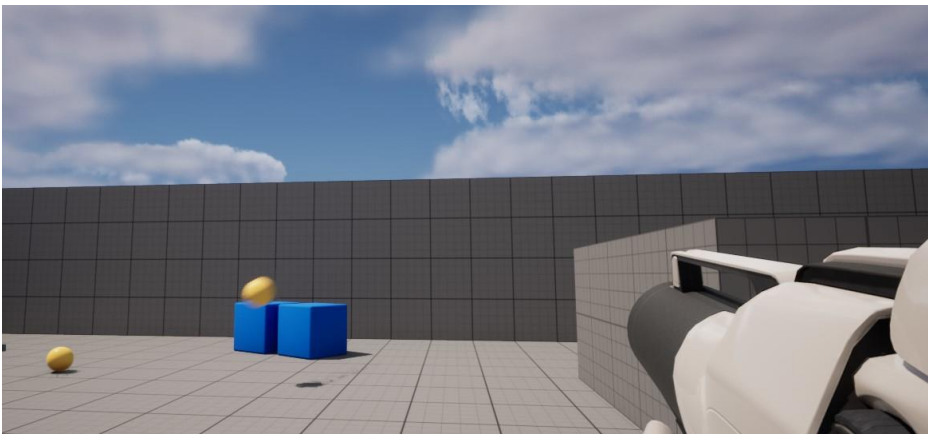
The black hole projectile is also fairly simple. I set the projectile to overlap all and I add a pulling effect. For the pulling effect, in the tick function I used an FCollisionShape called ExpositionSphere and then I made a FCollisionShape::MakeSphere(SweepSize) with the 1000.0f sweep size. Then, I used SweepMultiByChannel and added Impulse to the surrounding objects by -2000.0f so that it pulls the other objects to it instead of pushing it.

### Inspiration / Reference Images



Source : Mirror's Edge Catalyst

The inspiration for the parkour and the jumping mechanic came from the game Mirror's edge catalyst. Mirror's edge catalyst basically inspired the whole concept of the game in the first place since it is a first person action adventure parkour game.



Source : First Person Template of Unreal Engine 5

The first person template of the engine has a bouncy ball projectile. I saw it and thought to myself that it would fit perfectly to my game since my game will have enemies that I can trickshot with my bouncy balls.



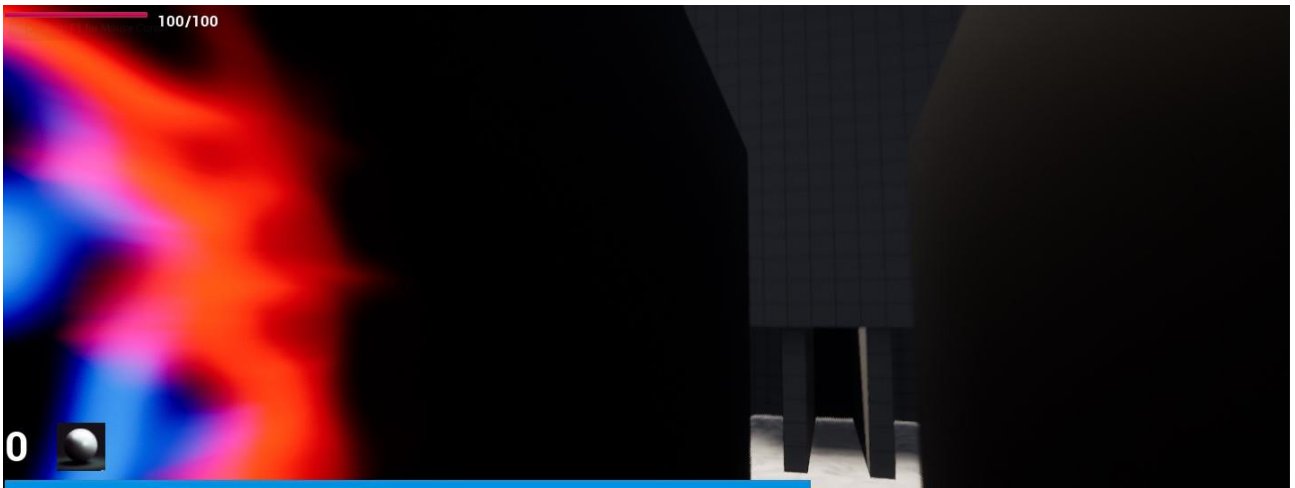
Source : Week 6 Labs of FIT2096

For the black hole projectile, I was inspired by one of the tasks in the labs which was to make a block that pulls all objects nearby to it. I envisioned it to be a black hole when I used it and made it in the form of a projectile.

### In-Engine Screenshots



In this screenshot, it is shown that the player used the LaunchCharacter function upwards increasing the Z value by 500.



In this screenshot, it is shown that the player is getting launched based on where the player is looking at which in this case is forward.



In this screenshot, it can be seen that the bouncy ball projectile bounced back from where I shot it, which is the black wall in front of the player.



This screenshot shows the black hole projectile without any objects nearby. It just pushes itself until it is destroyed after a couple of seconds.



This screenshot shows the black hole projectile when there are objects nearby. The objects will get pulled and it will swirl similar to a black hole effect. This effect runs every tick until the projectile is gone.



This screenshot shows the objects when the black hole projectile is gone. The objects will be pushed away giving an effect of explosion of the projectile.

### Properties and Values

Property	Description of Purpose	Value
JumpBoost	Acts as a variable for the jumping mechanic of when the player is on the wall.	FVector (GetActorForwardVector().X, GetActorForwardVector().Y, 1.5f)
LaunchCharacter()	The LaunchCharacter	PlayerCharacter->LaunchChar

	is used to launch the character whenever the player is jumping but the value of it depends on whether the player is wall running or not.	acter(JumpBoost.GetSafeNormal()*1000, false, false) for when the player is wall running or PlayerCharacter->LaunchCharacter(FVector(0,0,500), false, false)
bShouldBounce	This boolean is a function in the MovementComponent that allows the actor to bounce when hitting another object in the world.	MovementComponent->bShouldBounce = true
SweepSize	This variable is used to adjust the size of the temporary hitbox around the projectile	float SweepSize = 1000.0f
OutHits	This array variable holds out all the hit results from the black hole projectile.	TArray<FHitResult> OutHits = *Objects around the projectile*
Location	This FVector value gets the location of the actor which is the black hole projectile.	FVector Location = GetActorLocation()
ExplosionSphere	This FCollisionShape makes a sphere around the actor	FCollisionShape ExplosionSphere = FCollisionShape::MakeSphere(SweepSize)
SweepMultiByChannel	This function performs a collision check around the ExplosionSphere.	if(GetWorld()->SweepMultiByChannel(OutHits, Location, FQuat::Identity, ECC_WorldStatic, ExplosionSphere))
Hit	This is a variable to	for(auto& Hit : OutHits)

	loop with the array called OutHits.	
Mesh	This UStaticMeshComponent variable is there to cast the actor that get hit by the collision sphere into a UStaticMeshComponent	UStaticMeshComponent* Mesh = Cast<UStaticMeshComponent>(Hit.GetActor()->GetRootComponent());
AddRadialImpulse	This function is used to add impulse to the objects that were detected nearby. The value of the impulse is negative since it pulls the object to the actor rather than pushing it.	Mesh->AddRadialImpulse(Location, SweepSize, -2000.Of, ERadialImpulseFalloff::RIF_Linear, true)

## Artificial Intelligence

### Overview of AI

The main purpose of my main AI is to be a simple enemy to the player. The AI patrols through a random location through the navmesh and looks at the player and shoots the player when the player is in their sight. In a parkour game, making an enemy is a bit tricky since parkour games tend to not have enemies, but by adding enemies to my game, the AI implementation itself is simple enough that it is viable for me to put AI in the game to give the player a bit of a challenge and stakes when playing the game. The AI can also give the player satisfaction by killing them with trickshots and possibilities that are available within the game mechanics such as the slow down time mechanic, the bouncy ball mechanic, etc. With these mechanics there are a number of ways for the player to kill the enemies. The AI also has a switching Gun task to allow the AI for more variety of ways to attack the player. The AI switches guns after it runs out of ammo of the gun that the AI is using. This can be an addition to the threat and danger of when the player is facing the AI.

### AI Description

#### AI Abilities

The AI uses sight stimuli as one of his abilities. So the AI is able to detect the player when the player is close enough and is in front of the AI. The AI is also able to shoot the player when it detects the player with its stimuli. The way the AI shoots the player is that the AI will rotate towards the player every time it sees the

player and it will spawn a projectile in front of it based on where the player is. The AI then will have a cooldown for 2 seconds after shooting and then shoot again after the cooldown ends.

When it is not engaging a player, the AI generates random locations throughout the navmesh and goes into patrol mode where the AI is going through the environment and patrolling randomly. It uses an UNavigationSystemV1 variable called NavigationSystem that is declared on the header file. Then it checks if the NavigationSystem is there and an FNavLocation called ReturnLocation is made. Then, it gets a random point in the navmesh using GetRandomPointInNavigableRadius in a radius of 2000 and puts it in the ReturnLocation variable. Lastly, it accesses one of the Blackboard Component's values which is the PatrolPoint and puts the ReturnLocation variable inside the PatrolPoint variable.

The AI also switches guns everytime time the int variable GunAmmo is below or equals to 0. At the moment, the AI only has 2 guns in total but there will be more guns to be added to the game in the future.

## Inputs & Senses

### AI Senses

Sense Name	Property	Value
Sight	SightRadius	1000
	SightAge	10
	LoseSightRadius	SightRadius + 30
	FieldOfView	100

### Blackboard Values

Property	Description	Related Actions
SelfActor	An object variable that accesses the actor of the AI by itself.	Actions have not been implemented but is planning to be in the future
PatrolPoint	A vector variable that gives the AI a vector to go to when patrolling.	GenerateNewRandomLocation, Patrolling

PlayerPosition	A vector variable that gives the AI information of where the player's position is.	Rotate facing Player, Shooting Player
ShootPlayer	A boolean that gives the Ai information whether it should start shooting or not.	Shooting Player
GunVariable	An int variable that determines what kind of guns that the player will switch into,	Switch Gun
GunAmmo	An int variable that indicates how much ammo the enemy AI has left to shoot before switching the gun.	Shooting Player, Switch Gun

Basically, the sight stimuli of the AI acts as a trigger for the AI to start shooting at the player. The priority of the AI shooting is higher than its patrolling because the AI immediately stops patrolling when the player is in sight of the AI. The priority of the rotation of the enemy to the player and shooting at the player is the same since I want the enemy AI to run both of those at the same time when the AI senses the player.

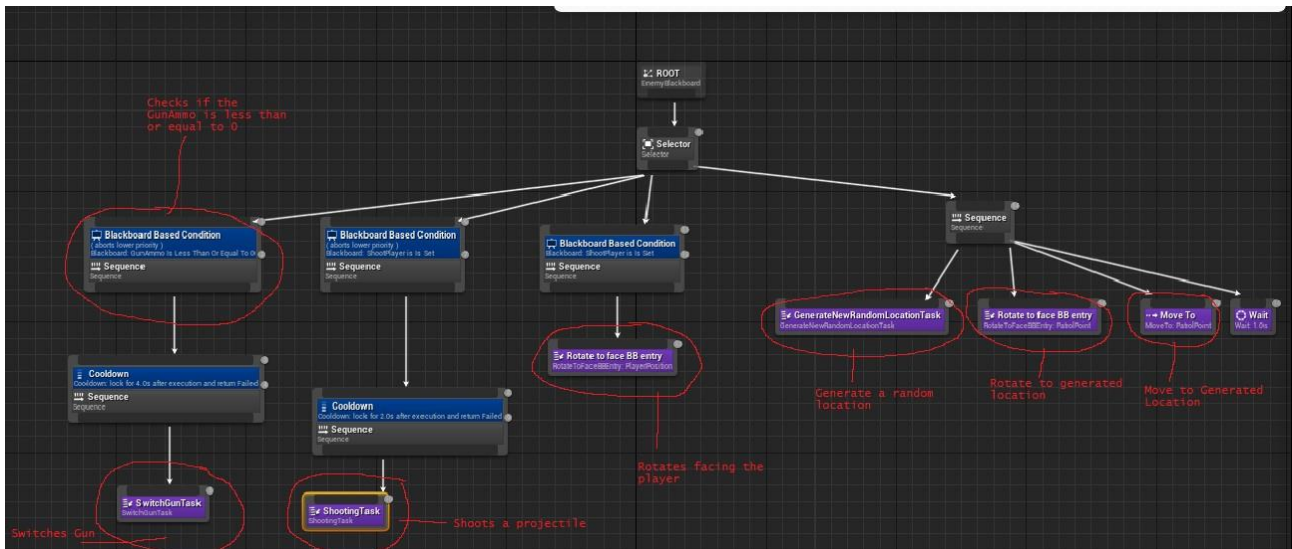
When patrolling, the blackboard variables that are used are mainly PatrolPoint. It uses a custom C++ task called GenerateNewRandomLocationTask to provide for the vector variable of PatrolPoint. Then it rotates to the PatrolPoint and moves to it. It also waits for 1.0s after completing the task.

The 2 other sequences are the Rotate to the player and the ShootingTask. The Rotate to the player has a decorator which is basically telling it that it can only be activated if the ShootPlayer variable is true that is activated when the player is within sight of the sight stimuli. Another sequence that has the same priority as the rotate to player is the ShootingTask sequence. It starts with a Sequence with a decorator that is also enabled when the ShootPlayer boolean is true. Then it goes to a cooldown of 2.0s after execution so that the enemies shoot in a predicted and timely manner. Then after that, it connects to the custom C++ task called ShootingTask.



The Switch Gun task is highly dependent on the GunVariable variable. The switch gun task basically checks which gun the enemy AI has in the moment and changes it into the other gun that the enemy AI is not using based on the GunVariable variable.

## Behaviour Tree Graph



Overall the behaviour tree is fairly simple. It starts with a selector that selects the actions based on the priority and the conditions of the decorator. Since there is only 1 sequence that has no condition decorator, the AI will execute the patrolling sequence every time at the start. This will keep going until the AI senses the player with its sight stimuli which makes the ShootPlayer variable to true. This then makes the player rotate to the player every time the player is on sight and shoots at the player every 2 seconds. The reason why I didn't combine the rotation and the shooting in the same branch is because I want to make sure that the AI always looks at the player even when it is not currently shooting at it. The priority of the shooting is high and it aborts the lower priority tasks which are the generated random location task. The reason why the priority of the shooting is high is because I want the enemy to stay in place while they are shooting the player. This creates an effect that makes the enemy AI look like it's actually taking its time and focusing on shooting the player rather than them moving while shooting. Then there is also the SwitchGun task. This task is only executed based on the black board condition which is that the ammo of the gun has to be under or equal to 0. Then after that, it triggers the switching gun task after a 4 seconds wait. The switch gun task has the highest priority of them all. This is so that the enemy AI can focus on reloading rather than shooting and moving because logically, when the gun is out of ammo, the enemy AI isn't able to shoot anything unless it reloads first.

# Niagara Particles

## Niagara Particle Effect 1 - BlackHole

### Overview of Effect

This particle effect is being used for the black hole projectile. It is made to give a visual impact to the player when the player is firing the black hole projectile. The black hole projectile particle effect resembles a very commonly imagined shape of a black hole with an extra addition of the ring surrounding it. The black hole has an outline of a blue shade and the black colour in the middle. The ring has a different colour which is cyan. Even though the colour is different, it is not too different so that it complements each other. The particle effect is also emissive to give the player a sense of impact and satisfaction when using it even when the area of the game is dark or has low lighting. There are 4 different emitters that were used for the BlackHole particle effect. These emitters are the BlackHole\_Core, Core\_Particles, Ring\_Particles, and the Swirl. The BlackHole\_Core provides the black circle with the blue outline while the Core\_Particles provide the particle spread of the black circle. The Ring\_Particles however provides the particle spread on the ring of the black hole projectile particle and the Swirl provides the swirly pattern of the ring of the particle.

### Effect Description

As mentioned before in the overview, the effect resembles the look of a traditional black hole whether it's in video games and science fiction films. It is attached to the mesh component of the black hole projectile and it despawns when the projectile is destroyed. The effect has the ring that is attached to it to give it a sense of uniqueness and make it look more cooler. It also resembles the sphere shape since black holes that were used as a reference tend to look like a circle.

Inspiration / Reference Images:



Source : <https://www.pinterest.com.au/pin/180495897553589982/>

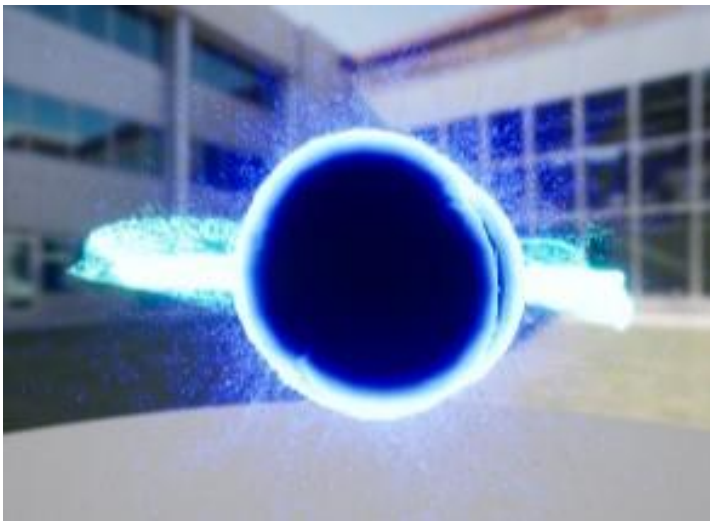
This image resembles perfectly how I want my black hole projectile to look like. It has the rings around the black circle with its outline that are very bright. The brightness aspects of it inspire the emissivity of the particles. The only difference between this reference image and the actual particles that I implemented is the colour of it with this being overall brown coded and my particles being blue coded.



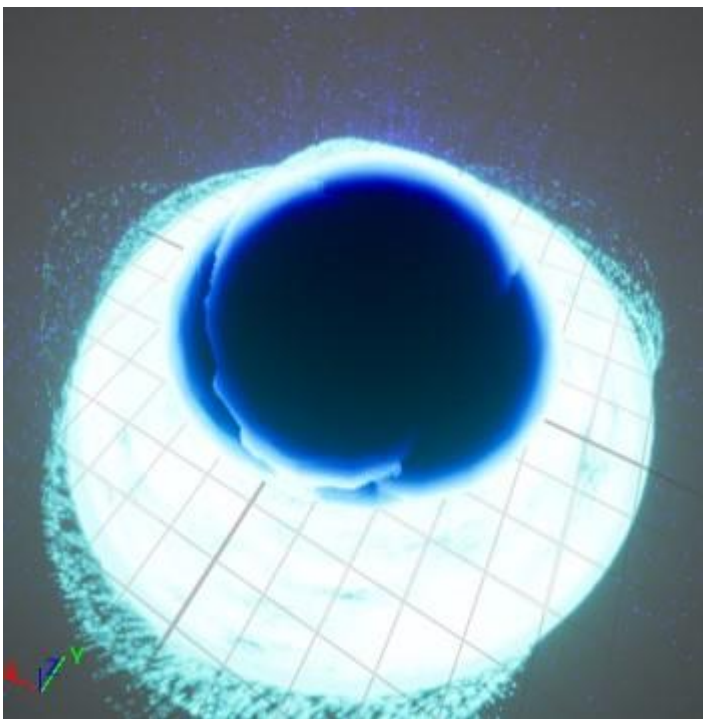
Source : <https://youtu.be/YFqRdSZEi20>

This video helps me in the process of making the particles.

In-Engine Screenshots:



This is how my black hole projectile particle effect looks like. As it can be seen, it has a lot of different components in it. The particle spread of the main core can be seen from the blue particles that are spreading. The ring can also be seen with it being cyan.



It can also be seen from this screenshot that the particle spread from the ring has a specific pattern in it. It can also be seen that the swirl ring has a very textured look due to the fact that it uses a specific material that I made.

#### Properties and Values

##### BlackHole\_Core Emitter

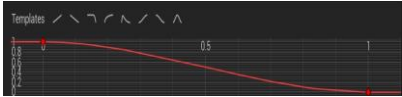

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	5.0

Spawn Rate	Number of particles per second to spawn	90.0
Spawn Burst Instantaneous (Spawn Count)	Spawns a burst of particles instantaneously	1
Lifetime Mode	Lifetime of the particle	Random (Min (1.4), Max (1.75))
Color Mode	The color of the particle	Direct Set : R 0.0, G 41.186333, B 1000.0, A 1.0
Mass Mode	The mass of the particle	Random (Min (0.75), Max (1.25))
Mesh Scale Mode	Determine the scale of the mesh of the particle	Uniform (0.2)
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Mesh Size	Takes the initial mesh scale as set in the spawn script, and scales it by a scale factor.	Sine (Normalized Angle, Period 0.1, Scale 0.02, Bias 1.0)
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Jitter Amount	Amount of jittering of the particle	0.5
Jitter Offset	Offset of the particle jittering	Random Vector
Override Materials	Overrides the material with a specific selected material	M_ShapeFresnel

## Core\_Particles Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	5.0
Spawn Rate	Number of particles per second to spawn	50000.0

Lifetime Mode	Lifetime of the particle	Direct Set (1.5)
Color Mode	The color of the particle	Direct Set : R 0.0, G 0.698129, B 41.186096, A 1.0
Mass Mode	The mass of the particle	Unset
Sprite Size Mode	Determine the size of the sprite of the particle	Random Uniform (Min 6.0, Max 8.0)
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the shape primitive	20.0

Camera Offset Amount	Amount of the offset of the particle along the vector between the particle and the camera	-100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Mesh Size	Takes the initial particle scale, and scales it by a scale factor.	Uniforms Curve Sprite Scale
Scale Curve	The curve of the Uniform Curve Sprite Scale	
Uniform Curve Scale	The scale of the curve uniformly	0.2
Noise Strength	Scales the sampled curl noise force vector.	300.0
Noise Frequency	Modulates the position to increase or decrease the rate at which curl noise is sampled	100.0
Noise Quality / Cost	Determines the quality of the noise	Baked (Medium)
Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample.	X 0.2, Y 0.5, Z 1.0
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	Float from Curve -> FloatCurve->Curve
Drag Curve	The curve of the drag	


Scale Curve	The scale of the drag curve	1.0
Rotational Drag	Reduces each particle's rotational velocity	1.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Material	The material for the particle	M_Radial_Gradient

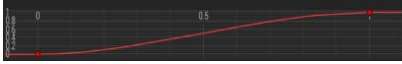
Ring\_Particles Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)

Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	5.0
Spawn Rate	Number of particles per second to spawn	50000.0
Lifetime Mode	Lifetime of the particle	Random (Min (0.1), Max (1.5))



Color Mode	The color of the particle	Direct Set : R 0.0, G 9.104832, B 10.0, A 1.0
Mass Mode	The mass of the particle	Unset
Sprite Size Mode	Determine the size of the sprite of the particle	Random Uniform (Min 6.0, Max 8.0)
Shape Primitive	The shape of what the particle will spawn into	Ring / Disc
Ring Radius	The radius of the Ring primitive	60.0
Camera Offset Amount	Amount of the offset of the particle along the vector between the particle and the camera	-100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Sprite Size	Takes the initial particle scale, and scales it by a scale factor.	Uniforms Curve Sprite Scale
Scale Curve	The curve of the Uniform Curve Sprite Scale	
Uniform Curve Scale	The scale of the curve uniformly	1.0
Noise Strength	Scales the sampled curl noise force vector.	100.0
Noise Frequency	Modulates the position to increase or decrease the rate at which curl noise is sampled	10.0
Noise Quality / Cost	Determines the quality of the noise	Baked (Medium)

Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample.	X 2.0, Y 3.0, Z 5.0
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	Float from Curve -> FloatCurve->Curve
Drag Curve	The curve of the drag	
Scale Curve	The scale of the drag curve	1.0
Rotational Drag	Reduces each particle's rotational velocity	1.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Material	The material for the particle	M_Radial_Gradient

Swirl Emitter

Property	Description of Purpose	Value
----------	------------------------	-------

Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)

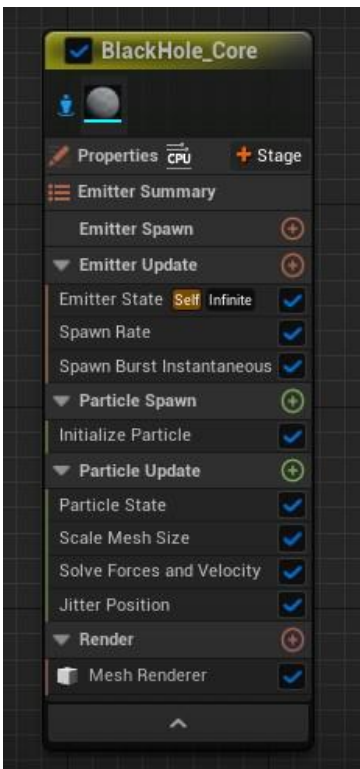
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	5.0
Spawn Burst Instantaneous (Spawn Count)	Spawns a burst of particles instantaneously	1
Apply Force to Velocity	Converts the Physics Force value generated by force modules and applies it to Particle Velocity.	true
Apply Rotational Force to Rotational Velocity	Converts the Physics RotationalForce value generated by rotational force modules and applies it to Particle RotationalVelocity.	true
Lifetime Mode	Lifetime of the particle	Direct Set (1.5)
Color Mode	The color of the particle	Random Hue/Saturation/Value : R 0.0, G 41.186333, B 1000.0, A 1.0
Alpha Scale Range	The random range of the color's alpha	X 0.8 Y 1.0
Mass Mode	The mass of the particle	Unset
Sprite Size Mode	Determine the size of the sprite of the particle	Uniform (120.0)
Mesh Orientation Relative Sprite Facing Vector	Adjust the orientation of the particle of the Sprite relative to the mesh	X 0.0, Y 0.0, Z 1.0

Mesh Orientation Relative Sprite Facing Vector	Adjust the orientation of the particle of the Sprite relative to the mesh	X 0.0, Y 1.0, Z 0.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Dynamic Material Parameters	Control the strength, pan and the x and y of the swirl	Swirl 0.001, Pan 2.0, X 1.0, Y 1.0

### Niagara System / Emitters Breakdown



This is the overview of the emitters. There are a total of 4 emitters and it will be explained one by one.



## BlackHole\_Core

This emitter emits the core part of the black hole projectile. It uses the sphere fresnel material that I made and turns it into a black hole core. It has a dark colour in the middle with bright blue outlines. In this emitter I basically put it so that it emits in a specific spot and then added jitter into it so that it appears as a couple of spheres overlapping each other. This creates a cool effect of a black hole core.

This emitter is responsible for the particle spread of the black hole core. This emitter takes the shape of a sphere and then creates a spread of particles. This particle also has the Curl Noise Force and the Drag node to add into the unpredictability of the particle itself.



## Core\_Particles

This emitter is responsible for the particle spread of the black hole core. This emitter takes the shape of a sphere and then creates a spread of particles. This particle also has the Curl Noise Force and the Drag node to add into the unpredictability of the particle itself.



## Ring\_Particles

The ring particles emitter is responsible for the ring outline of the swirl effect of the black hole projectile. It has similar properties as the Core\_Particle emitter but the difference is the shape on where it is being emitted which is the Ring / Disc shape.



This swirl emitter creates the swirl effect of the ring in the black hole particle. This has the Align Sprite to Mesh Orientation node to align the sprite itself to the other particle mesh. This also has Dynamic Material Parameters so that the material that is stored in this particle can be controlled dynamically in the outlier of the particle.

## Niagara Particle Effect 2 - RockTrail

### Overview of Effect

Basically, this effect is present to showcase the trail behind a character. Originally I wanted to add this rock trail to the player character but since the player is in first person, it is hard to tell that the effect is present. Therefore, I put this effect on the enemies. The impact of this effect is to provide more sense of dread and danger to the enemies. The rock trail adds a sense of power to the enemies since having a trail that is as rugged as the rock trail makes the enemy look more powerful when they are walking and turning. The effect itself is made with the SM\_Rock mesh that is already available in the engine.

The effect only has 2 emitters. The first emitter is the Main\_Rock\_Trail which makes the big rocks appear when the enemies are walking. Another emitter is the Small\_Rock\_Pieces which shows the debris of the rocks that are going everywhere when the enemies are walking or turning around. These emitters combined creates the perfect rock trail for the enemies. Even though it only has 2 emitters in contrast with the black hole particles that have 4 emitters, it still creates a very cool rock trail effect that is perfect for the concept that I imagined.

### Effect Description

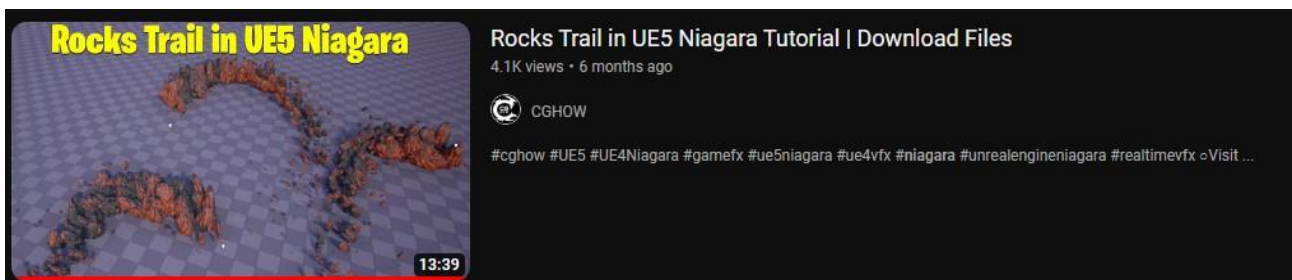
The effect itself has 2 main parts as mentioned before in the overview. It has big rocks and small rocks. The big rocks spawned in a random rotation and it spawned behind the enemy character. It also spawned in the height of where the ground is so it looks like it spawned from the ground. The small rocks spawned on the top part of the particles. It has a wider spread and looks like debris that was spreading everywhere when the rocks spawned.

Inspiration / Reference Images:



Source : Avatar the Last Airbender (TV Show)

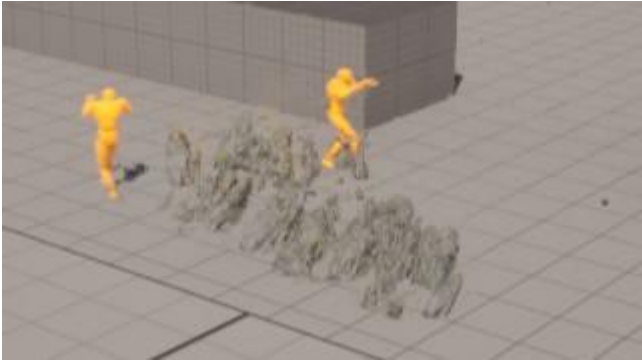
The first thing that I reference this particle from is the TV show “Avatar the last airbender.” In the TV show, there are people that can control rocks and things which are called the EarthBenders. I based my particle highly out of these EarthBenders techniques especially when they bring out rocks from the ground. It is worth noting that I referenced it heavily from the TV show not the live action movie since the movie didn’t portray any powerful feats of the earthbenders and it took them like 7 earthbenders to move 1 single rock.



Source : <https://youtu.be/a2vyMVQmx3E>

This youtube video helped me in the process of making the rocks trail for my video game. I adjusted the values of the trail so that it matches the aesthetic and the visual aspect of it that I wanted.

In-Engine Screenshots:



This is the particle in action. It can be seen that there are the big rocks and the small rocks spawning. It can also be seen that it disappears after a couple of seconds the enemy AI moves since there is a specific lifetime of the particle effect. I also grabbed this screenshot when in play mode because in the niagara particle itself, it appears as nothing since there are no movements on the preview. This means the particle only appears when the object that it is attached to is moving.

Properties and Values

Main\_Rock\_Trail Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Spacing	Spacing between spawned particles in units (cm)	20.0



Max Movement Threshold	If the emitter speed is greater than this value in a single frame, stop spawning particles on that frame	5000.0
Movement Tolerance	If the amount of movement on a given frame is below this threshold, don't spawn any particles. Prevents small emitter movements from spawning particles.	1.0
Lifetime Mode	Lifetime of the particle	Random (Min (0.9), Max (1.0))
Color Mode	The color of the particle	Direct Set : R 1.0, G 1.0, B 1.0, A 1.0
Mass Mode	The mass of the particle	Random (Min (0.75), Max (1.25))
Mesh Scale Mode	Determine the scale of the mesh of the particle	Random Non-Uniform (Min (0.2, 0.2, 0.4), Max (0.3, 0.3, 0.5))
Shape Primitive	The shape of what the particle will spawn into	Cylinder
Cylinder Height	The height of the cylinder primitive	0.0
Cylinder Radius	The radius of the cylinder primitive	100.0
Cylinder Height Midpoint	The Cylinder height midpoint. 0 is the bottom of the cylinder, 0.5 is the middle, 1 is the top.	0.5
Initial Mesh Orientation	Align a mesh to a vector or rotate it into a place using the rotation vector.	Rotation Random Range Vector

Rotation random Range Vector	Rotates the orientation quat on any axis. A value of 1 represents a full rotation on a given axis.	Minimum (0.0, 0.0, -1.0), Maximum (0.0, 0.0, 1.0)
Rotation Coordinate Space	Defines the originating space of the vector before being transformed into the destination space	Mesh
Meshes	Inserts the mesh that the particle will appear as	SM_Rock

Small-Rock\_Pieces Emitter

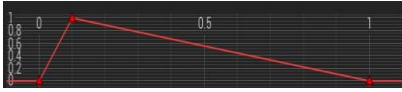
Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self

Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Spacing	Spacing between spawned particles in units (cm)	10.0
Max Movement Threshold	If the emitter speed is greater than this value in a single frame, stop spawning particles on that frame	5000.0

Movement Tolerance	If the amount of movement on a given frame is below this threshold, don't spawn any particles. Prevents small emitter movements from spawning particles.	1.0
Lifetime Mode	Lifetime of the particle	Random (Min (0.9), Max (2.0))
Color Mode	The color of the particle	Direct Set : R 1.0, G 1.0, B 1.0, A 1.0
Mass Mode	The mass of the particle	Random (Min (0.75), Max (1.25))
Mesh Scale Mode	Determine the scale of the mesh of the particle	Random Non-Uniform (Min (0.025, 0.025, 0.025), Max (0.1, 0.1, 0.1))
Shape Primitive	The shape of what the particle will spawn into	Cylinder
Cylinder Height	The height of the cylinder primitive	0.0
Cylinder Radius	The radius of the cylinder primitive	100.0

Cylinder Height Midpoint	The Cylinder height midpoint. 0 is the bottom of the cylinder, 0.5 is the middle, 1 is the top.	0.5
Initial Mesh Orientation	Align a mesh to a vector or rotate it into a place using the rotation vector.	Rotation Random Range Vector
Velocity Mode	The added velocity mode to the particle	In Cone

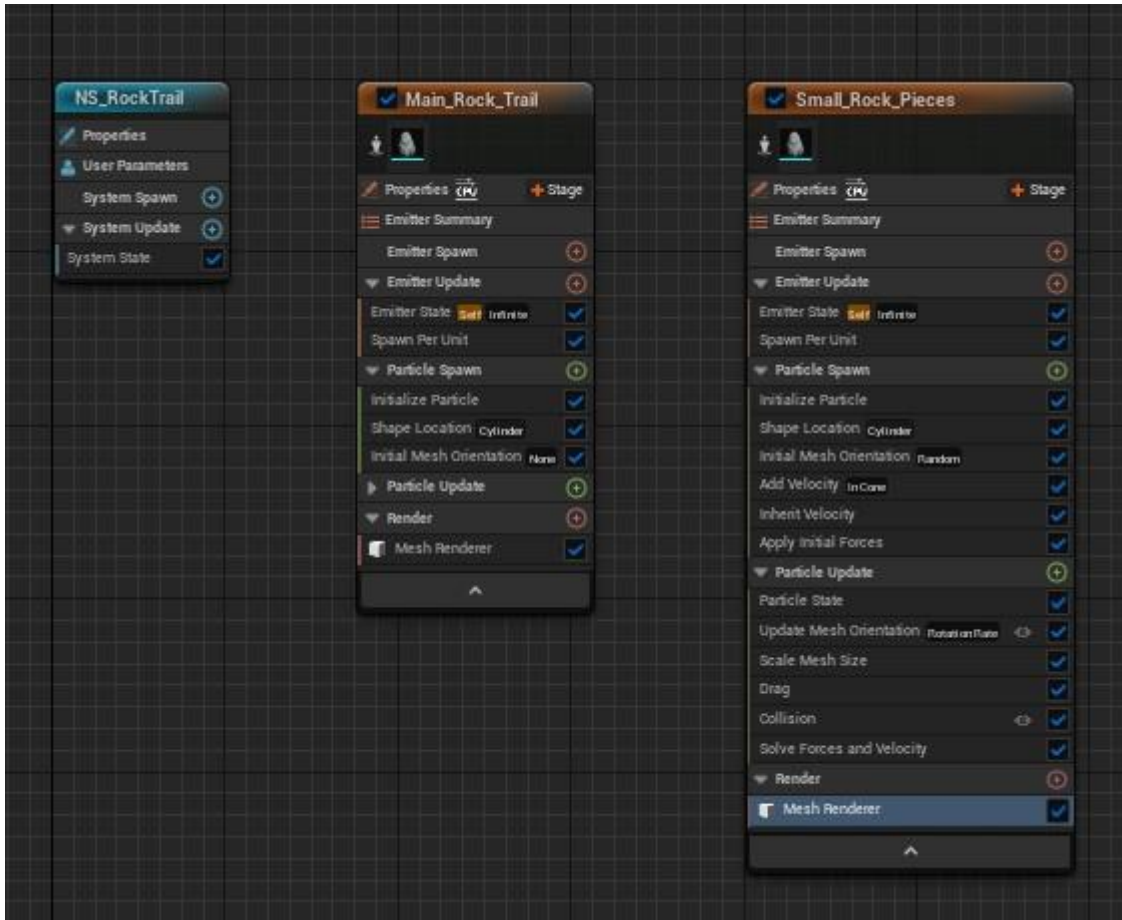
Velocity Speed	The added speed of the velocity of the particle	Random range Float (Min 100.0, Max 400.0)
Cone Axis	Axis of the Cone Velocity Mode	1.0, 0.0, 1.0
Cone Angle	Angle of the Cone Velocity Mode	45.0
Inherit Velocity Amount Scale	A scale factor on the amount of velocity to inherit.	Vector From Float (0.5, Inherit Velocity Speed Limit 100.0)
Apply Force to Velocity	Converts the Physics Force value generated by force modules and applies it to Particle Velocity.	true
Apply Rotational Force to Rotational Velocity	Converts the Physics RotationalForce value generated by rotational force modules and applies it to Particle RotationalVelocity.	true
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Orientation Method	Choose from a selection of options for updating the orientation of mesh particles, including a simple rotation rate, orienting directly to a vector or position in space, flight behavior	Rotation Rate
	such as banking and turn rate, and a simple automatic roll behavior based on the radius of a particle and how fast it is moving.	

Rotation Vector	Rotation per axis in the coordinate space of choosing	Random Range Vector (Minimum (-1.0, -1.0, -1.0), Maximum (1.0, 1.0, 1.0), Evaluation Type : Spawn Only)
Rotation Rate	Scale factor on Delta Time for global speedup (or bypassing) of rotation.	Random Range Float (Minimum -0.5, Maximum 0.5, Evaluation Type : Spawn Only)
Scale Mesh Size	Takes the initial mesh scale as set in the spawn script, and scales it by a scale factor	Vector From Float->Float from curve->Curve for Floats
Scale Mesh Size Curve	The curve for Scale Mesh Size	
Scale Curve	The scale of the curve of the Scale Mesh Size	1.0
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.5
Rotational Drag	Reduces each particle's rotational velocity	5.0
Collision Enabled	Enable or disable the module's affect on the effect	enabled
Radius Calculation Type	Each particle's collision radius will be calculated for you, per frame , using the methods laid out below.	Sprite
Method for Calculating Particles Radius	This enum changes the way that each particle's collision radius is calculated.	Bounds

Particle Radius Scale	This value scales the calculated particle collision radius	1.0
Restitution	This controls the particle's bounce coefficient. 1 will retain all of the particle's energy along the impact normal vector and 0 will remove it.	Random Range Float (Min 0.0, Max 0.3)
Simple Friction	If true, fewer parameters will be used to control the friction coefficient.	true
Friction	The friction coefficient defines how quickly a particle will slow down as it slides across a surface	0.25
Enable Rest State	This will pause particles that penetrate surfaces more often than the specified rate and particles that have penetrated a surface more deeply than allowed.	true
Maximum Penetration Correction	This number specifies the maximum number of units that a particle can be pulled out of another surface before being instantaneously forced into a rest state	0.5
Percentage of Penetration Before	Particles will enter a rest state if they are found to penetrate surfaces more often than this float allows over the "Rest State Time Range"	1.0
Rest State Time Range	This is the amount of time that a particle's interpenetrations will be	0.5
	tracked over when determining if it should enter a rest state.	

Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Meshes	The mesh this particle will take shape as	SM_Rock

### Niagara System / Emitters Breakdown



This is the overview of the emitters. Unlike the black hole particle, this particle effect only has 2 emitters but the complexity of it is quite complex. Again, each emitter will be explained and broken down one by one.



## Main\_Rock\_Trail

This emitter is fairly simple. How it basically works is that it spawns on a cylinder shape and then it spawns whenever it moves. The way for it to spawn whenever it moves is through the Spawn Per Unit node. Then I adjusted the values accordingly to create the optimal effect in the end.

## Small\_Rock\_Pieces

This emitter is the complicated emitter basically portrays the enemies spawns the rock trail. previous emitter and then I Add Velocity, Inherit Velocity, responsible for the offset force spawned. The particle also has to give the collision effect



one out of the two emitters. This rock pieces that fly out when the The way I do this is I duplicated the added a couple of nodes into it. The and the Apply Initial Forces nodes are of the particle when the particle is Drag to help offset itself and Collision whenever it collides with an object.

## Sequencing /

### Overview of Sequence

Basically the cinematic that I level design overview of the the start of the game after the menu. There is a bit of a to give the player a clear view to do this cinematic is because the player if the player doesn't know where to go or what the level looks like. By giving the player a preview of the level, it can increase the motivation of the player to finish the level and it will make the game more immersive.

## Cinematic

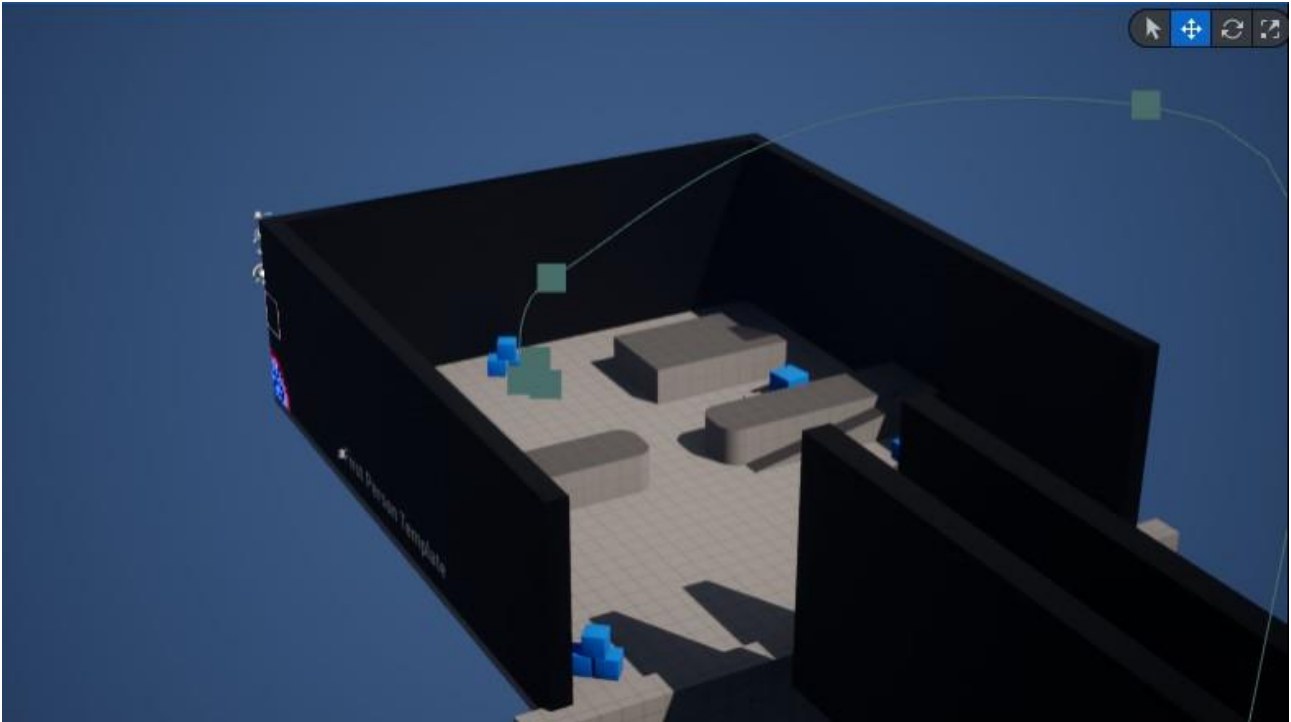
have in my game portrays the whole game. The cinematic takes place at player presses start on the main distance from the camera to the level of the levels. The reason why I chose a parkour game tends to be tricky for



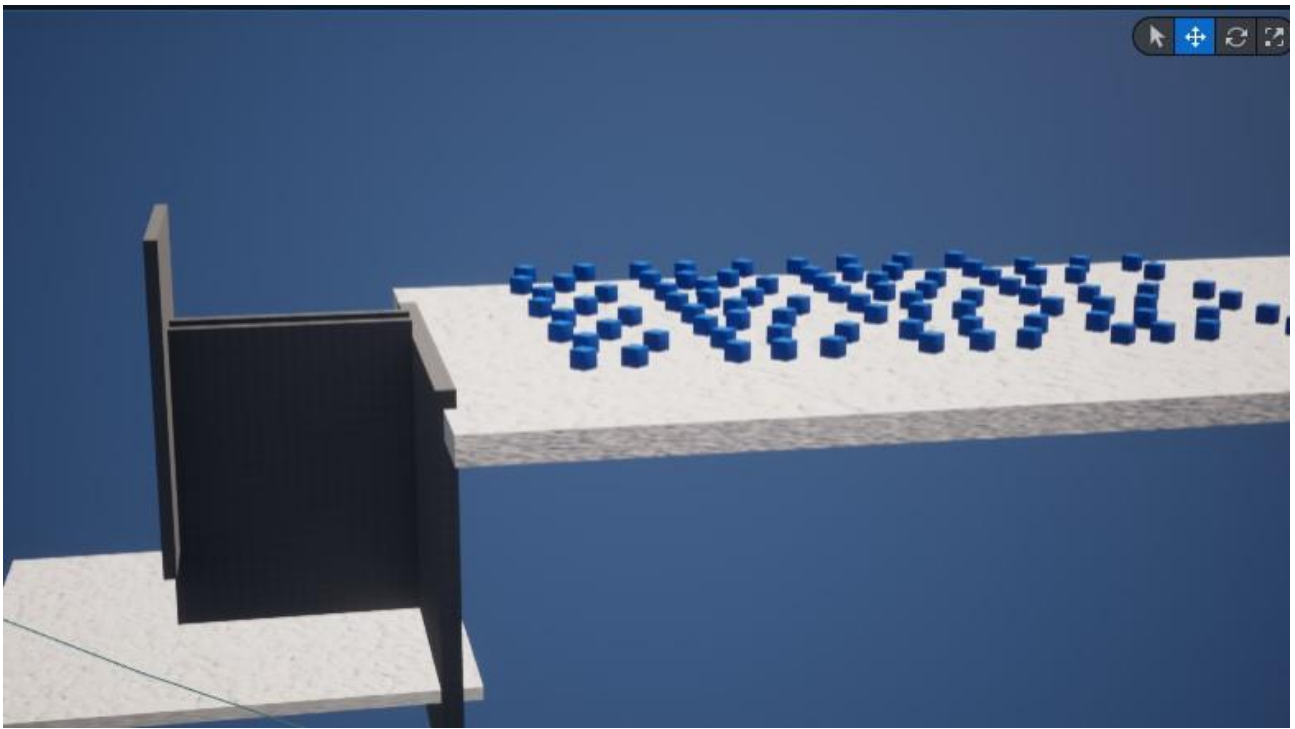
There is also a custom event that happens on the cinematic which is the enemy AI shooting when the cinematic is playing. This is to give a preview of what the Enemy is capable of and give the player a sense of danger and caution when approaching the level where the enemy AI is present. Even though this increases the predictability of the level, it also decreases the stress of the unknown of the player because sometimes in games, especially parkour games, it is better to provide the player with clear expectations of the level rather than relying on the element of the unknown.

### **Camera Angles / Properties**

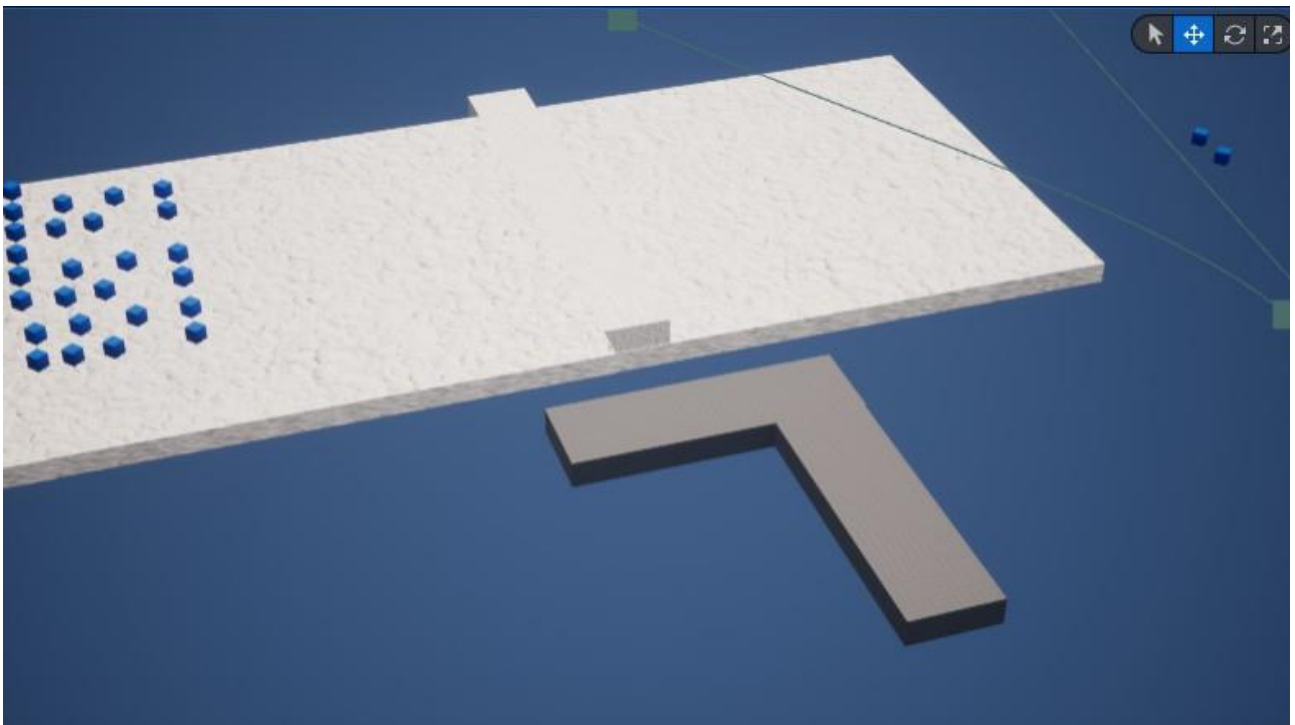
There are several keyframes that I made in order for the cinematic to work. These are the keyframes that are in the sequence with the description of it :



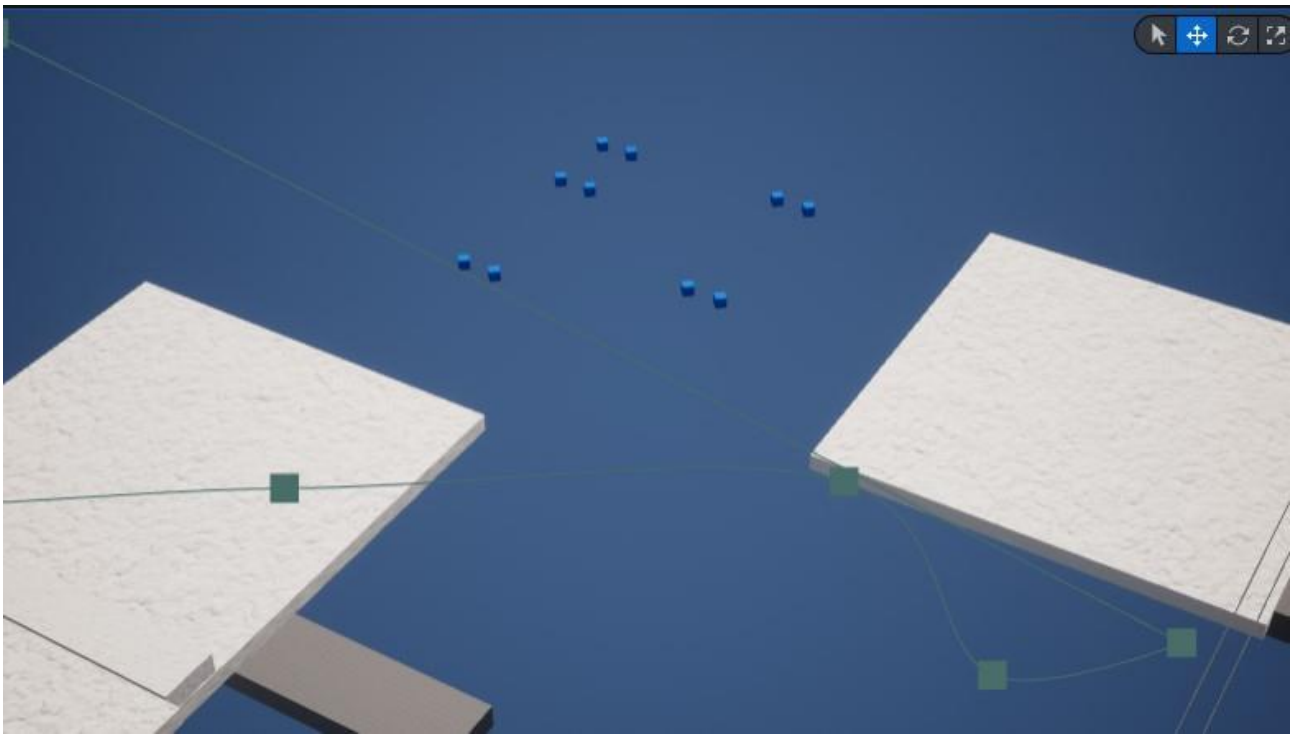
This is the first camera angle of the cutscene. The reason why I put it here is because it provides a great overview of the first part of the starting level of where the player is.



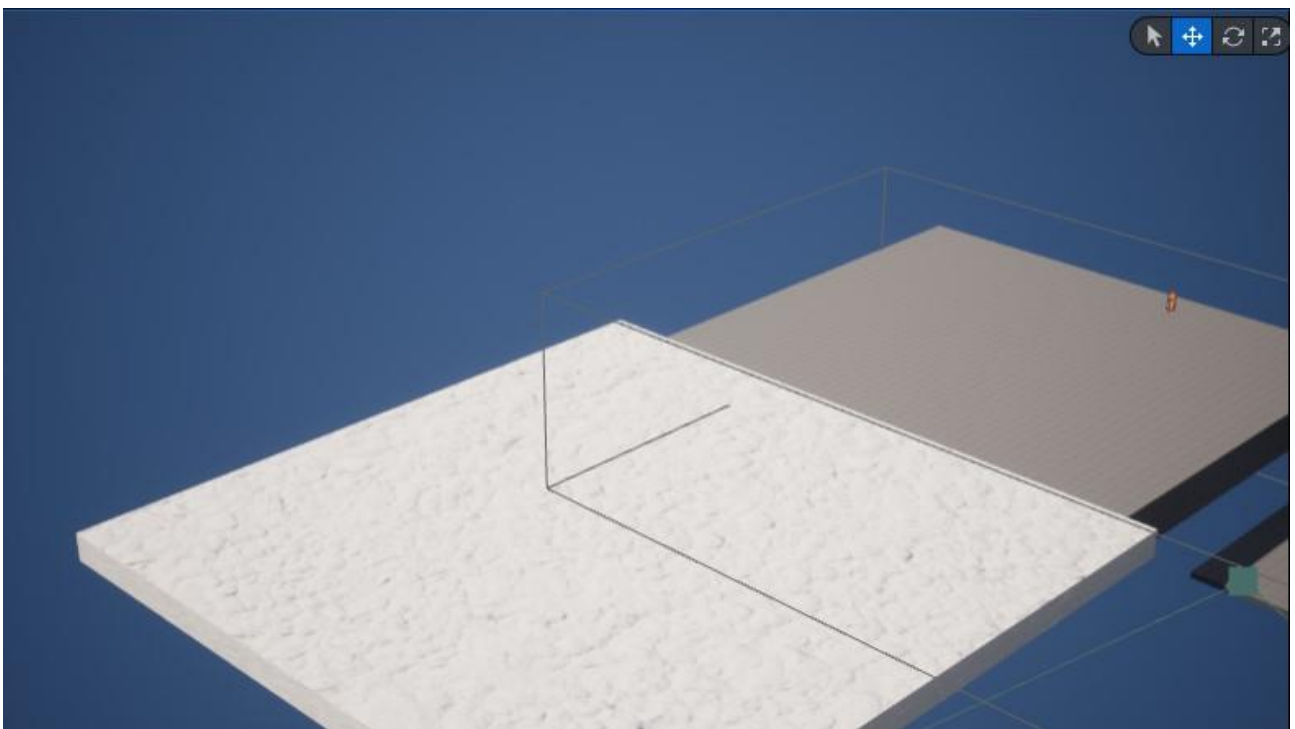
This is the 2nd angle of the camera. This is in a slightly different position than the last angle but it is rotated to another area. The reason why I chose this angle is because it is a good overview of another section of the level and while the camera is rotating it also showcases the first wall running level.



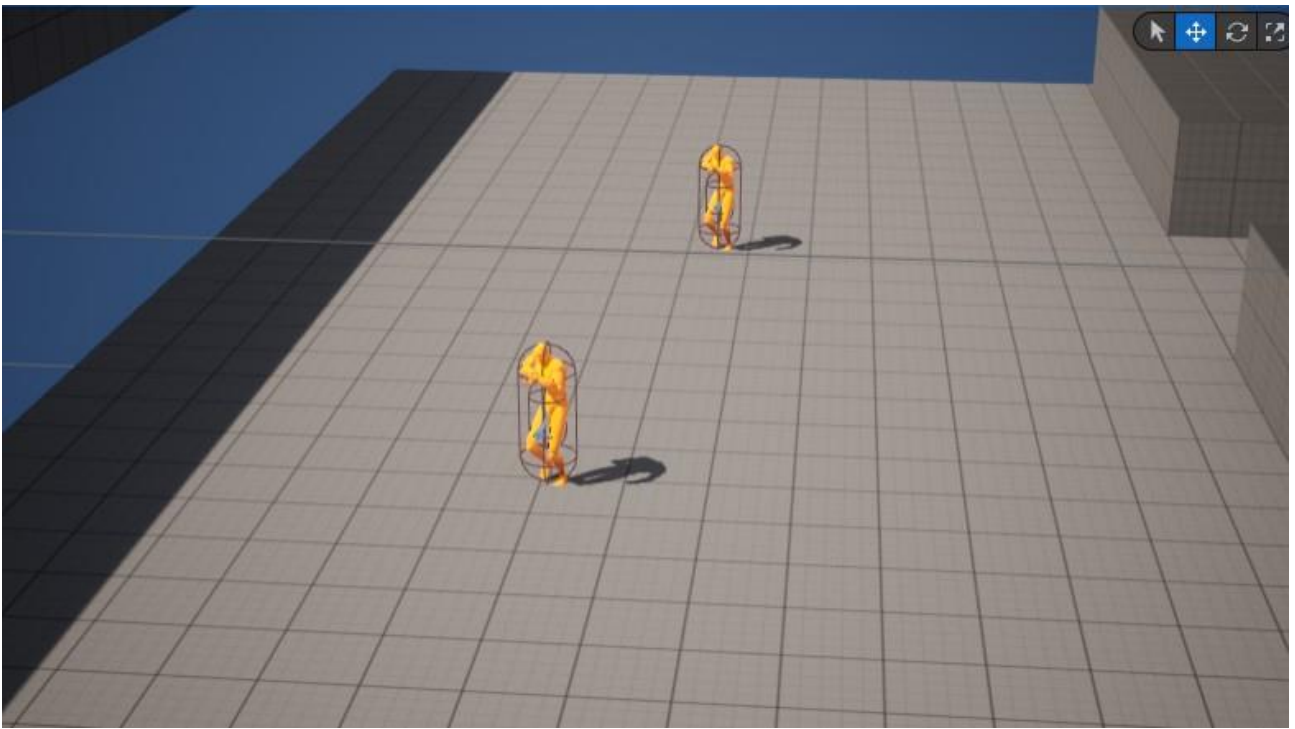
This camera angle is chosen because it showcases the next section of the level which is the invisible ground section. It also shows the hint of the level (The shape under the ground level) and the level itself.



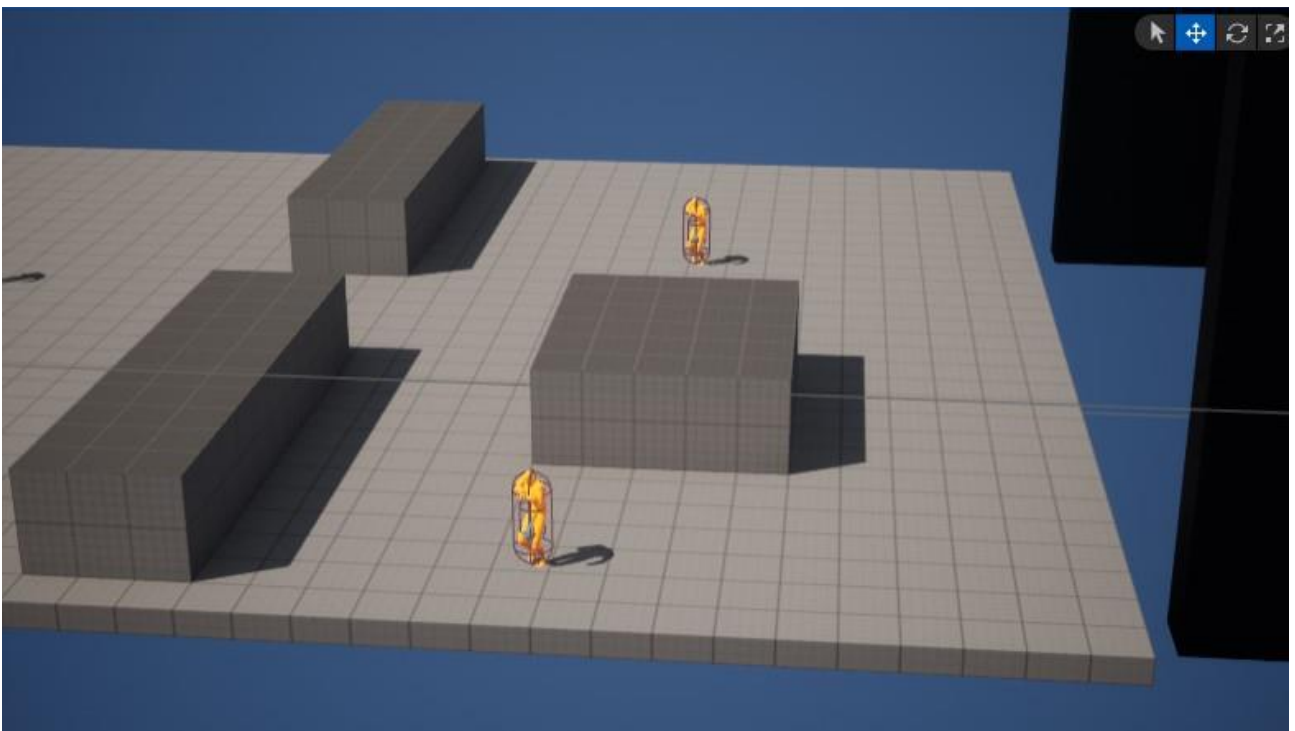
This camera angle provides a clearer view of the invisible ground level overall since it shows the start and the end of the invisible ground level.



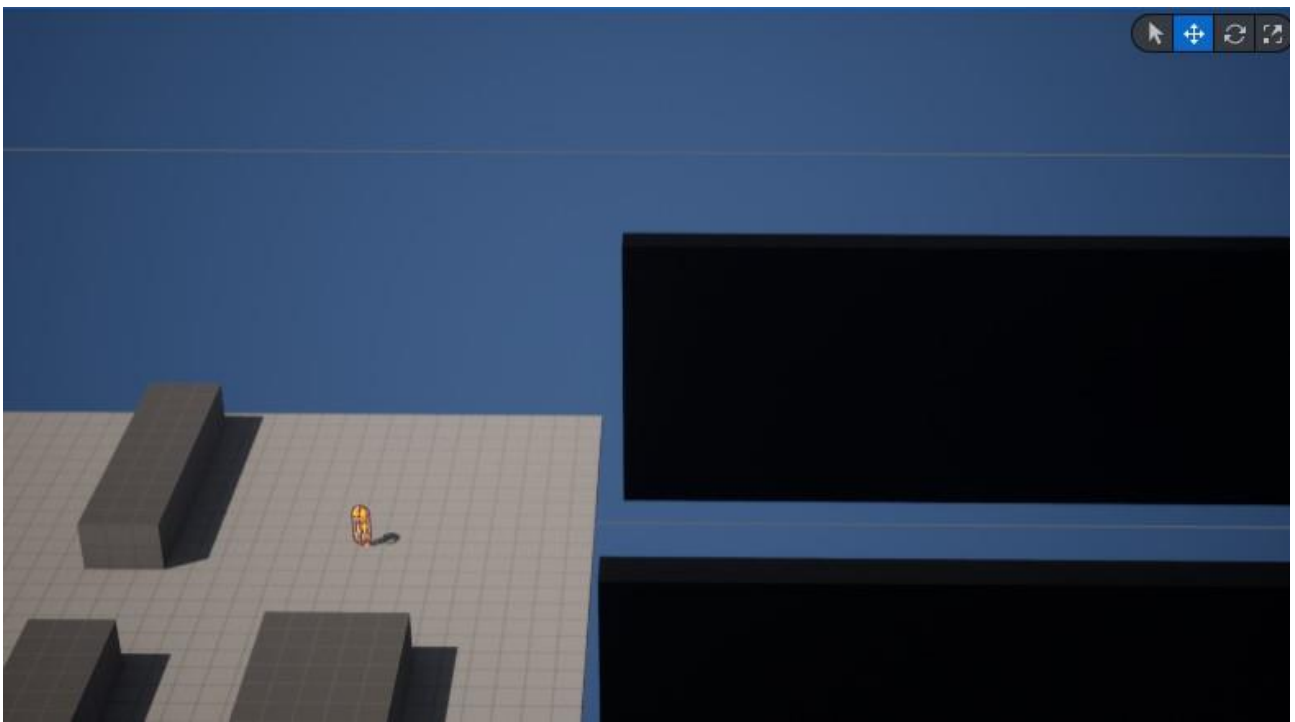
This camera angle shows the end of the invisible ground level and shows the next part of the level which is the AI enemy levels.



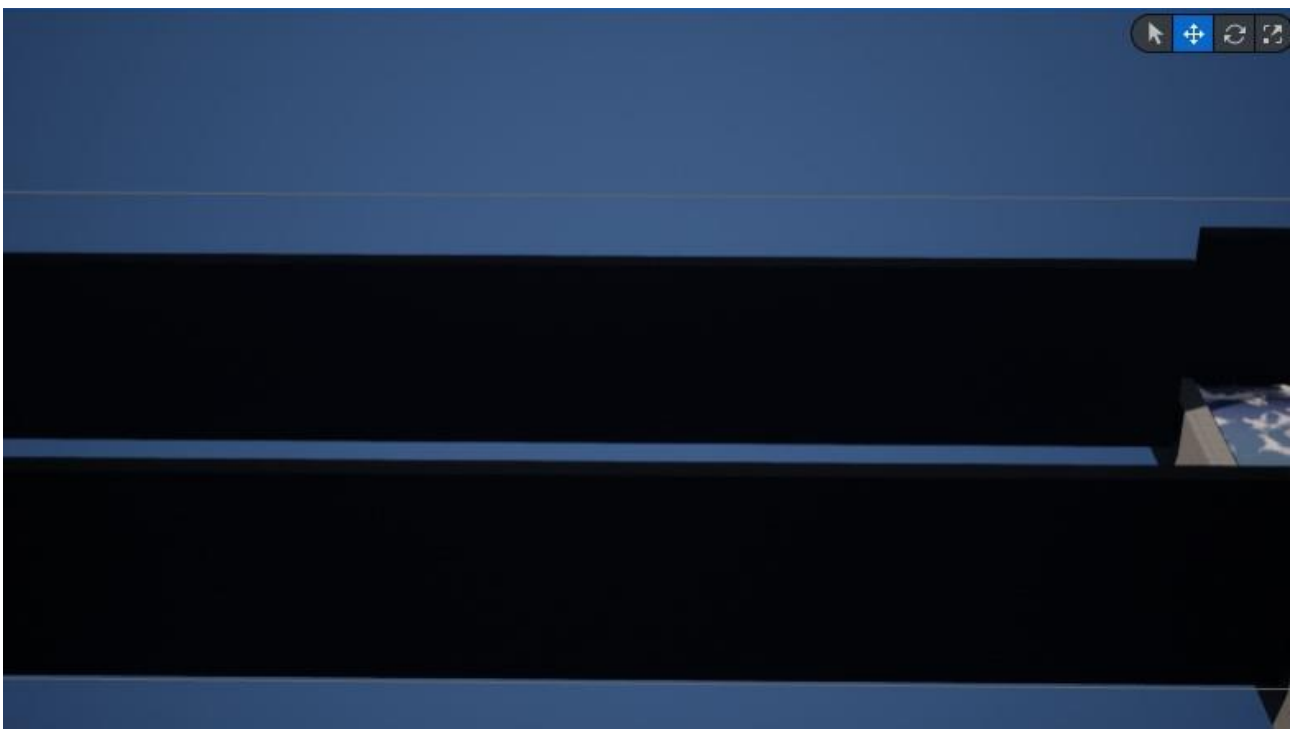
This is the first part of the AI Enemy level and it is located up close to the enemy AI. The enemy AI will also move while the cutscene is running, showing the patrolling nature of the AI.



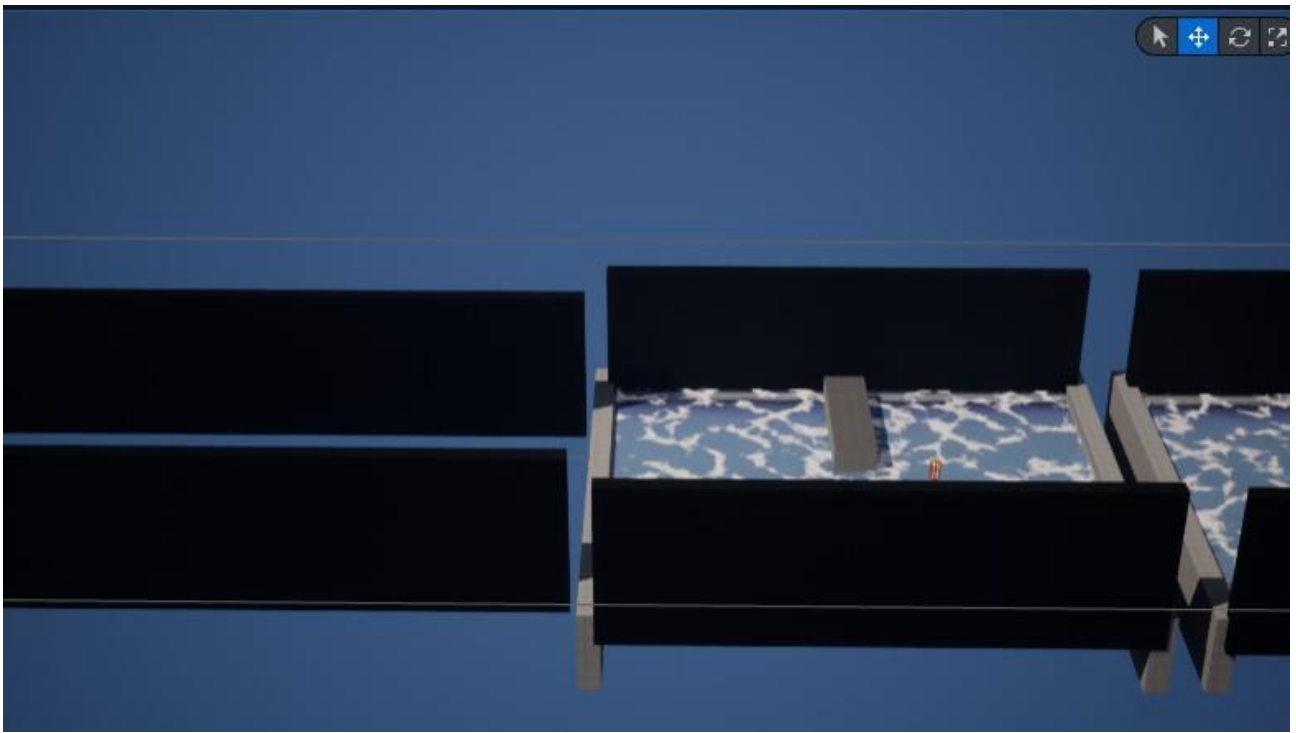
This camera angle is similar with the previous one but it is in a slightly different location but on the same level ground of the enemy AI level.



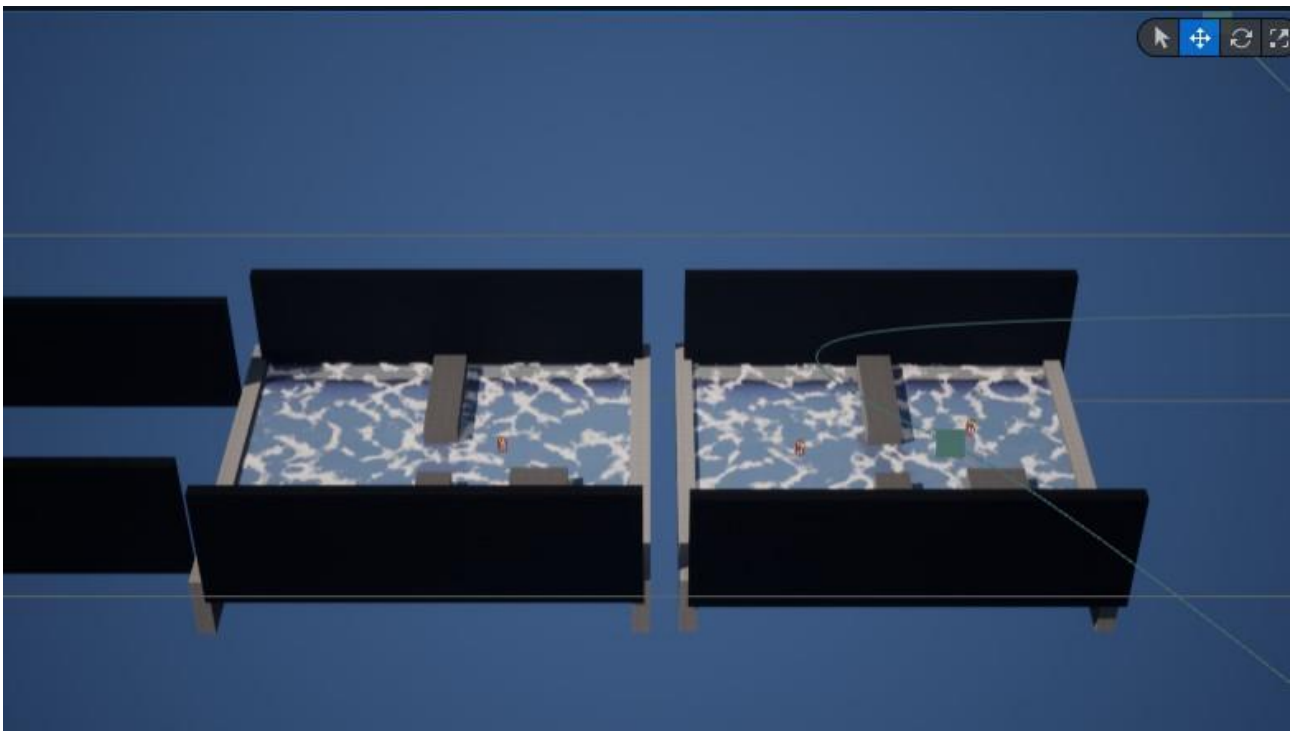
This keyframe shows the transition between the first part of the enemy AI level to the next enemy AI level but through another wall run level which is a sub level.



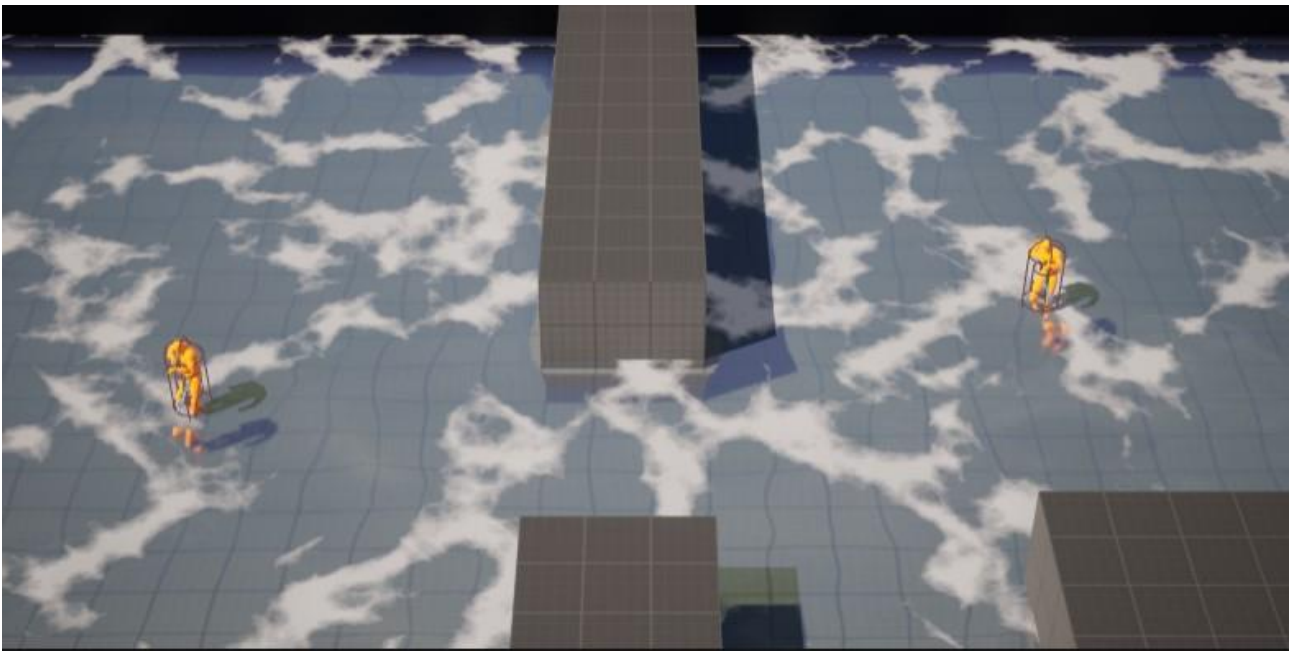
The wall run part of this level is quite long so the key frame to showcase how long the wall run part is needed.



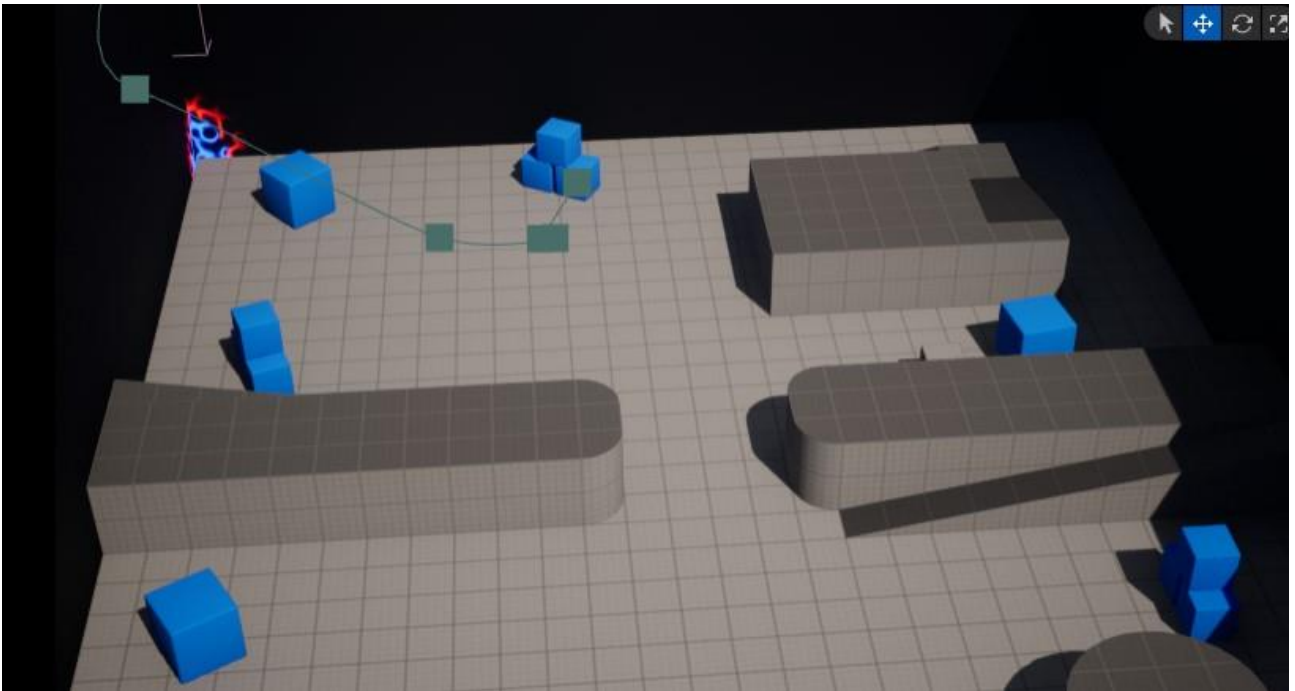
This keyframe showcases the next Enemy AI level which I call the “Enemy on Water.” This shows the first part of the Enemy On Water level.



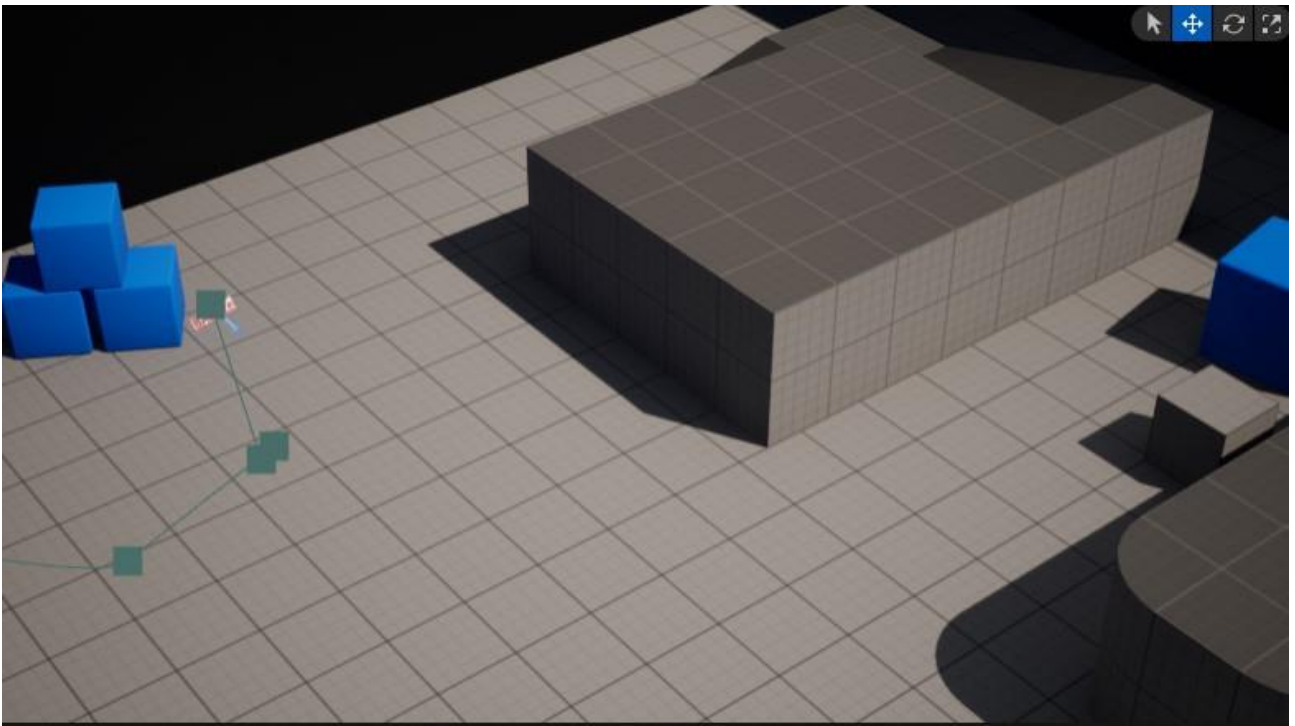
This keyframe showcases the whole Enemy On Water level which is separated into two parts.



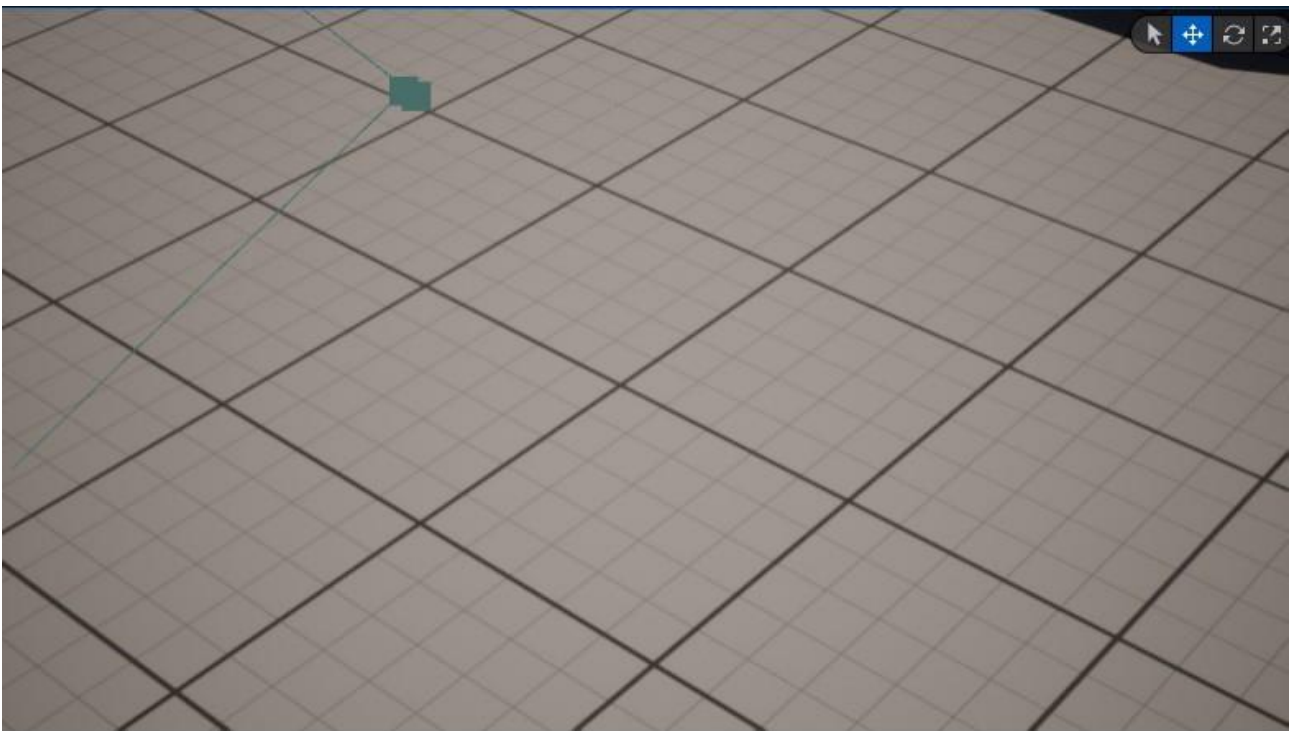
This keyframe shows the Enemy On Water level as well but with more depth into it. It shows the enemies with a more up close angle.



This keyframe shows the start of the level again since after the cutscene ends, it goes back to the start of the level.

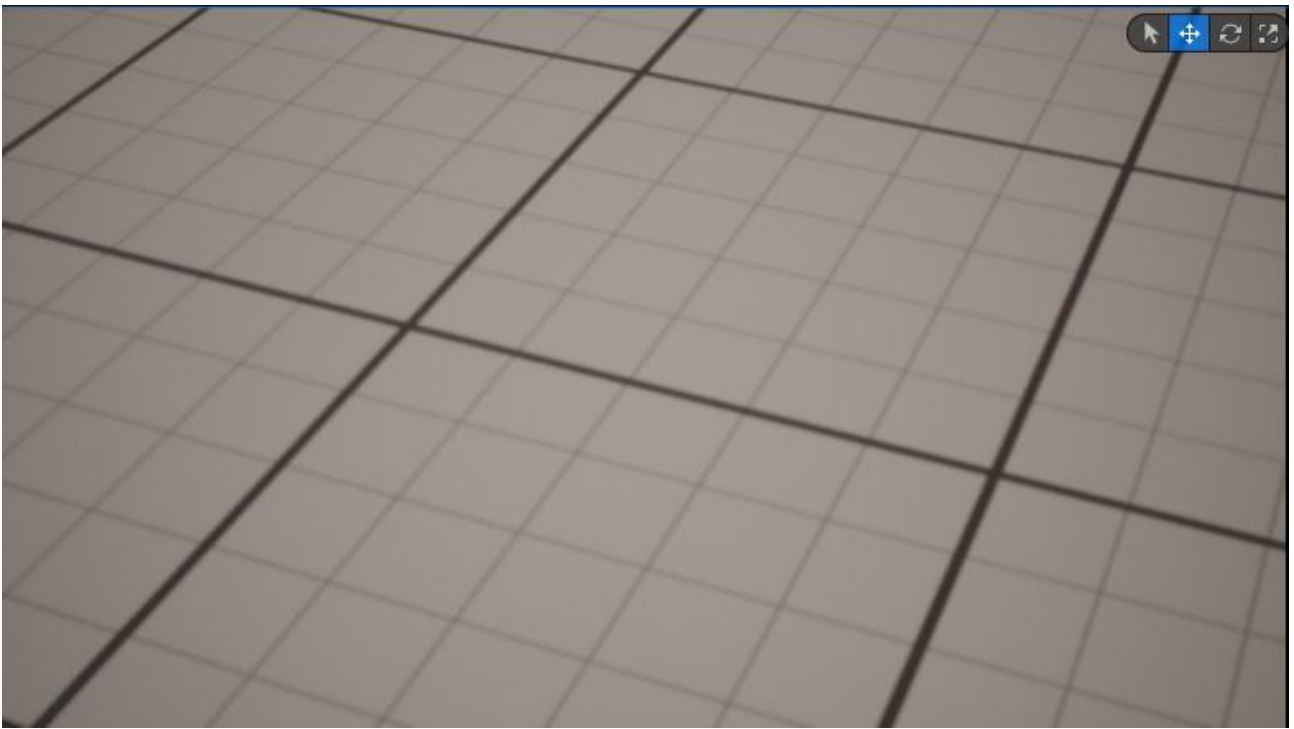


This keyframe zooms in to the spot of where the player is since the end goal of the cutscene is to go to where the player is facing and then transitions into it.



To avoid gimbal lock, this keyframe is needed before rotating the camera into where the player's starting point is facing.





This is the continuation of the previous keyframe and it is done to avoid gimbal lock.



This keyframe marks the start of the camera rotation to where the player is facing on the starting position.



This keyframe is the last keyframe of the cutscene. It captures where the player is facing when the player will start playing and it will transition to where the player can take control of the character.

### **Scripted Events**

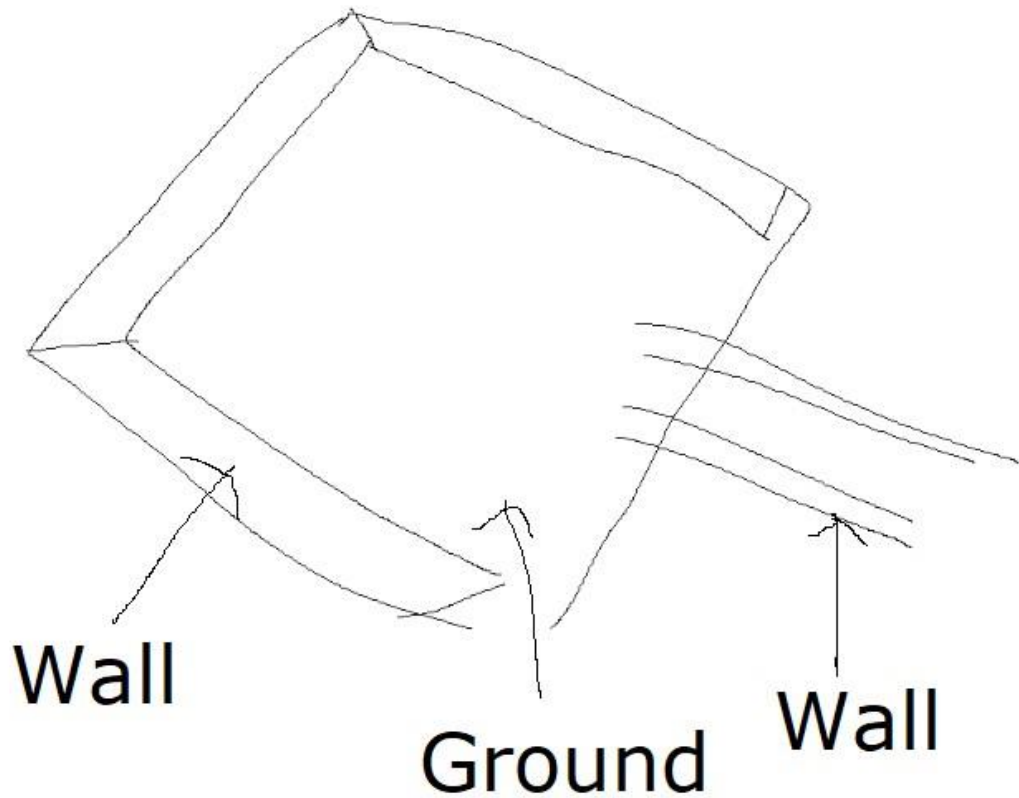
There is only one specific scripted event that happens during the cutscene. That scripted is the enemies shooting. The reason why I chose this scripted event is as mentioned before in the overview, it is to show the player what the enemy AI is capable of and to add that sense of danger and preparation to the player. This allows the player to prepare mentally or mechanically to face the challenge of the enemy AI before actually playing the game. I call the event through the game mode class of the game because there are 2 main ways to call scripted events that are through the game mode or a character class since it is called through blueprints.

Basically when the events are called, all the enemy character actors in the game will shoot a projectile in a specific direction. This happens when the cutscene shows the enemy AI and only happens in those instances. The function that was called is present in the EnemyAIController class since it is part of the AI mechanic called ShootPlayer.

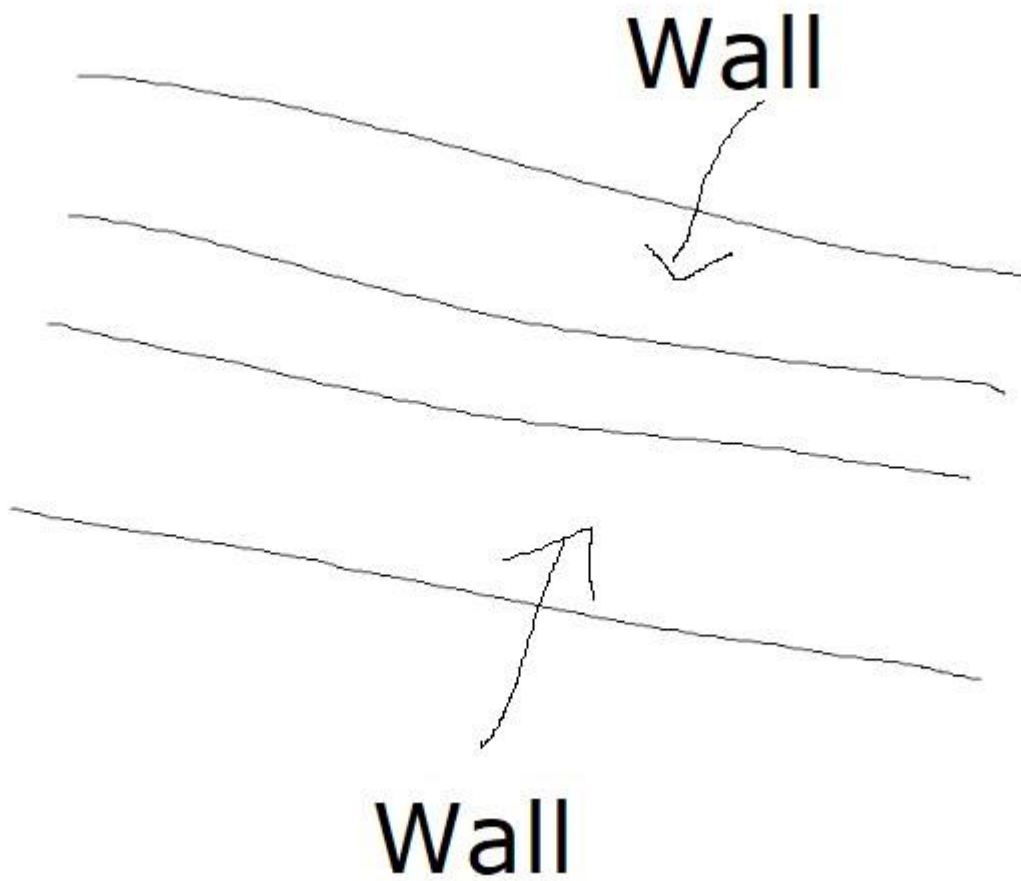
There are specific values used in the ShootPlayer function. First it takes an EnemyProjectileClass value which is declared as a TSubClassOf<AEnemyProjectile>. After that it gets the Pawn of the character. Then, it declared an ACharacter variable named PlayerCharacter which gets the player character from the world. Then it goes to the if statement to make sure the PlayerCharacter is not null then gets the CurrentLocation of the enemy AI. Then it also gets the ShootingDisplacement of the spawning of the projectile. After that, it gets the PLayerLocation of the player and the DirectionToPlayer. This is done to get the distance from the enemy AI to the player.

After that, it gets the rotation of the DirectionToPlayer to make sure the rotation is correct. Then it rotates the ShootingDisplacement Vector and then gets the SpawnLocation of the projectile. Lastly, it calls a variable called SpawnedProjectile with an AEnemyProjectile\* variable prefix and then cast it into AEnemyProjectile and Spawn the actor into the world on the SpawnLocation based on the previous location and rotation initialization. Then, the EnemyProjectile has a separate code which contains a simple projectile code to push the projectile forwards.

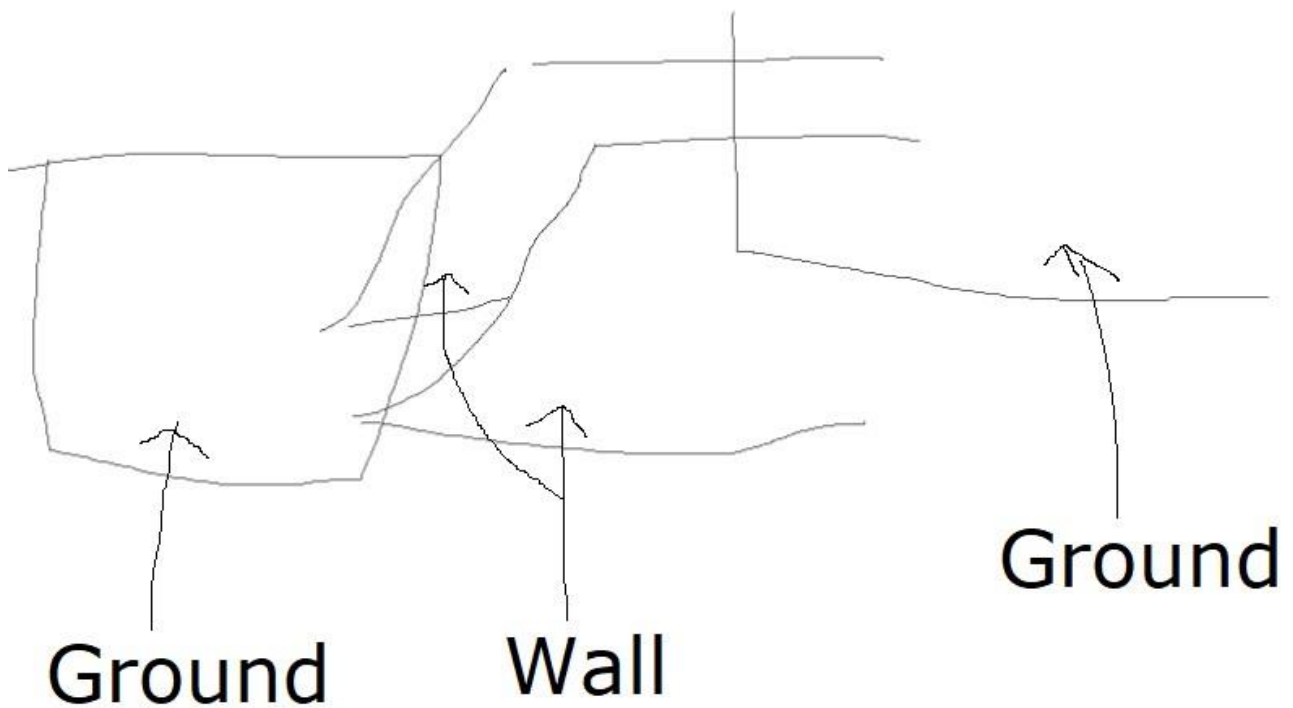
### Storyboard



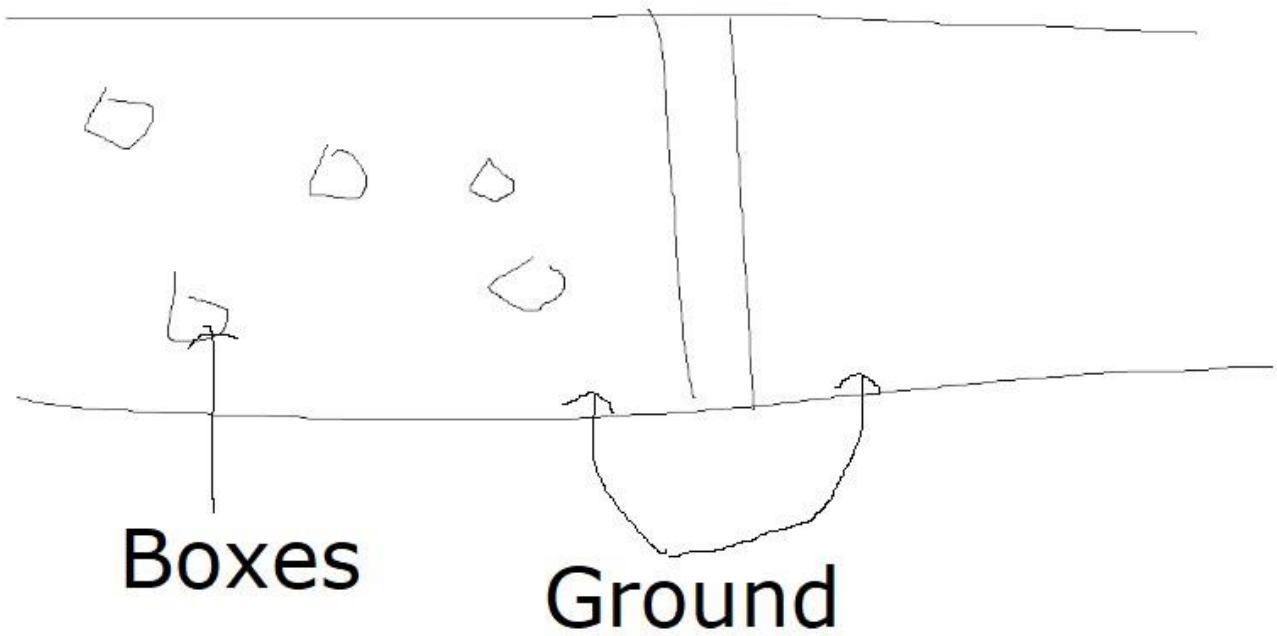
This is the drawing of the starting frame. This shows the start of the level which is indicated by the square. It also shows part of the next wall run part of the level.



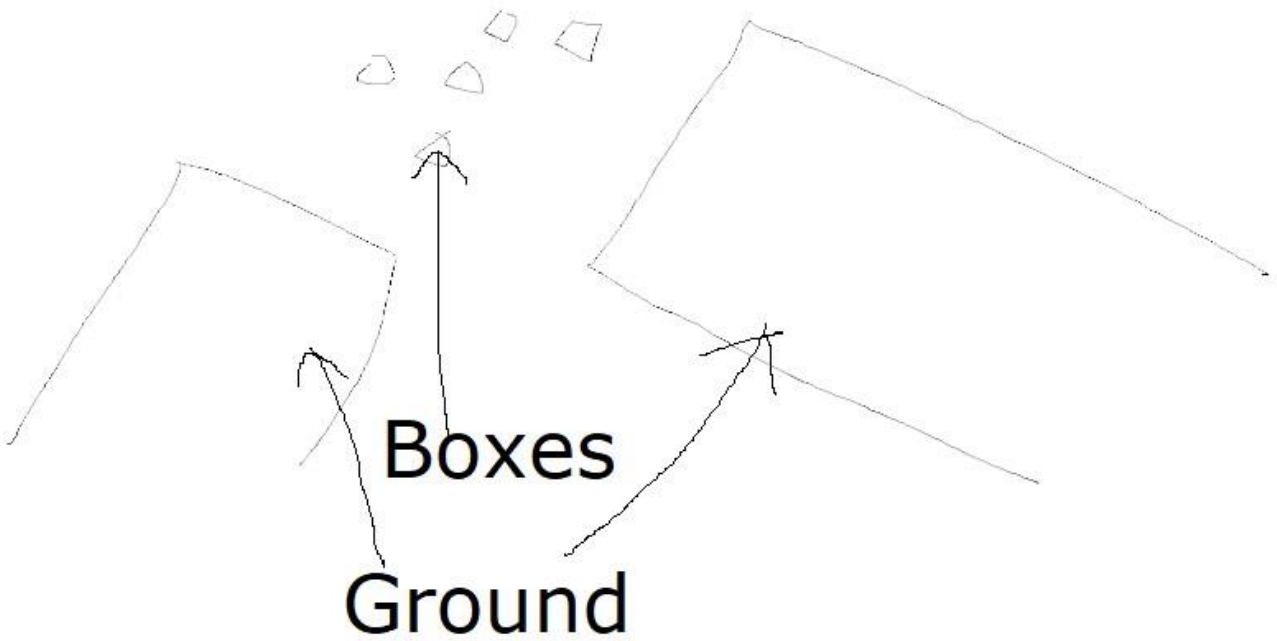
This is the wall run part of the level. The line shows the wall that exists in the next section of the level.



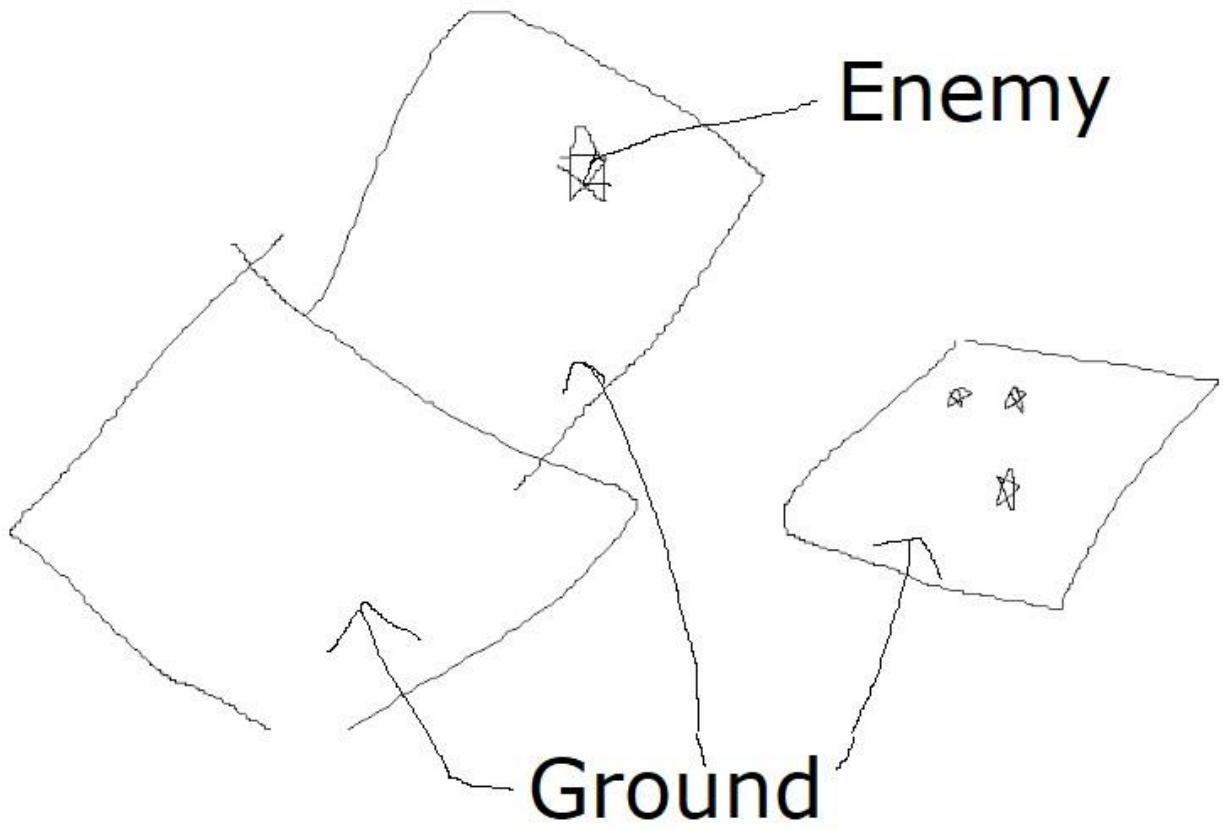
This is the next section of the level. This drawing shows the transition of the player going up the wall to get to the level located on the top part of the level.



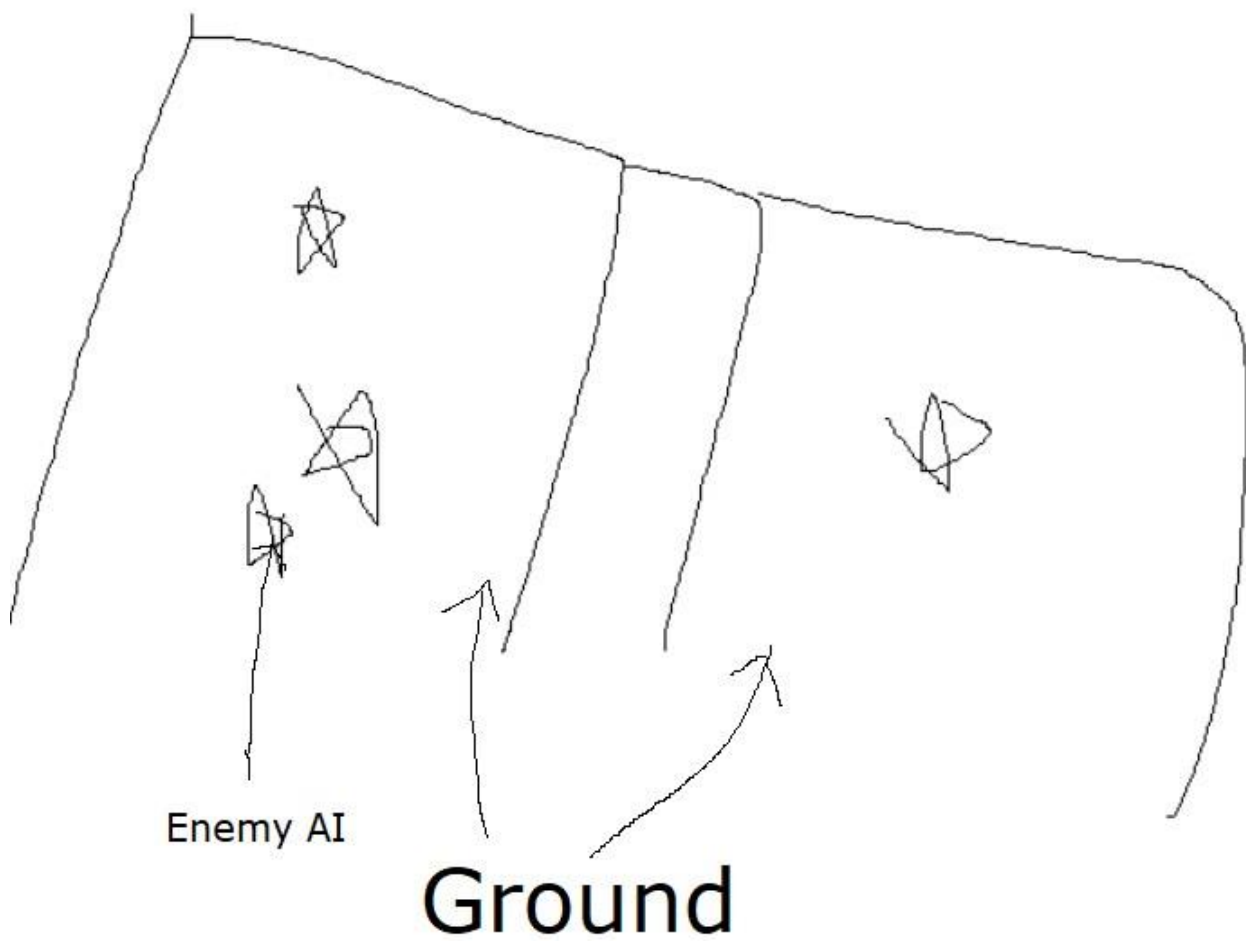
This part of the level shows another part of the level which is the boxes level. The box's level is the one where the player can test out the gravity gun.



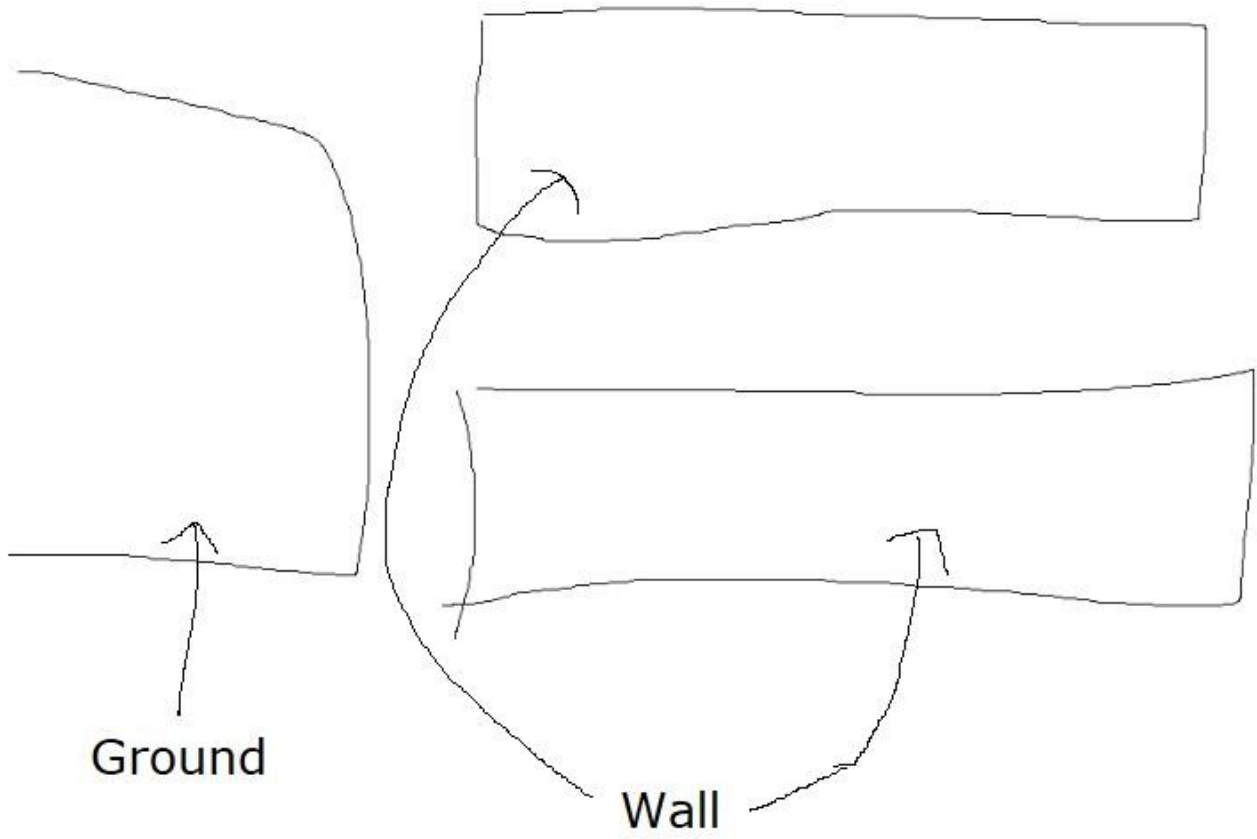
This part of the cutscene is the one where it showcases the transparent floor level part of the game.



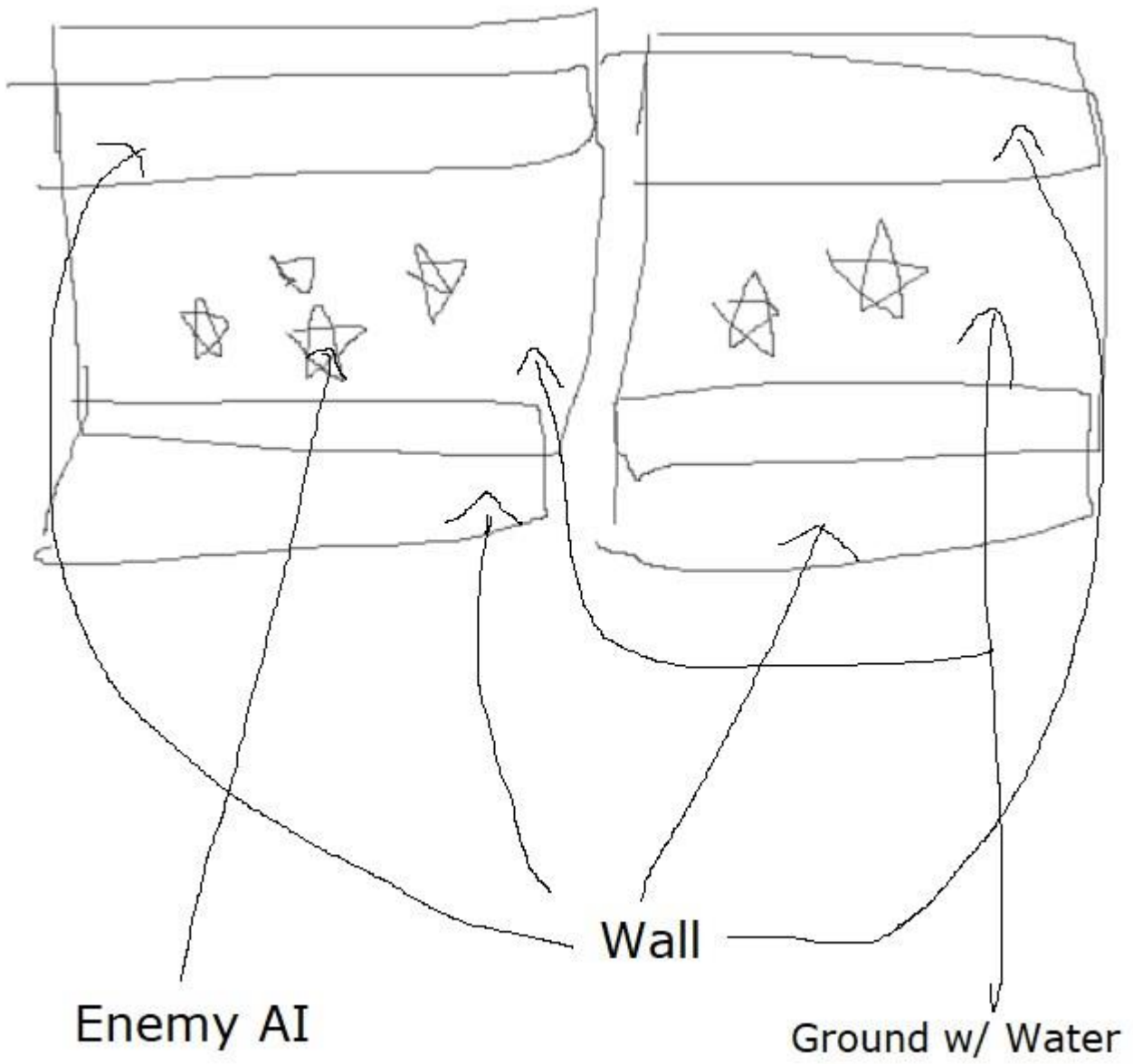
This is the part of the cutscene where it is approaching the enemy AI level. The stars symbolise the enemy AI.



This is the close up shot of the cutscene where it showcases the enemy AI.

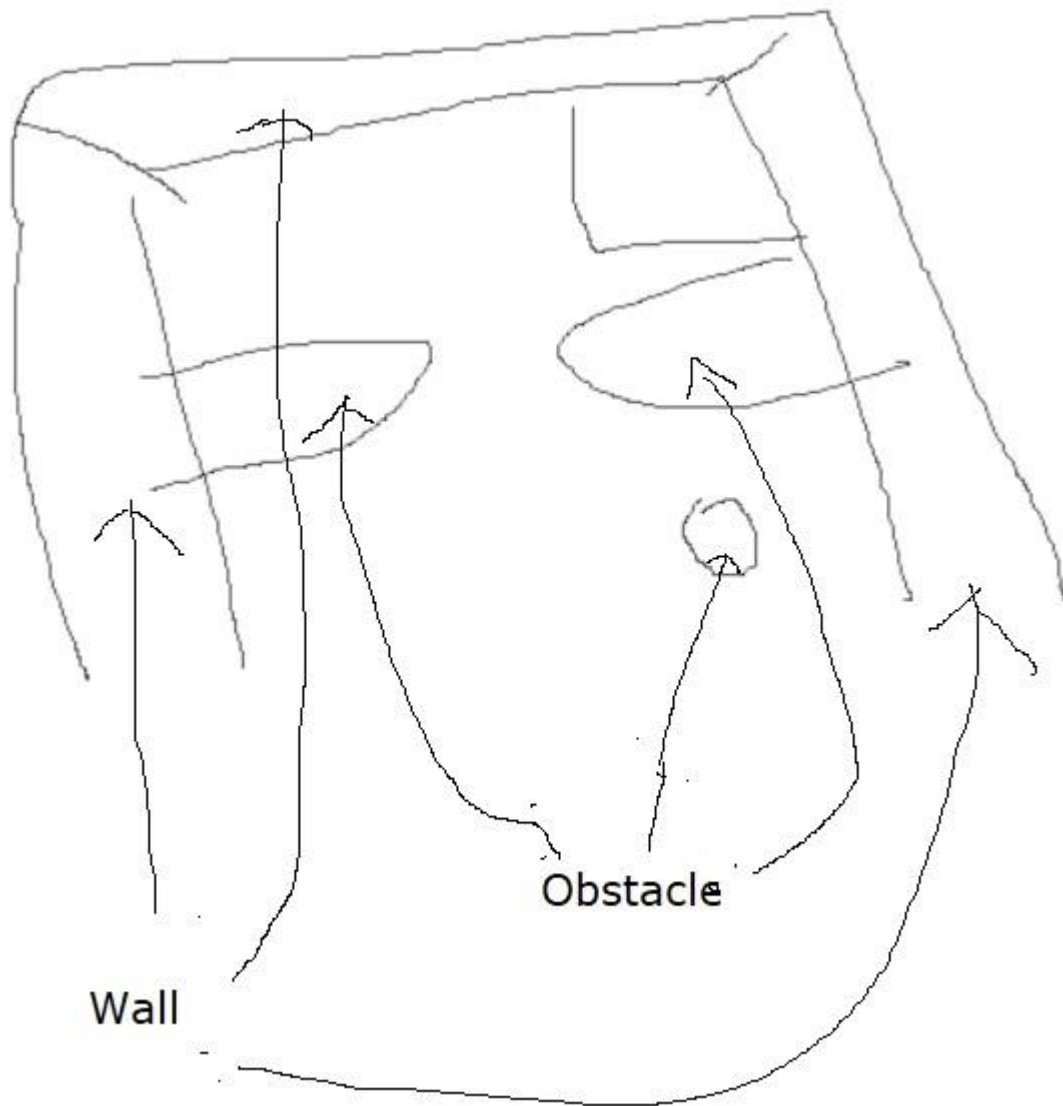


This part of the cutscene shows the next wall run part of the level.

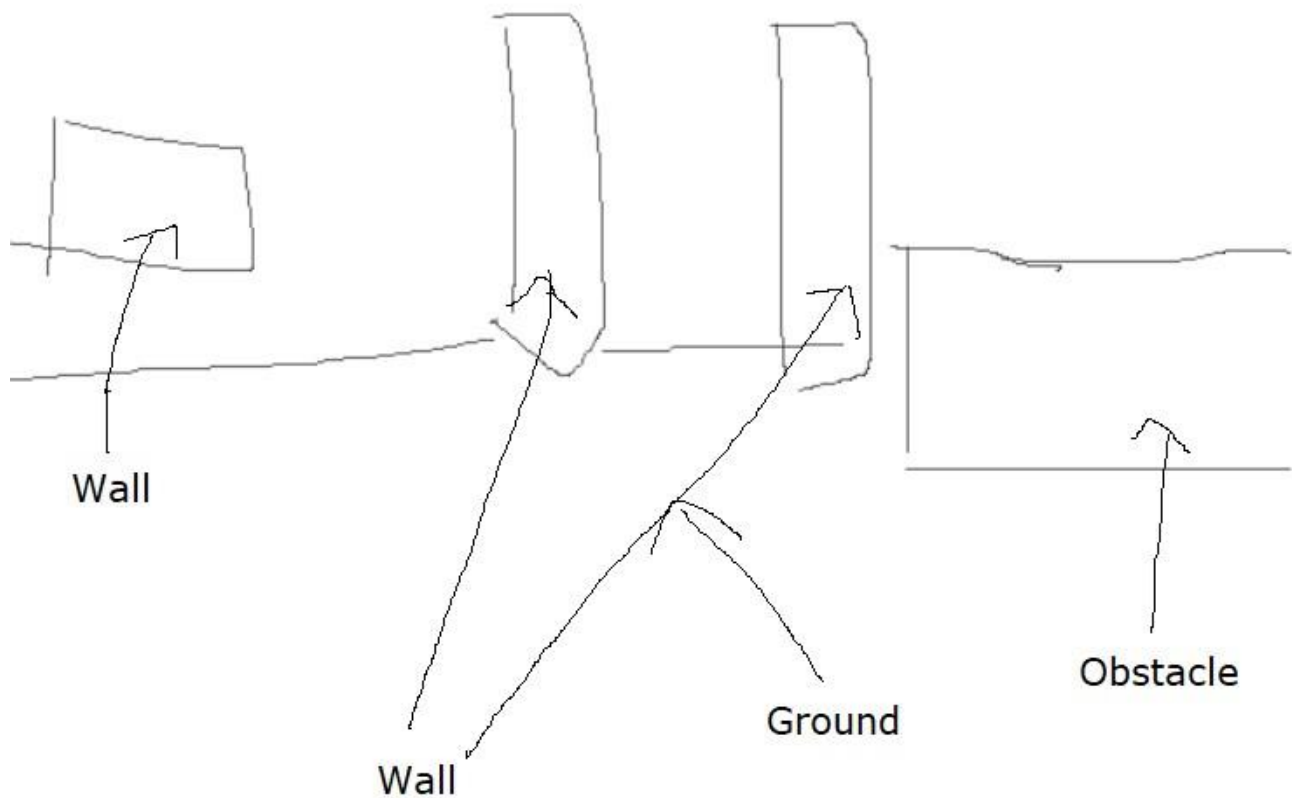


This part of the cinematic showcases the level on which it shows the enemies on the water level.





This keyframe part of the sequence shows the beginning of the level again but in a more close up manner.



This shows the fps preview of the player when the player is starting. It also marks the end of the cinematic.

## MetaSound

### Overview of Sound Effect

So the sound effect that I have in the meta sounds is the combination of three different sound effects. The three sound effects are the jumping sound effect, the wall run sound effect, and the shoot sound effect. It switches the sounds based on what happened with the player. When the player jumps, the sound will trigger once that will indicate that the user is jumping. The shoot sound effect also triggers once after the player shoots once. The wall run sounds a bit different. It repeats until the player ends the wall run. These sounds are implemented because it adds impact to the player when the player is playing the game. Sounds are one of the most important aspects of the game since it adds the immersivity to the game and by adding these sounds to the game, it makes the game feel more complete.

When the player does the things that the sound is implemented in which are the wall run, shoot, and the jump action, the player will feel more immersed when doing those actions and it will make the actions more obvious in terms of using it. The nodes of the meta sound use several triggers to trigger the sound differently based on which actions the player is doing.

### Effect Description

Basically the sound effects that I implemented are three different sound effects combined into one meta sound. This is to make for easier development. As mentioned before in the overview, the sounds are the

Jump sound effect, the shoot sound effect, and the wall run sound effect. The wall run sound effect uses the VR\_Object\_Grabbed\_Loop sound from the engine since it sounds futuristic and it fits the Wall Run mechanic. The Jump sound effect uses the VR\_Grab sound effect from the engine because it sounds like a player jumping off a plank which is what I am aiming for. Lastly, the shooting sound effect uses the FirstPersonTemplateWeaponFire02 which is just basically the shooting sound from the first person template which is perfect for my shooting sounds.

**Inspiration / Reference:**

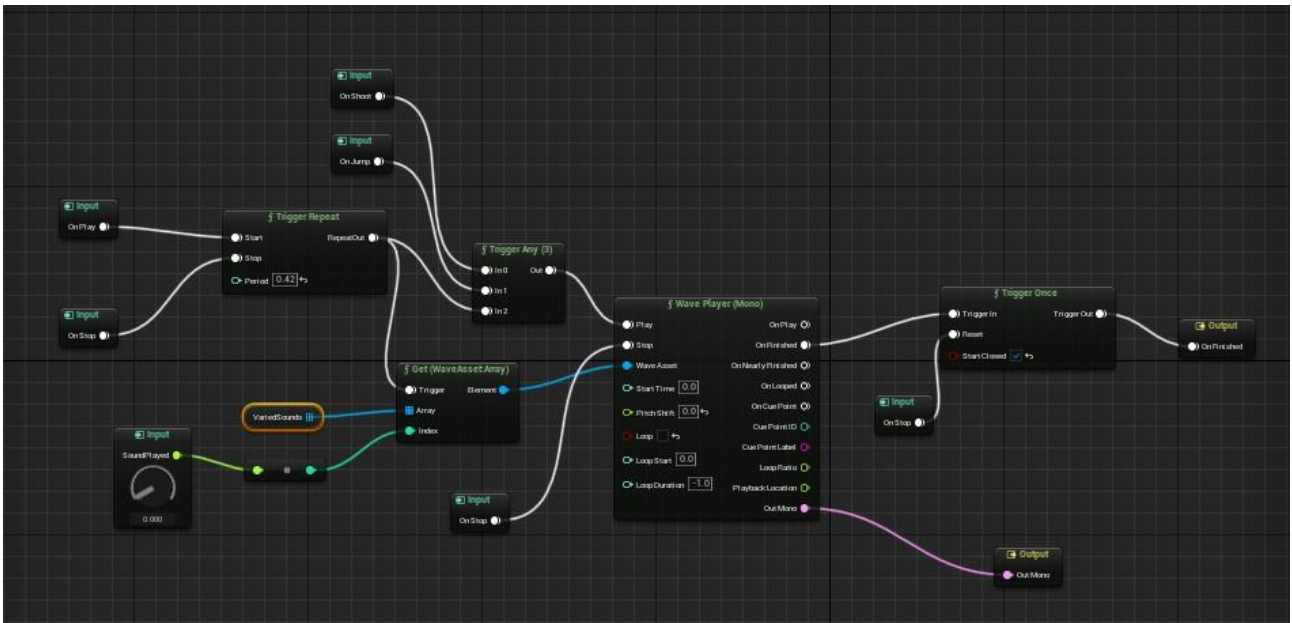
I cannot provide my inspiration with screenshots since it is impossible to provide the inspiration of sounds through screenshots but I will describe my overall thought process of thinking about these sounds.

So when I started gathering the idea for the sound, an idea came up to me in the form of “What if I could trigger three different sounds in just one sequence.” This idea then made me think of what kind of sound effects I would want to implement then I thought why not just include every aspect of my game that is connected to my player. Then I decided that since the player can wall run, jump, move and shoot I would want to complete the main mechanics that the player has sound wise. I didn’t have to do the move sounds since it is already covered in the labs but I decided to do the rest.

**Properties and Values**

Property	Description of Purpose	Value
Attenuation (Jumping)	Value used to represent sound drop off over a distance. This sound is going to be placed on the bottom side of the player’s body.	500.0f
Attenuation (Wall Run)	Value used to represent sound drop off over a distance. This sound is going to be placed on the side part of the player’s body depending on which side the wall is.	500.0f
Attenuation (Shooting)	Value used to represent sound drop off over a distance. This sound is going to be placed on the middle side of the player’s body.	500.0f

## MetaSound Diagram



This is the overall node graph of the metasound. Since there are different sounds that will play depending on the action of the player there are different inputs present. The most obvious ones are the On Play input which plays the audio and the On Stop input which stops the sound. There are other inputs which are the On Shoot input which triggers when the player shoots, the On Jump input which triggers when the player jumps and the SoundPlayed Input which decides which sound is going to be played based on the sound array. The VariedSounds array has 3 different sounds. In the 0 index the sound is the sound for the wall run action, the 1 index is the sound for the jump action and the 2 index is the sound for the shoot action. The SoundPlayed input is changed dynamically through code based on the player's action. This then is connected to the Get(WaveAsset: Array) node which is then connected to the Wave Asset part of the Wave Player (Mono) node. The Trigger Repeat node is for the wall run audio since when wall running, the audio should keep repeating over and over until the player stops or finishes wall running. The period of it is 0.42 since it is a good period repeat of the wall run audio. The On Shoot and the On Jump input is connected to the Trigger Any (3) node which connects to the Play node. This allows the play node to be triggered in 3 different ways rather than 1. The Wave Player (Mono) plays the sound based on the Wave Asset that was selected. The reason why there is an On Stop trigger on the Stop part of the Wave Player is because the Wall Run audio doesn't stop immediately and has a delay on when it stops so using this On Stop input trigger on it stops it immediately. Then, when the Wave Player is finished it goes to the Trigger Once node to mark the meta sound as finished.