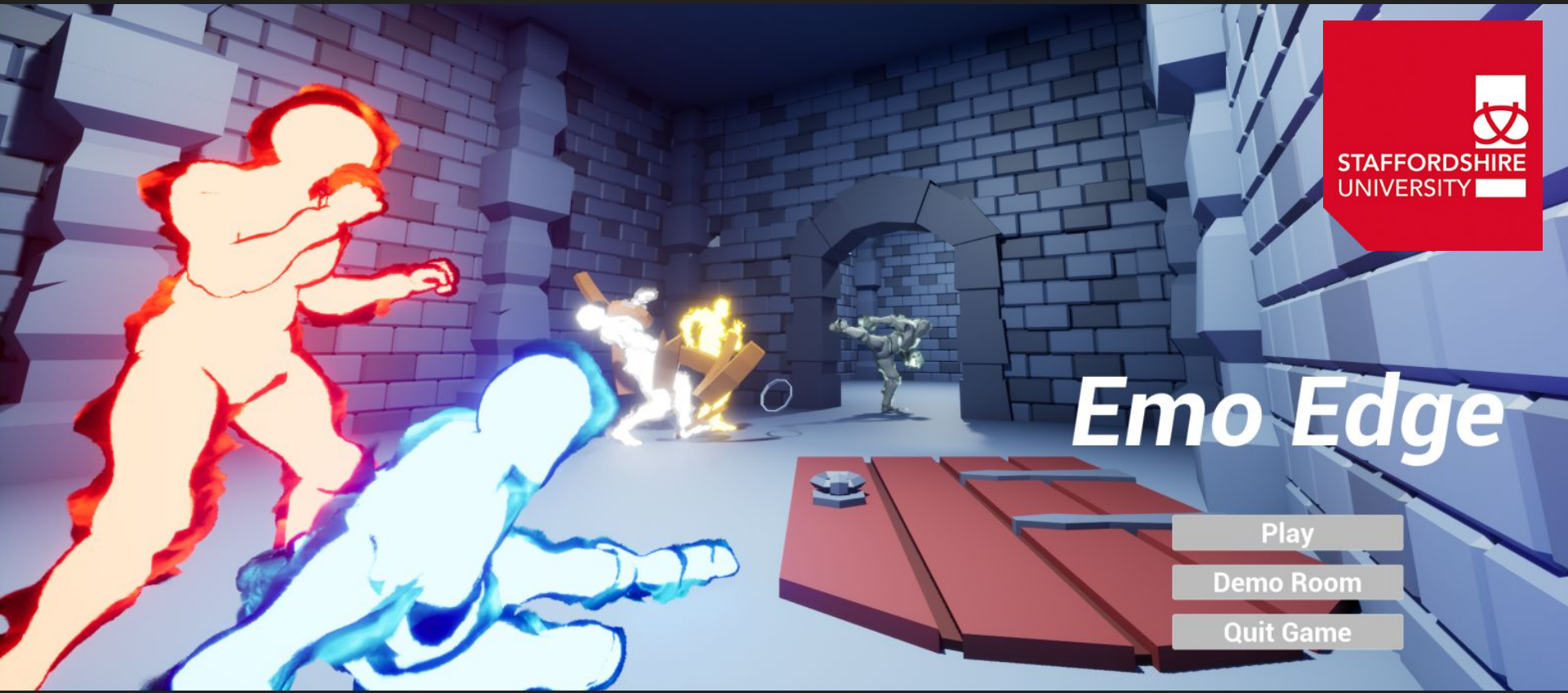

STAFFORDSHIRE
UNIVERSITY
Emo Edge
Play
Demo Room
Quit Game

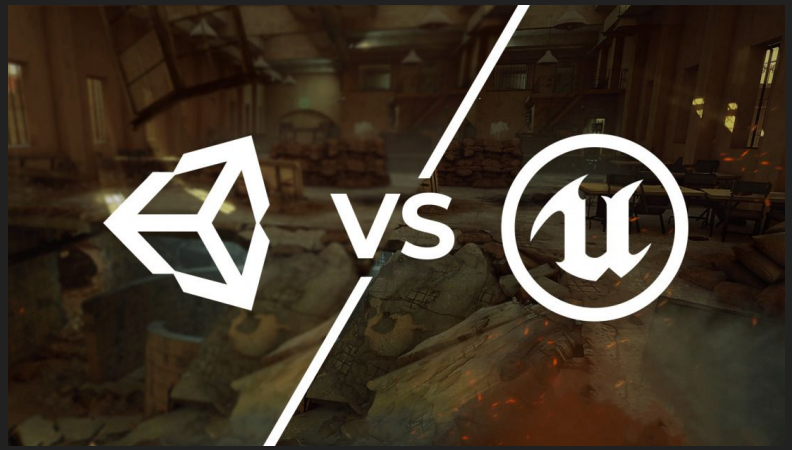
Table Of Contents

The Core goals of the project stemmed down to:

- ➔ Wanting to Develop Knowledge as a Gameplay/Technical Designer
- ➔ A High Quality Portfolio Piece
- ➔ Something that can help get me into US Games Industry

To Achieve these desired outcomes I looked at doing…..

Playable Game Project!

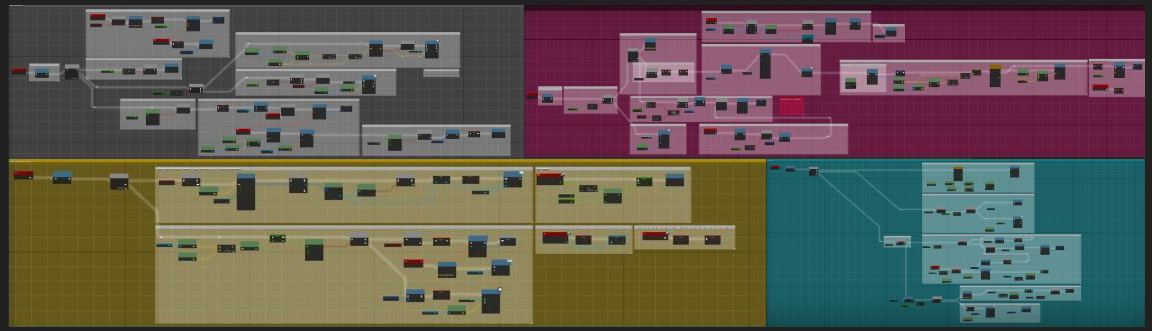


- ➔ When I first started my project, I could use either Unity or Unreal as my game engine. I chose to complete this using Unreal after giving it some thought since I wanted to delve deeper into the domain and have a better understanding of the Unreal blueprint system.

- ➔ I wanted my portfolio to feature a greater variety of projects using the Unreal Engine.

With these goal In mind I looked into covering 3 Core Topics/ Goals for my Development Project, those being

- Core Set of fun gameplay mechanics
- AI Enemy Development
- Develop deeper understanding and implementation of Blueprint as well as Unreal Engine 5



After Honing in of my targeted idea with my mentor I looked into, what more specifically I wanted to produce.

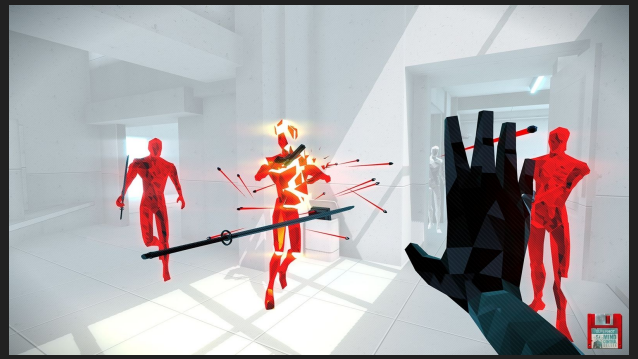
For this I wanted to look at making an Vertical Slice High Octane Shooter.

Initial Research into the project had me look into looking at similar Titles and mechanics I was looking to Replicate.

This lead me to looking into games such as

- Angerfoot
- Doom
- Superhot
- Control
- Ghost Runner

Delving into these games helped shape Inspiration for potential mechanics I could look at developing and what makes these mechanics fun



Emo Edge is a first person shooter with the twist of where emotion has power. In this game the player goes through the mind of a person clearing emotional aspects that are trapped within. Throughout the game cycle the player will need to use the range of emotional powers they have to destroy the enemies and threats that come along his way. Throughout gameplay the player has control over:

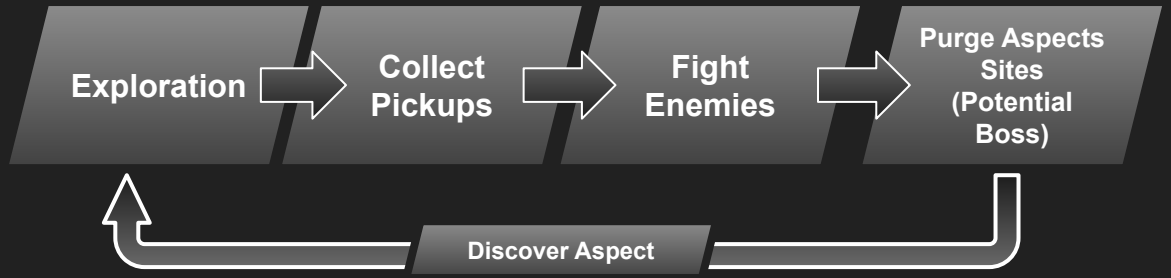
- Joy
- Dread
- Rage
- Sorrow

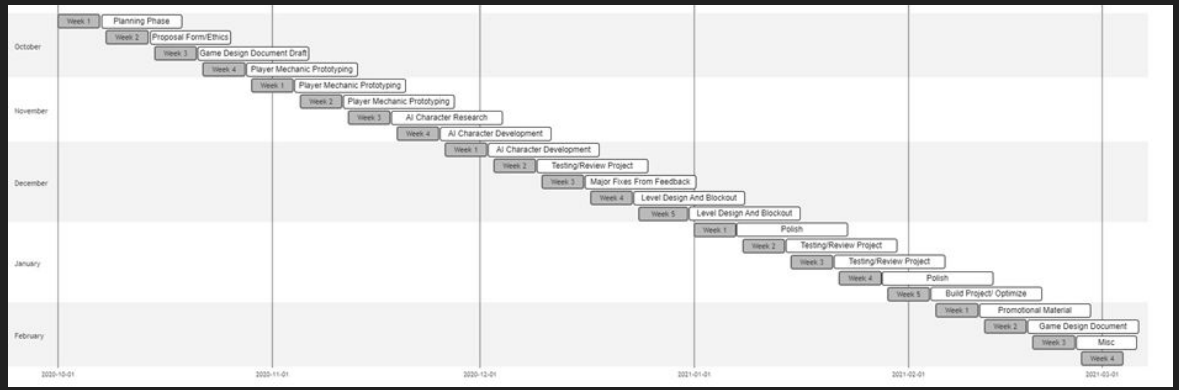
Each aspect has a range of corresponding powers to go with it. For the player to succeed they need to use all in tandem at the right moment to survive against the onslaught of different aspects in the oncoming rooms and waves

The Tech Demo Emo Edge would cover the core themes of First Person Shooter, Emotion, Arcade, Fast paced gameplay.

The game is designed for having fast, quick paced battles in multiple rooms in quick succession. There would be cycles of intensity and duration where both would increase in value over time. Testing the player until failure.

Game Loop





Planning started with Timelines,MosCoW's and Game Loops

MosCow Break Down

Must Have
- Enemy Aspects/ AI
- Emotional Powers
- Movement Based Abilities

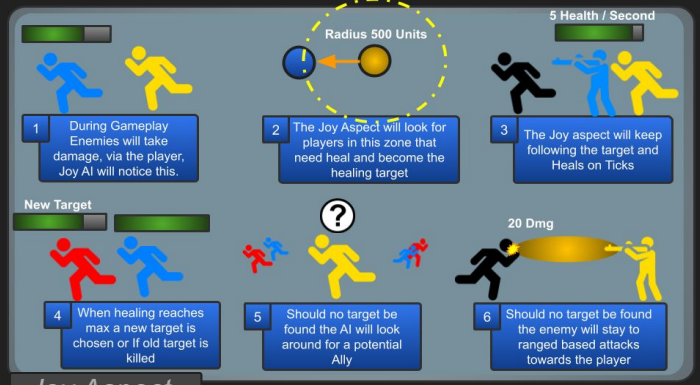
Should Have
- UI Elements
- Particle Fxs/VFxs
- Animations

Could Have
- Multiple Enemy Behaviour Patterns
- Save Data/ Leaderboards
- Cutscenes
- Pause / Restart Menu Functions

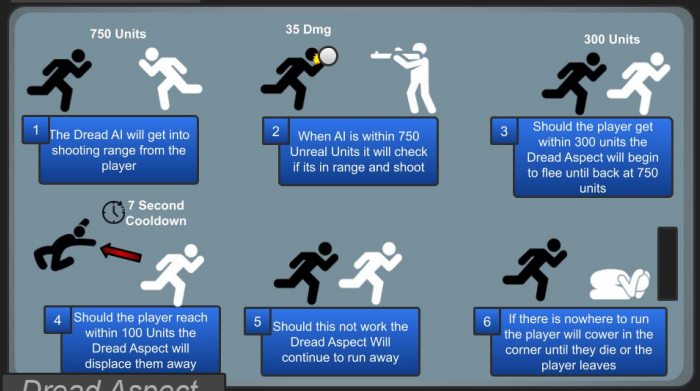
Won't Have
- Multiplayer
- Co-Op/ Local Co-Op
- Advanced Options Menu


Reloading!
0/5
POWER
HEALTH
RUSH
Ability Ready!
Floor - 0
1
5/5
Hold U To Interact
2/5
Exploration
Collect Pickups
Fight Enemies
Purge Aspects Sites (Potential Boss)
Discover Aspect

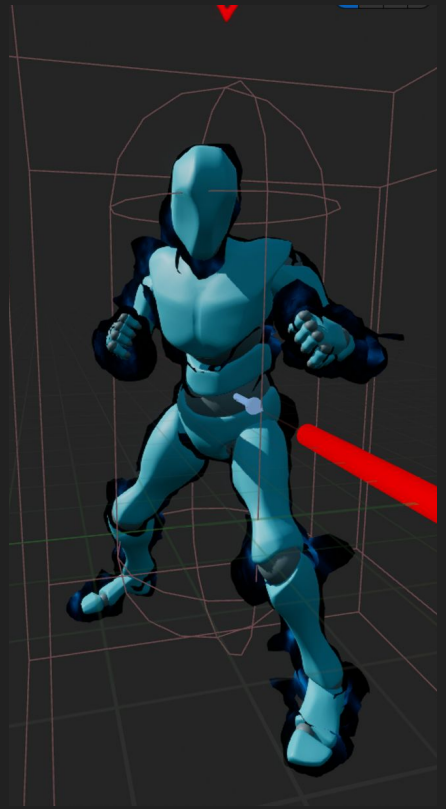
When Starting this Project I looked into planning out the AI before delving in. For this I looked at making 4 Different Behaviours

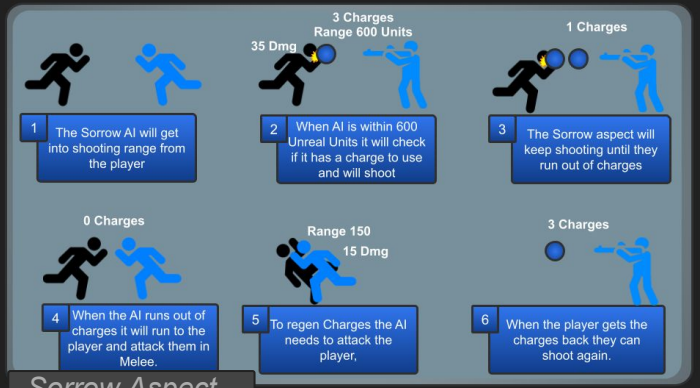


Joy Aspect

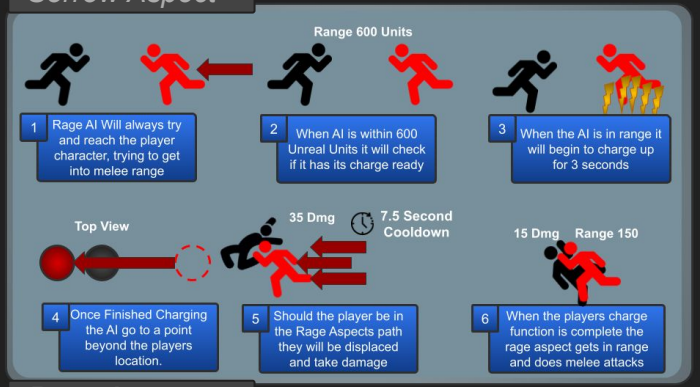


Dread Aspect



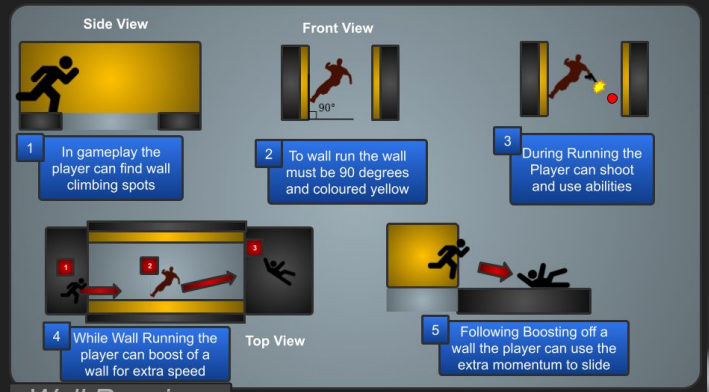


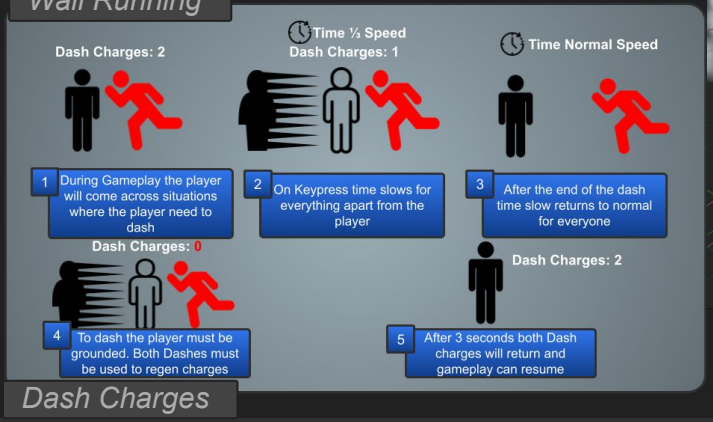
Sorrow Aspect

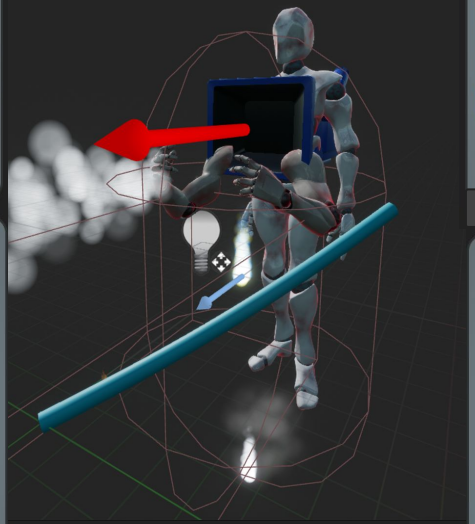


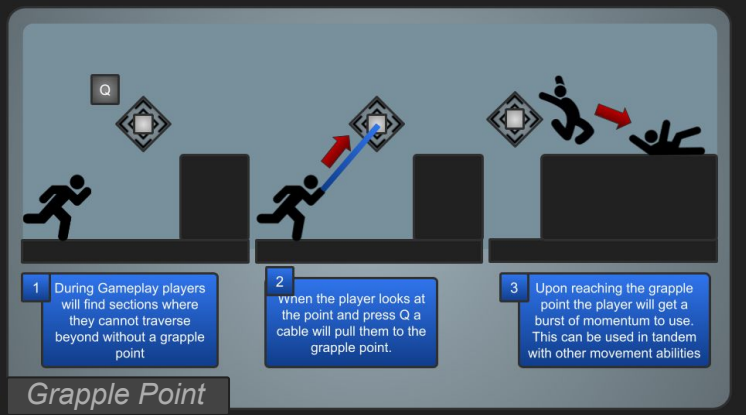
Rage Aspect

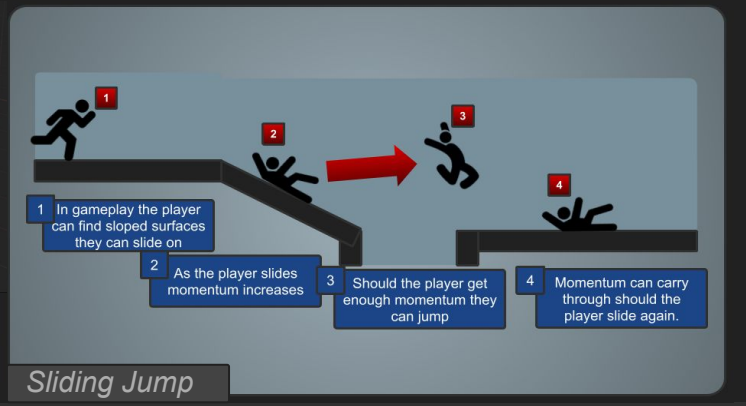
For the Player Character 3 Movement Mechanics Were Planned out, this Inclusion was vital for a High Octane Game Project











When Starting this Project I looked into planning out the Player Pickups for the game

Double Shot Pickup

Across gameplay the player can find the Immune powerup, when the player picks it up the powerup is destroyed. From here the player is given temporary double shots whenever they shoot from any of the weapon profiles

The goal of this powerup is to give the player boost to damage making rooms easier to manage, this also mixes up gameplay
Only One Pick up can be in use at a time.

Immune Pickup

Across gameplay the player can find the Immune powerup, when the player picks it up the powerup is destroyed. From here the player is given temporary Immunity to all forms of incoming damage

The goal of this powerup is to give the player a survivability boost and mix up gameplay

Only One Pick up can be in use at a time.

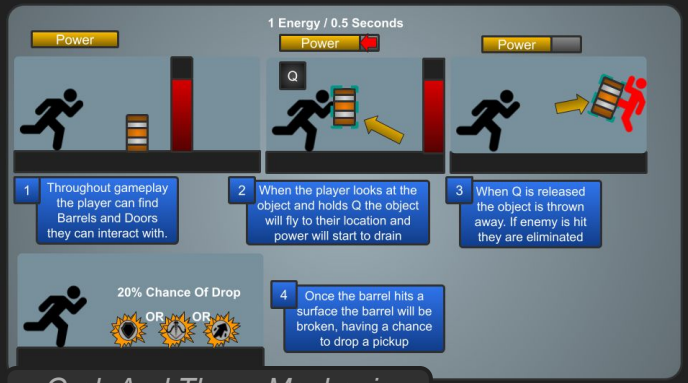
Dash Pickups

Across gameplay the player can find the Dash Powerup, when the player picks it up the powerup is destroyed. From here the player is given temporary infinite dash to use over the period of 5 seconds.

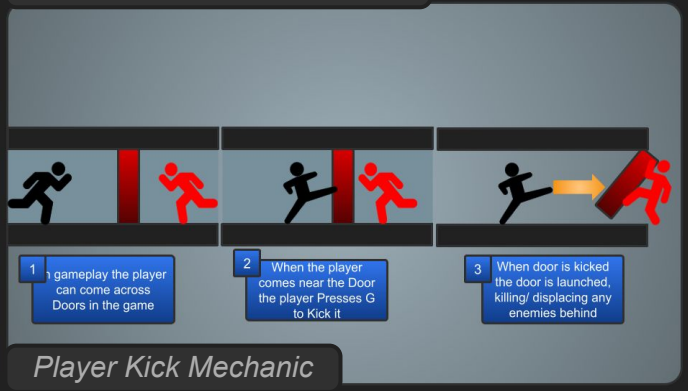
The goal of this powerup is to give the player temporary speed boost enhancing maneuverability.

Only One Pick up can be in use at a time

When Starting this Project I looked into planning out the AI before delving in. For this I looked at making 4 Different Behaviours

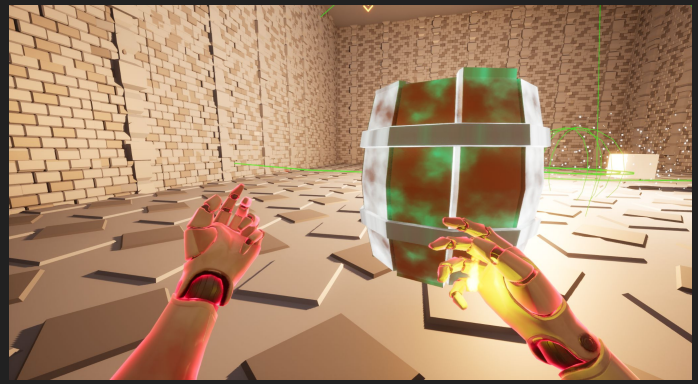


Grab And Throw Mechanic



Player Kick Mechanic

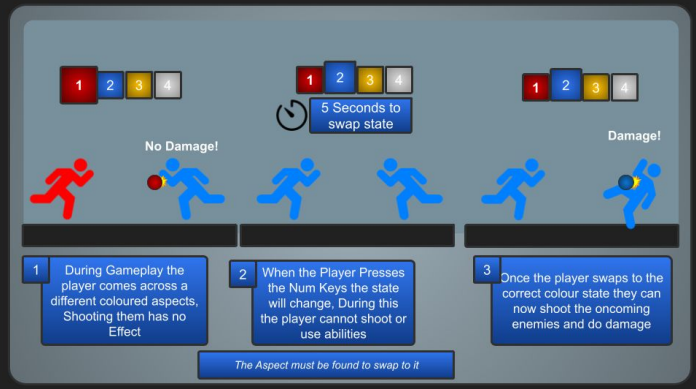
Player Kicking A Door

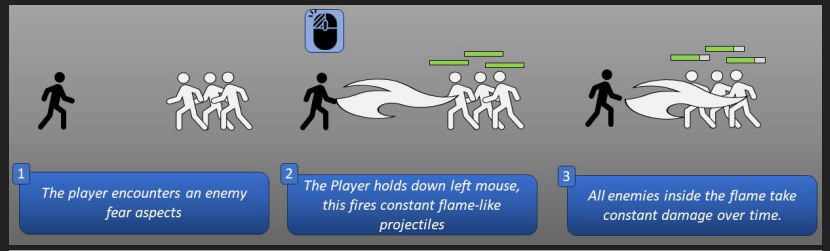


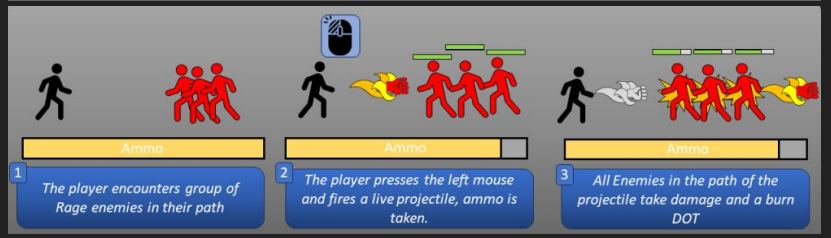
In gameplay States are used for dynamic play, the player must be in the correct state to harm enemies, abilities reflect differently for each state

The player during gameplay finds these aspects and uses them in tandem with other abilities

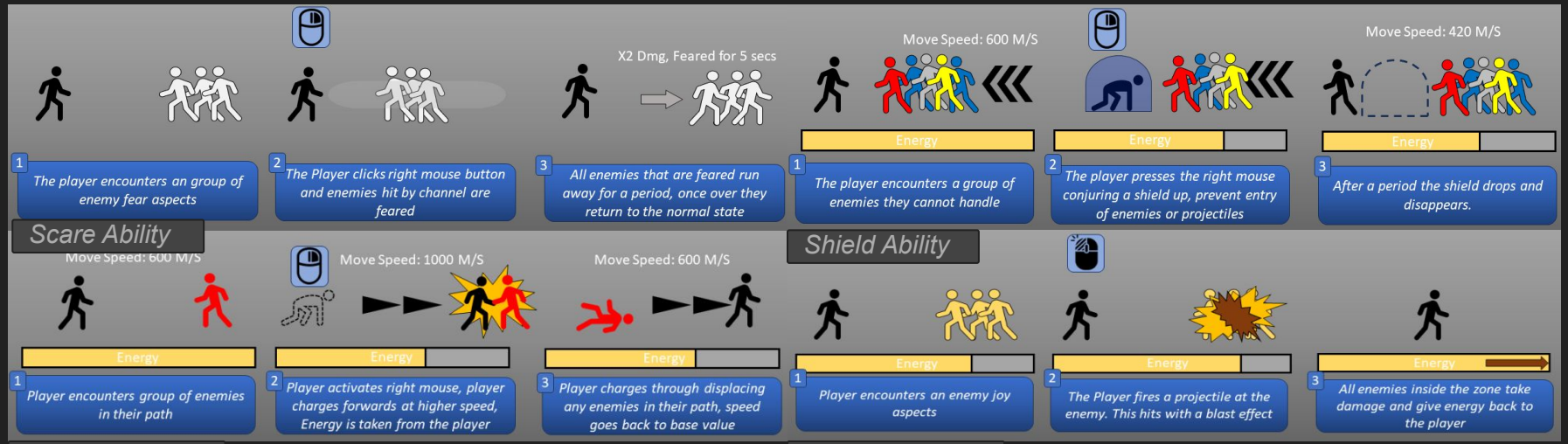
The state are found throughout the gameplay.





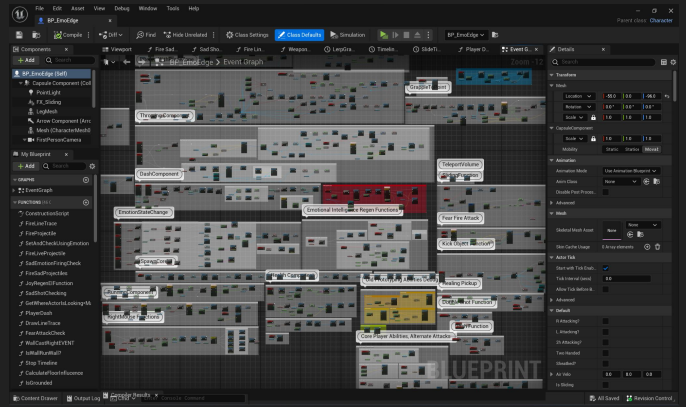


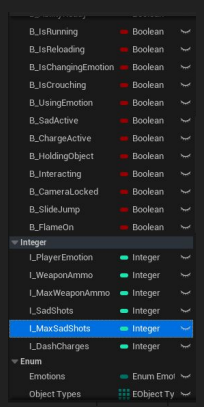
Alongside the primary attacks the player has in each state the player also has access to alternate abilities for each class, with some being more offensive and others being more Ultitlity each has a role and different purpose

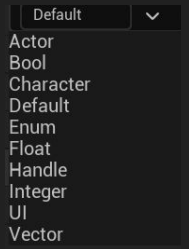


EmoEdge Minimum/Recommended Hardware Requirements	
GPU (Graphics Card)	GTX 1660 Super
CPU (Central Processing Unit)	Ryzen 5 2500
Dedicated RAM	16 GB
Software	Windows 10/11

Before Starting this Project I set Hardware Goals on the Project to ensure Optimization







Commenting and Categories were made to ensure readability for me and others

Naming Conventions

Variable names camel cased - myVariable

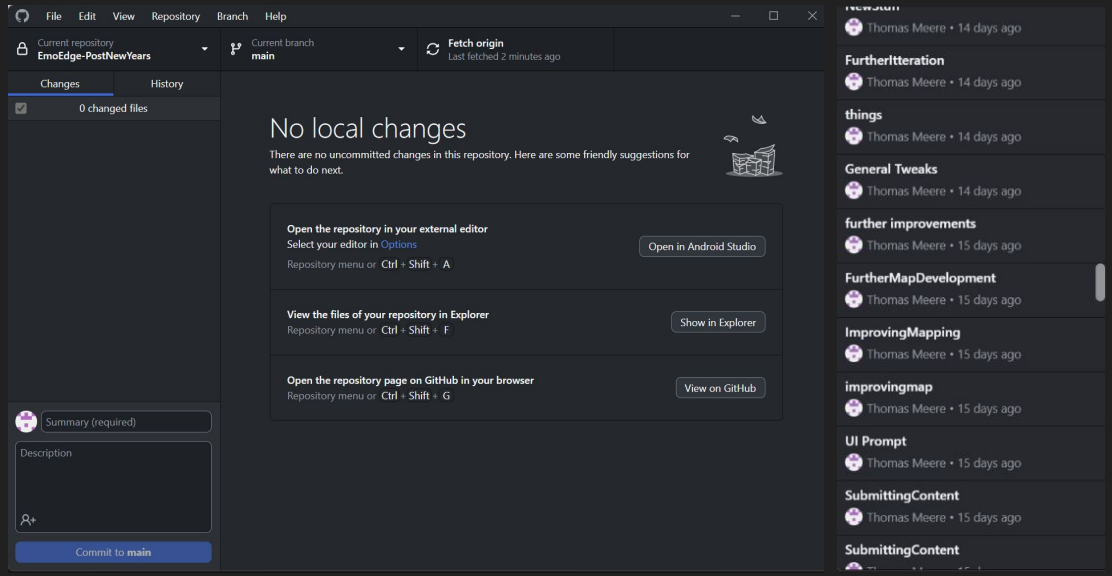
Class Variables/Member Variables – m_variable

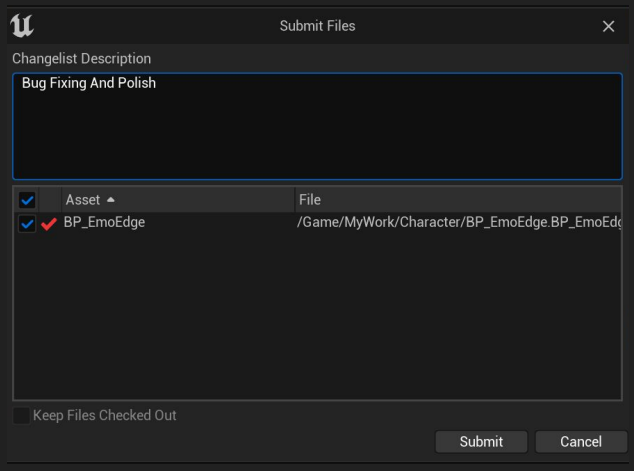
Global Variables (instanced variables) – g_variable

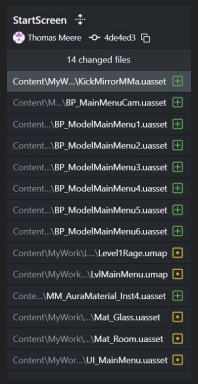
Class Name	Class Prefix
Blueprint	BP_
Material	MM_
Material Instance	MI_
Texture	T_
Game Mode	GM_
Game Instance	GI_
Game State	GS_
Player State	PS_
Player Controller	PC_
Animation	Anim_
Animation Blendspace	BS_
Animation Blendspace 1D	AnimBS1D_
Animation Blueprint	AnimBP_
Animation Montage	AnimMon_
Master	Mast_
Child	Child_
Blueprint Interface	BPI_
Widget Blueprint	UI_
Function Library	FL_
Structure	Struct_

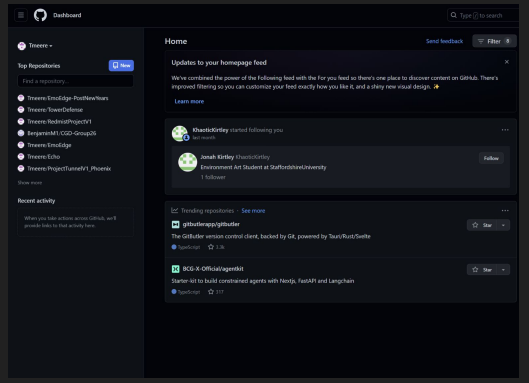
Emo Edge, Game Design Document, Thomas Meere 32

Naming Conventions Were Also Made to Made Sure Everything was Consistent

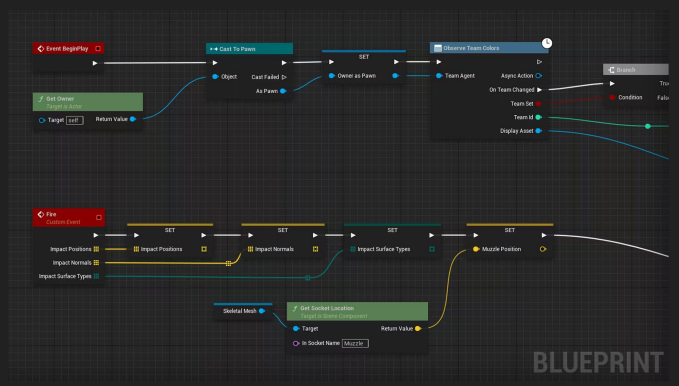


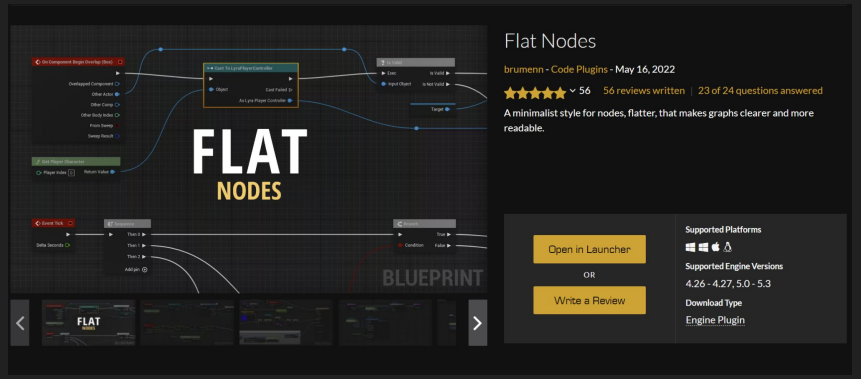




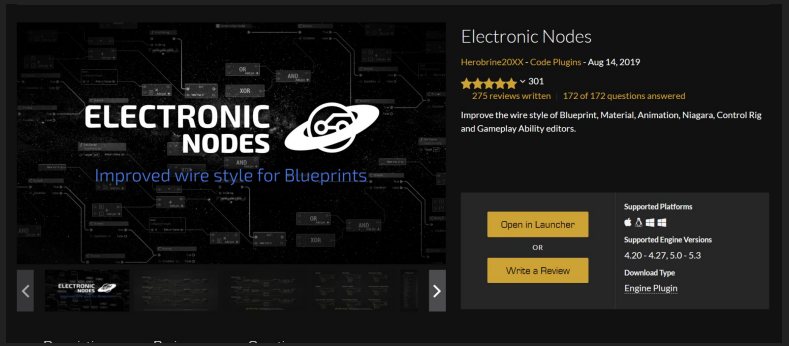


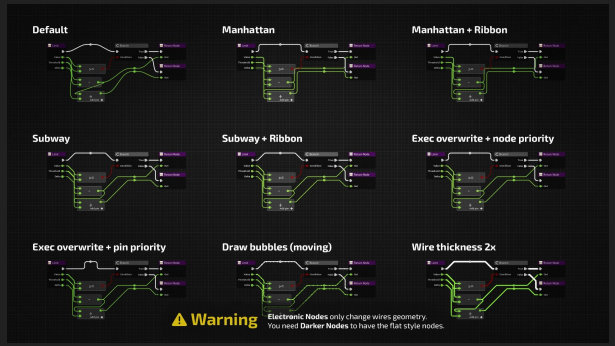
Github Desktop Was Used for the projects cycle, this ensured a non destructive workflow and allowance for mistakes or errors to be quickly rectified with source control





During the project add ons were used as Visual Aids, Improving the Clarity of Work for me, this aided with workflow development



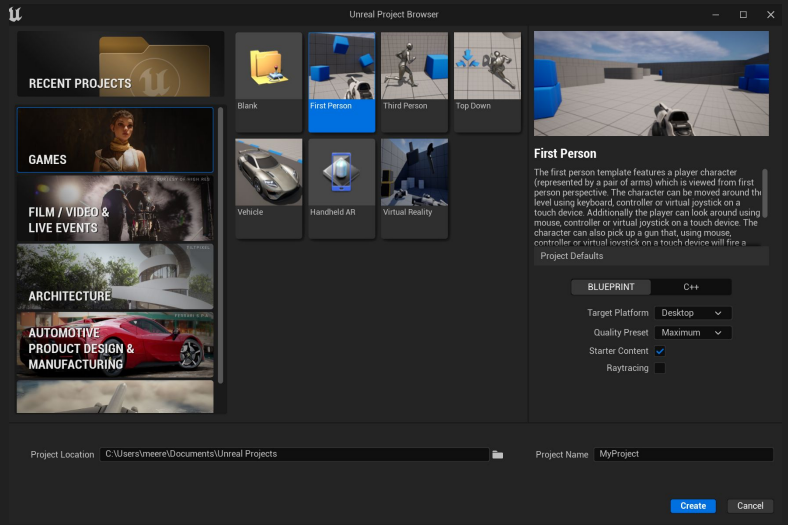


Organizing Workflow

During the projects cycle I found the scripting to get more and more intertwined and messy causing myself confusion and others who would review my code. For this reason I looked into sectioning my blueprint into commented sections and commented categories depending on what the thing was. For the player character this was broken down into sections of:

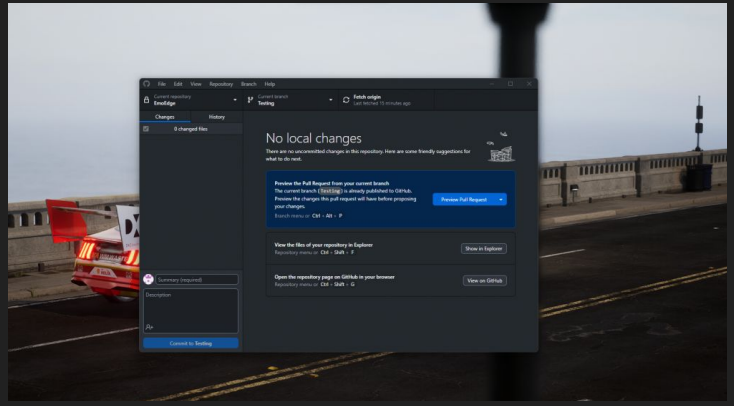
- Shooting Mechanics
- Event Begin Play Functions
- Base Player Movement
- Extended Abilities
- Player Stats
- Power Ups/ Miscellaneous

This helped improve clarity and readability for the player character script. This format was also followed for the enemy AI class.



To start the project I first looked into what framework would best be suitable for the project and the goals in mind of a high octane shooter, the First person template was chosen as a starting point to work from.

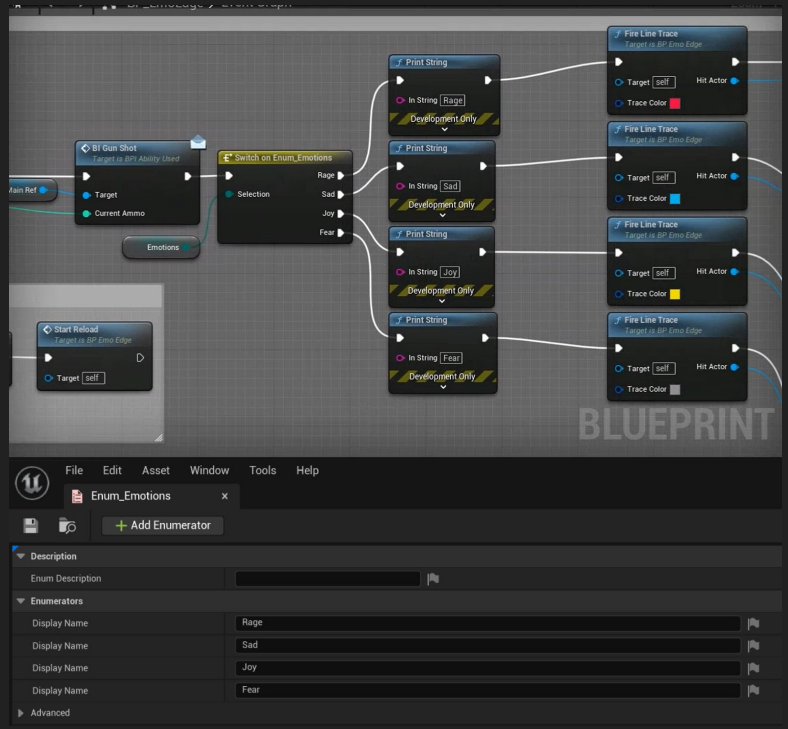
Alongside the project being made a repository on github was set up for the project to allow non destructive workflow and quick change should work be altered.



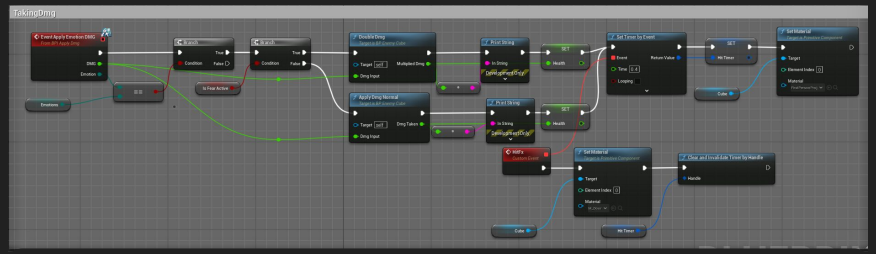
EmoEdge Asset List

Asset Name	Description	Links	Found Yet?
Enemy Character Model	*Character model for the enemy archetype, this will be a stylized model that has a simplistic for or shape*		
First Person Arms Model	*First person arms will be needed for the project as it will be a first-person game. This will again be a simplistic stylized model*		
General Niagara Fxs	*Any and all Fxs needed to enhance the feel of the game, can range from the environment Fxs to hit Fxs*		
Environment Meshes – Walls/Floors	*Environment meshes will be needed once going past the block out phase for the room designs.*		
Environment Meshes – General Objects	*Once Blockouts are completed rooms will need to be filled with various stylized props for interest and diversity in the levels*		
Projectile Meshes/Fxs	*For the 4-character emotional states custom projectiles will need to be made or found, some will be physical meshes while some FXs(for things such as line traces)*		
Player Animations	*Since I am not looking at creating animations for this project animations will need to be found. This will include a range of things including Idle, Running, Reloading and shooting states*		
Enemy Animations	*Since I am not looking at creating animations for this project animations will need to be found. This will include a range of things including Idle, Running, Reloading, shooting, flinch, death animations etc.*		
Stylized Materials	*For the games stylized look I may look at outsourcing Materials or take reference in some to shape the looks of the game. This is not a primary concern as of this moment*		
Post Process Materials	*For the game's atmosphere Post Process materials will aid in enhancing the look of the game with the changing emotions.*		

During the start I also put together an Asset list of potentially needed items for the projects development, this aided with work to come for the project



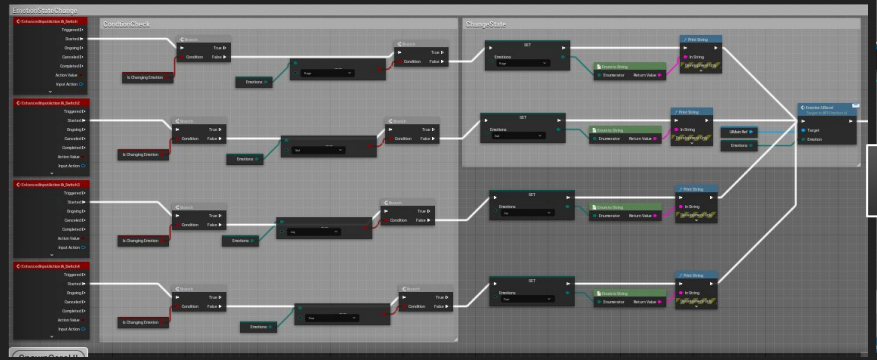
When starting the project I looked in learning a new variable class and how it could be used for my project, for this I used an Enum and initially made some Debug line trace fires. These would only hurt the debug cubes with the same corresponding Enum



The Enemy Testing Cube had a new health made function that would only take damage if the Input Enum and its own was the same

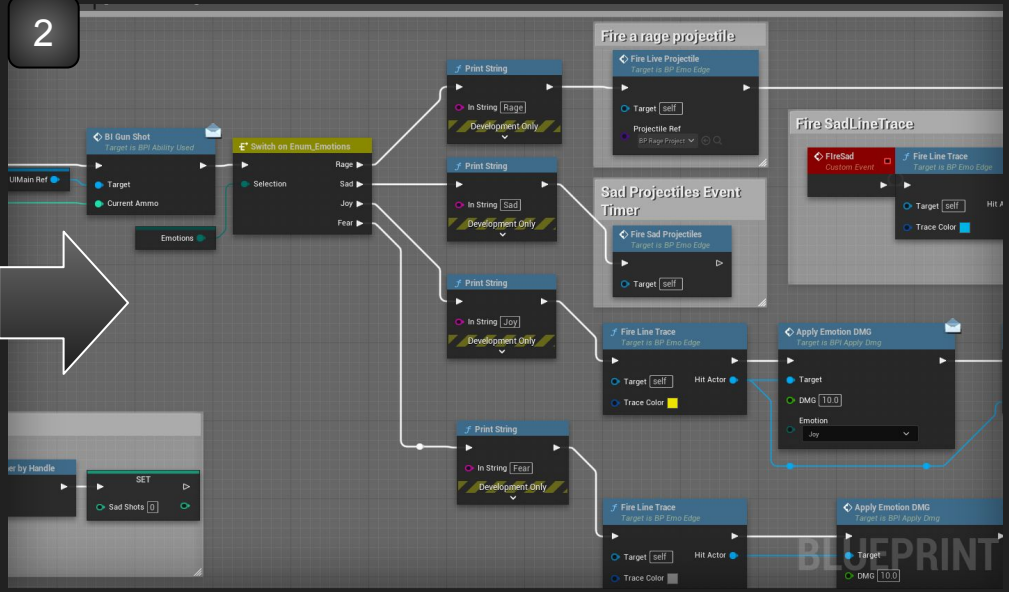
This initially was hard coded and would only work as a instance edible variable. This would need to be something dynamic the player could control.

1

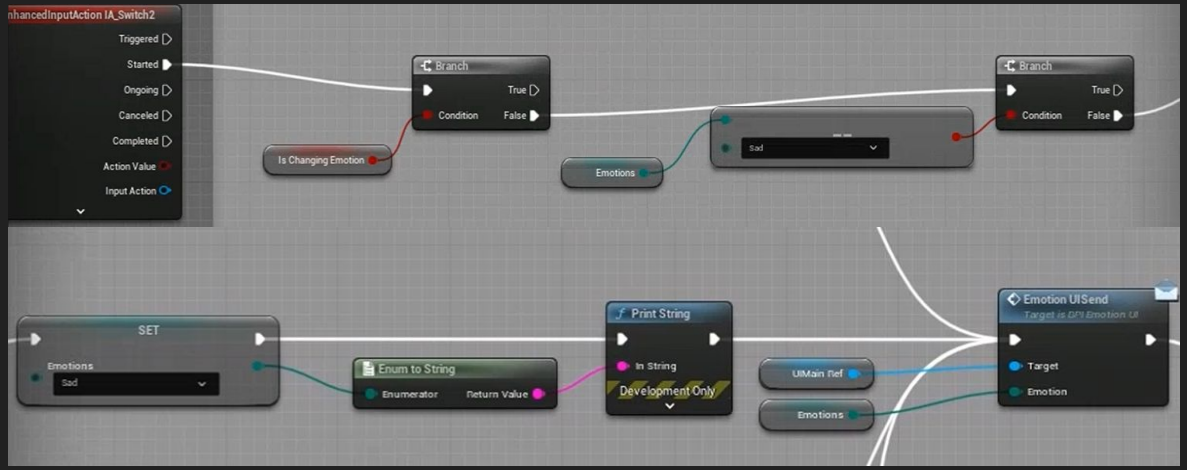


Change

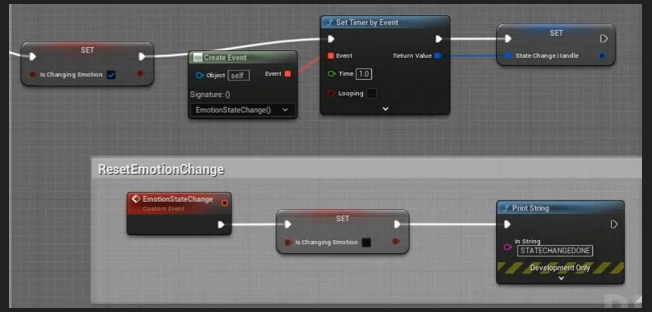
2



Fire State Function

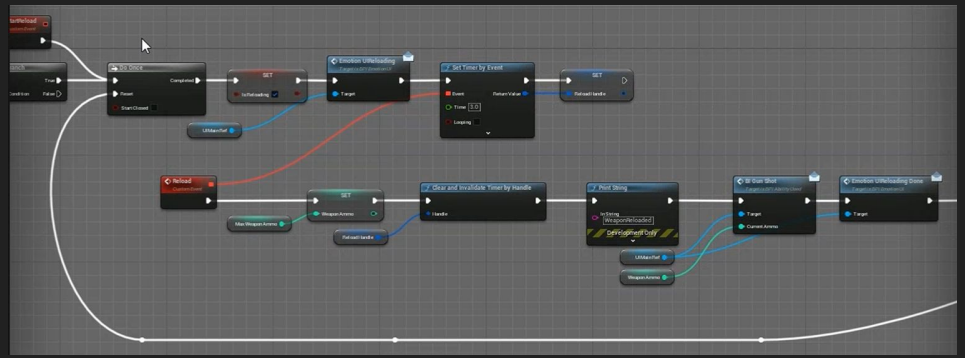


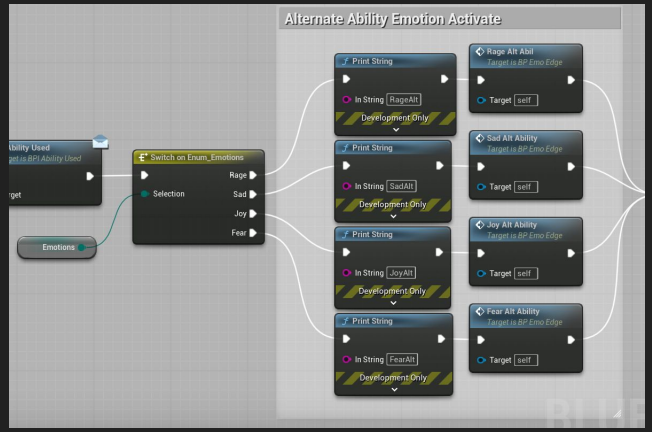
Following making the debug shooting. New actions were made so the player could change their state on the go, this function would check if its a new enum and if so begin to change.



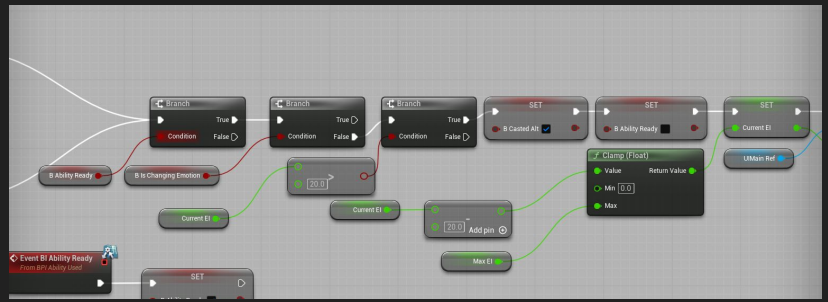
During this, it was found that the immediate state changes were overly abrupt, reducing the impact of each state. To fix this, delays and boolean variables were implemented to enhance the feel.

Additionally, a base reload mechanism and ammo counter were introduced alongside shooting to prevent players from spamming gunfire, promoting a less spammy approach to gameplay.

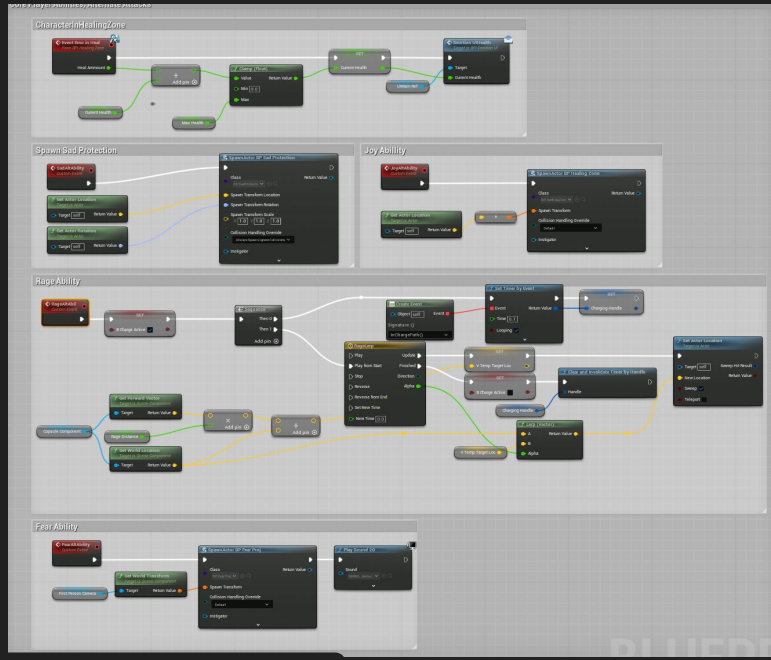




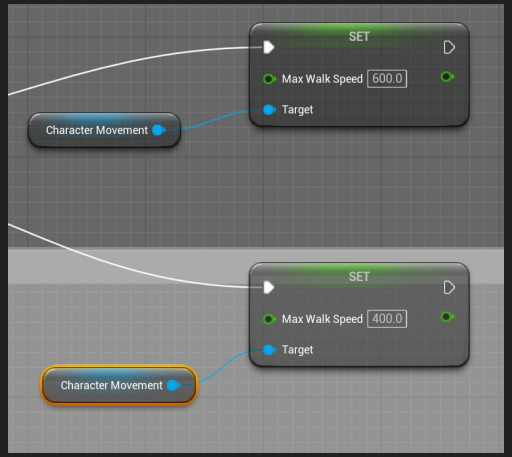
From making the player states I next looked into how I was going to be making the players alternate abilities. For this I referred to my design documents and made some prototype abilities, some were custom spawned actors while some were new components to the player char, this would be controlled by right mouse or left trigger



From this conditions and energy were taken from the player to stop them spamming abilities, this helped with balance and making the abilities meaningful

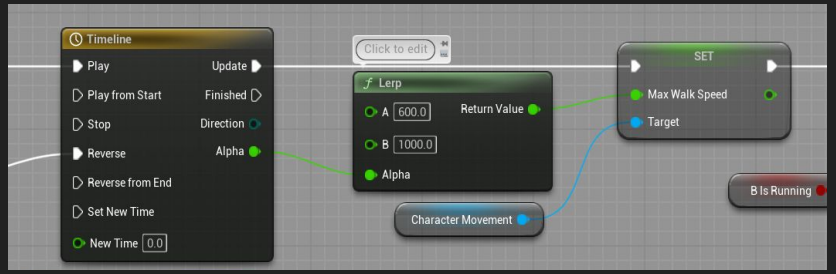


Initial Alternate Abilities



From this point I next looked into developing a running state for the character, Initially this started as a set character speed to a new value.

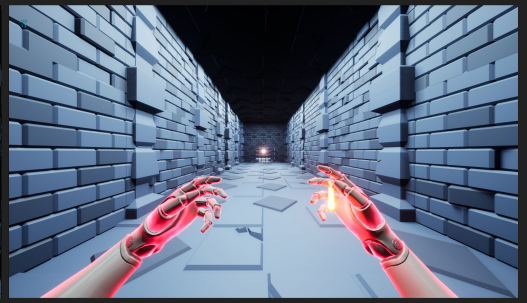
While this did work it came with the issue of limited satisfaction. As it more felt the player went from walking to speed walking.



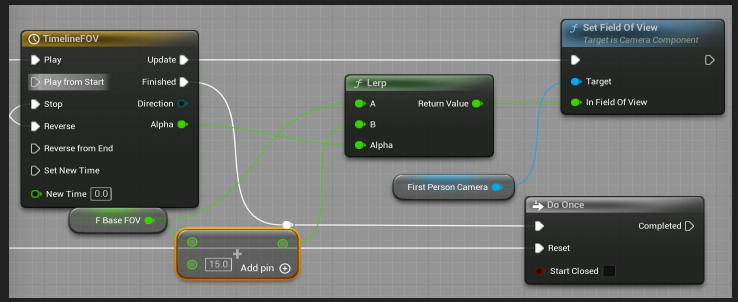
To rectify this I looked into using 2 things to emphasise the change in speed:

- FOV Scaling FOVs
- Timeline Float Lerps

Base FOV



Increased FOV



The Key when doing this was keeping it smooth and subtle, large obvious changes made it jarring for the player

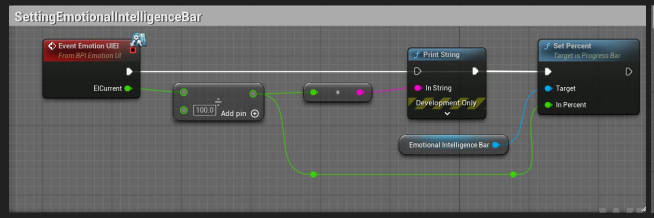
Upon initial testing while I found a few issues and bugs that would needed to be fixed most of the feedback was good, Initial testing found that the concept made sense but players were unclear on what state they were in beyond print string information.

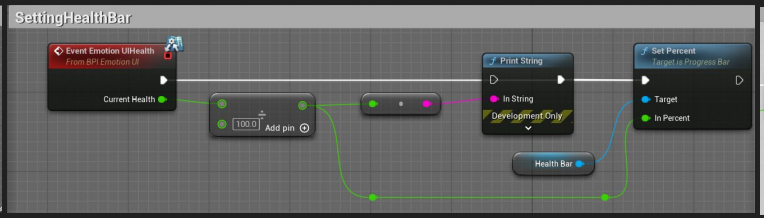
For this reason I looked into starting some base UI for the player. This would track core functions and would help ground the player.

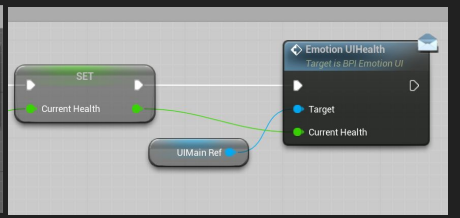
For this I made a UI_Main that dealt with the player information. It controlled things like:

- Ammo
- Health/ Power
- Alternate Ability
- Reticle

Initial Viewport and BPI Examples

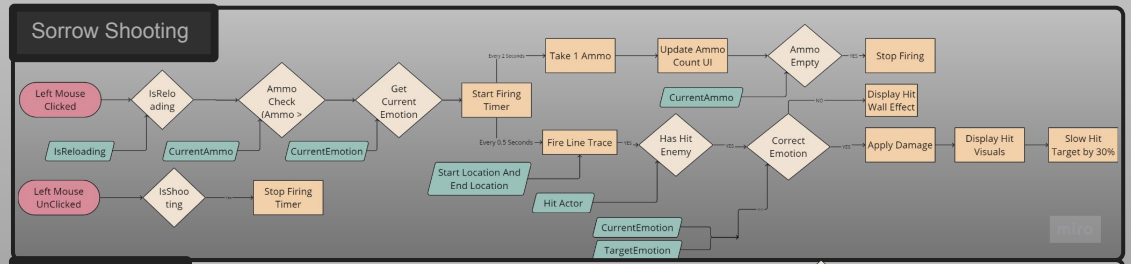


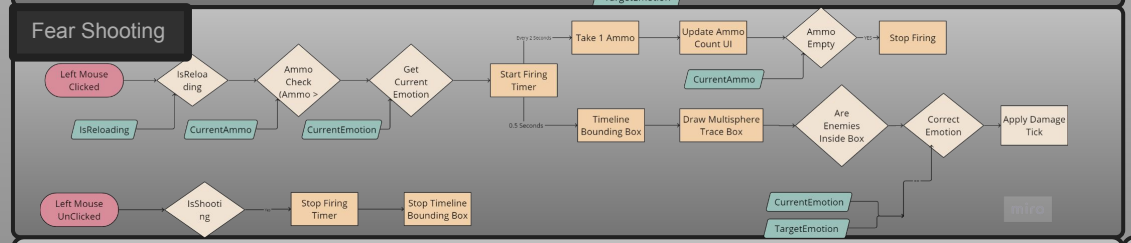


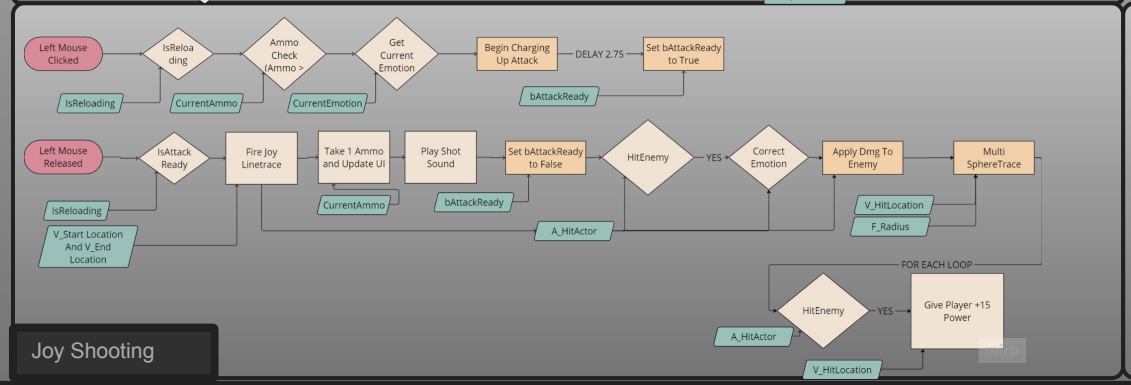


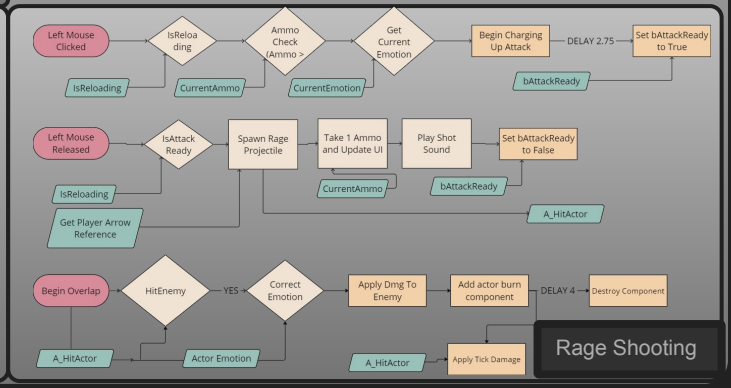
With these initial developments here was the initial progress

https://youtu.be/hokJAwHM5uc

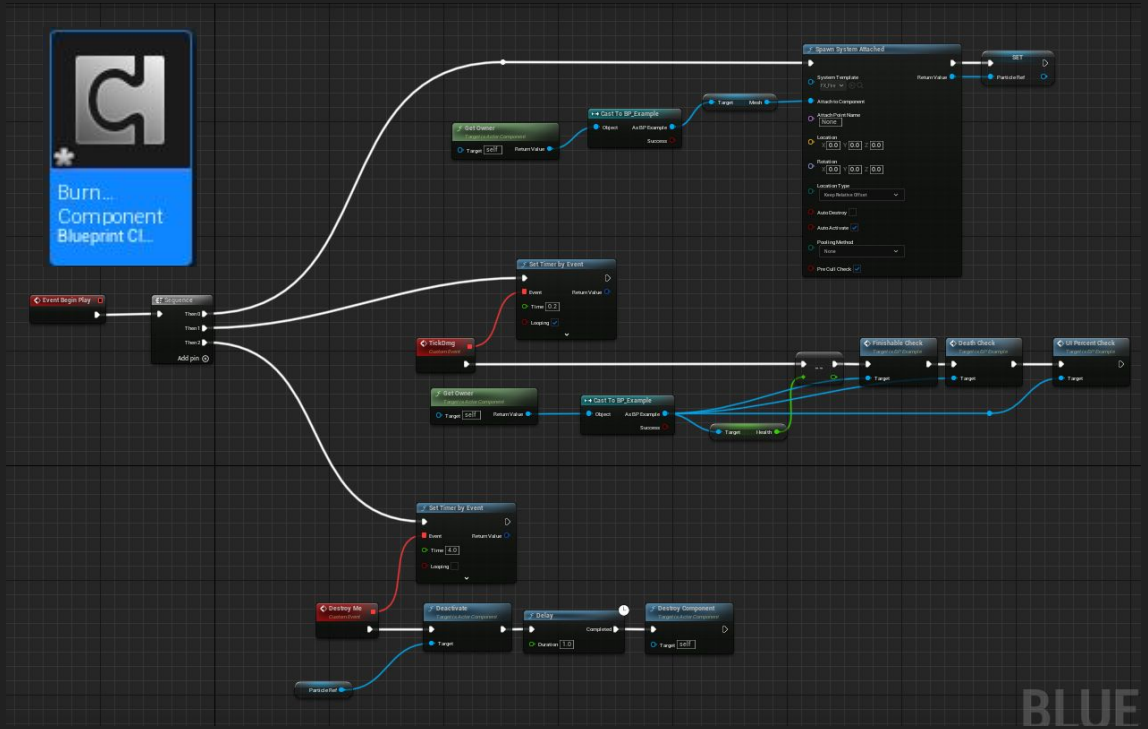




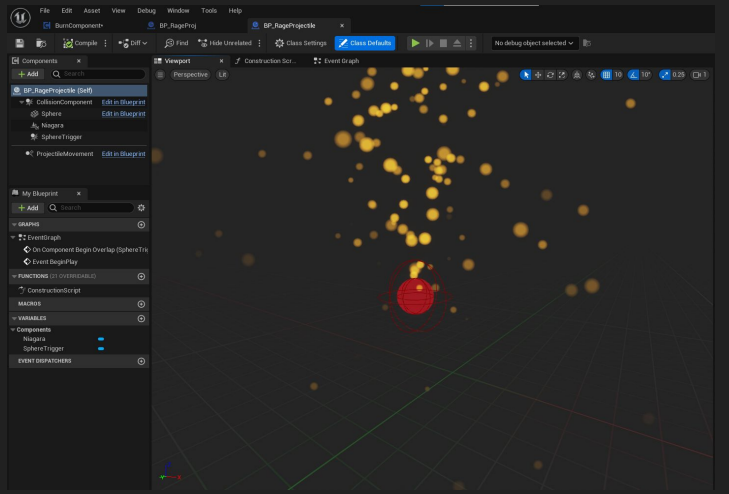




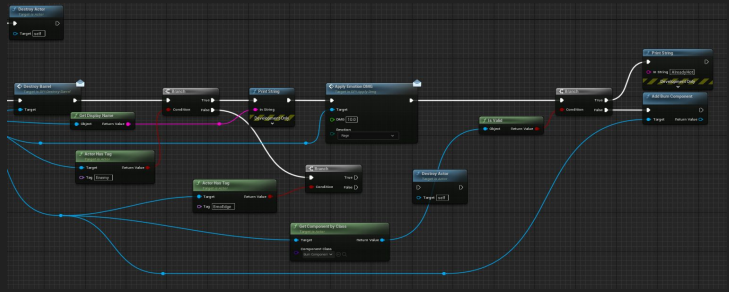
Following on from this stage of development I looked into taking the Player System Further. For this I begun to work on the player's Primary and Secondary Functions Unique, going from line traces to more unique states. For this I looked back at my design doc as well as the Flow states.



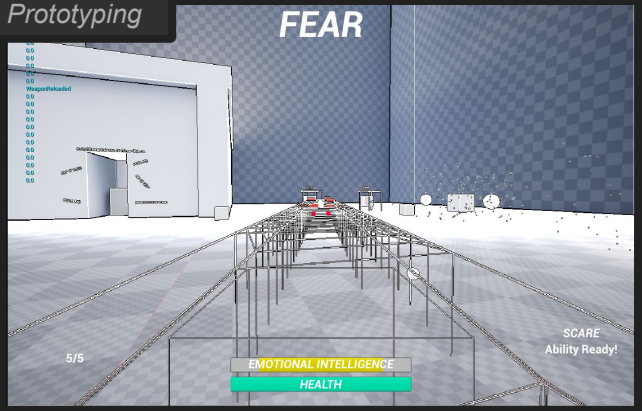
For the projects cycle I looked into making certain mechanics actor components, this allowed for universal changes without having to make the function on each new class

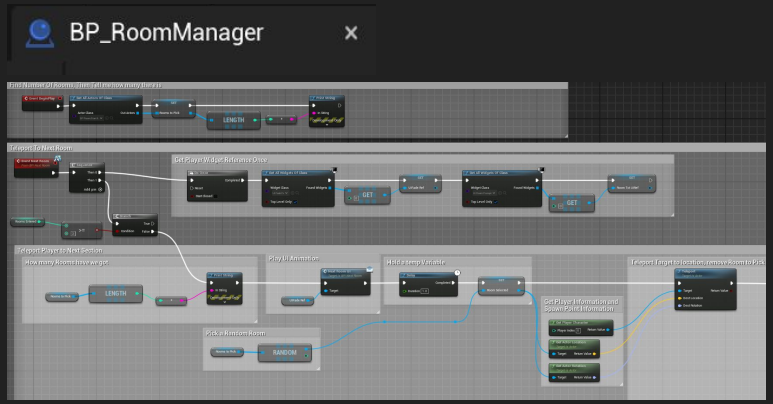


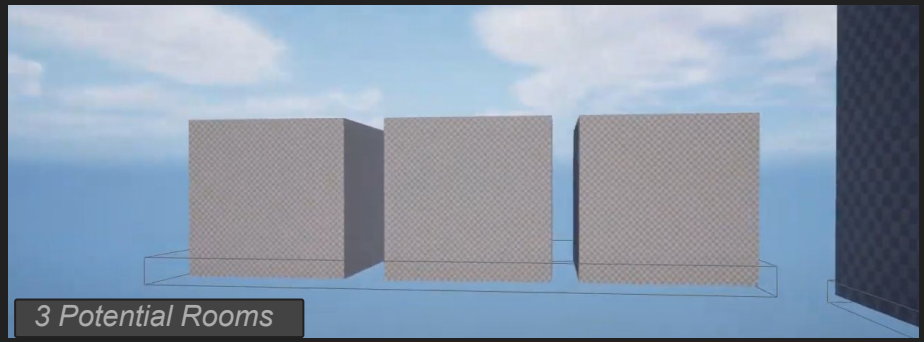
The rage projectile was made as a child actor from the default projectile class and iterated on.



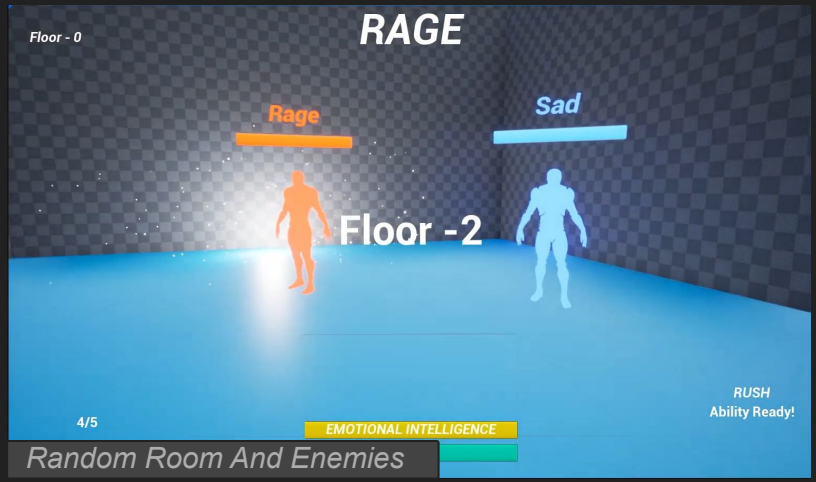
Initial Player Debug Prototyping







During the early stages of the project, I developed a Room Generator/Selector. This system would randomly select a room and spawn random enemies with varying quantities. While this approach technically functioned, testing and player feedback revealed several issues. Testers did not enjoyed the unpredictability of the system and how it fragmented the gameplay into shorter, disconnected experiences. Instead, they desired a more continuous connection and fluid movement from room to room. Due to these concerns and time constraints related to achieving other core project goals, I decided to set aside the level generator for the time being.



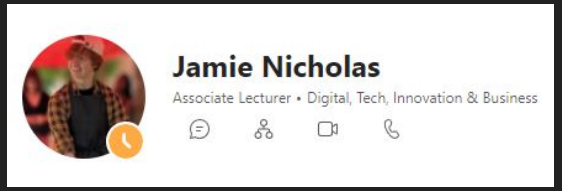
Random Room And Enemies

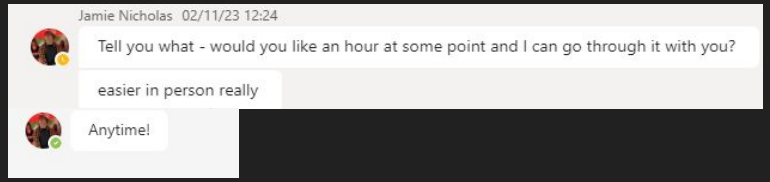
During the project, I had to learn AI prototyping, which was the second key component I looked at initially. For initial testing, I created a simple AI class and made a cube go to the player's position. Although this was a good place to start, it required more, so I spoke with Jamie Nicholas about it.

After we spoke and I received instruction from him, I had two choices:

AI with finite state machines or Behaviour Trees

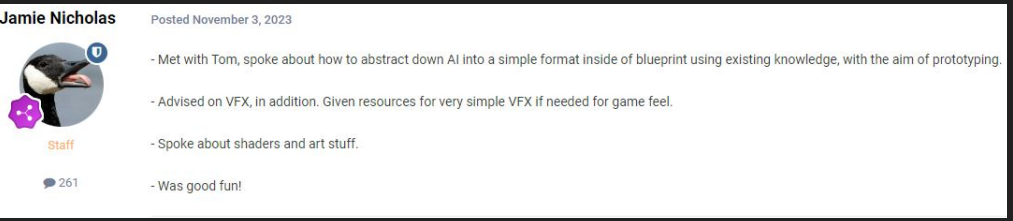
Following a sit-down conversation with Jamie regarding the project's objectives, he suggested starting with learning finite state machines.
He provided me some starting and iteration tips based on this.

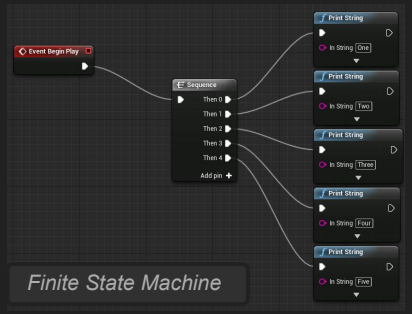




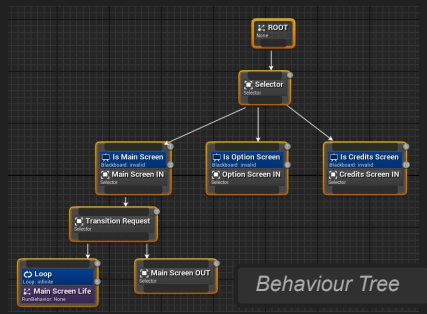
Basic AI Chasing the player

Basic AI Chasing the player



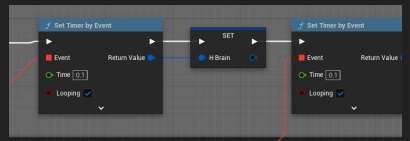


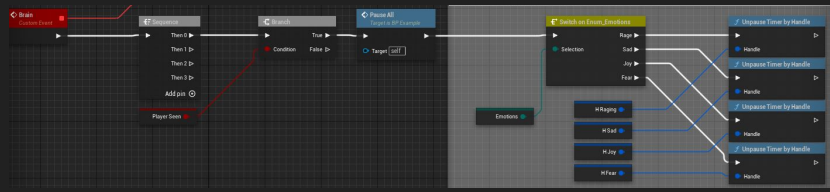
Finite State Machine

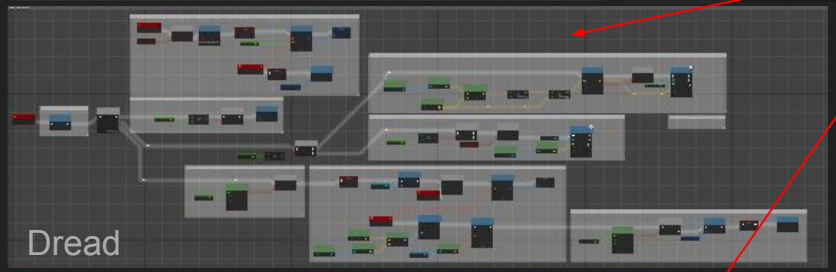


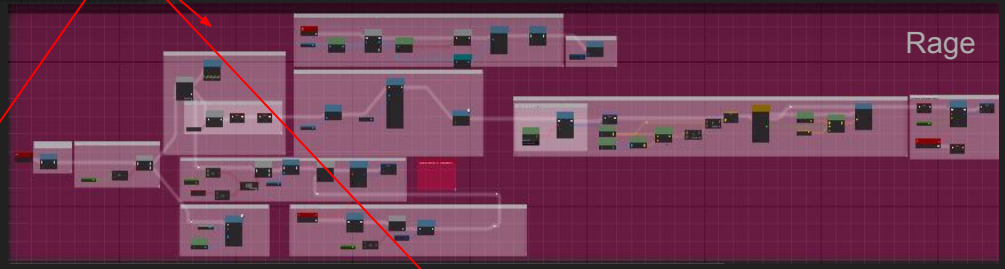
Behaviour Tree

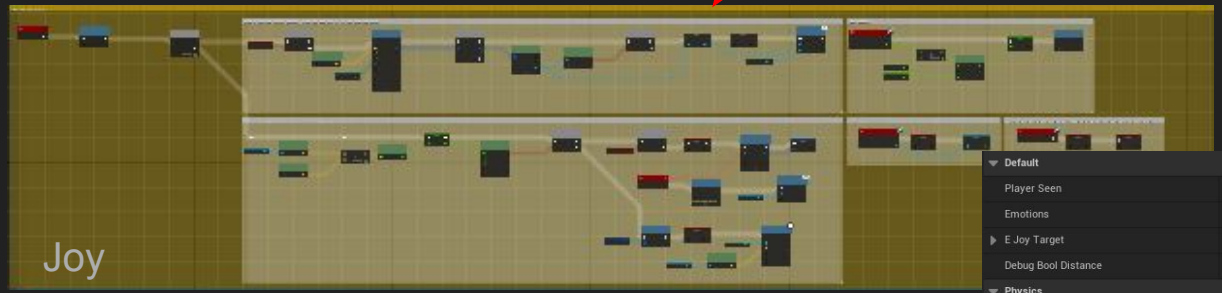
From chatting with him I then looked into fleshing out the 4 states from the design documentation, these included the base states, the AI was made within one state to allow the character to change to different states on the the go. When not in action the other states are turned off.

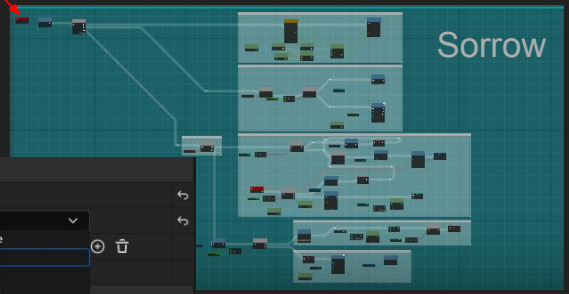


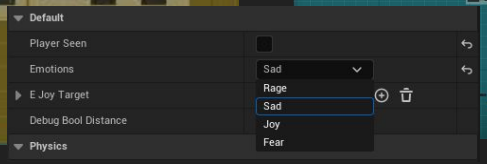






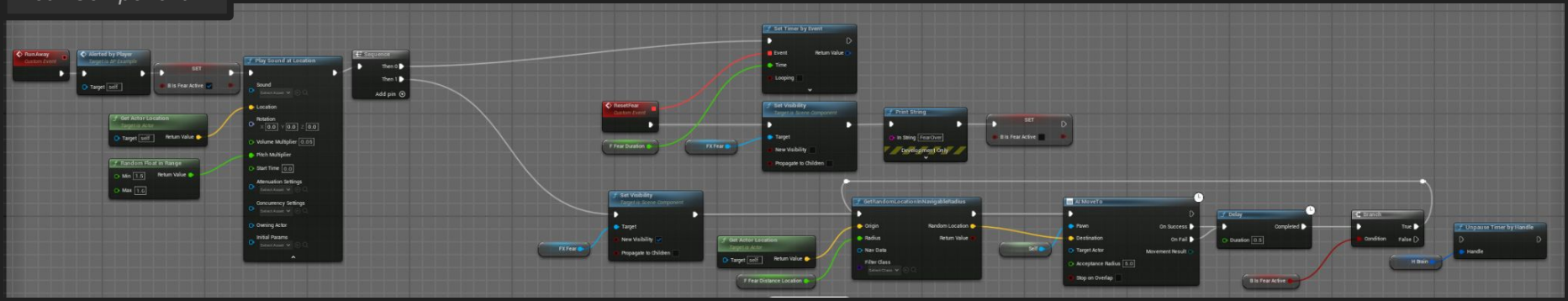




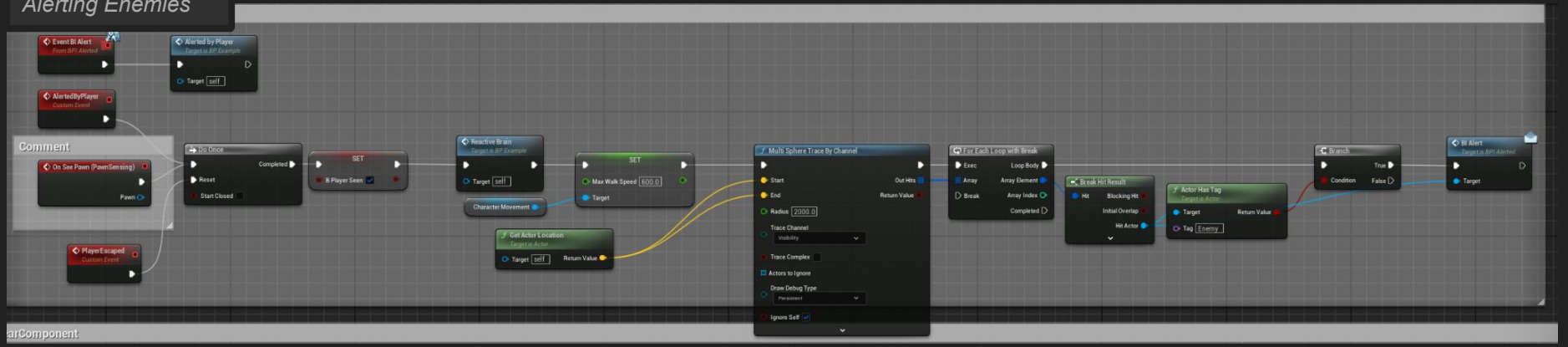


From here additional states for the player were made, these were added for more interaction

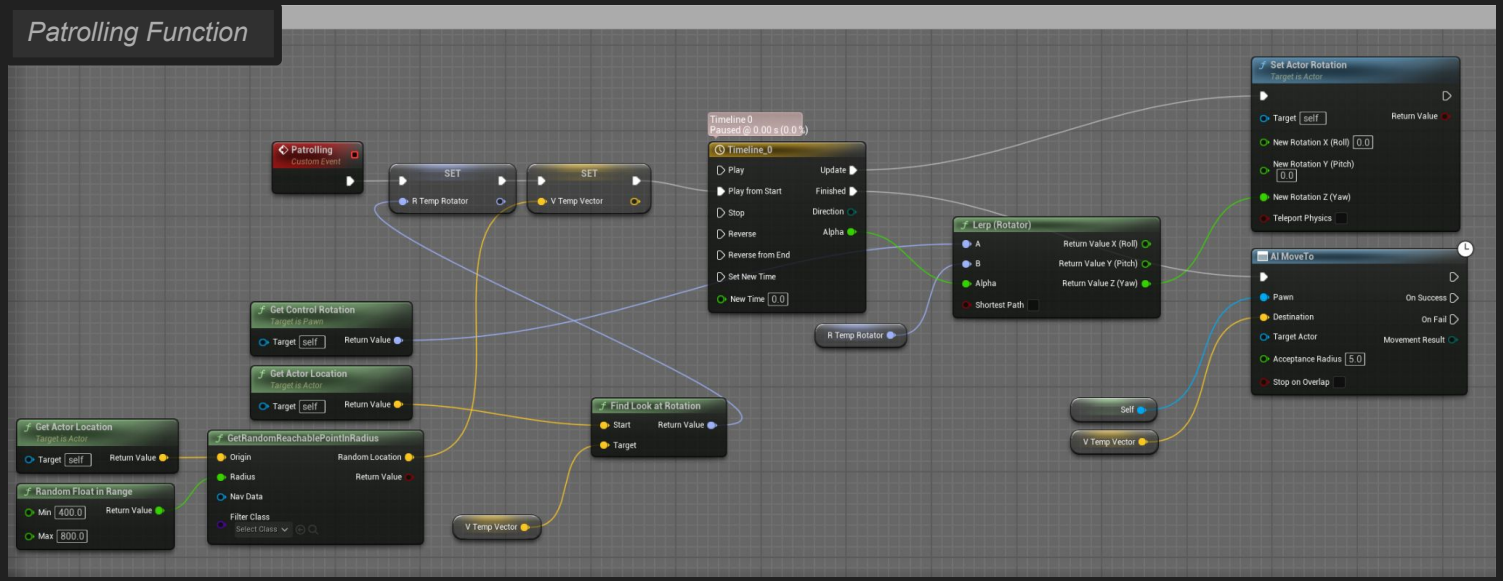
Fear Component



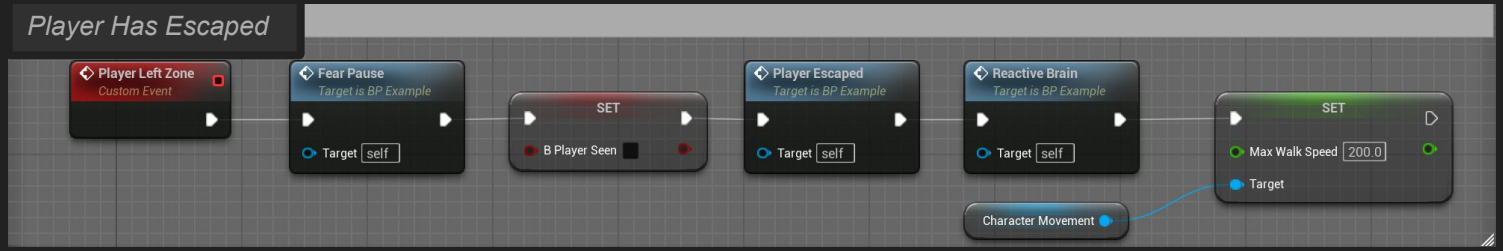
Alerting Enemies



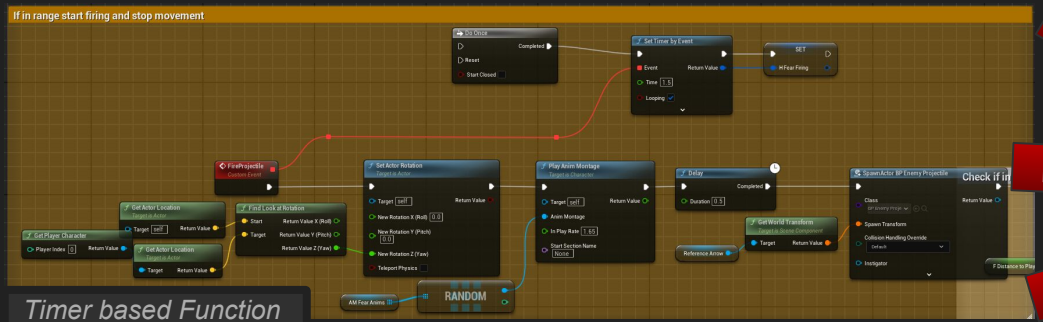
Patrolling Function



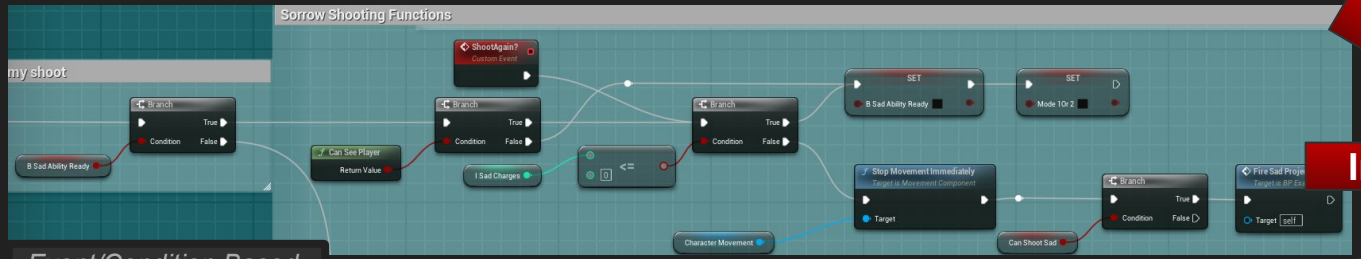
Player Has Escaped



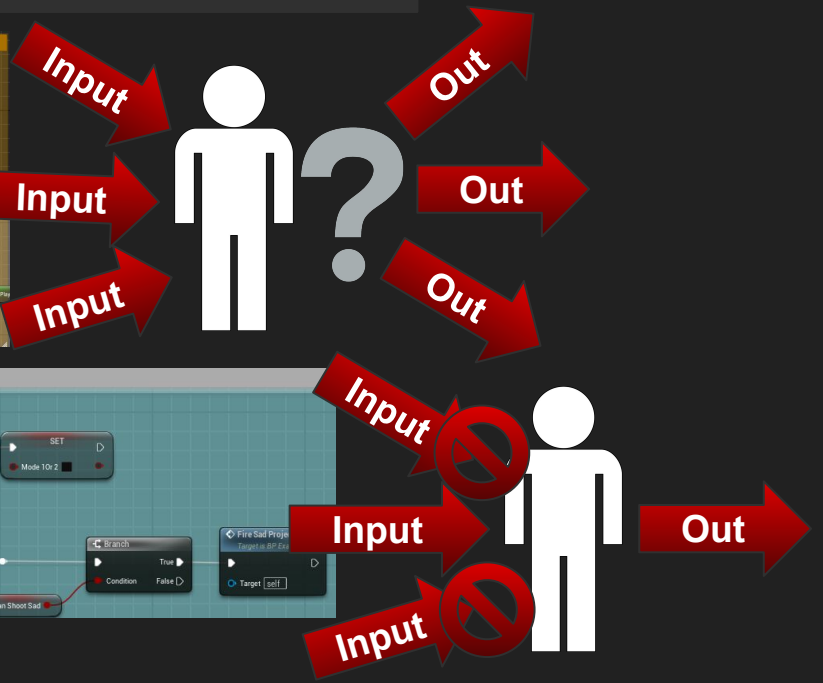
When I first developed the AI, I encountered issues stemming from nested timers conflicting with each other within the blueprint. This resulted in incorrect states being triggered at the wrong times. To resolve this, I revamped the structure by converting nearly all events into tasks. Now, each task must be completed before the next function can fire or be triggered again. This is mostly controlled by bools and gates.



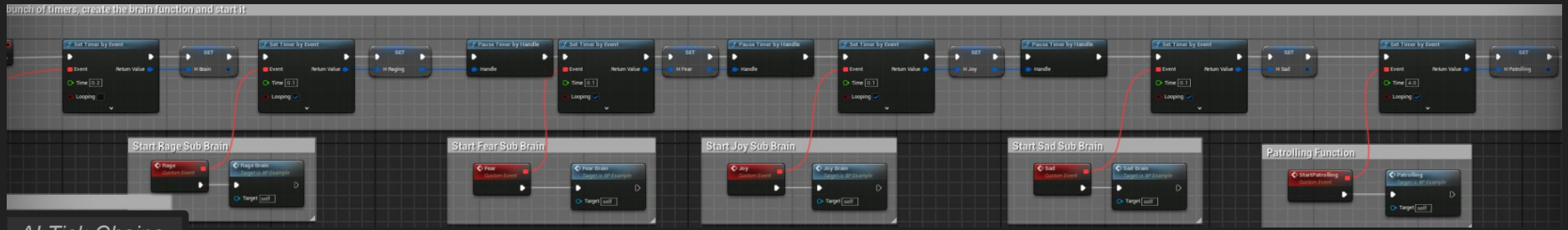
Timer based Function



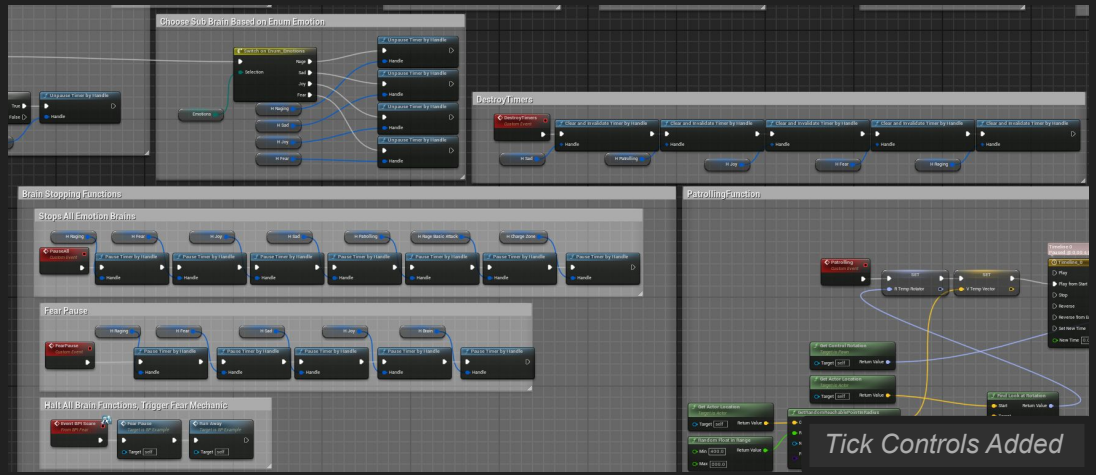
Event/Condition Based

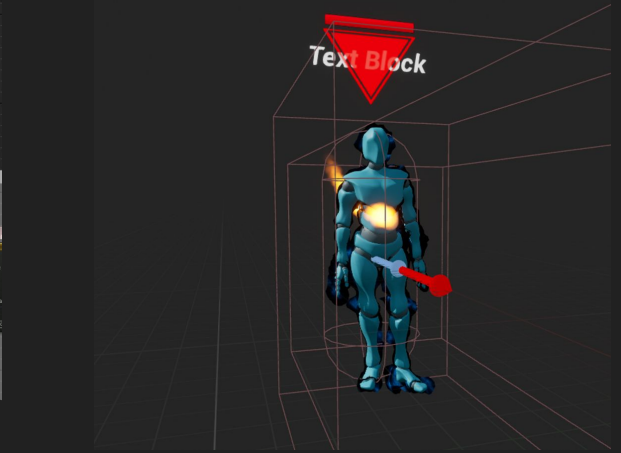


Following getting my work Reviewed I found I could optimize my AI more. During evaluation with a lecturer I found a lot of unnecessary timers were active when they were not needed. For this I looked into deactivating unescacry chains.



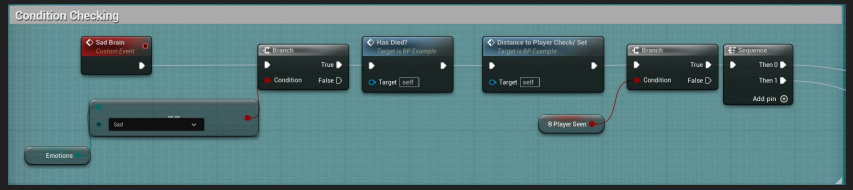
AI Tick Chains

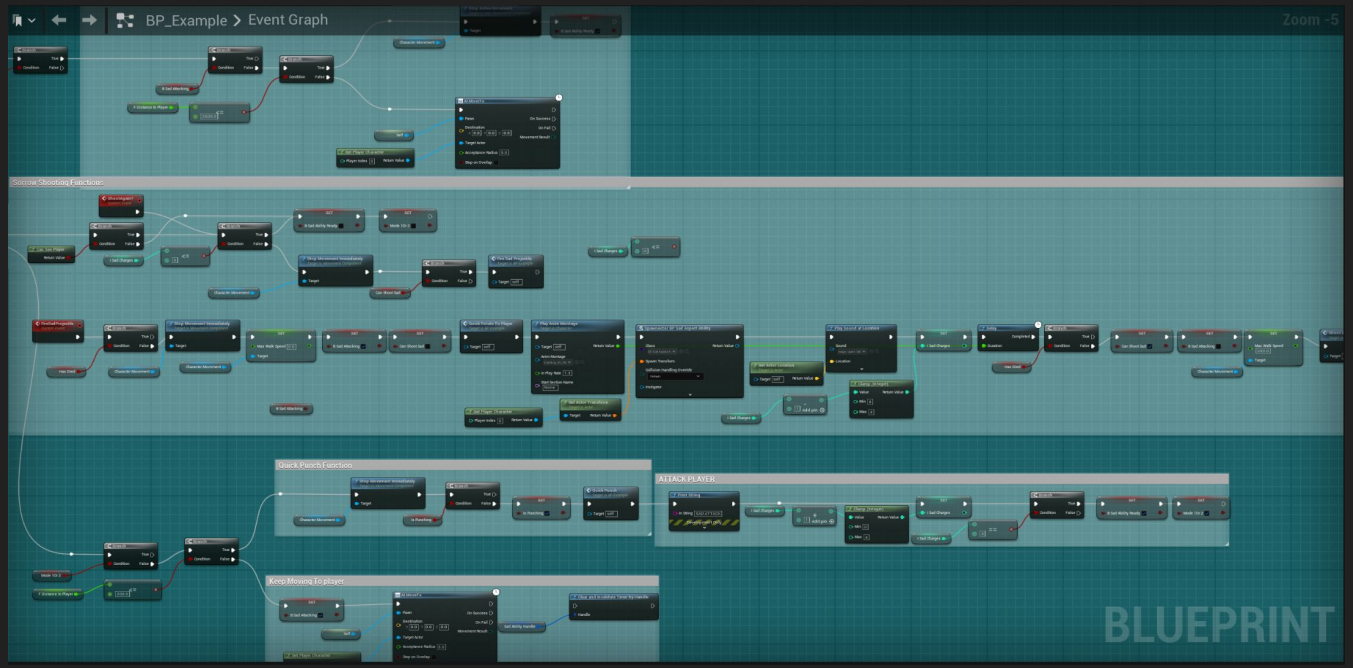




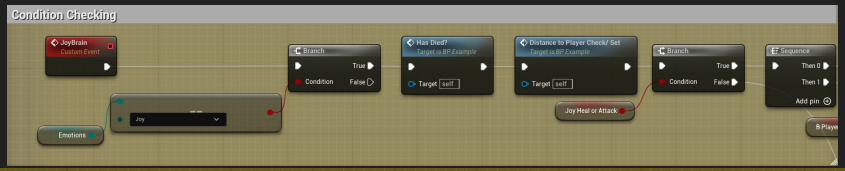
Tick Controls Added

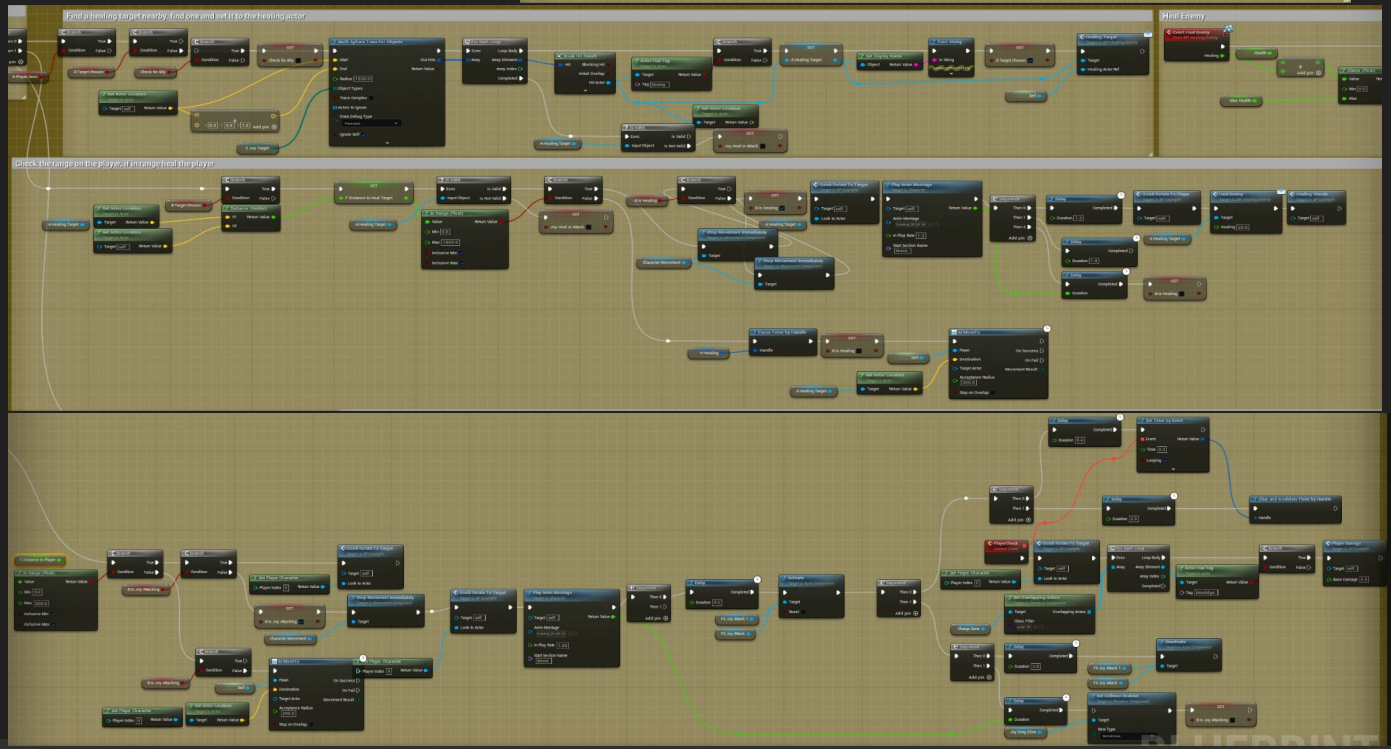
Sorrow Character State Machine



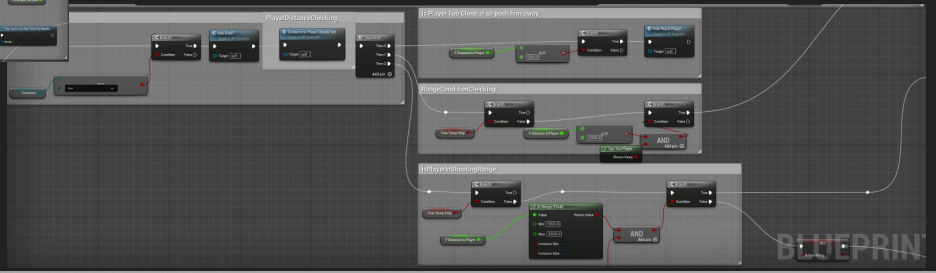


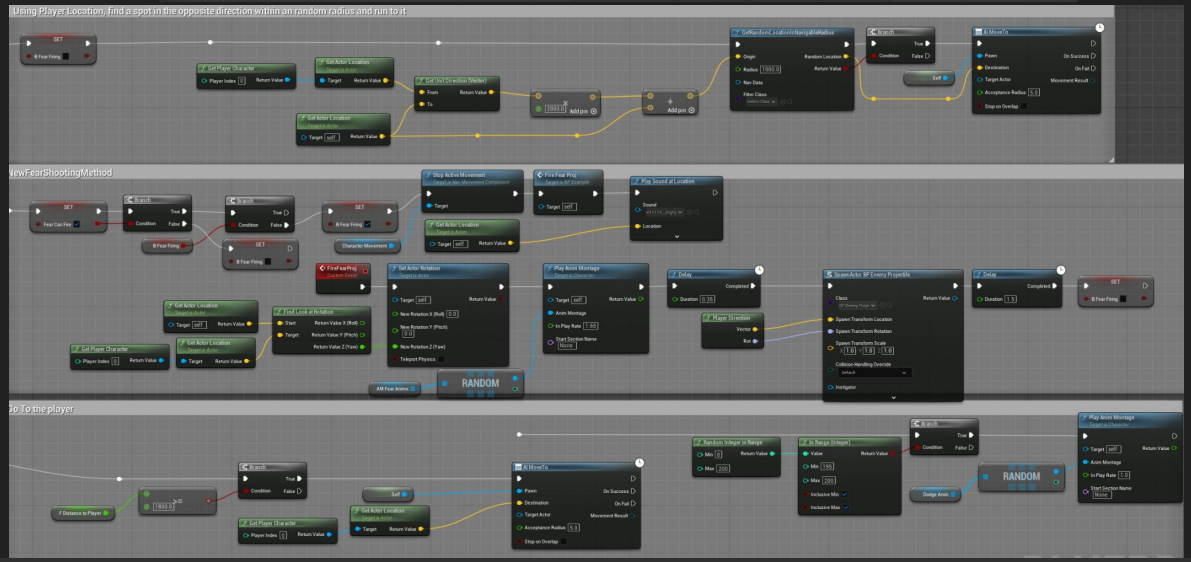
Joy Character State Machine



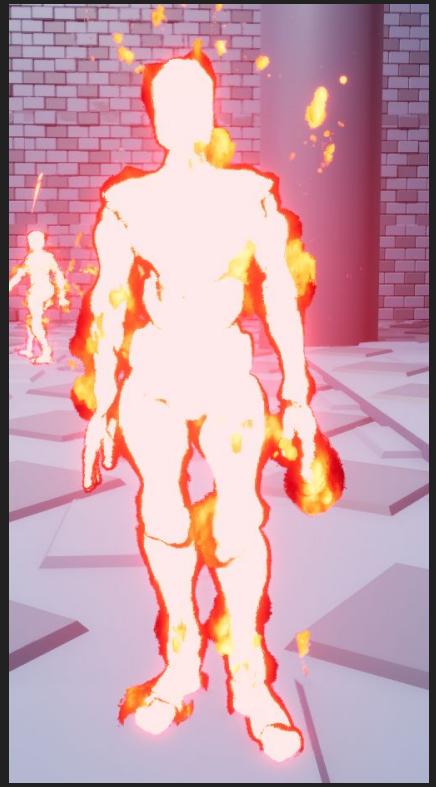


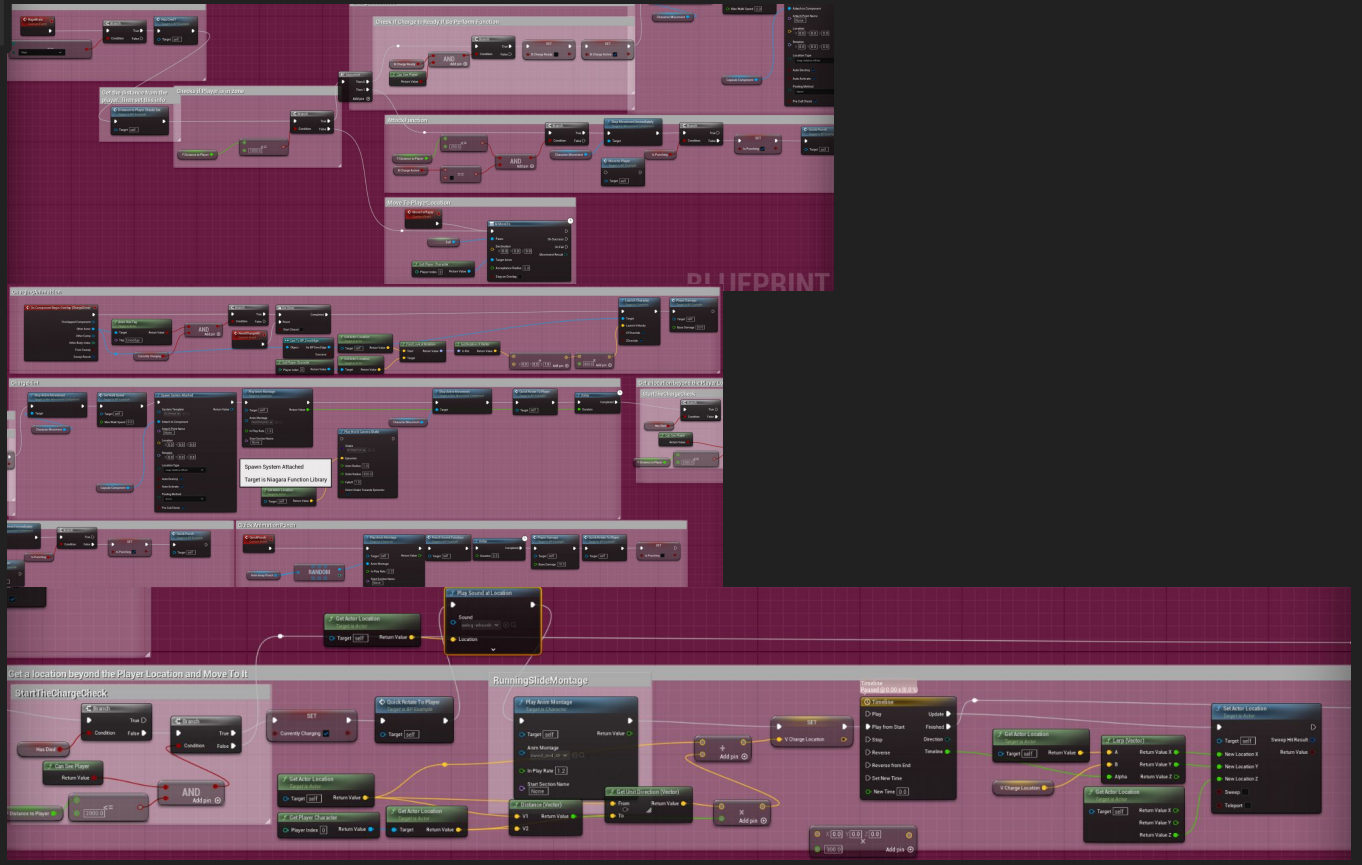
Fear Character State Machine



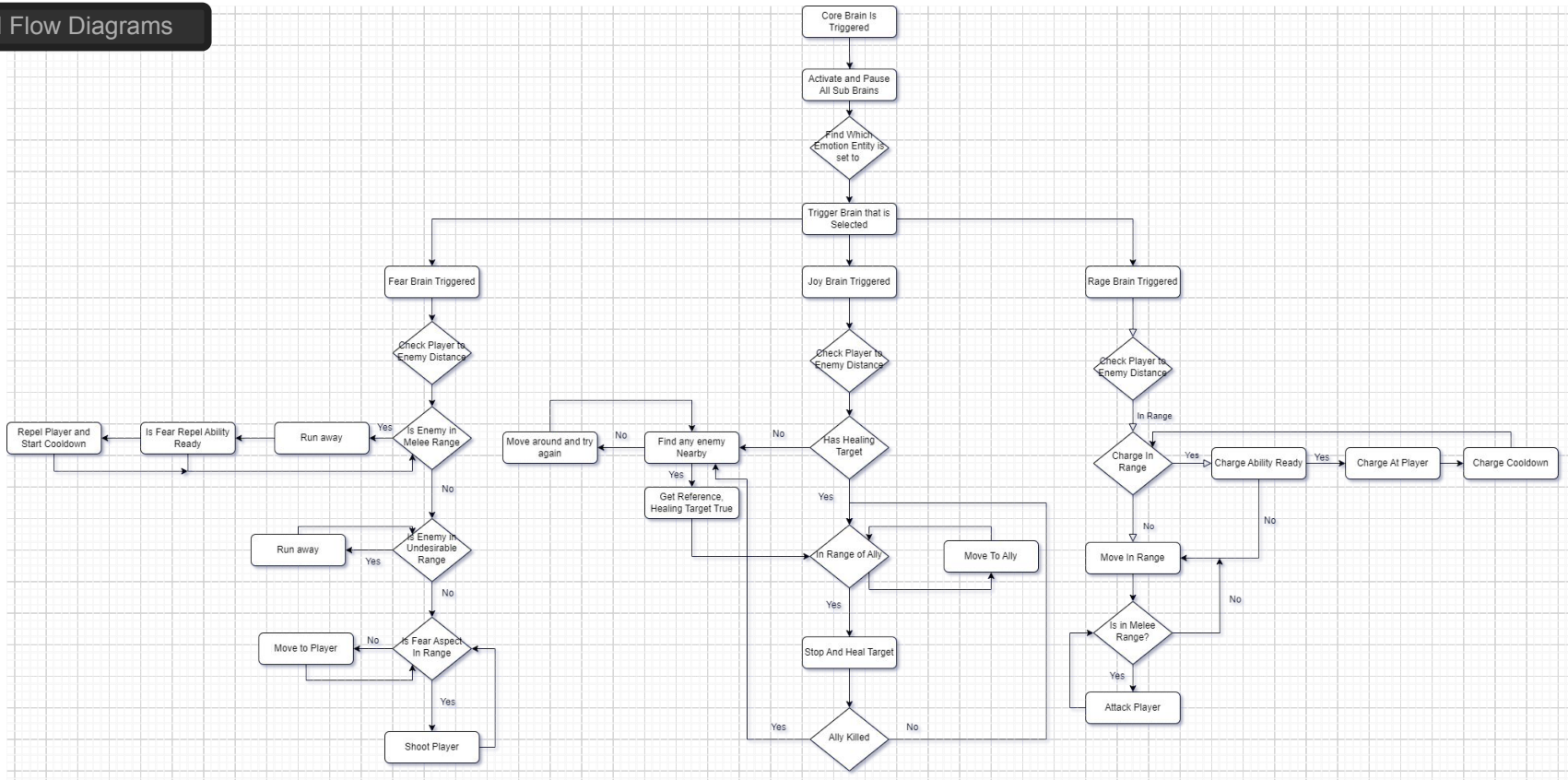


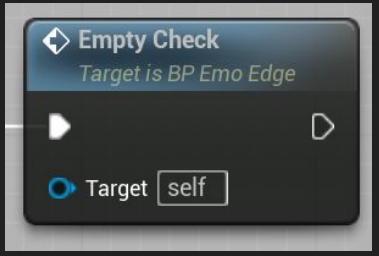
Rage Character State Machine

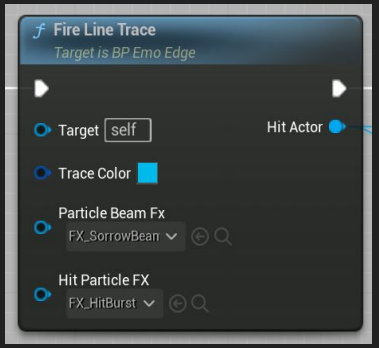


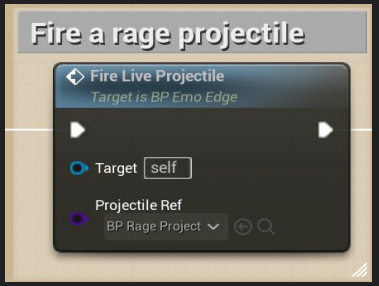


AI Flow Diagrams

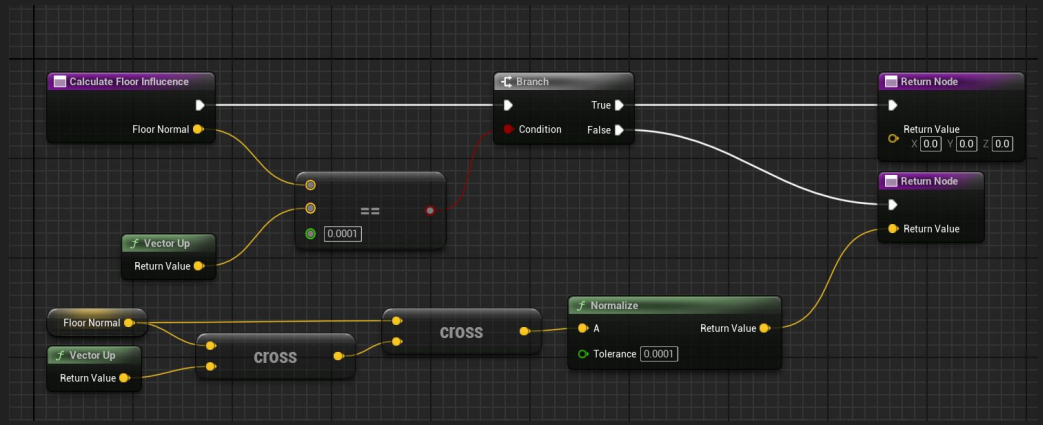




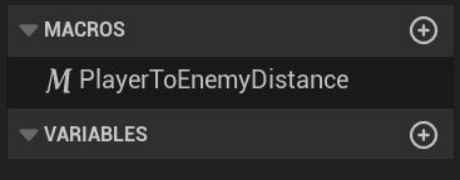




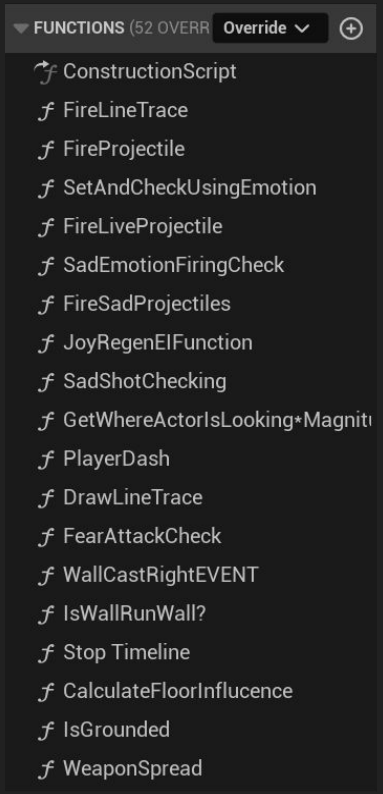
For common events many systems would use, I made functions, to reduce clutter and prevented me wasting time making the same function again and again.



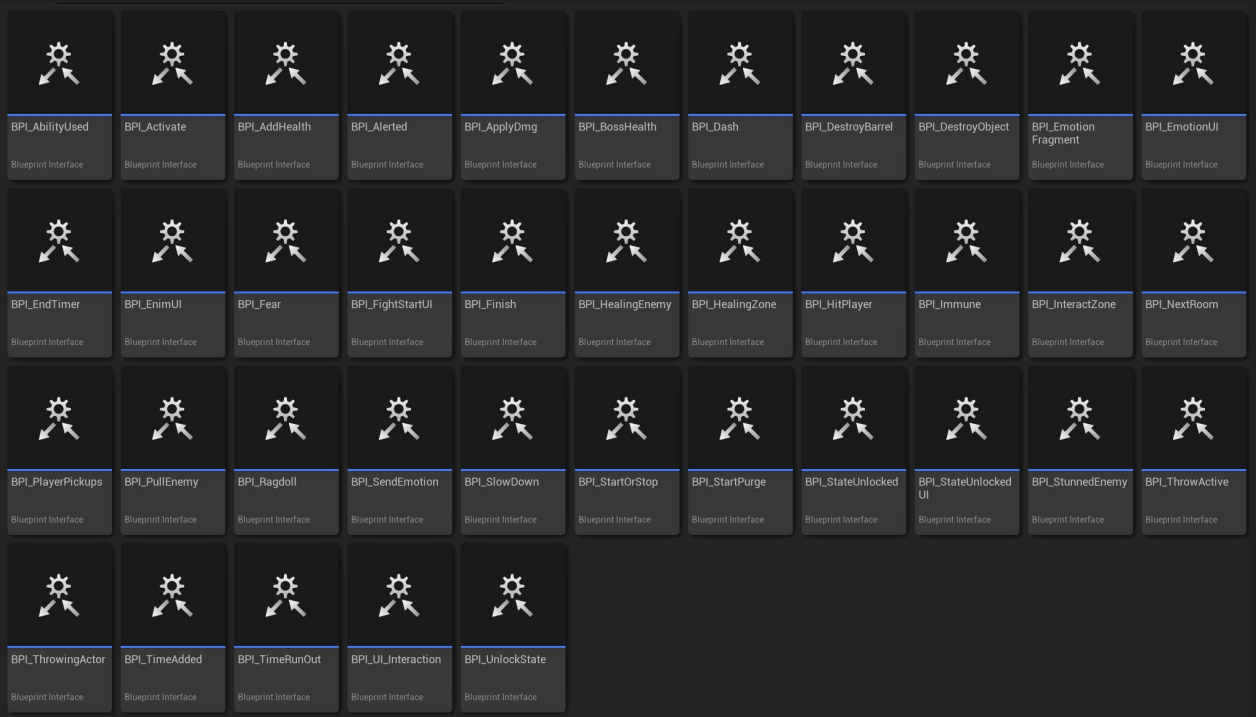
During the projects time I found while I could use macros they were not needed as many functions did not correlate much with other gameplay mechanics.



Custom Pure Function Nodes

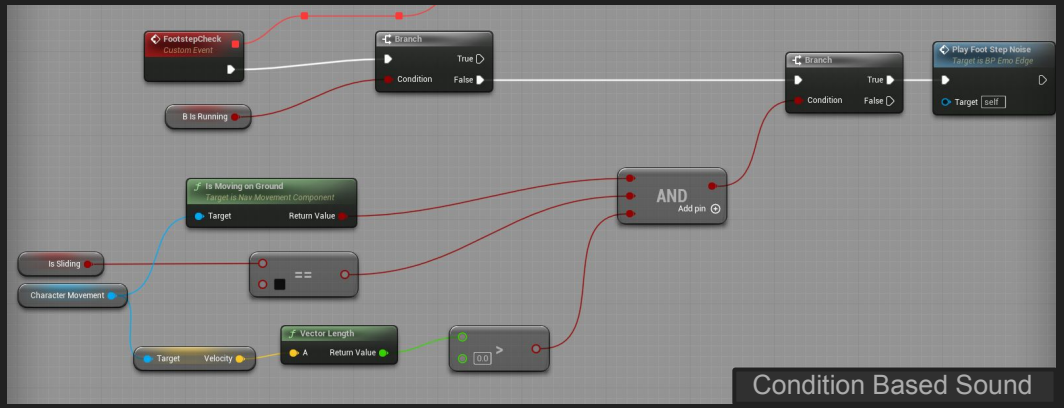


Player Functions



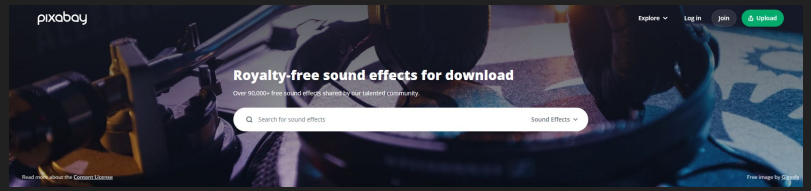
Functions and Blueprint Interfaces (BPIs) were extensively utilized to maintain a clean and efficient workflow. These were used for events and sending information, This improve efficiency and readability and is good practice.

To improve the feel of the game I looked into adding sounds for the project, this helped with improving the feeling of impact the player,enemies and environment had on the game.

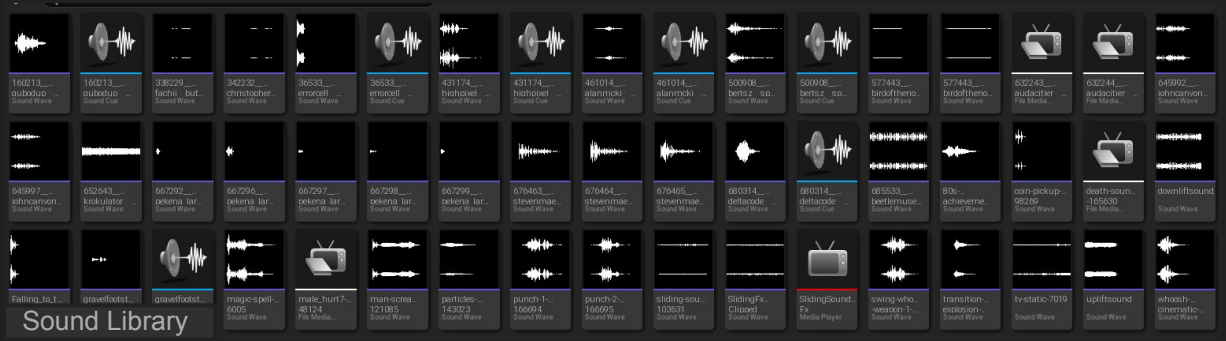


Condition Based Sound

Methods Of Playing Sound

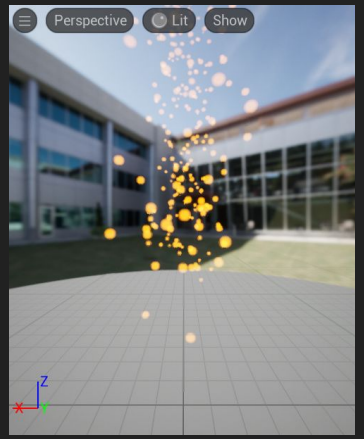


For sound on the project a mixture of personally making on FL Studio and royalty free sourcing was done.

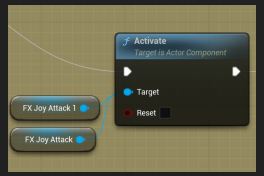


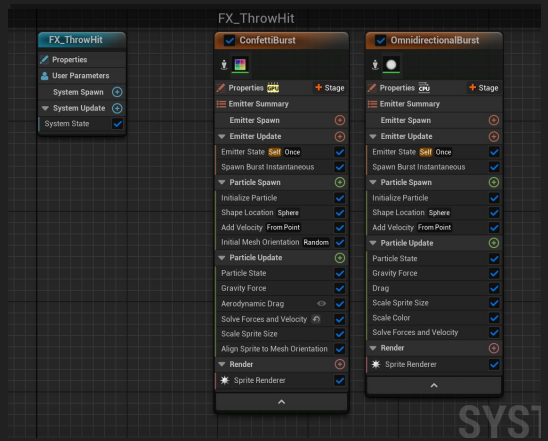
Sound Library

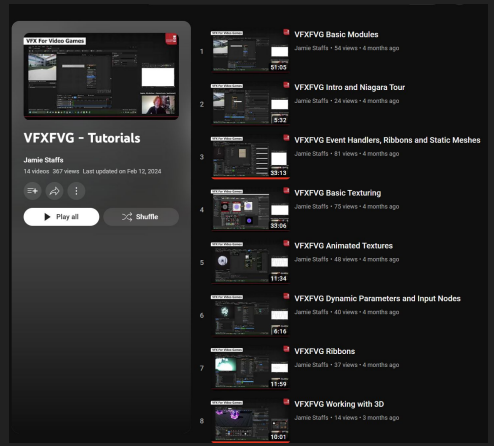
FL Studio

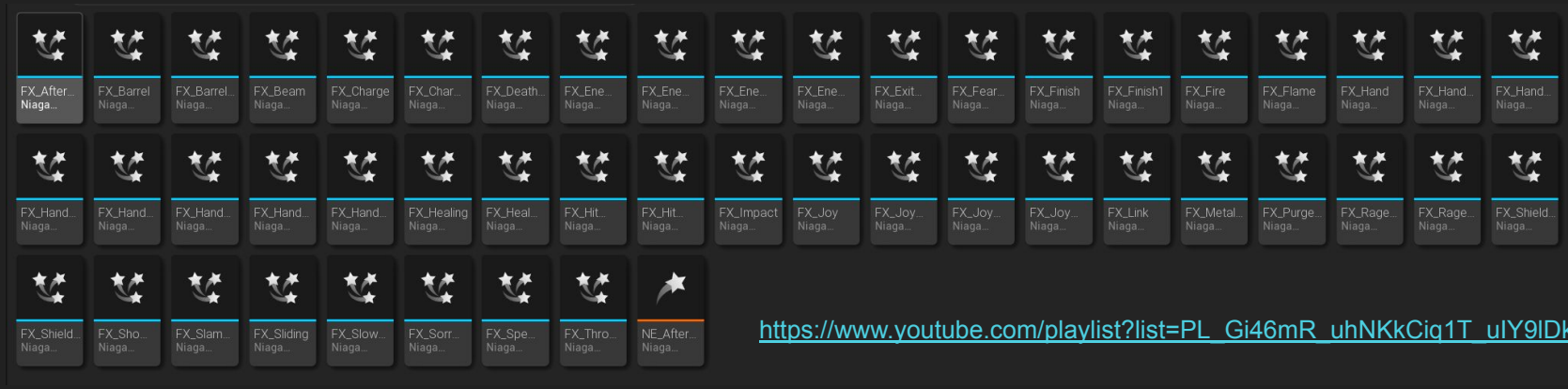


During the Project I looked into making small amounts of VFXs for the environment and its characters. For this I followed a series of VFXs Tutorials made by Jamie Nicholas

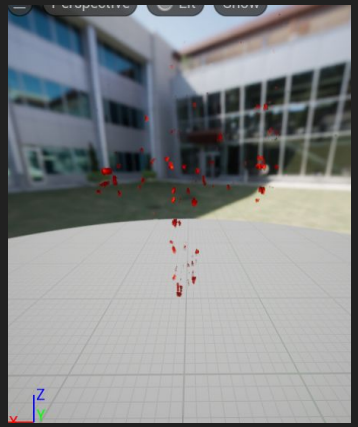




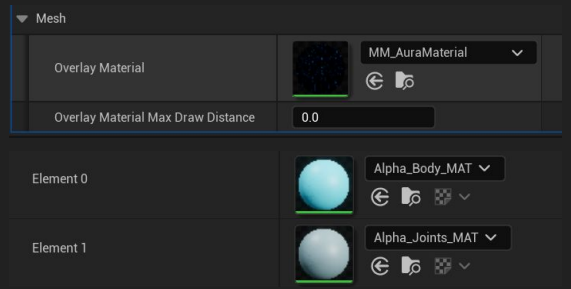




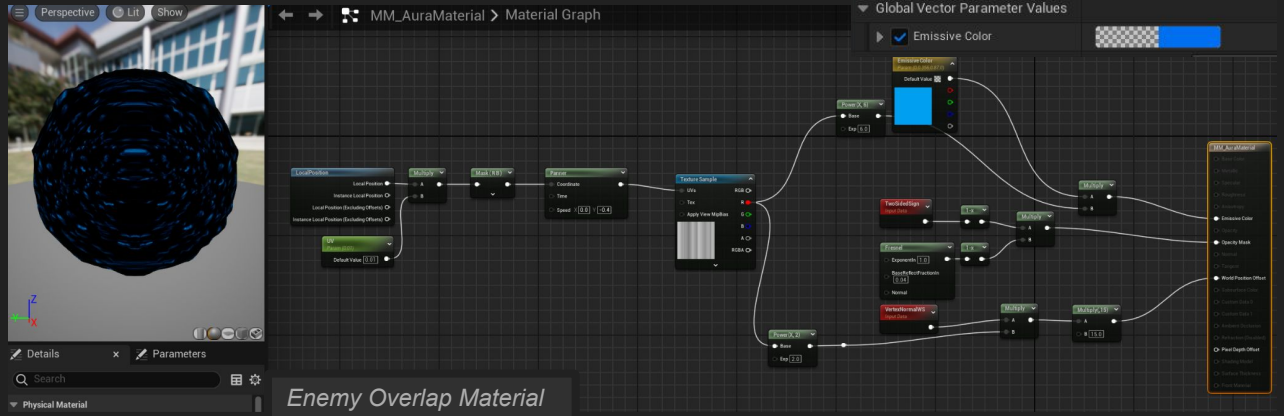
https://www.youtube.com/playlist?list=PL_Gi46mR_uhNKkCiq1T_uIY9lDkYYg9jk



Alongside Particles VFXs I looked into making custom Materials for the enemy chars. This started as a basic solid colour material, from here I learned about Overlay Materials as well as Mesh Based Particle Systems

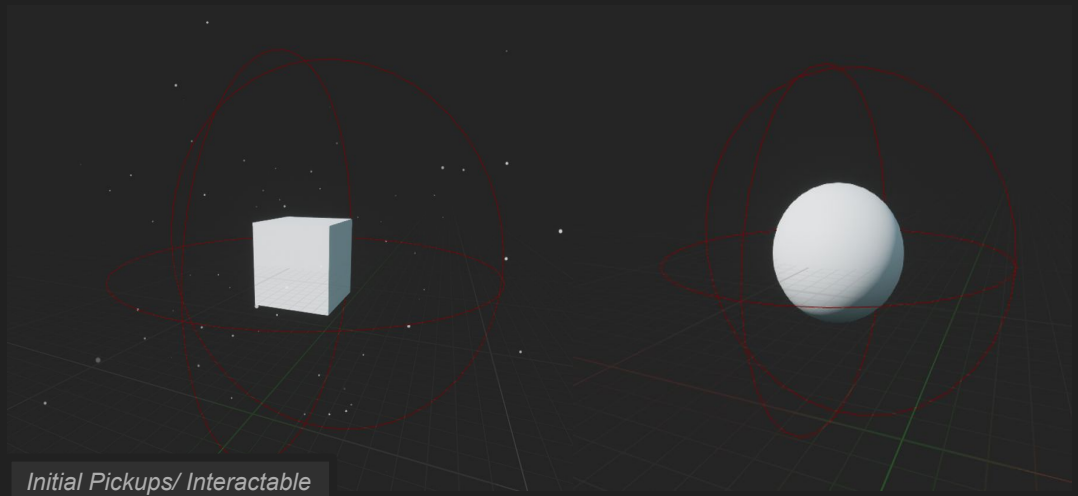


Making this had me learn about material blueprint and how to make more complex systems



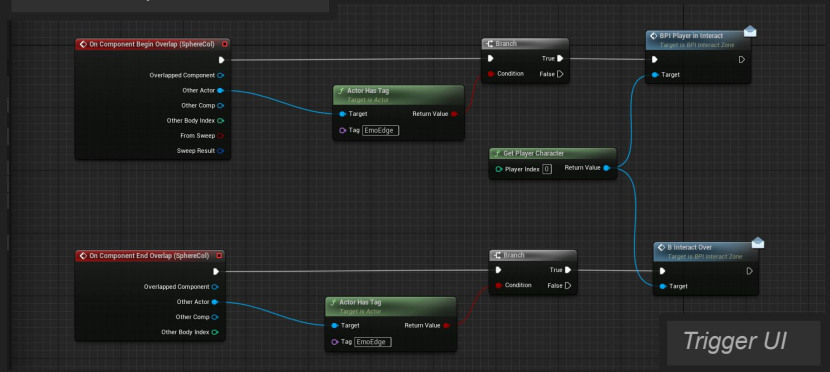
Enemy Overlap Material

https://www.youtube.com/watch?v=6lW5XeClv8I



Initial Pickups/ Interactable

Interactable UI



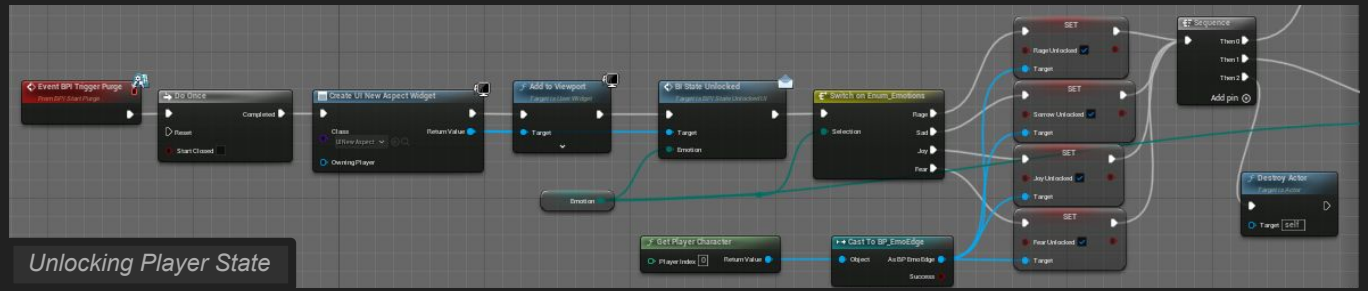
Trigger UI

Initially, development for interactables began with a debug cube. When the player held the "F" key, a line trace was fired, triggering a timer to start before a UI text prompt appeared. However, after testing, it was clear that players did not enjoy the abrupt interruptions to gameplay caused by interacting for small snippets of information or the sudden enemies from random spawners. With this I revised the interactable mechanics to be less essential to the game's flow and less disruptive overall.

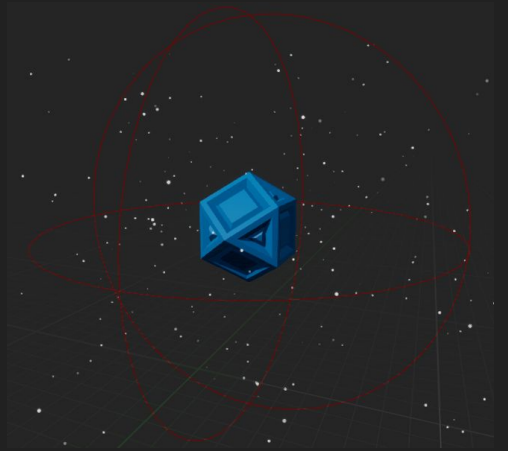
Developed Design And VFXs

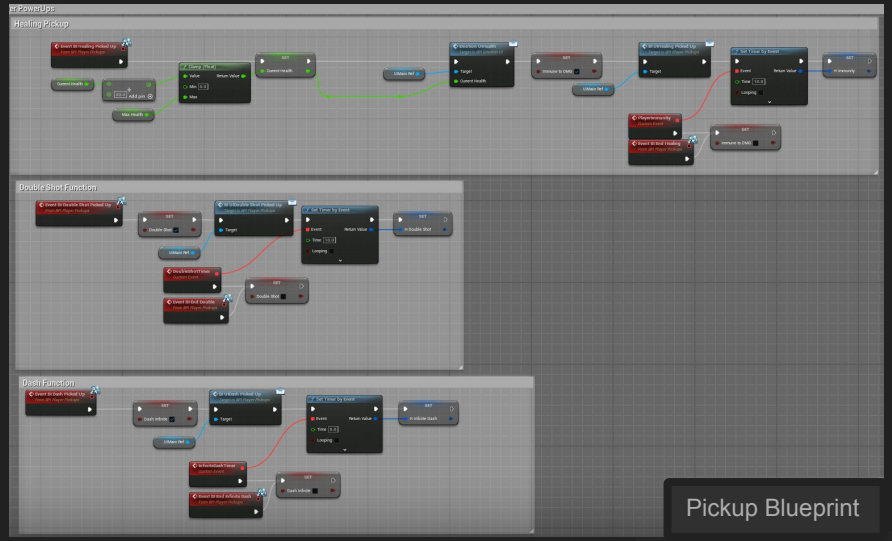
UI Widget Added When Grabbed

Based on previous feedback, I revisited the role of interactables in gameplay. I designed them as a means for players to acquire new aspects of the game. These interactables now act as breaking points between sections, providing players with a checkpoint to prepare for upcoming content.



Unlocking Player State





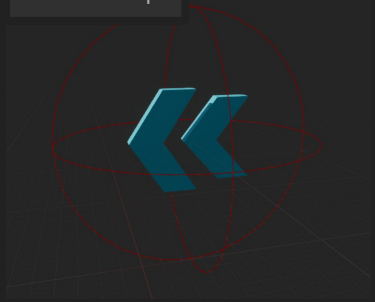
Pickup Blueprint

For more Dynamic gameplay pickups were made, these could be found or dropped by broken barrels.

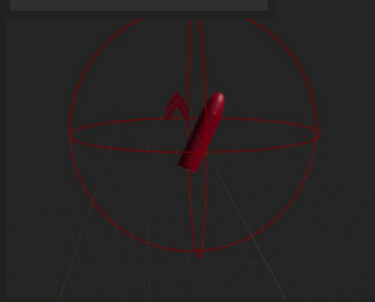
This was done following feedback suggesting to add more dynamic arcade elements.

While these were tertiary to the project it did improve gameplay flow

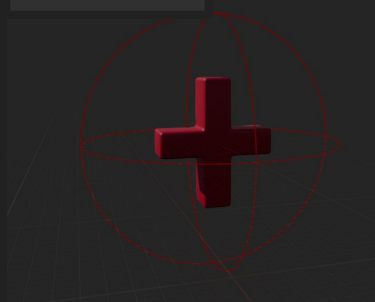
Dash Pickup



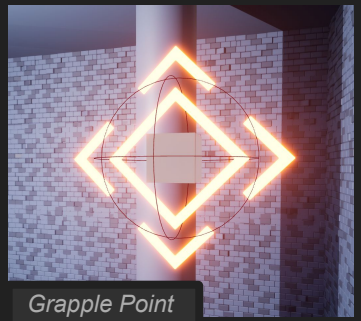
Double Shot Pickup



Healing Pickup

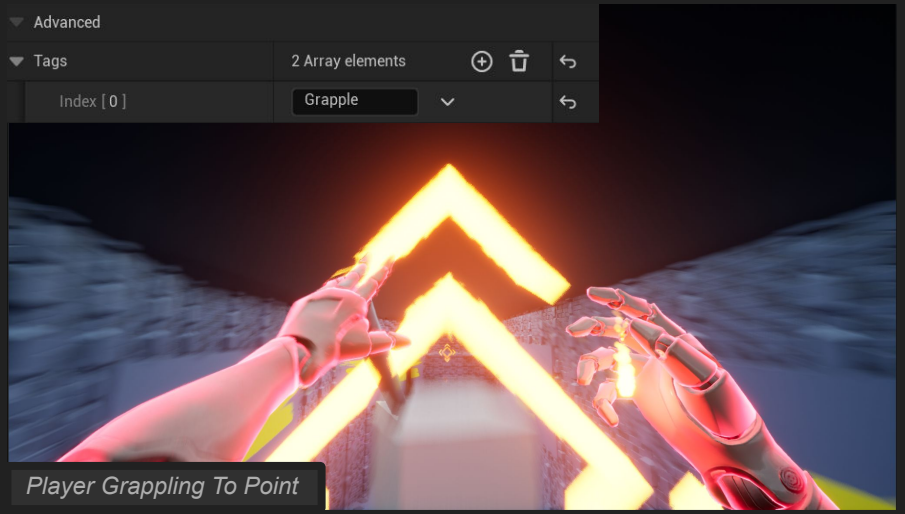


Grapple Mechanic Development

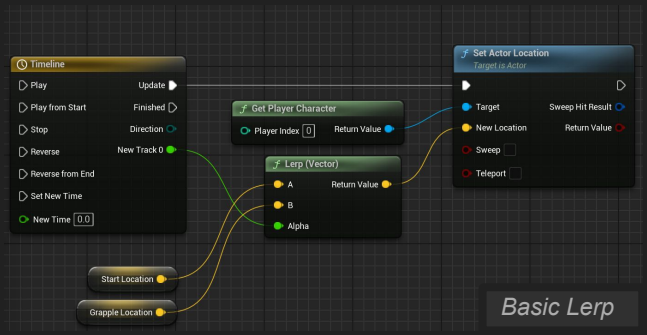


Grapple Point

Through Player Testing with the Initial Builds players found they were lacking with connectability between movement based mechanics and wanted more forms of High Octane Movement between, others. For this I looked into making an Grapple Mechanics. This would be something Similar to Ghost Runners Mechanic

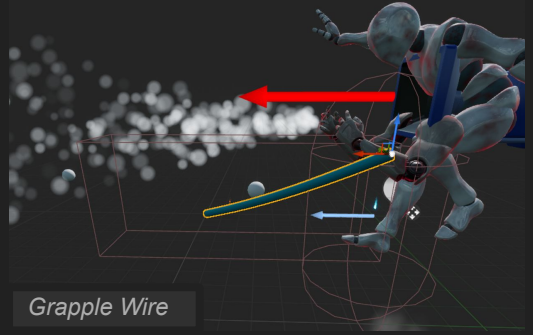


Player Grappling To Point



Basic Lerp

This Initially Started with a simple Lerp Vector Should a Line Trace hit a Actor with the Tag: Grapple



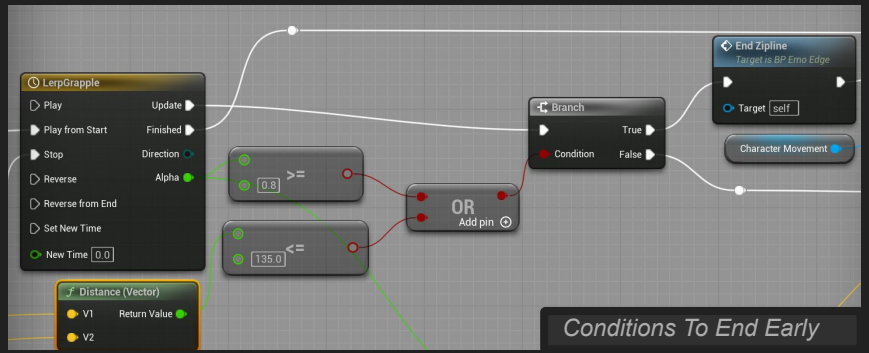
Grapple Wire

Alongside the Initial Simple Lerp a Cable Was Added to the Player and Bound the hand, this would appear only when grappling and would bind to the end position

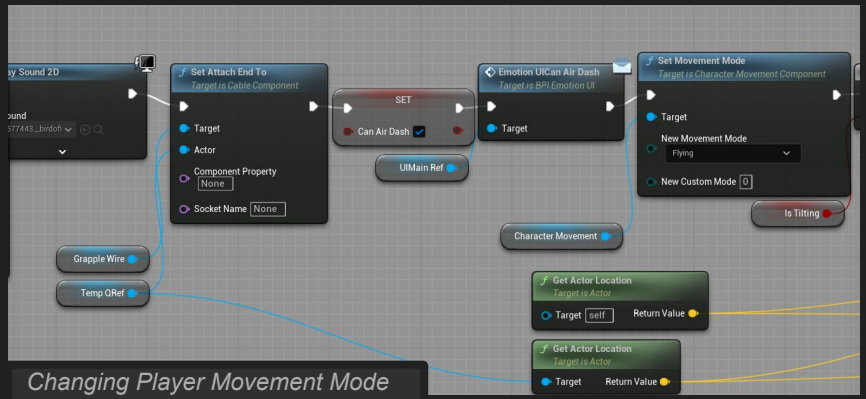
After creating the grapple, I encountered two issues:

- The player experienced glitches while attempting to reach the target point.
- There was a lack of satisfaction when using the grapple.

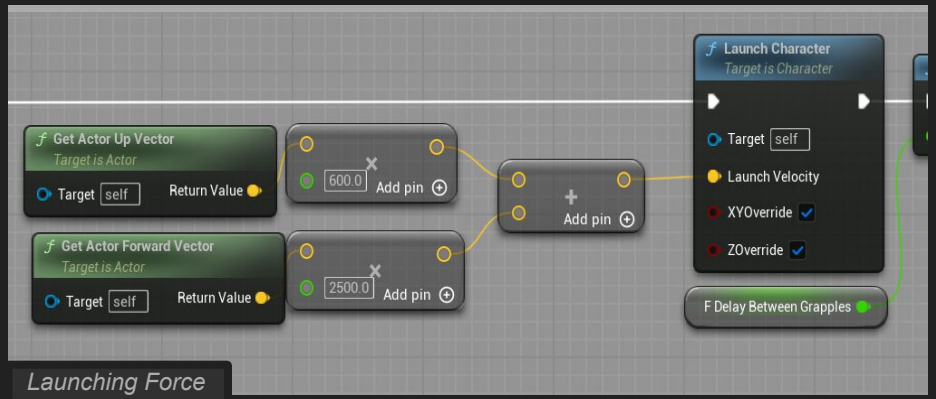
To Improve this I looked at ending the grapple when the player got near to the end of it, additionally when the reaches the end they get a burst of force added in the directions of connection, this addition was well received during testing.



Conditions To End Early



Changing Player Movement Mode



Launching Force

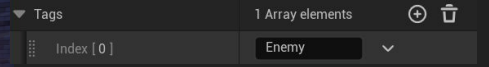
Further Grapple Development

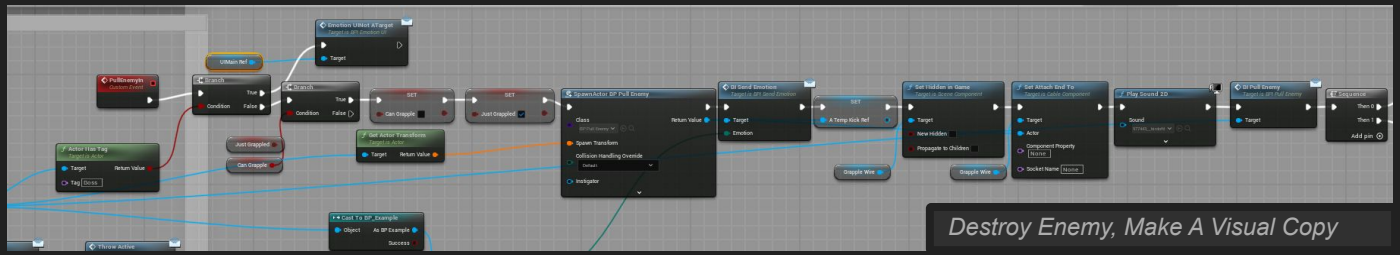
Following making the grapple and doing some testing players wanted more mechanics based around the grapple, between shooting and reloading a way to kill enemies. For this I looked into making a pull mechanic for enemies. This would instantly kill them.

This system was added to the grapple line trace and would work if a Enemy was hit by the line trace, a copy is made and thrown

Pulling Enemy

Kicking Enemy



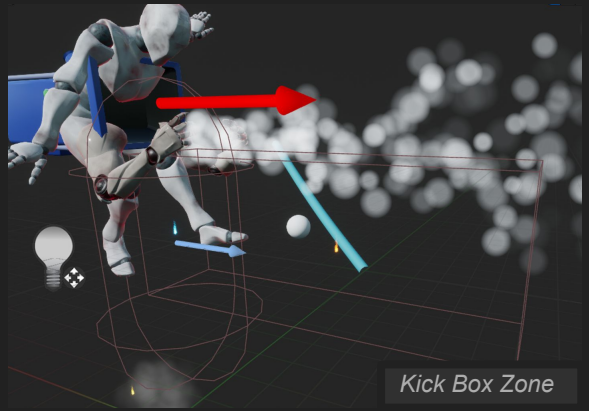


Destroy Enemy, Make A Visual Copy

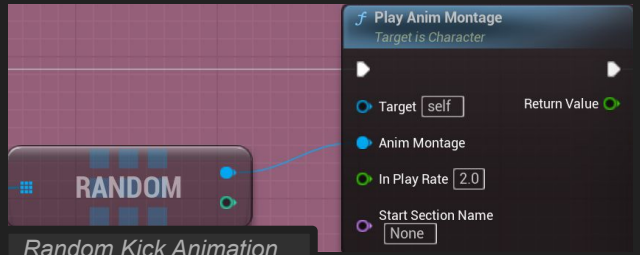
New Actor Spawned

Following making these changes I looked into adding more methods of defeating enemies. For this a kick mechanic was added. This started as an simple box trigger that checked if any enemy or item is inside it.

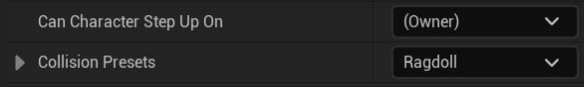
This initially was a box collision that would destroy any actor with the right tag. From Here I added Physics and Ragdolls to enemies for extra flair, alongside this a kick montage was added to improve the feel of impact



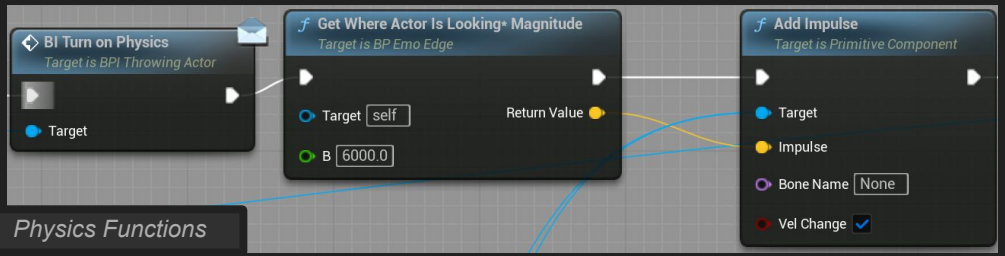
Kick Box Zone



Random Kick Animation



Enemy Collision To Ragdoll



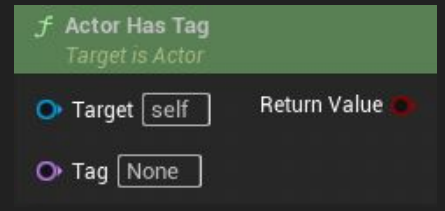
Physics Functions

Angerfoot Gameplay

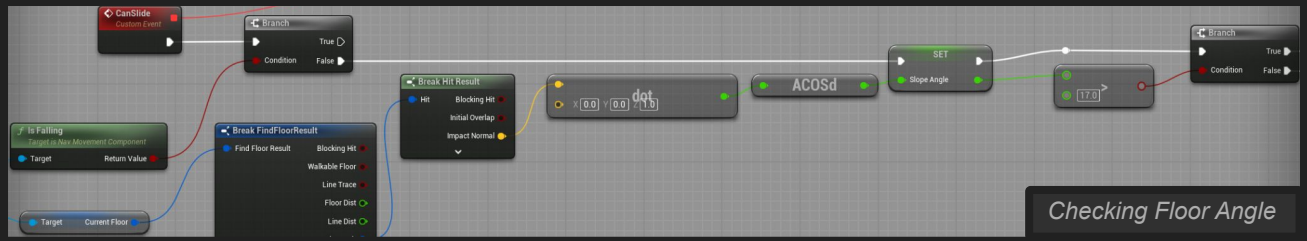
Kicking Door At Enemy

https://www.youtube.com/watch?v=HewLoAfsAeY

For the project, I starting creating a sliding mechanic. Initially, I attempted to constantly check the floor for a Sliding Tag Actor, which proved inefficient. Instead, I made a more optimized approach. The new system does a floor checking mechanism when the player holds the crouch button. It calculates the floor angle based on two points on the object and the player's feet, forming a triangle to determine if the floor angle exceeds a certain threshold.



Mirror's Edge Sliding



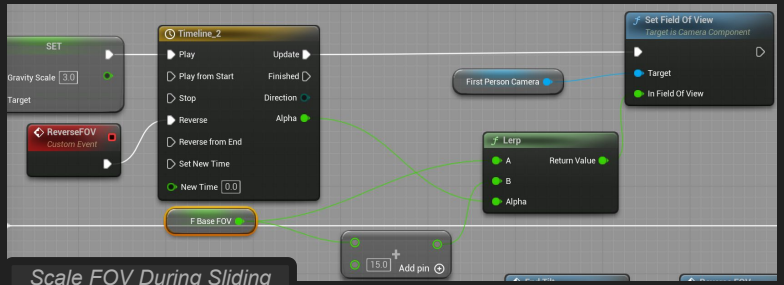
Checking Floor Angle

While this did initially work there was not much satisfaction in doing so and little reward. For this I looked into adding more polish into the sliding mechanic. For this I looked into adding three things. Those being:

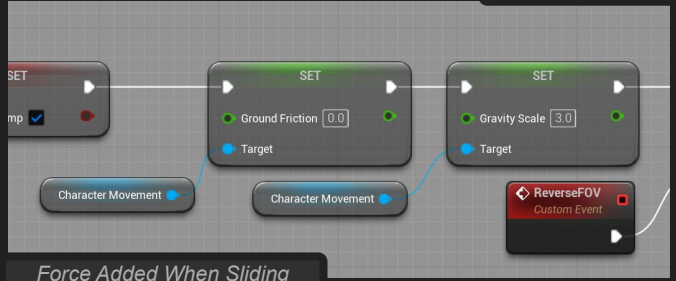
- More Speed
- Feeling Of Speed
- Connection to Other Gameplay Mechanics

This helped to improve game feel and flow and connectivity between sections

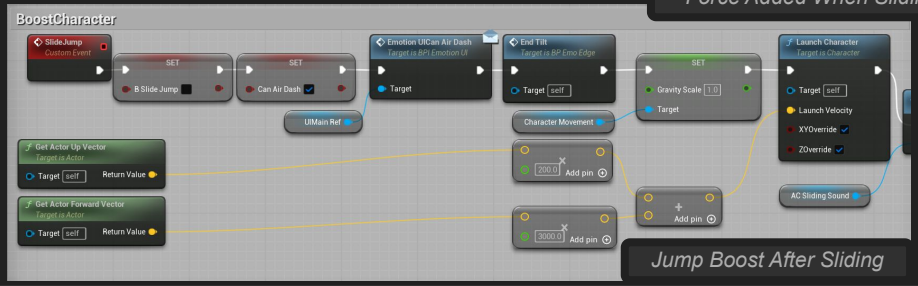
Player Sliding With Tilt



Scale FOV During Sliding



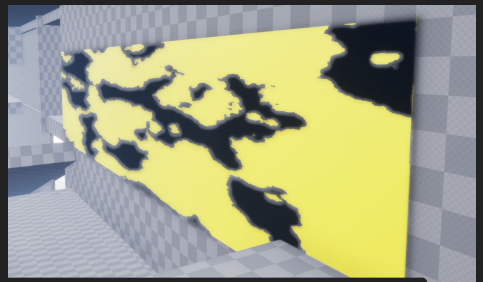
Force Added When Sliding

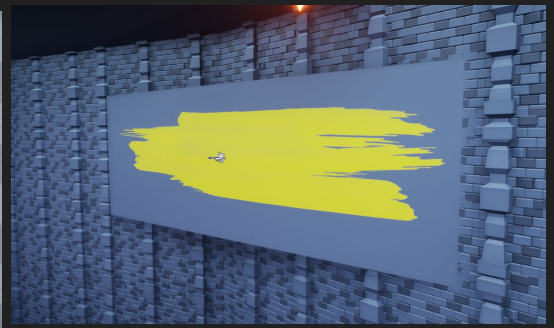


Jump Boost After Sliding

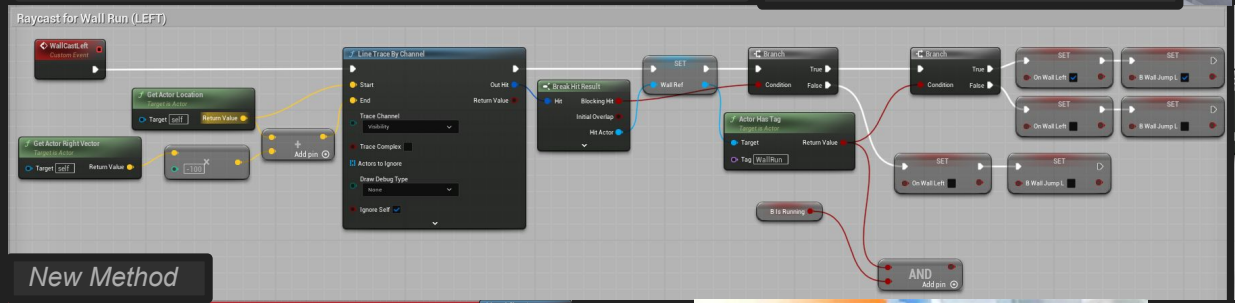
During the project, I explored implementing a wall running mechanic to enhance the high-octane goal I was going for. Initially, I attempted a wall check method that continuously applied force to push the player against the wall. However, this approach was unreliable and temperamental.

To improve this mechanic, I made a new system that switches the player's movement mode to flying while near a wall running section, this helped improve gameplay and make it feels smoother.

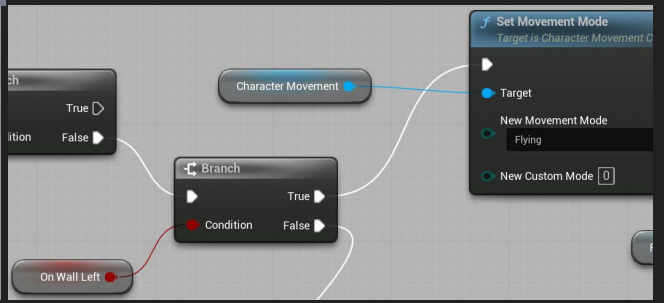


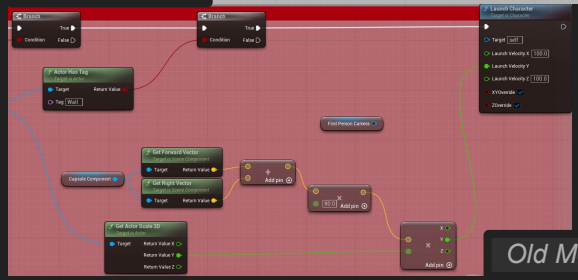


Old/New Wall Running Sections

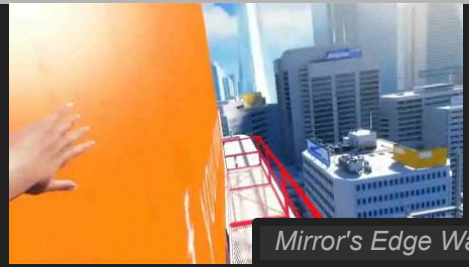


New Method





Old Method

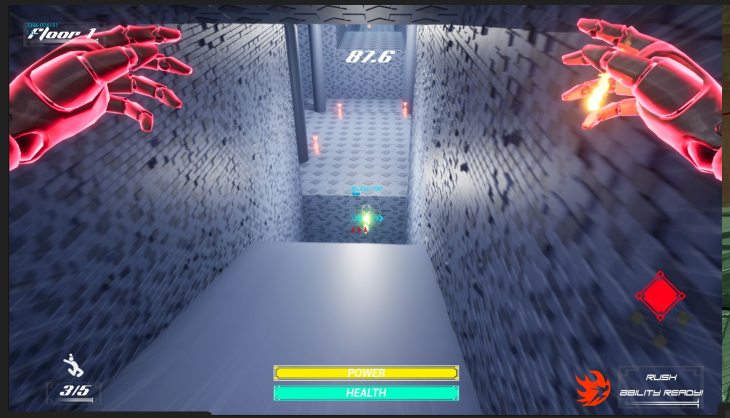


Mirror's Edge Wall Running

Wall Running

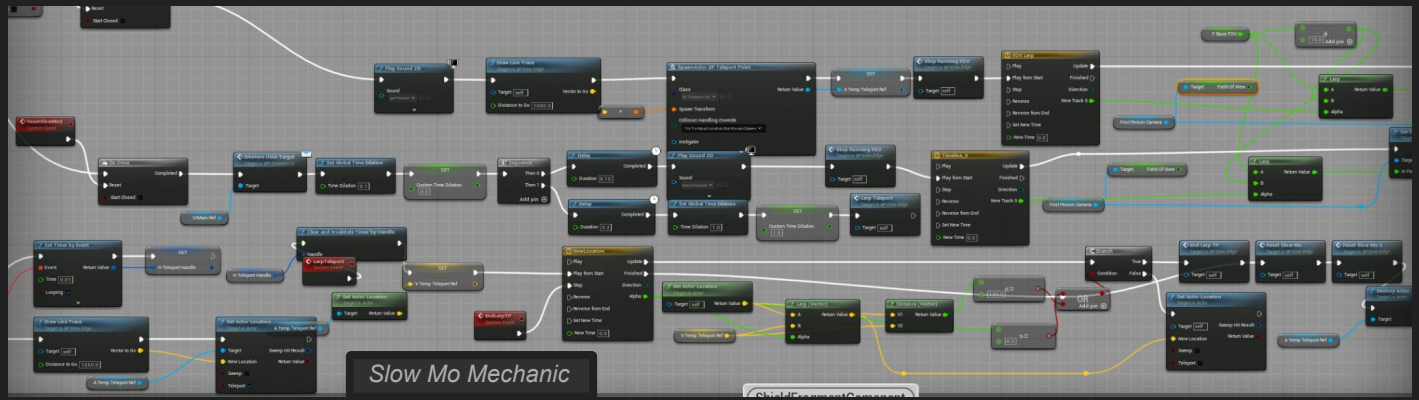
In addition to developing the wall running mechanic, I focused on polish by incorporating two key elements: camera tilt and UI indicators. These additions help tell the player when they are actively wall running, this helped with the feeling of High Octane.

Alongside this for indication to the player Yellow paint was added to wall running sections to make it clear to the player where they can and cannot wall climb



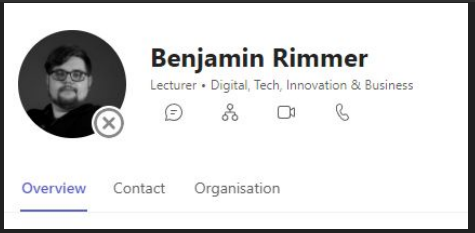
Slow Mo Dash

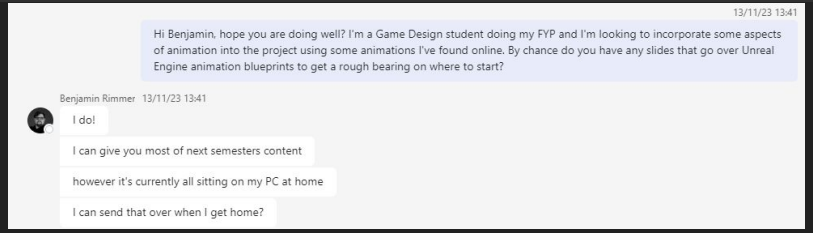
Ghost Runner

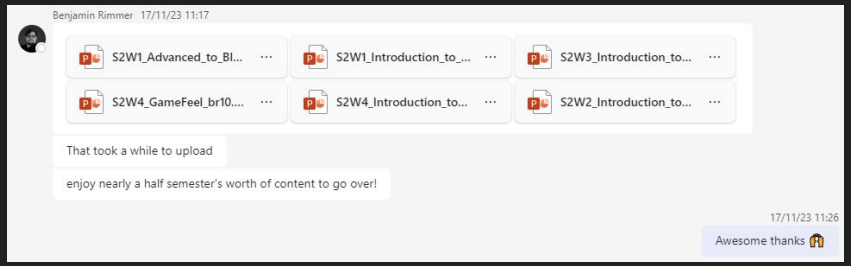


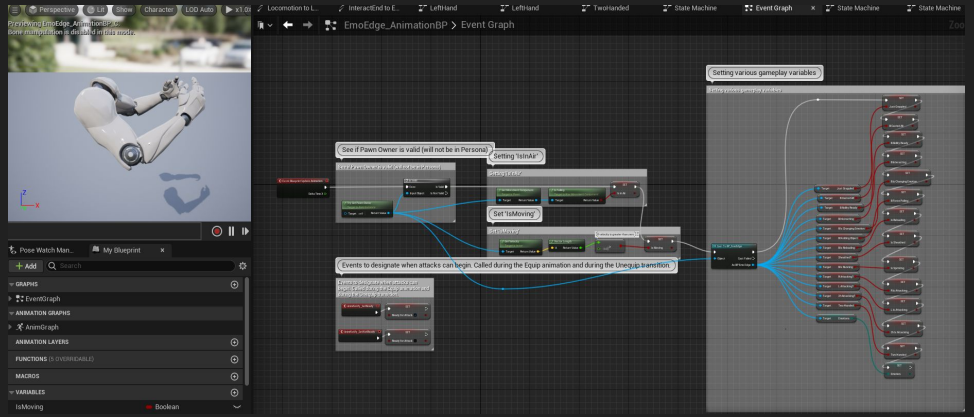
Slow Mo Mechanic

In addition to the new mechanics, I looked at creating a final element to connect movement elements: the slow-motion dash. In gameplay, I saw the need for more sudden elements of connection for a slow-motion mechanic. This feature temporarily slows down time, allowing the player to smoothly move to a new position, akin to the teleport mechanic in Ghost Runner. This mechanic Is given to the player when they jump off Walls, Sliding or Grapple Points.





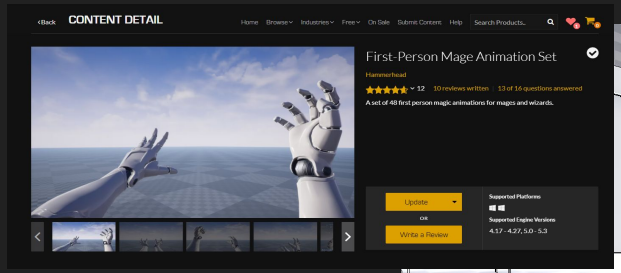


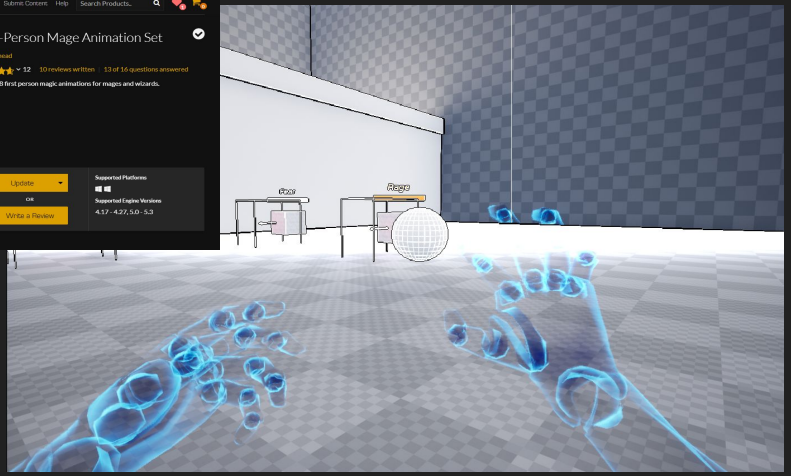


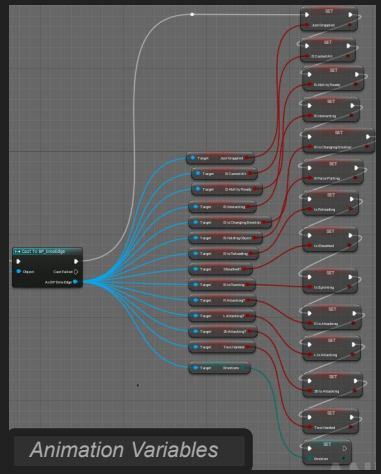
When beginning to polish the player character for the game I looked into implementing animation into the char.
This came from feedback saying it didn't feel immersive

Shared Booleans for Animation and Tech

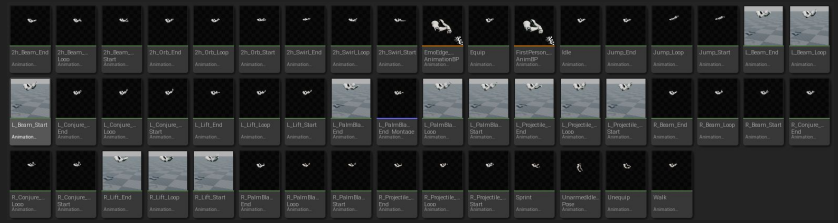
https://www.youtube.com/watch?v=daNOJIX1_8g

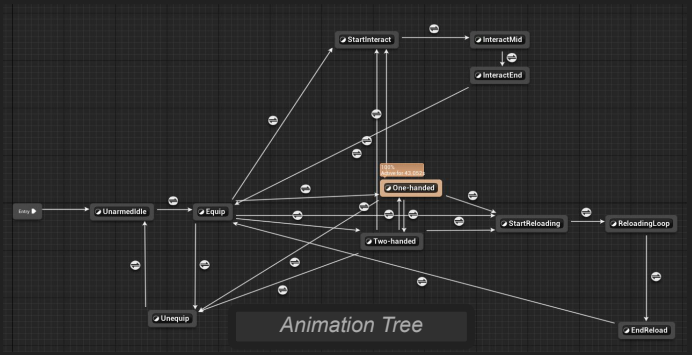




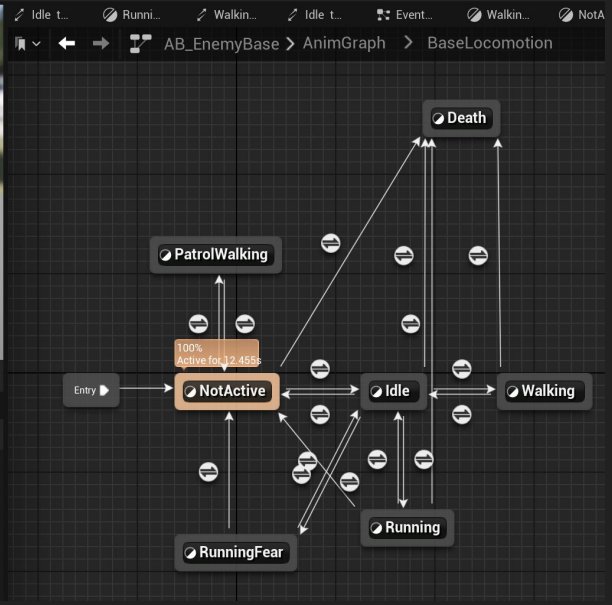


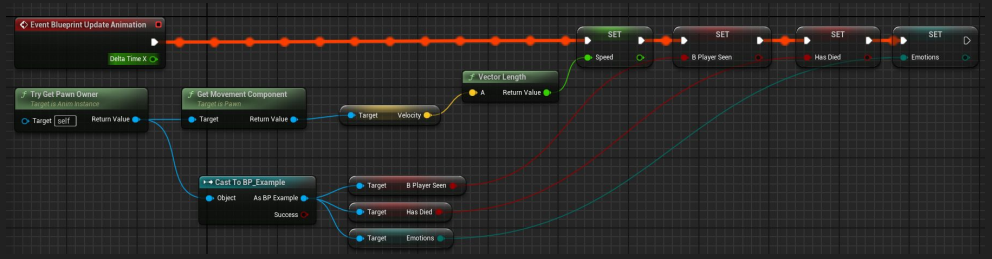
Animations were sourced for the player char, the animations were found on the Unreal Marketplace and were implemented,

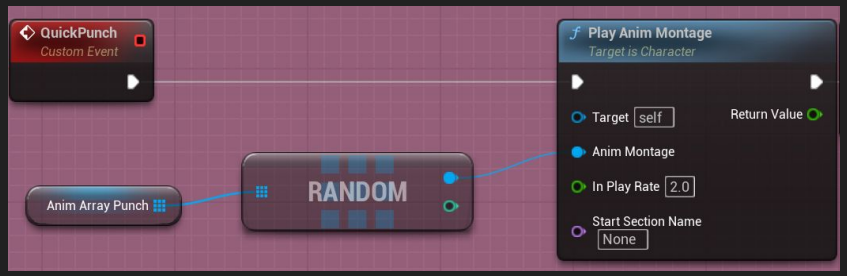




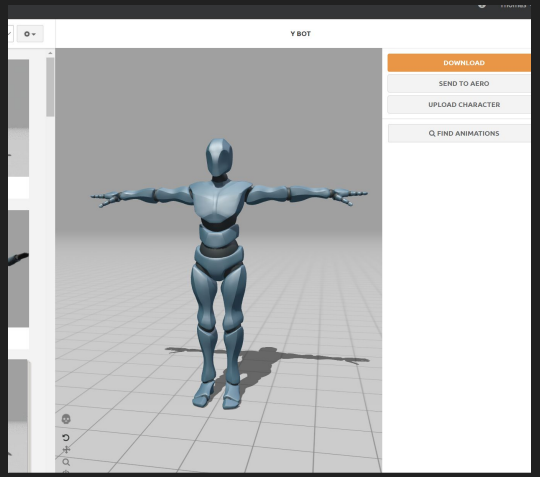
For this I looked into learning about animation blueprint and how to make them work with two hand systems in a range of different states

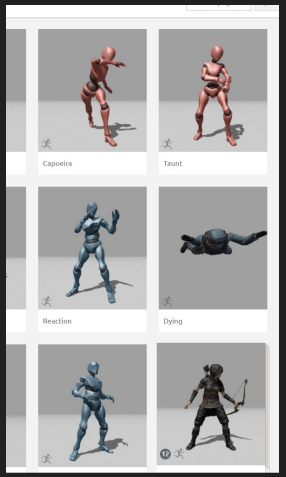






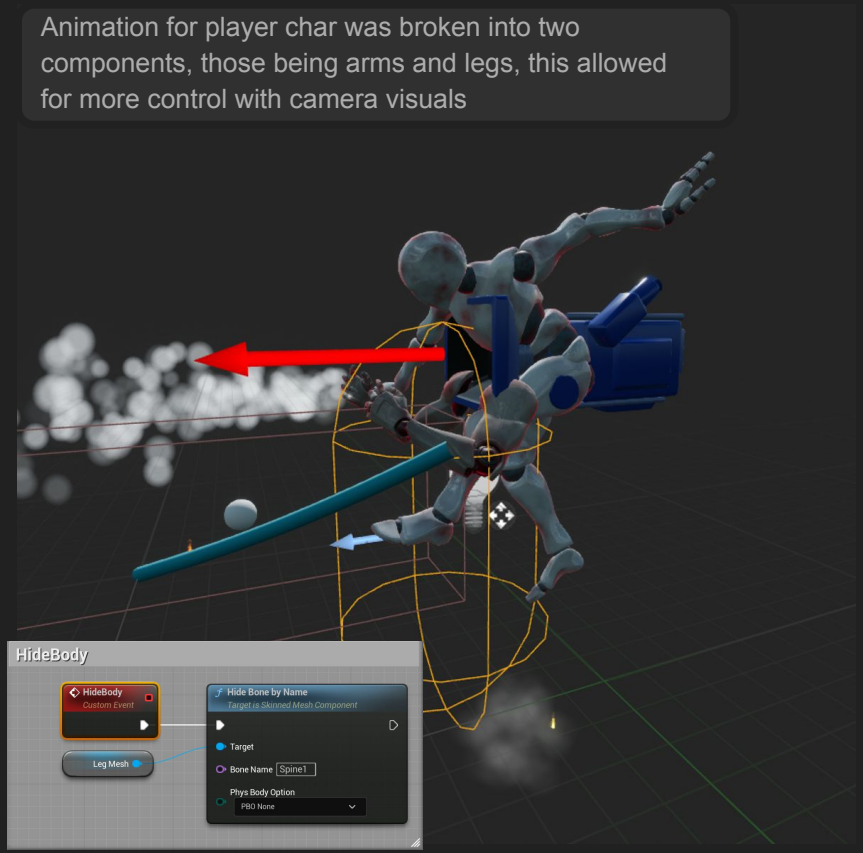
Anim Montages, being used for one time events EG Punching

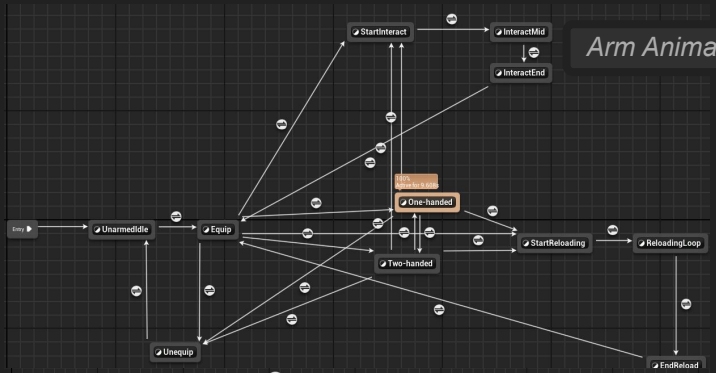




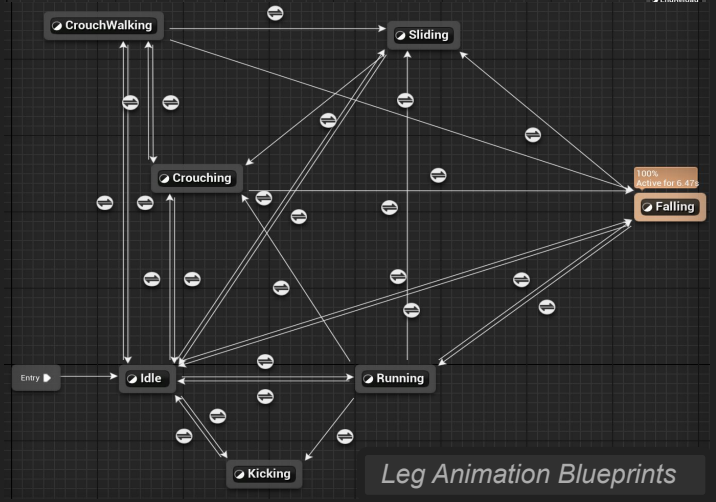
Mixamo Animations were used for a Pre Rigged Character

Animation for player char was broken into two components, those being arms and legs, this allowed for more control with camera visuals





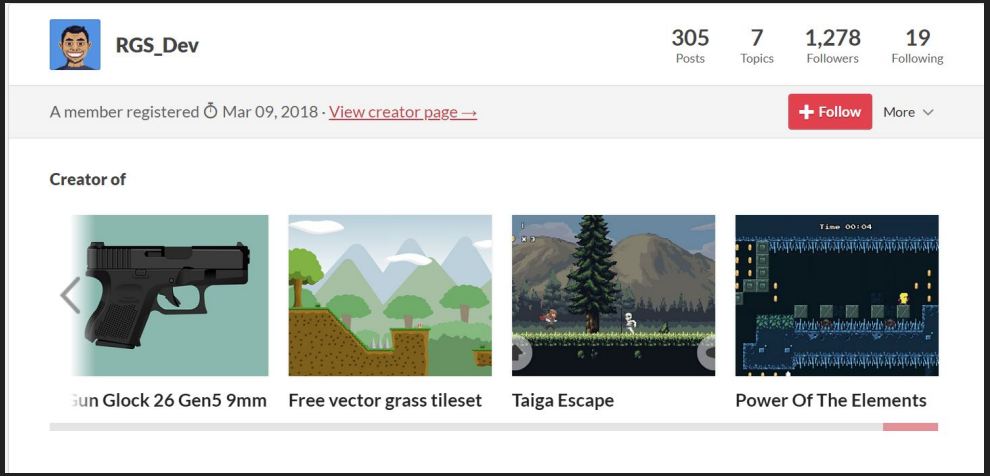
Arm Animation Blueprint

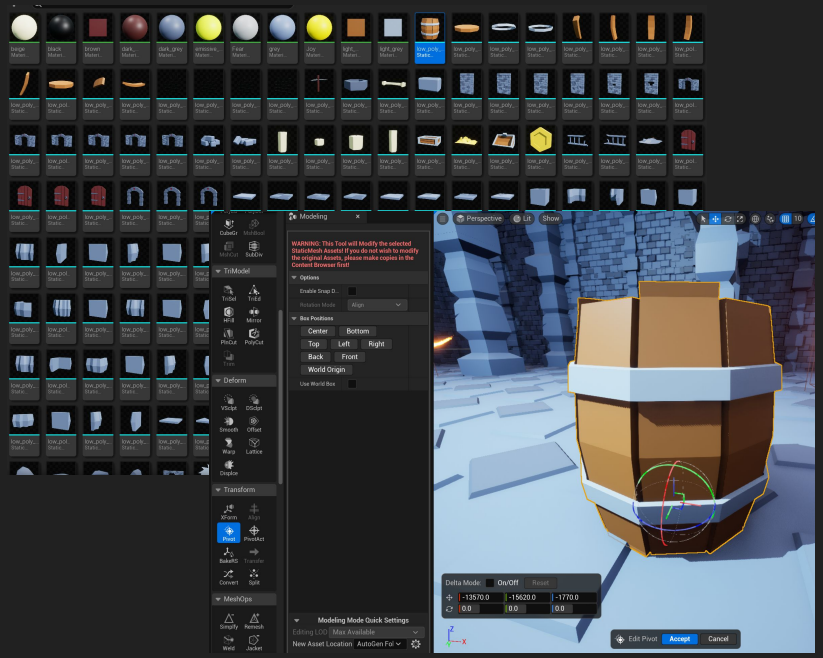


Leg Animation Blueprints

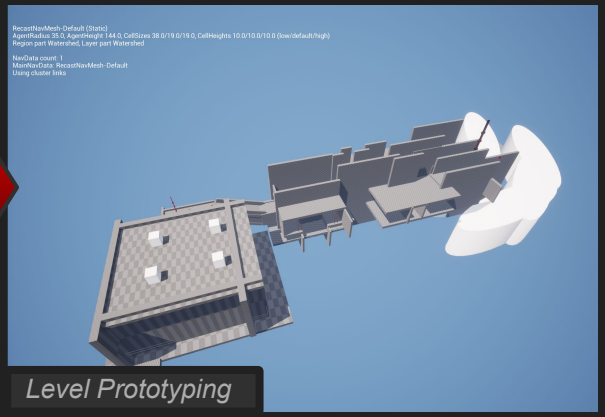
For the project I found an Tileable asset pack. While the assets were great the X-Forms Were messed up and were needed to be manually set.

https://itch.io/profile/rgsdev





Debug Testing World

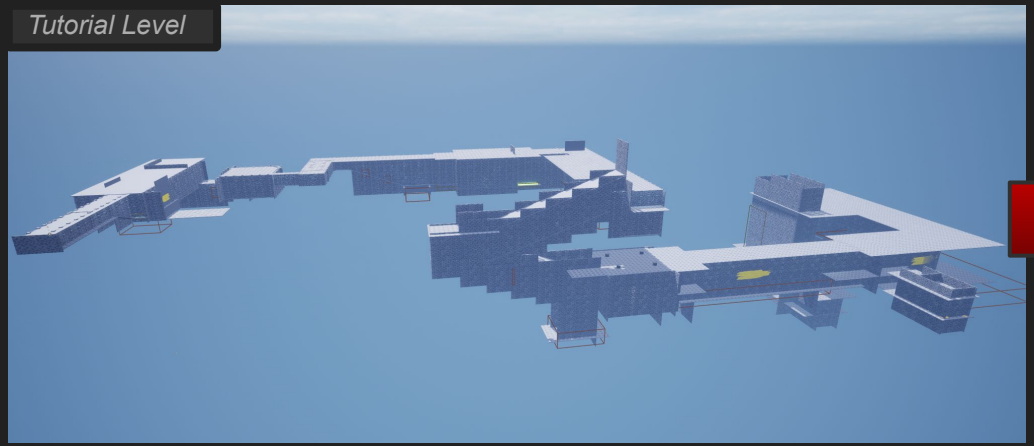


Level Prototyping

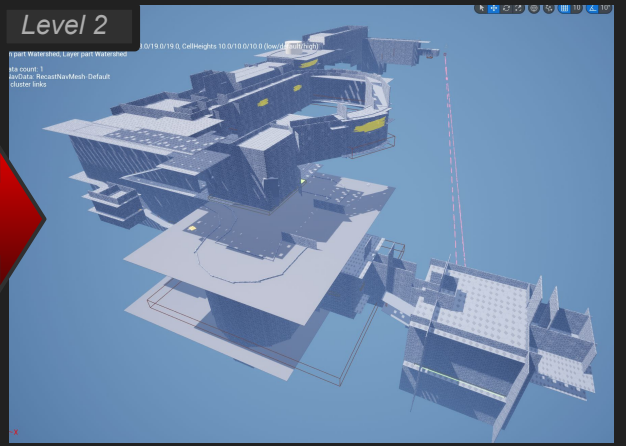
Different maps were made over the projects cycle

Initially these started as testing ground for the player mechanics. From here full levels were made. These were user tested and iterated on.

Tutorial Level

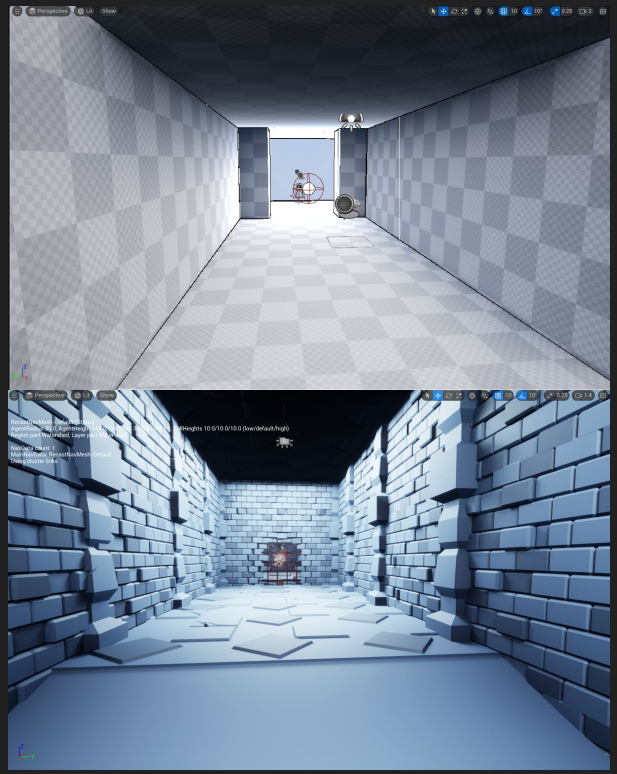


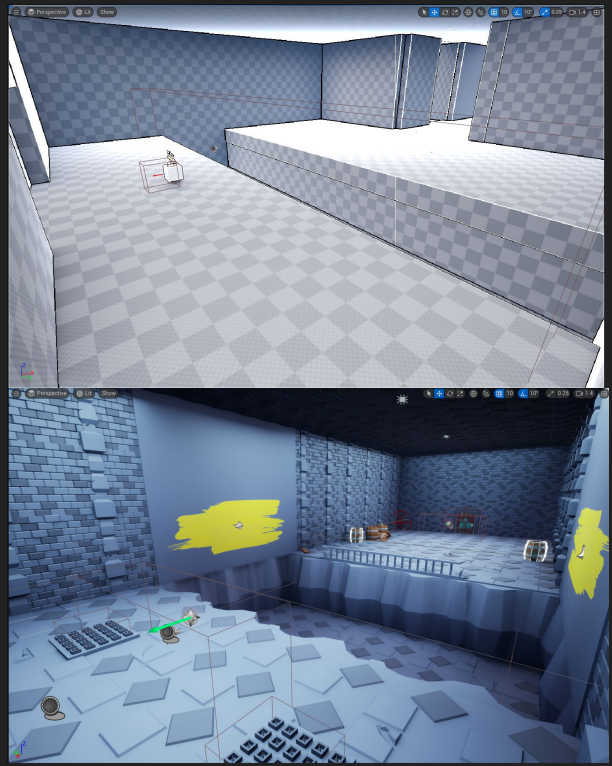
Level 2

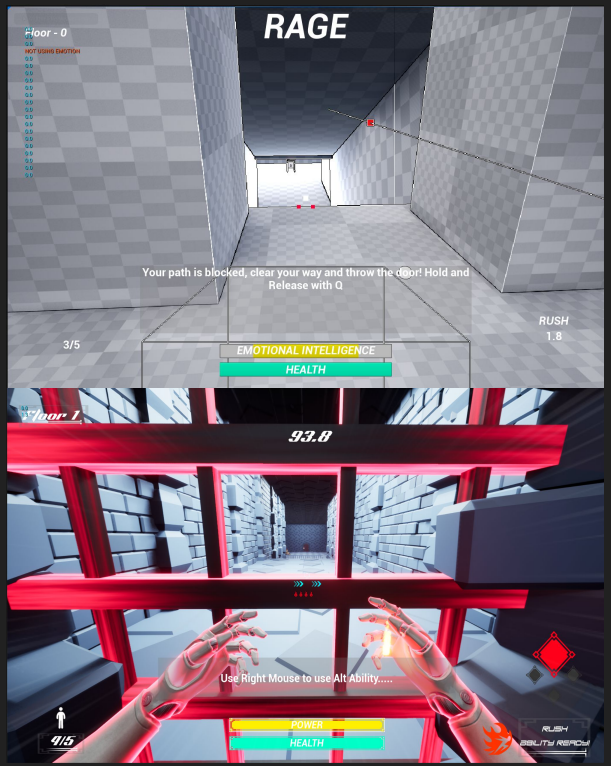


Level Design Iteration

Levels functionality was tested and made before full fledged levels were made, function before form

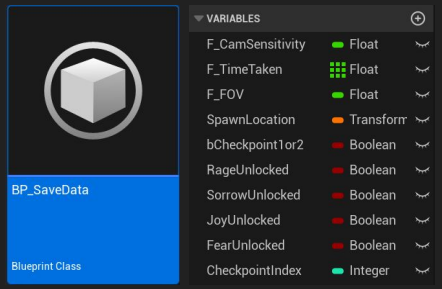


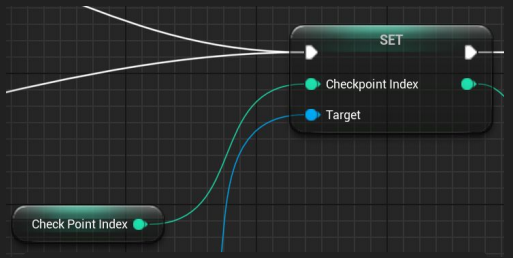




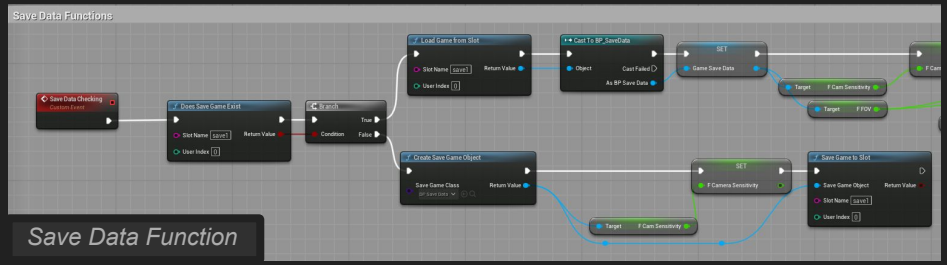
Level Blockout Vs Development

Development Of Save Data

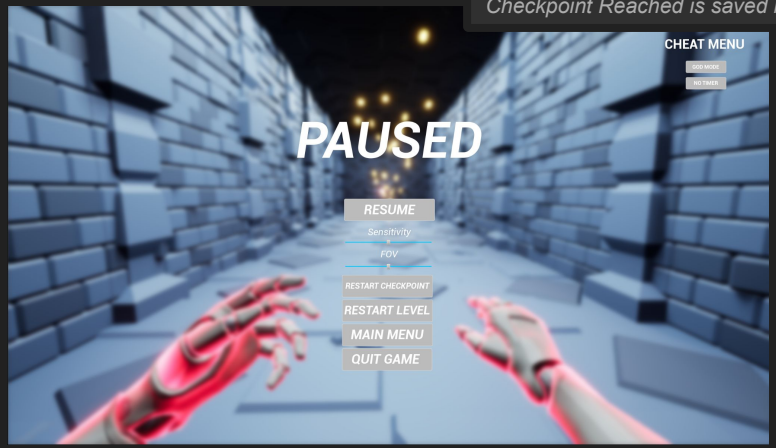


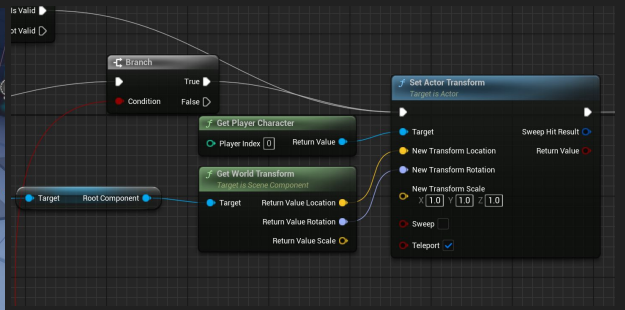


Checkpoint Reached is saved by the game



Save Data Function





When I made the change to making levels more continuous and longer I found there was a need to make save data as players did not like going all the way to the start of the map again.

New states found are saved by the game, these are given when the player restarts or resets.

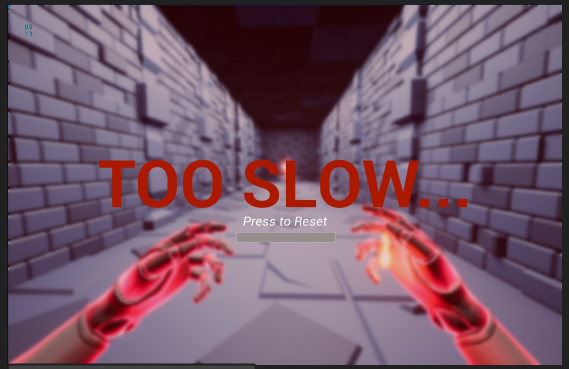
Should there be save data the game finds the checkpoint with the corresponding index and movies the player to it

Alongside this as the player changes FOV and Sensitivity this is put into save data to prevent the player having to input it again.

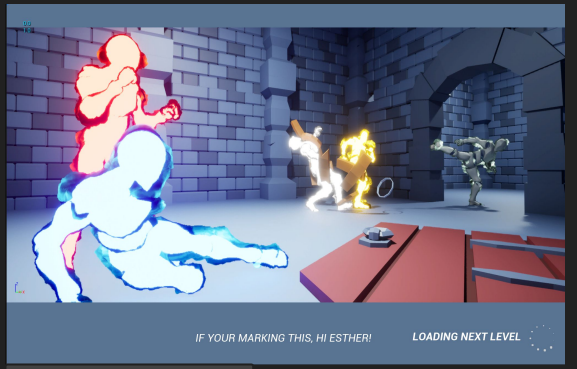
Main Menu

Finished Level

Death Screen



Time Run Out

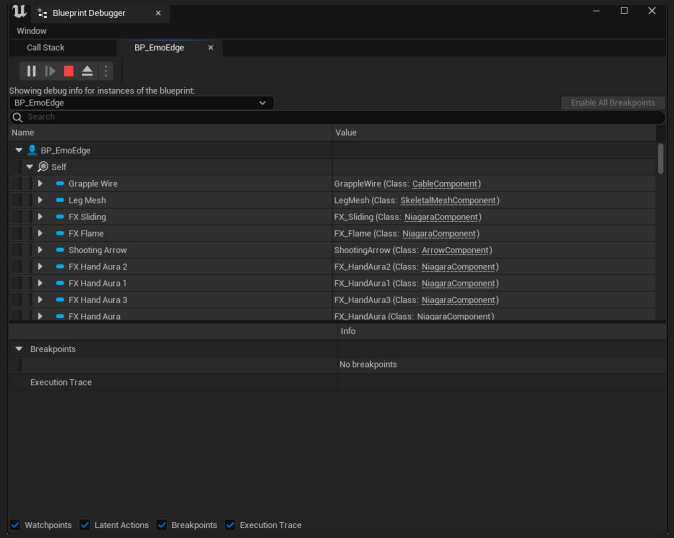


Loading Screen

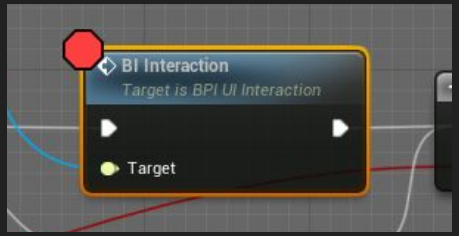
For the Tech Demo I looked into developing UI screen states to help break up gameplay to make the game more understandable to players. This included Death Screen, Loading Screens, Menus ETC.

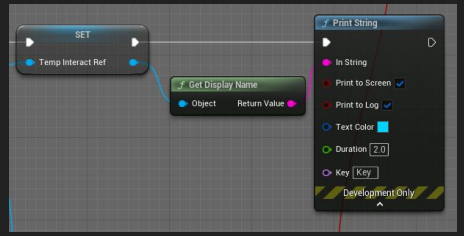
Doing this helped with polish and game feel.

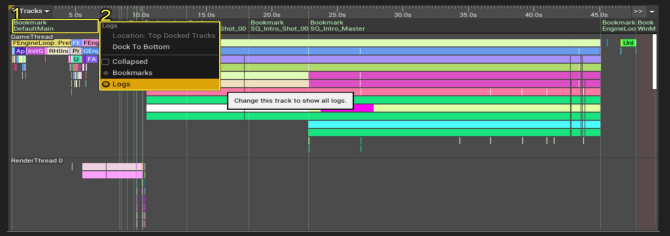
During the projects cycle I used debugging tools to ensure consistent frame rates and performance



Blueprint Debugger and GPU Visualiser for more complex Issues presented, during project cycle







Tmeere Today at 22:39
How did the project build run on your computer?

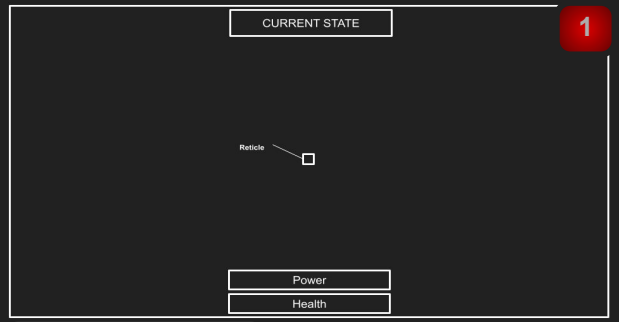
jOnah Today at 22:13
Ryzen 5 2500
GTX 1660 Super
16GB Ram
Gameplay felt smooth, certainly High-Octane if I do say so myself

Oneh Today at 22:37
i5
16GB RAM
RTX 3050
overall the game ran really well for me

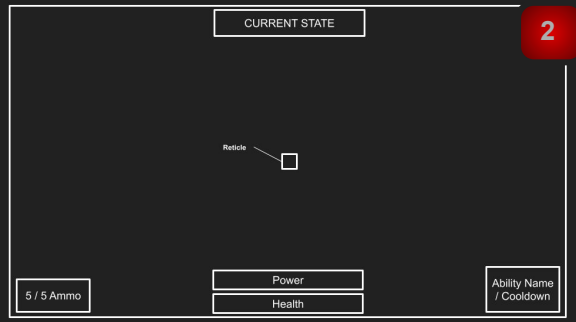
Testing was done on an range of different computers with different hardware to try and ensure consistent gameplay as much as possible.

While the game did run faster on some systems overall most devices could play the Tech Demo in built format

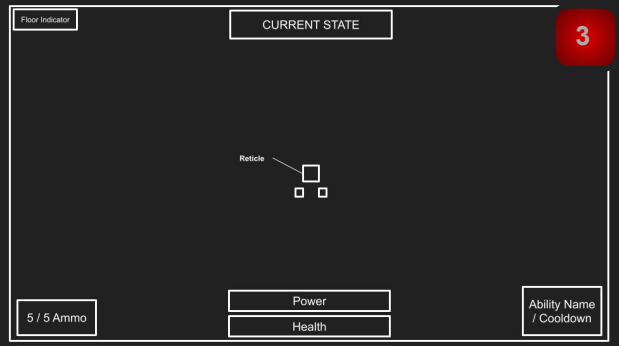
ByThePope Today at 22:55
Project build works fine on my PC, runs fairly smoothly at a comfortable frame rate. My specs are : 16GB RAM , AMD Ryzen 5 3600 6 core proccessor, GeForce GTX 1650



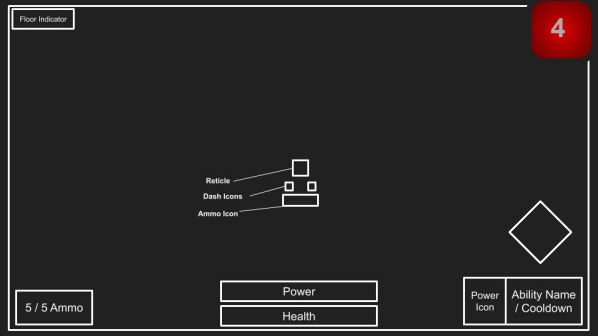
Initial Wireframe



More Panels with Info



Indications and Icons Added



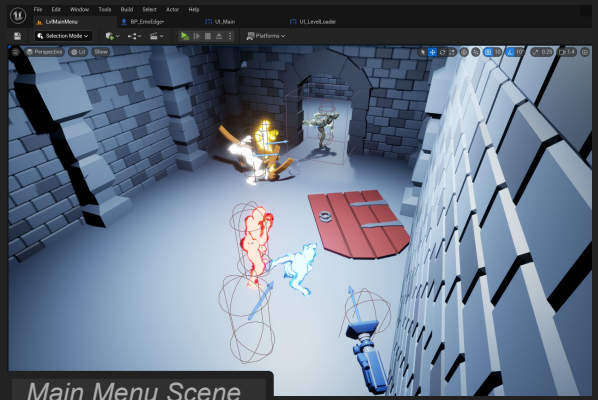
Removed Visual Clutter, Improved Clarity

Final Viewport Design

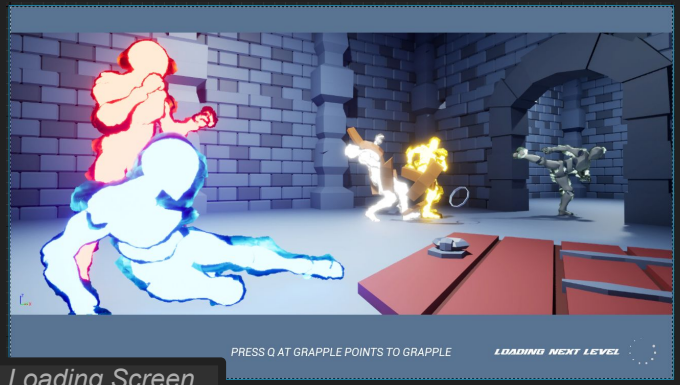
Viewports Design Changed Along the project, this went through various states and clarity was improved

Main Menu Viewports

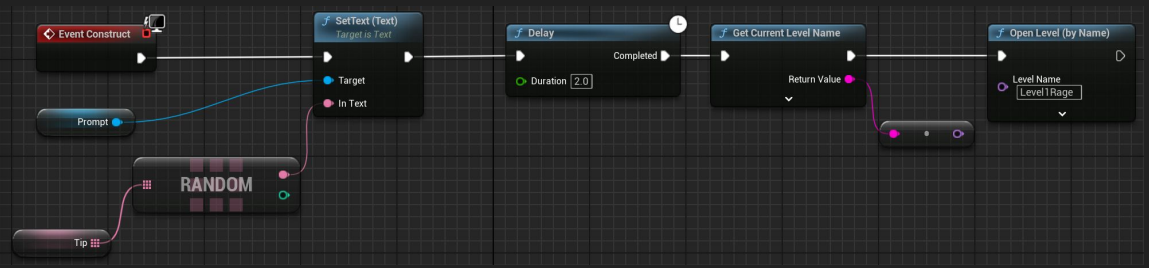
During development I looked into making Loading Screens alongside a Main menu, this helped to break up gameplay as well as making it feel more polished, alongside this leaderboard data is added on the main menu so players can compare times of the fastest runs done.



Main Menu Scene



Loading Screen



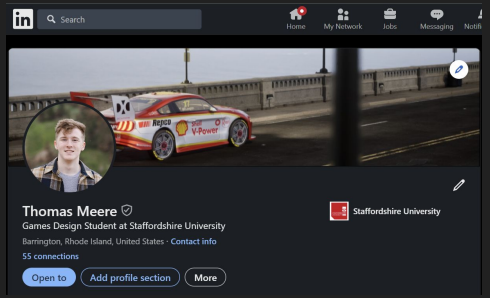
During Loading screen the player is given prompts and tips, I added this to give additional common info the player could miss

Overall I am very happy with the final results of the project. Overall my goals were to develop my understanding of unreal engine and learn about blueprint more. This came in the form of a Complex Player Character alongside Dynamic AI and mechanics. While this project was a great success there are many factors Id look on changing or improving.

Going on from these developments I am going to look into taking the AI even further for GradEx, With a now strong understanding of Finite State Machines and AI I would like to now go into the domain of Behaviour Trees as well as looking into programming them in C++.

In addition to learning AI, I would look at getting new asset packs and visuals to enhance the game's development. Although the primary objective of this project is to create a Vertical Slice Tech Demo, I am keen on prioritizing visual elements to give the project a distinctive and appealing style.

I'm proud to have achieved the goals outlined in my initial MoSCoW briefs. Although these goals have been accomplished, I recognize like with any project there's is always more to be done and to pursue / incorporate into the project to further demonstrate its value to the industry.These will be reflected before GradEx



After completing this project, I am currently preparing to present it on LinkedIn to showcase the work I have accomplished.

As additional material. Here below are links to my Tech Breakdown Video, as well as my gameplay video.

Tech Breakdown Video

https://youtu.be/7oGpFN4gbcg

Gameplay Video

https://youtu.be/idcMbMSL6Rk