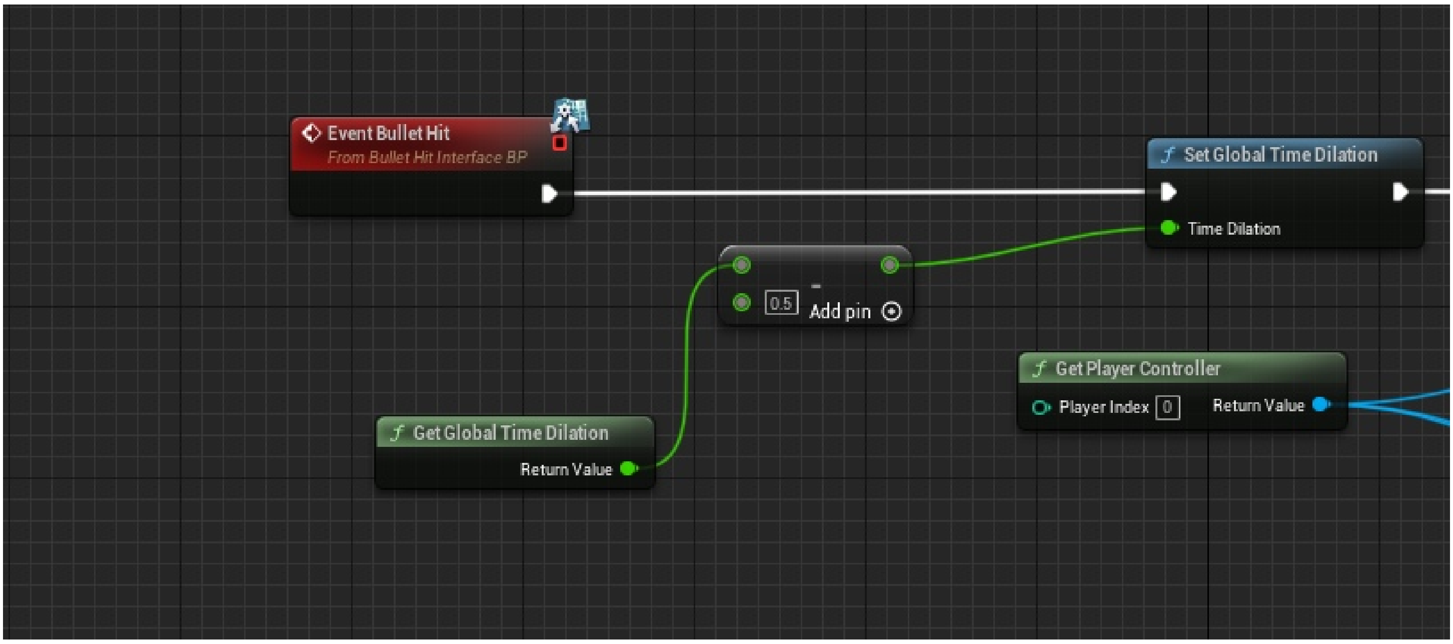
Time Dilation

Custom V Global

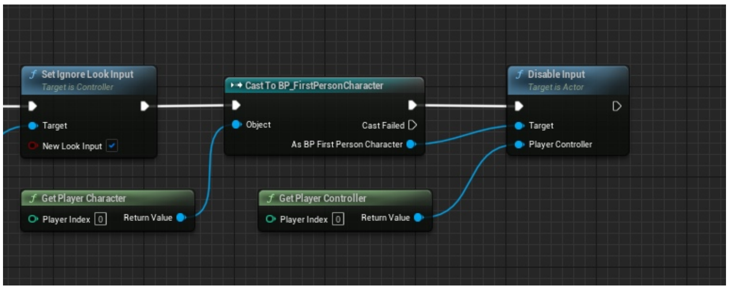
Walking in the Wild West's, Sniper Elite inspired, bullet time shooting mechanic requires time to slow down when the player fires a bullet. When a bullet is shot the camera will follow it in slow motion. To slow down time in the game their are 2 nodes that are available: custom time dilation and global time dilation. Using these Youtube videos (https://www.youtube.com/watch?v=VeA2K1coDVw) https://www.youtube.com/watch?v=y2DtVnOugVY&ab_channel=MathewWadstein) I decided that the global time dilation node is the superior option in this instance. This is because using custom dilation applies the dilation only to the singular blueprint in which its located; global applies the dilation to (as the name suggests) the entire level, which is more appropriate for my required use case. Using event dispatchers to call on this event when the gun hits a target is most optimal as it does not to be running at all times.

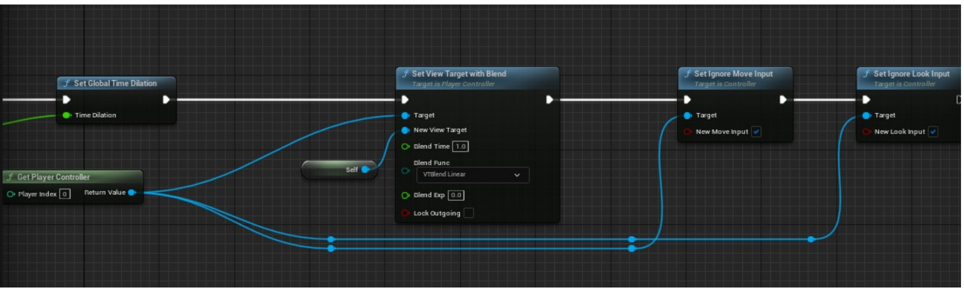


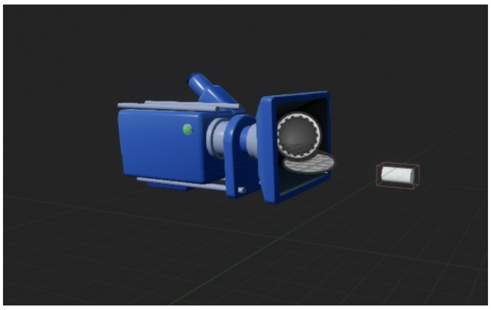
Camera Blend

Bullet POV

The previously mentioned bullet time mechanic has the camera follow the bullet to its target. Using the set view node allows me to change the players perspective to follow the bullet. Whilst following the bullet player inputs need to be disabled; the two nodes following set view target with blend achieve this, is what I believed. Upon testing they did not disable action mappings outside of just move and look. To resolve this I added the disable input node.







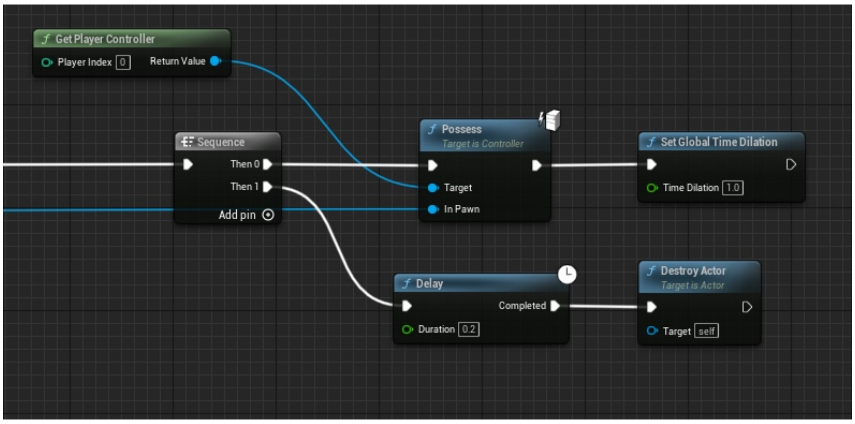
It is not possible to export **2023-01-08 21-34-17_Trim.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\2023-01-08 21-34-17_trim.mp4**

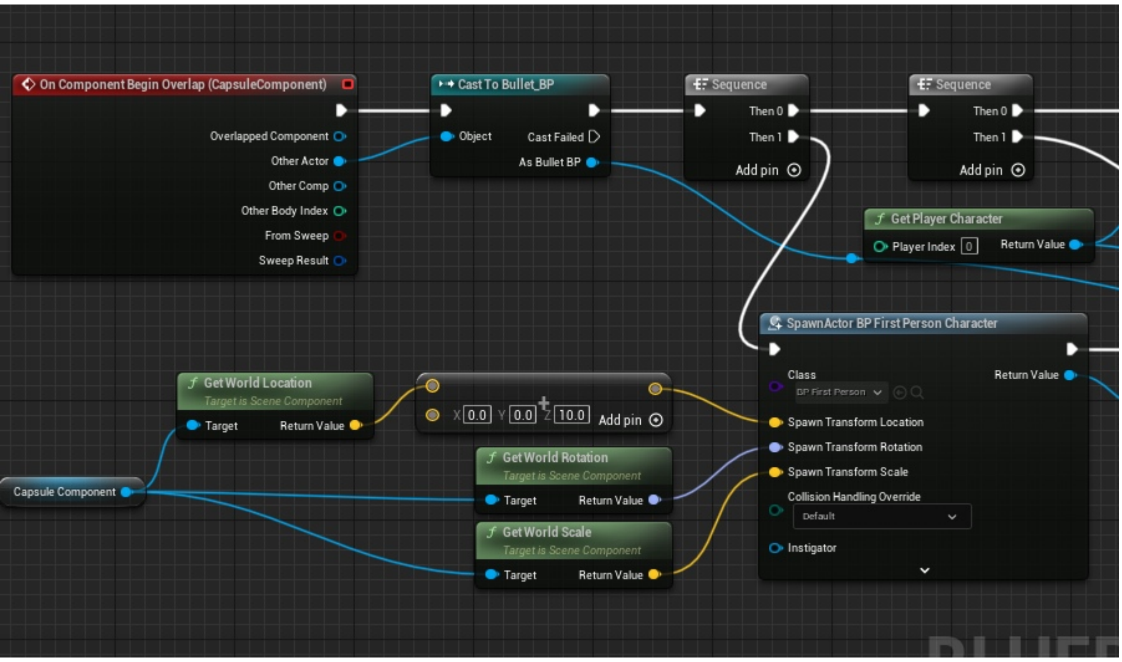
"Possessing" Enemies

Respawning the player character

It is not possible to export **2023-01-08 22-30-10_Trim.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\2023-01-08 22-30-10_trim.mp4**

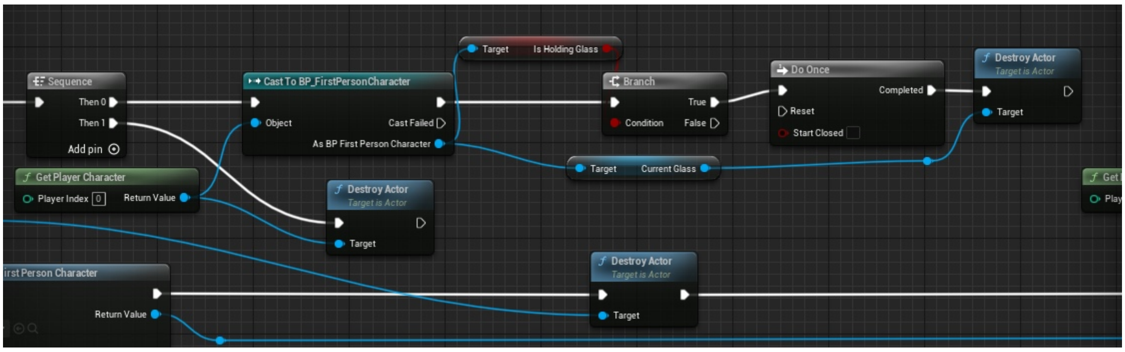
One of the primary mechanics in my game is possessing the enemy when they die. Unreal has the capabilities to actually possess pawn and take control of them, however, this would cause many issues relating to other systems. To make the game more efficient and easy to work on I decided to simply respawn the first person character at the place of the enemies death, possessing the first person character rather than the enemy.





The bullet sets of a chain of sequences that destroy all the actors involved that need to be destroyed. The sequences are used because I tried without them and it would cause issues with access none errors, due to the actors being destroyed before the code ran. The addition of 10 to the Z axis when respawning the first person character was to prevent the player from getting stuck in the floor.

The branch checking whether to destroy the "current glass" simply checks if the player is current holding a glass, otherwise the glass they were holding would float in the world.

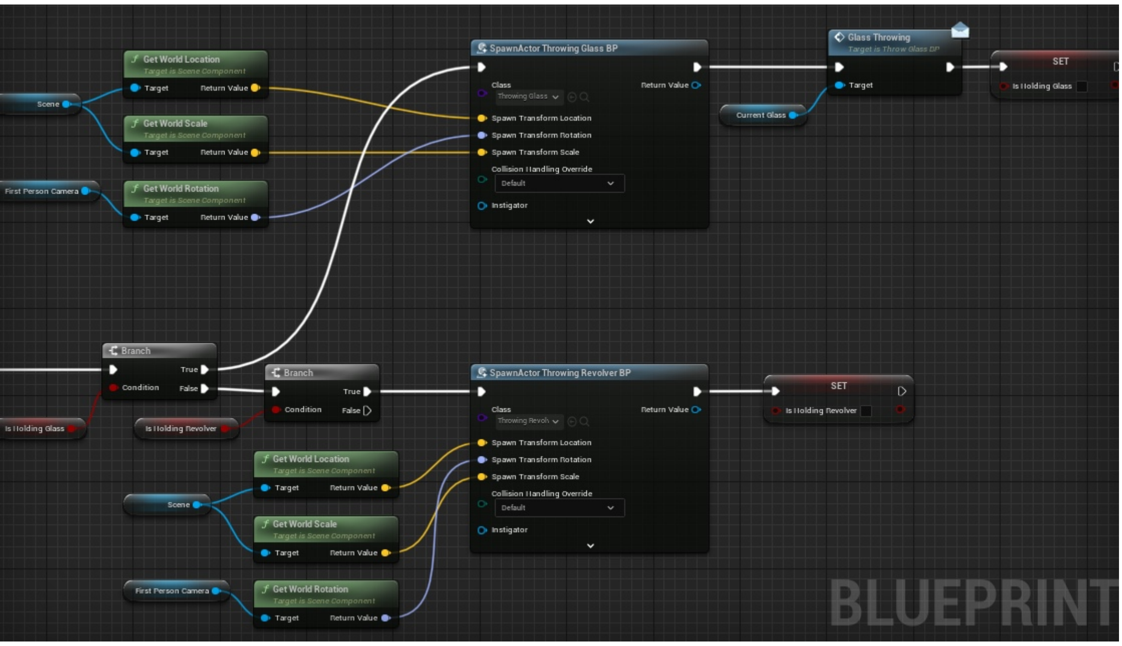


Glass/Revolver Throwing

Projectile and Rotating Movement

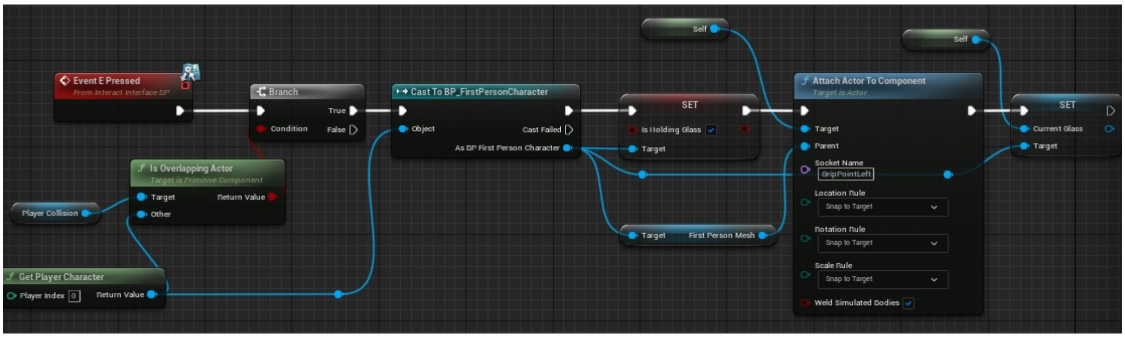
Weapons and glasses are able to thrown at enemies to disarm them (note I have to implemented the disarming aspect yet). In my first attempt, in order for the glass to remain stationary at event begin play, the projectile and rotating movement had "auto activate" set to false. I then used attach actor when in range of collision, and interacted with, to allow the player to hold the glass. The plan was to then activate movement and detach the glass from the player to throw it. However, for reasons I still do not understand, when detaching the glass it would not move or rotate at all (even with no collision). For this reason I decided to create a separate blueprint for the thrown glass with movement and rotate on auto (much like the bullet) then simply spawn the throwing glass and destroy the glass in hand.

It is not possible to export **2023-01-09 02-10-00_Trim.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\2023-01-09 02-10-00_trim.mp4**

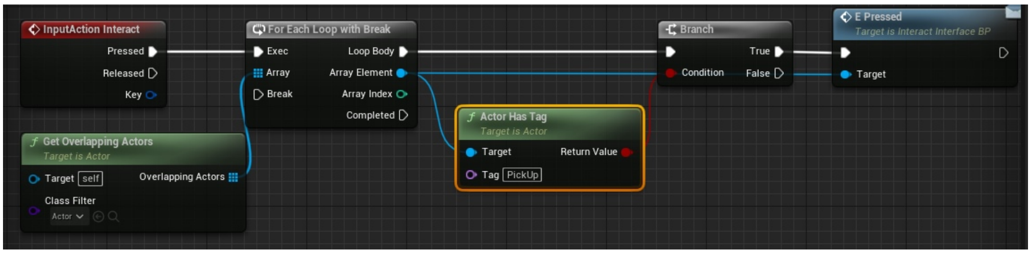


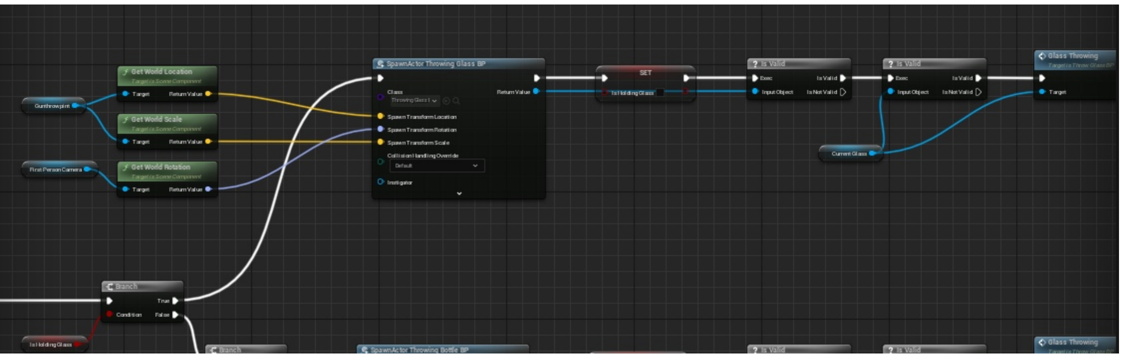
The system that decides whether the player should throw a glass or revolver is shown on the left. The blueprint interface that gets called is to delete the glass in the players hand.

The "current glass" variable is stored in the first person character and is used to make sure the glass in the players hand in the only one that is destroyed.



Once I added another throwable object, I ran a boolean that checked what the player was holding. This is set in each of the individual throwable blueprints and the system also features a way of checking if the player is holding any other items, if true then they will NOT pick up anything else.



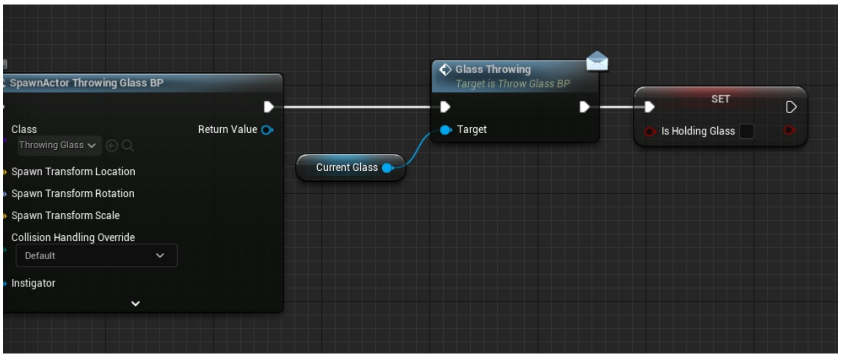


When input action "throw" occurs, after the line trace, a series of branches checks what item to spawn and check if they are valid to avoid access none errors.

Interfaces

Vs Event Dispatchers

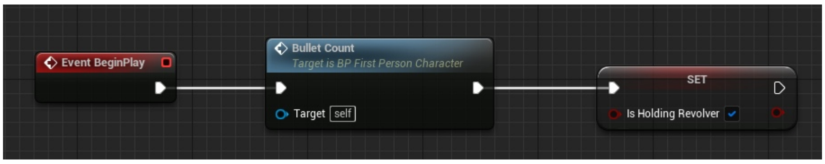
In the majority of cases in my project I have opted to use blueprint interfaces as the primary communication method between blueprints. This is because on more than 1 occasion event dispatchers unfortunately not function correctly. One such example is when I was trying to communicate that the glass had been thrown, to the glass. When using an event dispatcher, binding an event in the glass blueprint, this line of code would run 3 times for unknown reasons. Once I switched to a blueprint interface it worked without any issues.

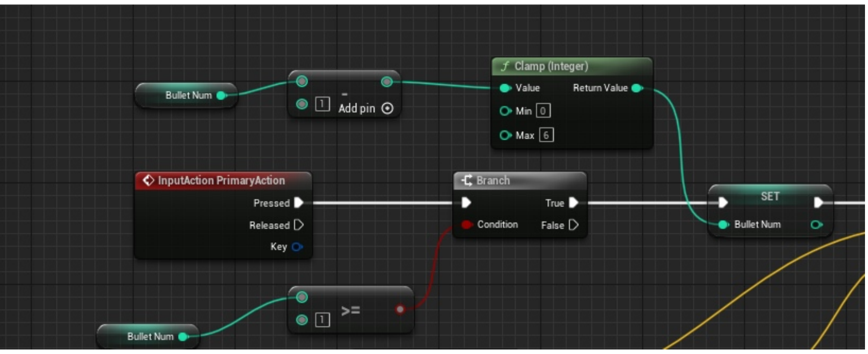


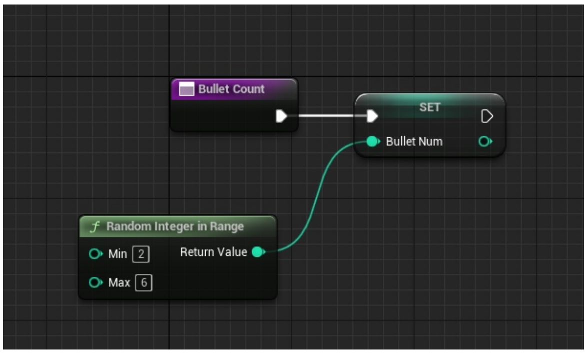
Bullet Count

Random counter

Most standard action revolvers hold 6 bullets in the magazine. This is why I choose 6 to be the maximum number of bullets the player can have. Whilst in a shootout it would be highly unlikely that every gun you picked up would have the same amount of bullets in; for this reason I decided to create a random number generator, this also keeps players from running through the game too easily with 6 bullets every kill.



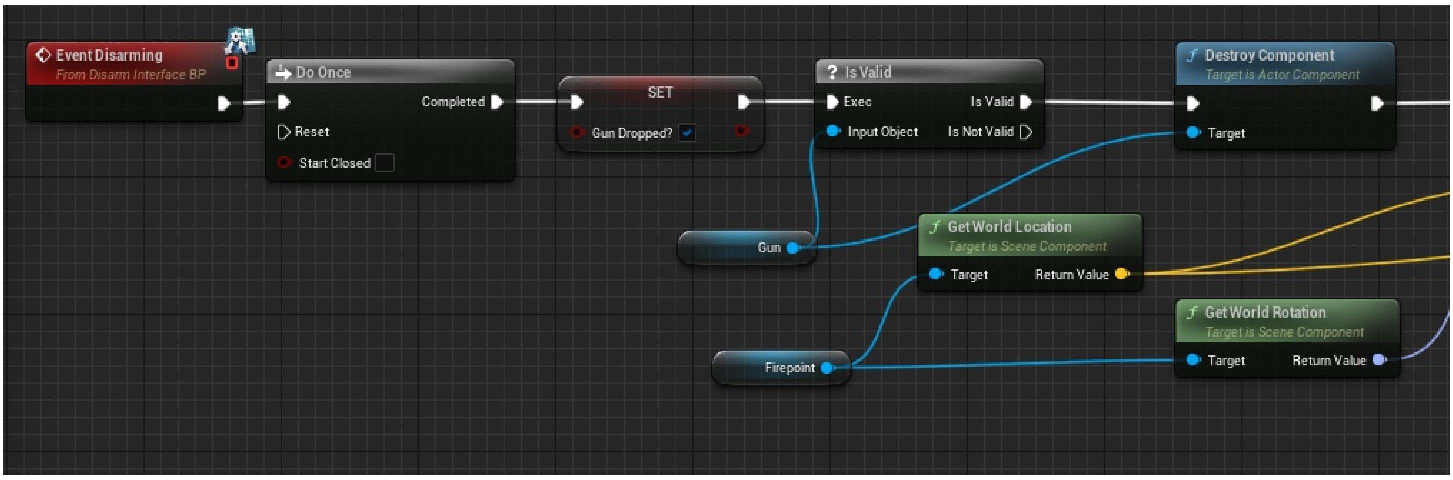




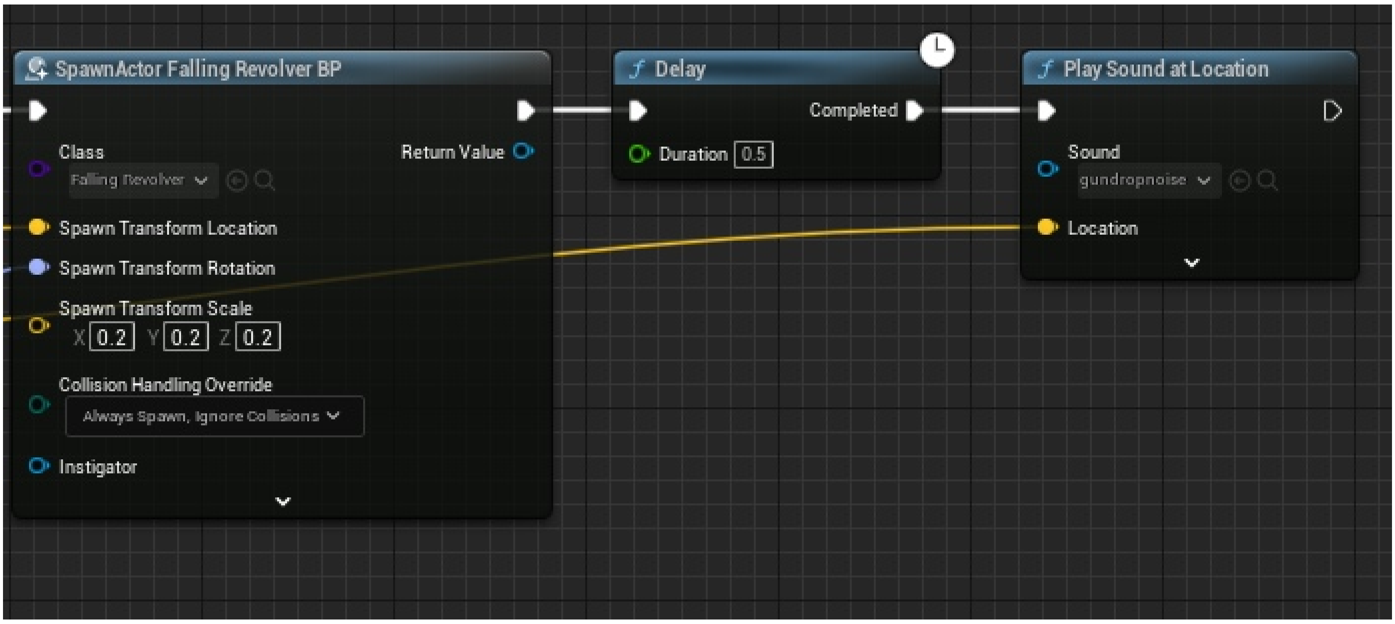
Disarming

Interfaces, sound

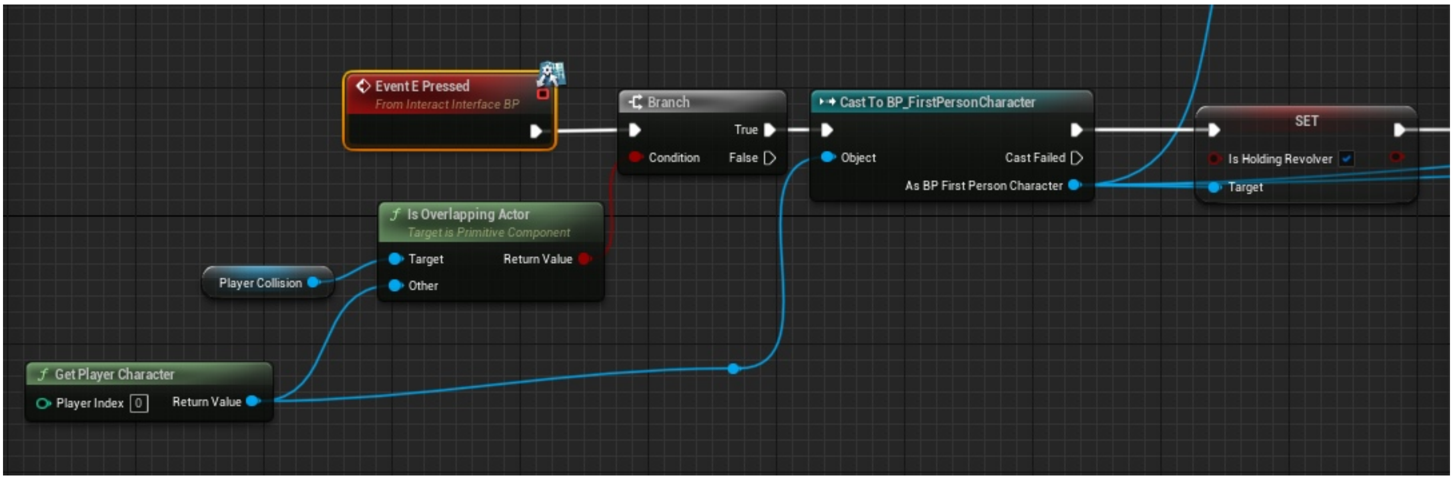
It is not possible to export **gundrop.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\gundrop.mp4**

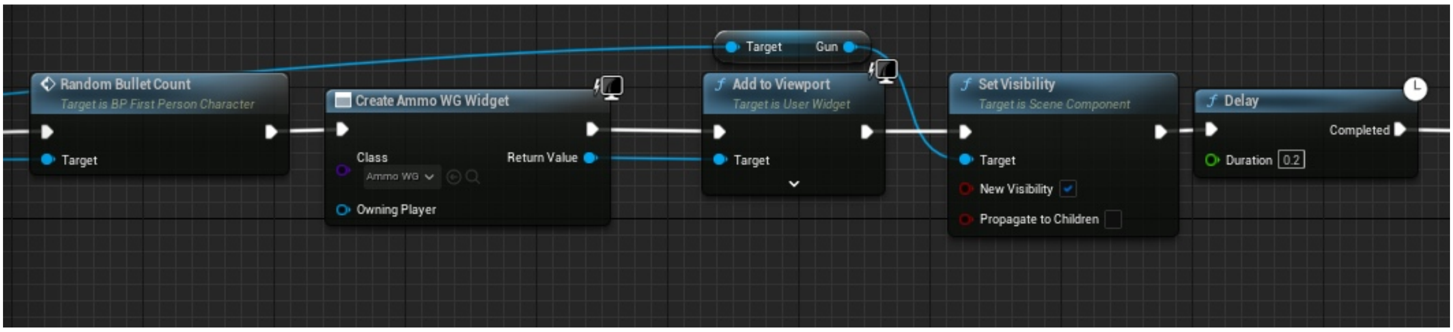


The interface for the disarming is ran from any throwable and is so that enemies drop their weapon when hit, by a throwable. This provides the main chunk of the ammo economy (along with possessing enemies) as you can pick up a revolver to gain more ammo. The disarming event sets the gun dropped variable to true and destroys the gun that the enemy is holding. It then spawns a new blueprint, falling revolver. It then plays a sound effect after a small delay so that it gives the revolver a chance to hit the ground.



The falling revolver bp is the gun model but with simulate physics turned on and a collision box around it to interact with. When the E pressed interface is ran (from the first person character) it checks if the first person character is overlapping it and if true then it sets the holding revolver variable to true. It also runs the random bullet count function, to give the player ammo, and then creates the ammo count widget. It then sets the visibility of the gun on the first person character to visible.

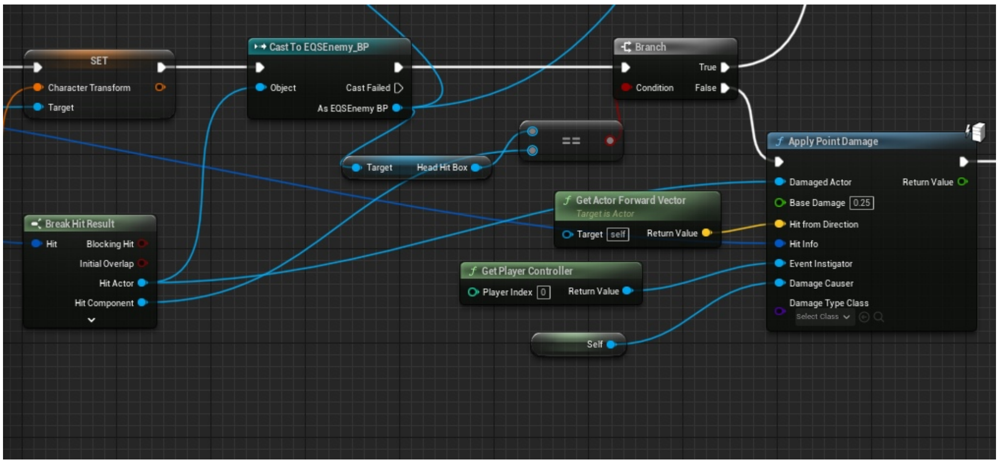


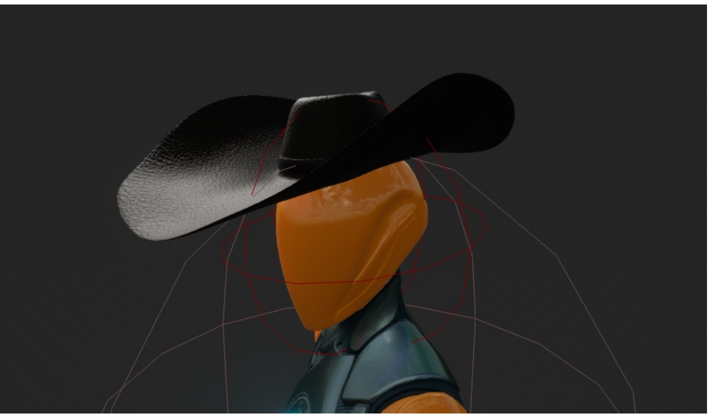


Changes to Headshot Bullet Cam

Custom Hitbox and apply damage

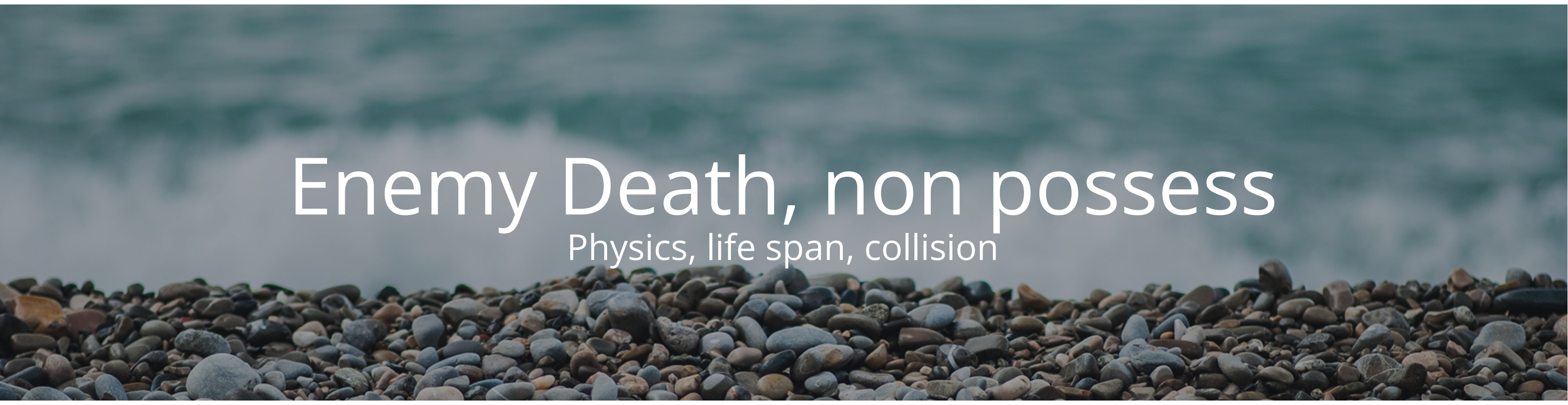
After play testing numerous times, the felt too easy and the bullet cam mechanic became boring. This was because it happened every single kill and allowed the player to be invulnerable whilst killing enemies. To fix this I made a custom head hitbox, via a sphere collision box, that meant the bullet cam event would only run when the player hit the enemy in the head. If the player hit the enemy elsewhere it would simply apply point damage and spawn a sound effect a location.





The headhit box is a sphere and quite forgiving because it is quite hard to aim in the game and with enemies running around a lot it is beneficial to allow for some forgiveness. I tested whether a more precise hitbox, using a capsule instead, would be better but I felt it was too difficult for most people to hit. Aiming for the head is very advantageous to players because it refreshes ammo and lives when they possess an enemy, helping them stay alive far longer.

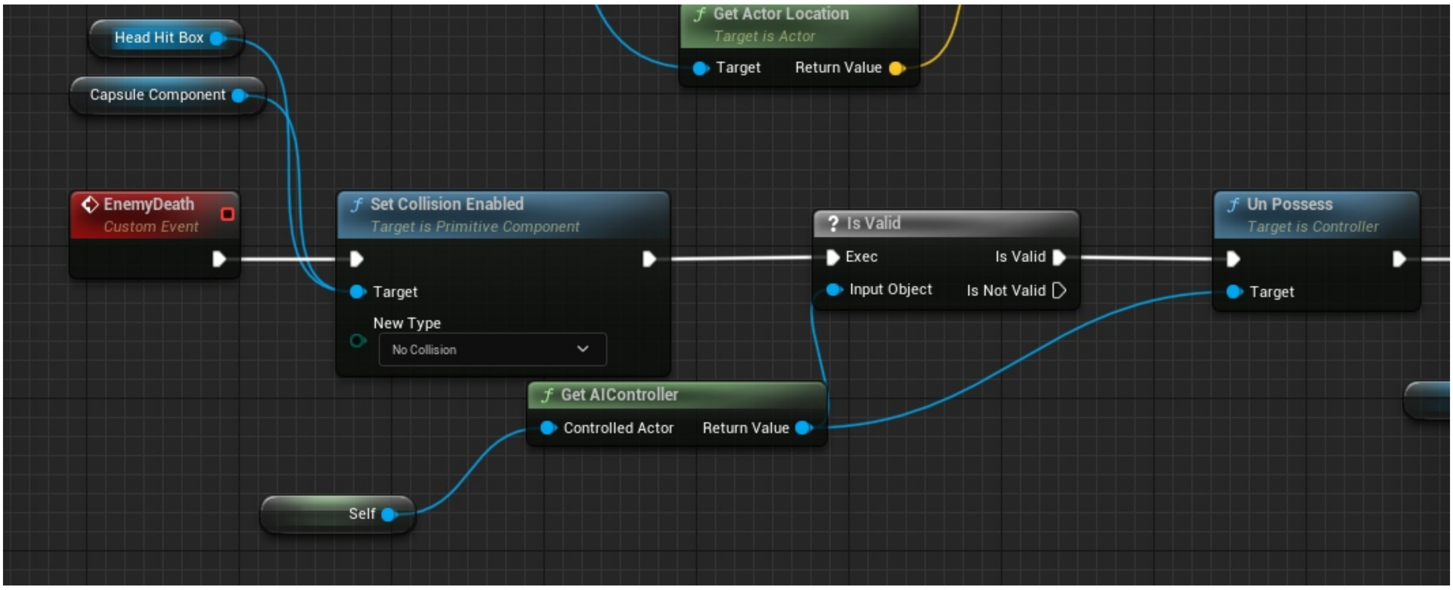
It is not possible to export **headshot.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\headshot.mp4**



Enemy Death, non possess

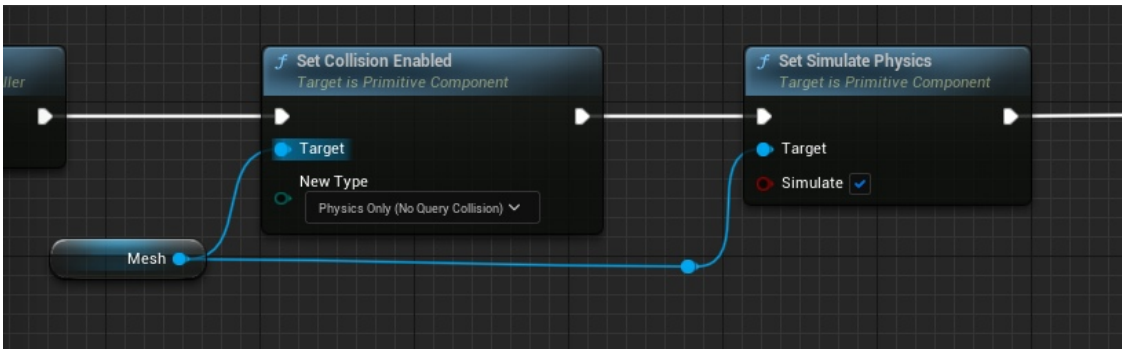
Physics, life span, collision

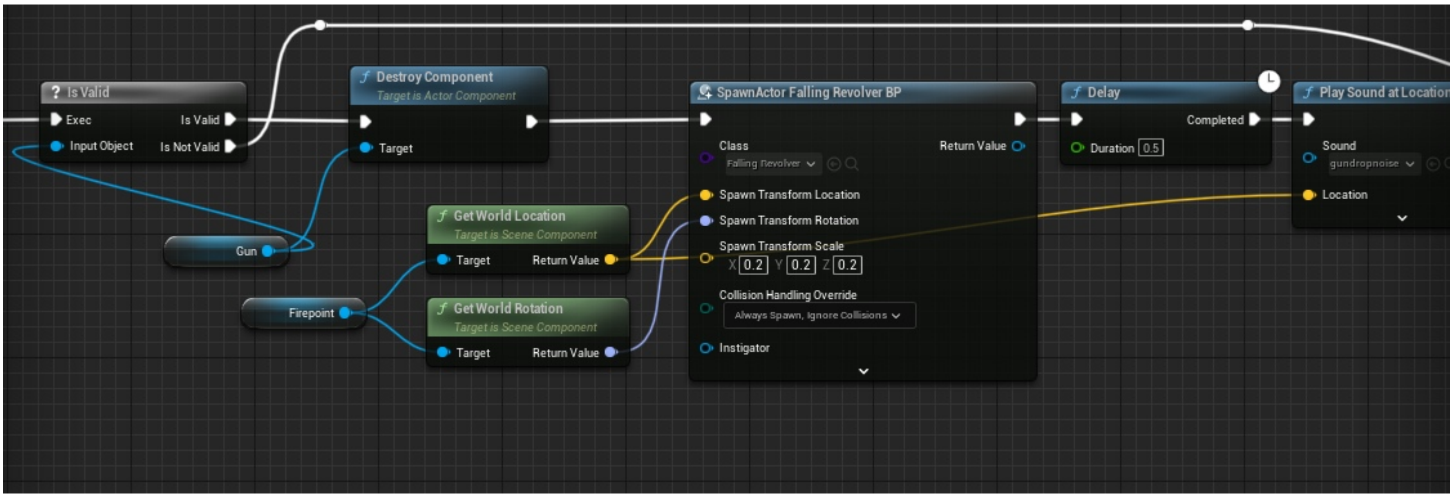
It is not possible to export **enemydeathvid.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\enemydeathvid.mp4**



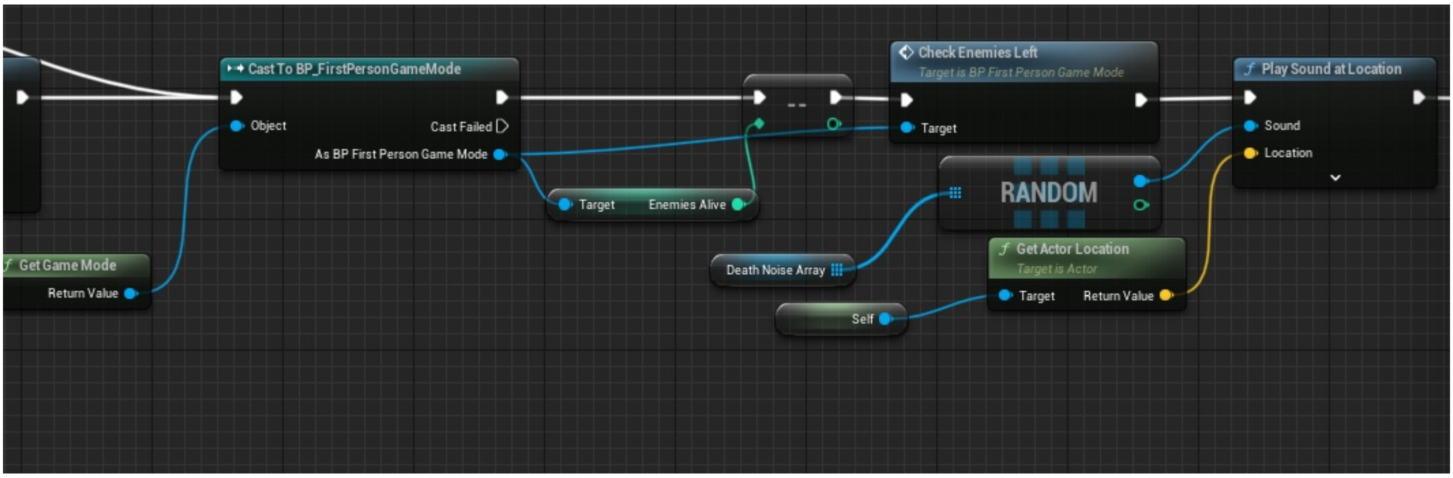
When the enemy dies to any damage, that isnt a headshot with bullet cam, the collision for the head hit box and capsule component are disabled, so that they dont interrupt player movement and for the headshot mechanic to be disabled. They are then unpossessed so that their AI behaviour no longer runs.

Collision is then set to physics only on the mesh, this means it will not interact with any overlap events or traces. Simulate physics is also turned on, to allow for ragdoll physics.


ller
Set Collision Enabled
Target is Primitive Component
Target
New Type
Physics Only (No Query Collision)
Mesh
Set Simulate Physics
Target is Primitive Component
Target
Simulate ☑



The gun in the enemies hand is destroyed because it then spawns a falling revolver bp (with the sound effect at location too), allowing the player to pick up their gun.

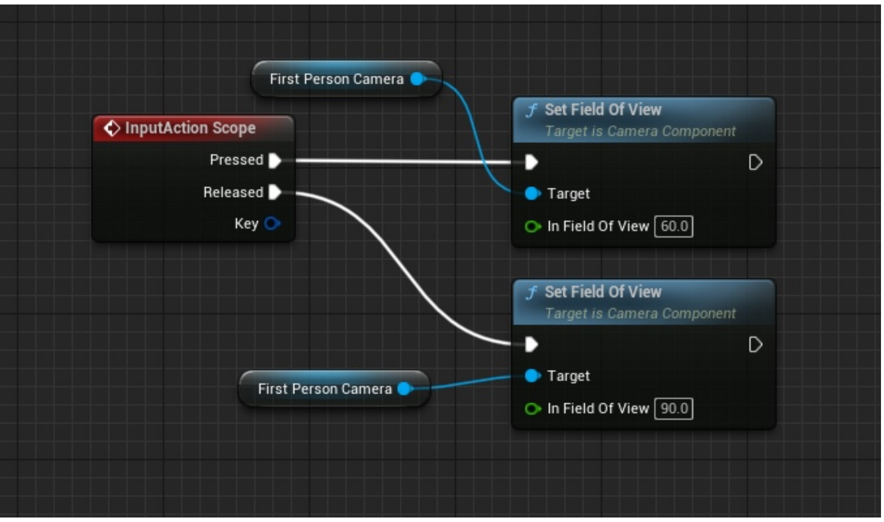


Finally, the enemies left function and decrement int is ran to make sure the enemy count is correct, for the wave spawn and enemy counter ui. A random death sound is played at location and then a life span is set. Life spans are useful to use over destroy actor because they allow for a short period where the actor is still in the level before being destroyed. This means you can do this such as ragdoll, or physics simulations, making it appear more realistic, before ultimately despawning the bp.

Scoping In

FOV

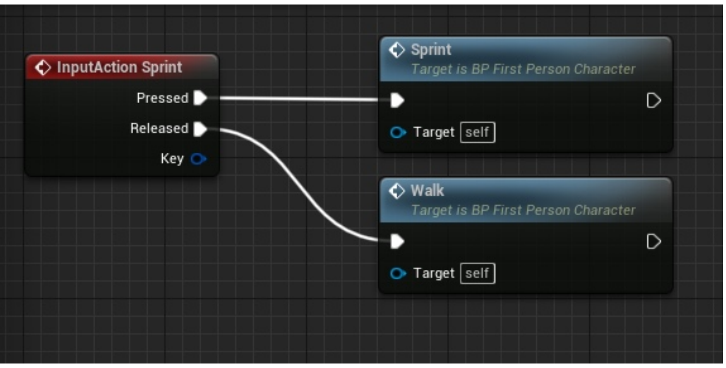
It is not possible to export **scoped.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\scoped.mp4**



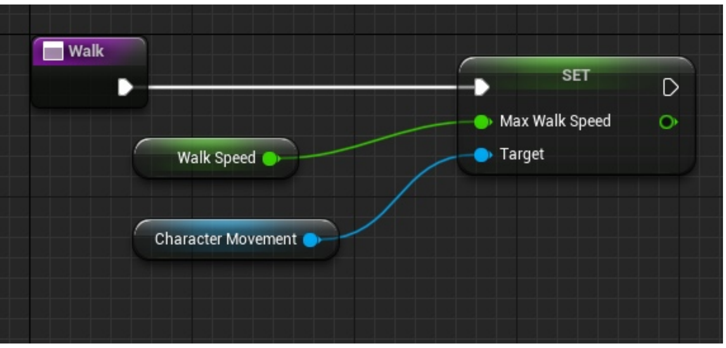
A basic scoping mechanic was added because of the aiming concerns. The simple change of the FOV allows the player to aim more precisely. If the revolver had a scope ontop of it, I would add a widget that had a scope crosshair that would be added and removed from the viewport based on the press and release of the scope button. Since their is no scope, the simply change in FOV works a charm.

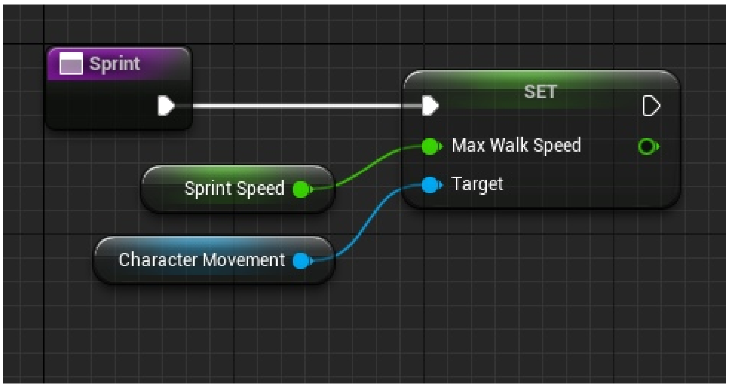
Sprinting

Move Speed


InputAction Sprint
Pressed
Released
Key
Sprint
Target is BP First Person Character
Target self
Walk
Target is BP First Person Character
Target self

The ability to sprint is standard in almost every FPS and is something that players often expect, this is why I added a sprint mechanic. The press and release run functions that set the max walk respectively. This allows the player to avoid bullets easier and overly enjoyment of the game increases because it feels far more fast paced and intense.





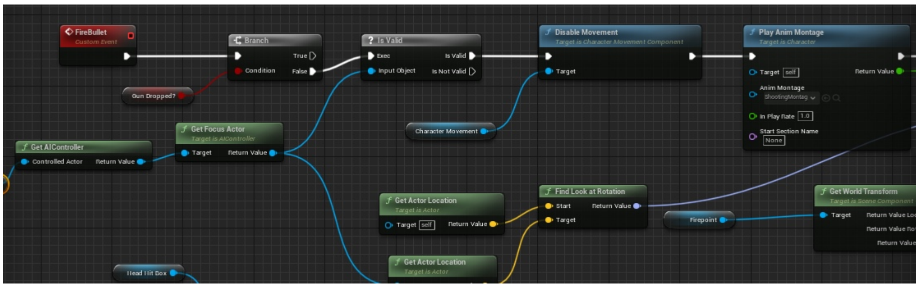
Enemy Shooting animation

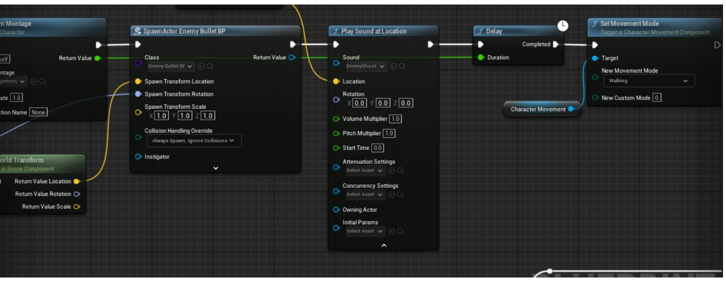
Montages

It is not possible to export **enemyshootingvid.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\enemyshootingvid.mp4**

When enemies run their shooting event they run an animation montage. Animation montages allow for blending and easy quick animations to play. Animation montages are assets that have a collection of animation clips that can be played in a particular sequence or are activated by certain triggers, such as a fire bullet custom event. You can adjust the logic and flow of the animation using their branching points, transitions, and notifications. Animation slots are named animation layers that can be applied to various skeleton segments of the character. They give you the option to combine animations from many sources, including the animation component, the animation montage, and the animation blueprint.

The fire bullet custom event is triggered from a behaviour tree service and makes sure to disable enemy movement before running the animation montage. This is because the animation features a stance that does not apply to a moving person and it makes more logical sense for them to stand still to be accurate. Before adding this disable movement node, the enemies would slide around whilst in their shooting animation which looked, and was, very broken. Adding this node fixed this bug.

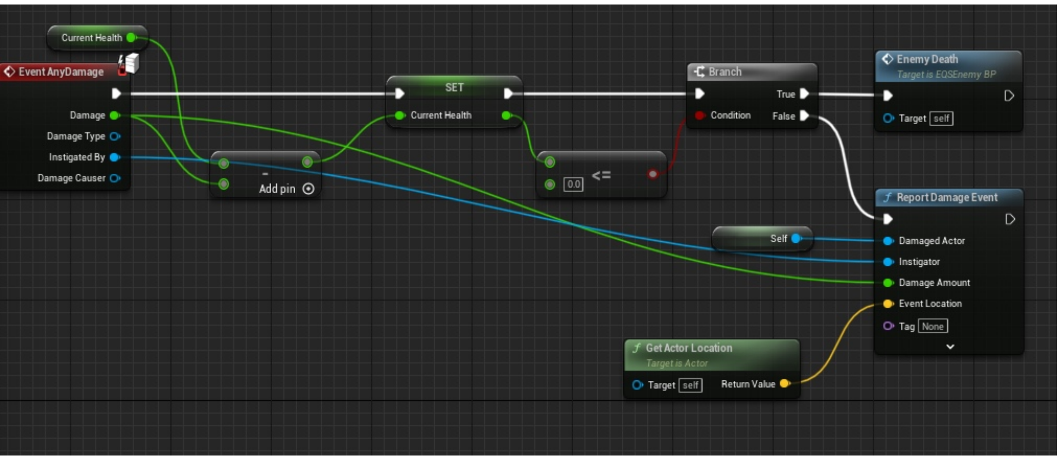




Once the animation is over the movement move is reset after a delay that has had its length set to the length of the animation. This ensures the enemies enter their normal animations properly. Without this node they would forever not move, or slide around in the animation.

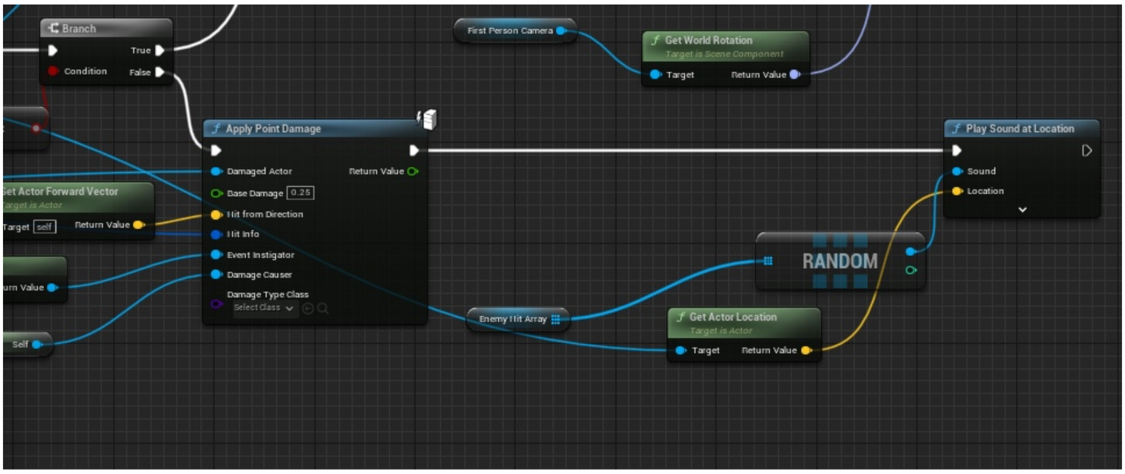
Enemy Health

Point Damage



When the player fires and their line trace hits an enemy, anywhere that is not the head, point damage is applied. It is standard practice to have enemies health be between 1 and 0 because it allows for easy interpretation into a percentage, needed for mechanics such as a health bar. The enemy health is checked to see if it is below 0 and if it is then the death event is called.

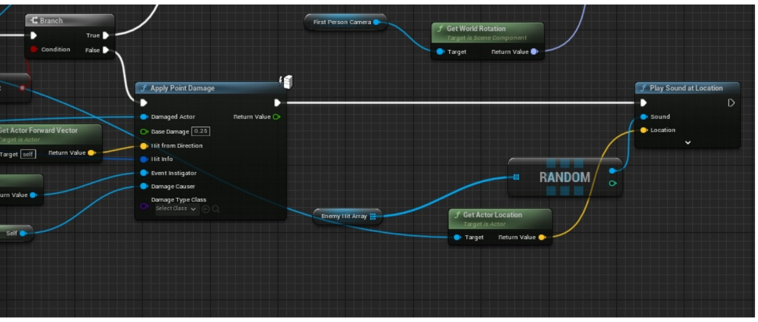
The point damage is applied by the FPC and then plays a sound at the location of the enemy.

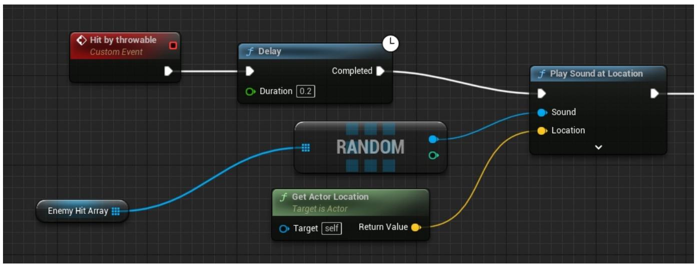


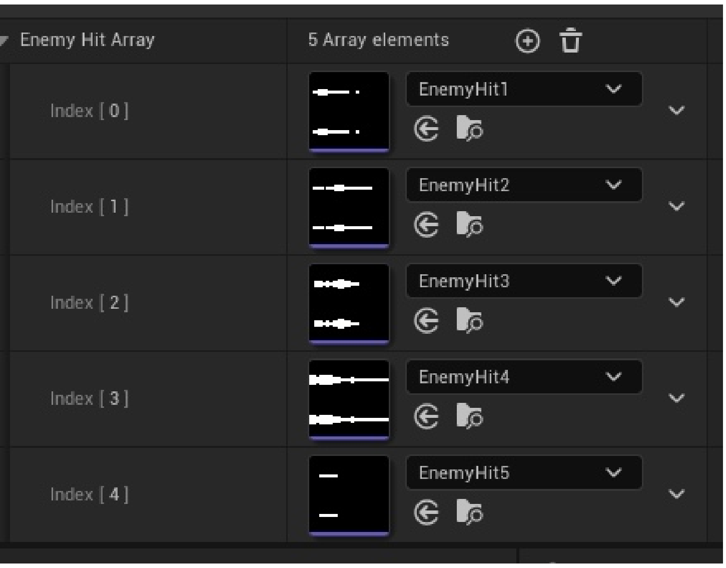
Random Hit/Death Sounds

Array

It is not possible to export **EnemyHitSounds.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\enemyhitsounds.mp4**



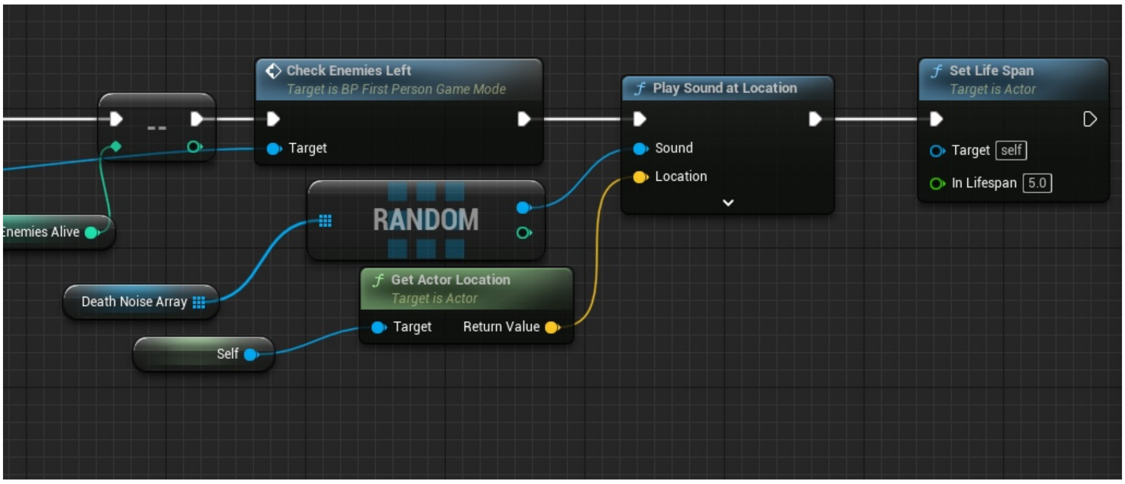




Enemy Hit Array	5 Array elements
Index [0]	EnemyHit1
Index [1]	EnemyHit2
Index [2]	EnemyHit3
Index [3]	EnemyHit4
Index [4]	EnemyHit5

When the enemy is hit by point damage and when they are hit by a throwable a random hit sound is played at their location. This is to make them getting hit not too annoying to the player because the same sound effect over and over would be irritating for some. The sounds are not spawned at event any damage because that caused an overlap with the throwable noises and hit noises. Adding the delay before playing the throwable hit sound effects fixed this issue.

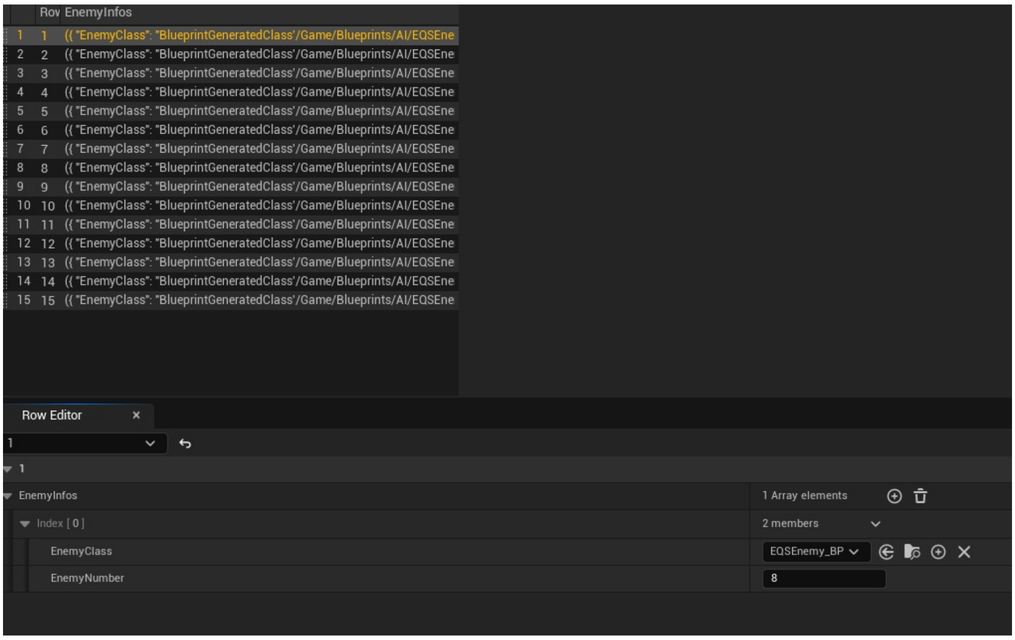
A very similar system is implemented for enemy deaths, allowing for more variation when enemies die.



Enemy Spawns

Data tables and arrays

Using 2 structures in a data table, 1 for enemy class and 1 for the number of enemies to spawn, I can use this data table to control the waves of enemies. This setup allows for very complex, customisable waves with different enemy types and how many of each you desire in the wave. It is import to set the row name to an integer so it can be referenced later.



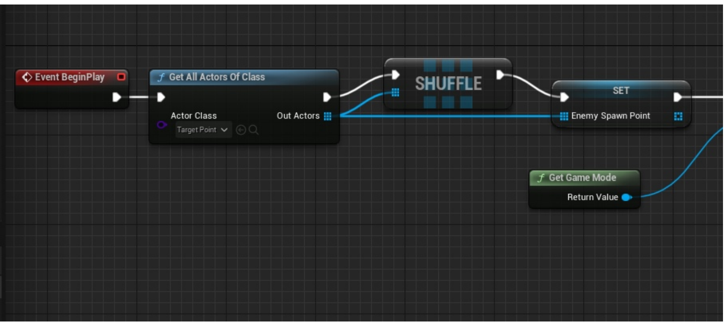
	Rov	EnemyInfos
1	1	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
2	2	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
3	3	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
4	4	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
5	5	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
6	6	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
7	7	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
8	8	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
9	9	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
10	10	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
11	11	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
12	12	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
13	13	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
14	14	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne
15	15	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne

Row Editor ×

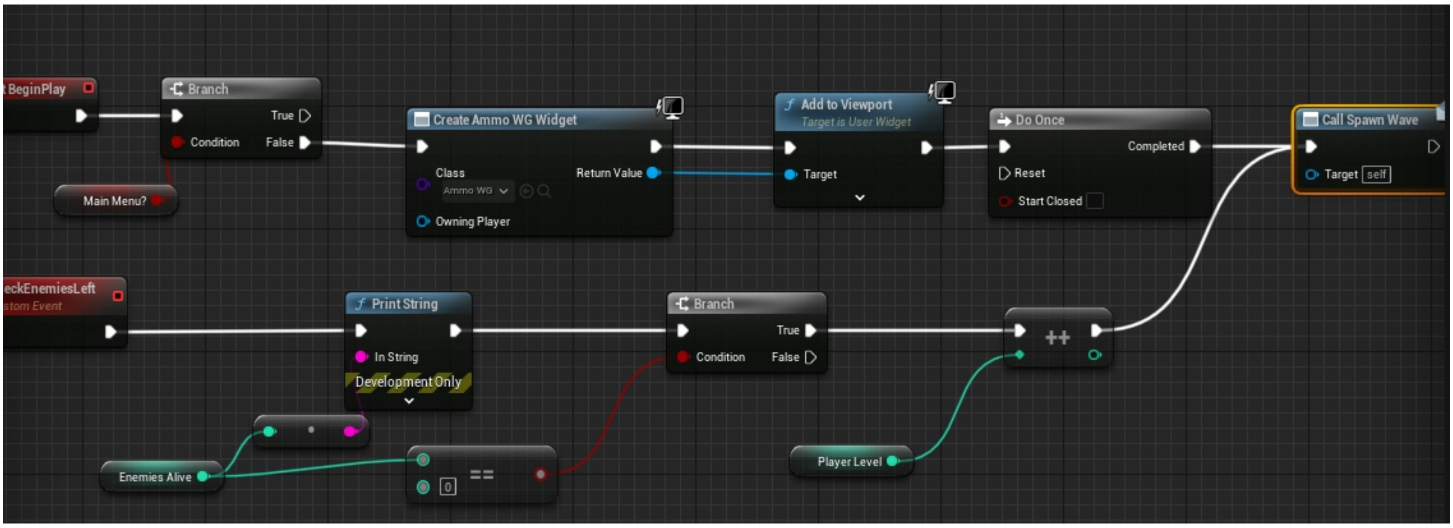
1

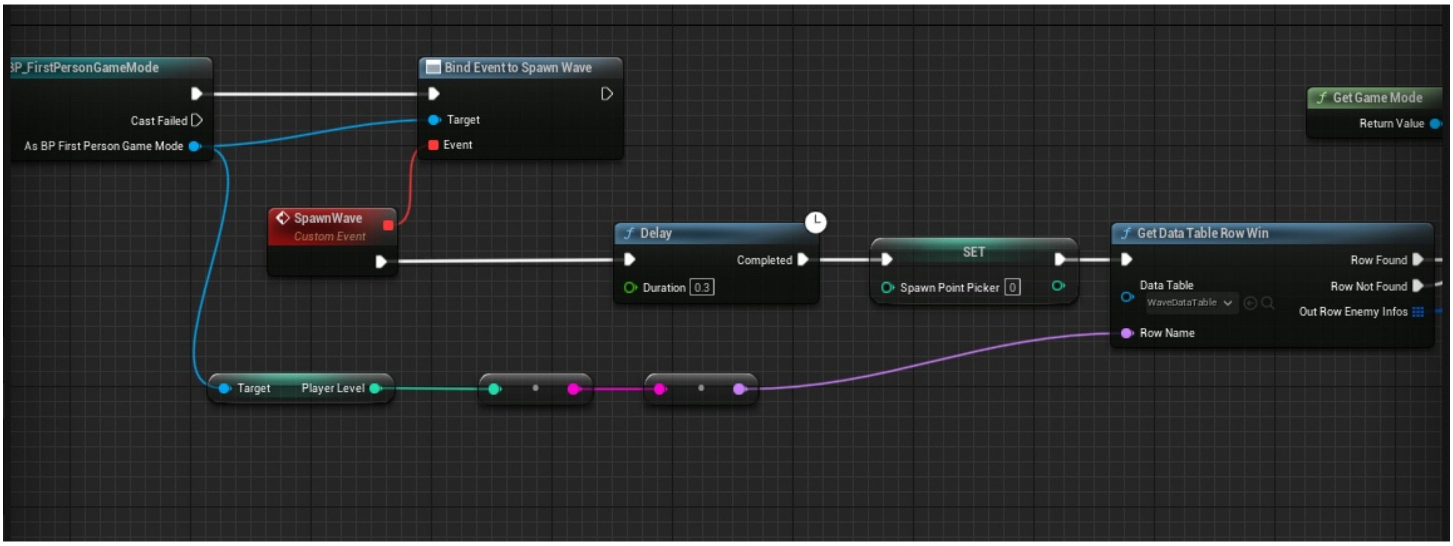
1

EnemyInfos	1 Array elements
Index [0]	2 members
EnemyClass	EQSEnemy_BP
EnemyNumber	8

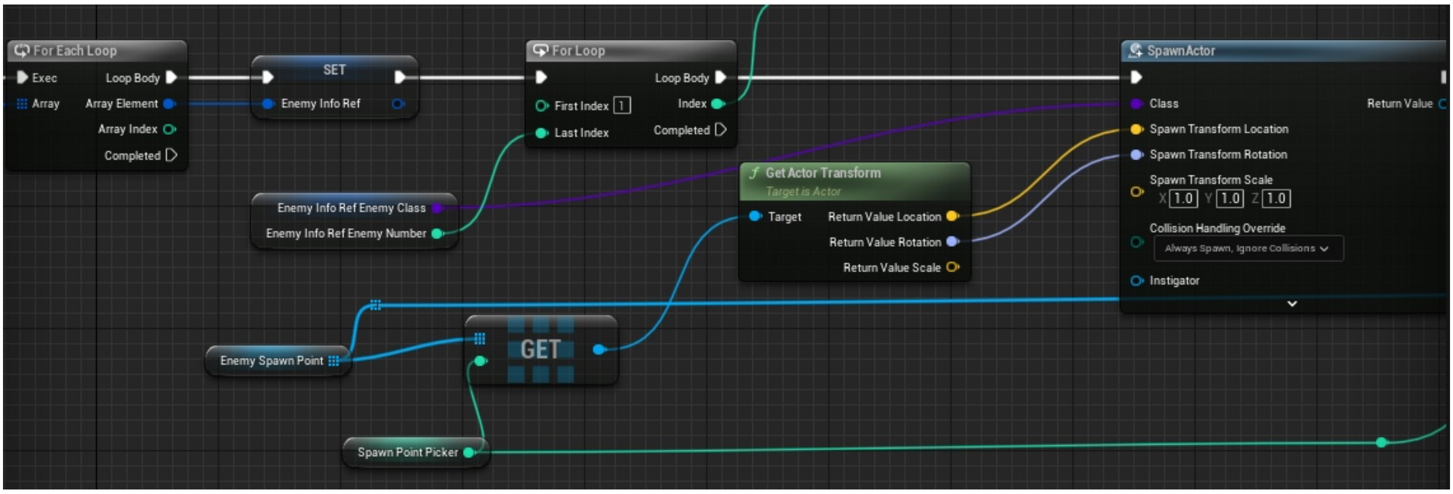


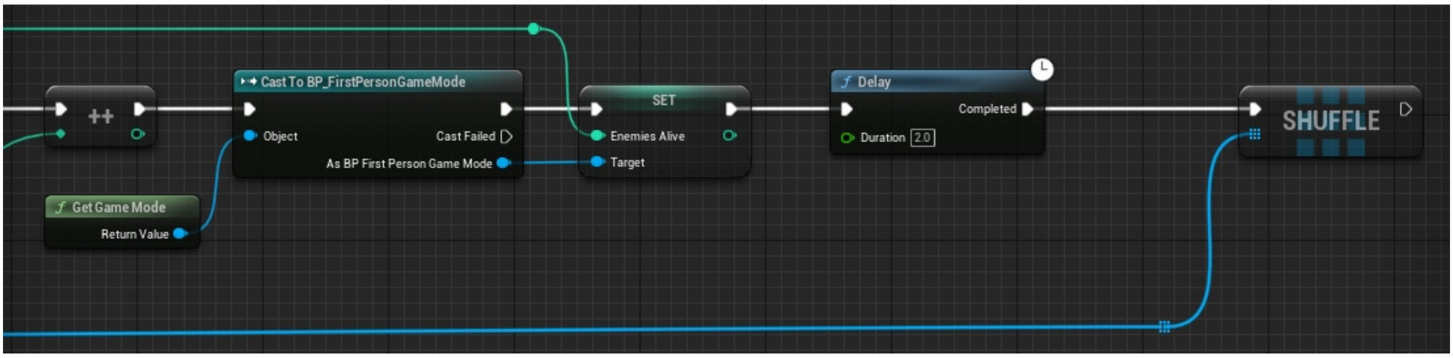
There are target points throughout the level that can be accessed through getting all actors of class. These target points act as spawn points for enemies and are used to set an array of spawn points that can be shuffled, allowing for random spawns for enemies.





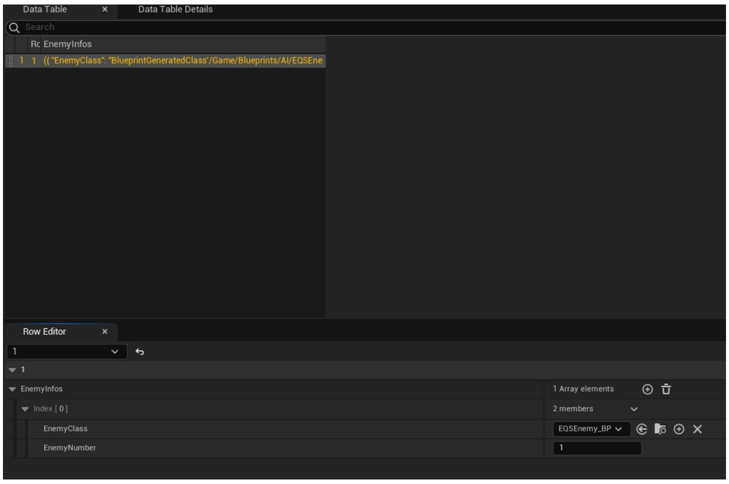
The spawn wave event is bound to an event dispatcher in the first person game mode that is ran on event begin play, for the first wave, and then every time the enemies alive variable is 0. The enemies alive variable has a decrement int ran every time an enemy dies, in the enemy bp. When the dispatcher calls the spawn wave event a small delay is ran to ensure that all the data table finds the correct row, via the player level variable. The spawn point picket is set to 0, which makes sure that it spawns the correct number of enemies each wave.





Once it gets the data table, the enemy spawn ref is set, which sets which actor to spawn from the data table and loops this for all the data table entries in that row. Then the number of enemies to spawn is set based on the data table info, via the last index in the for loop. Then this then runs into the spawn actor node. The location is a random target point, because of the shuffle at the start and end (ensuring all waves, including the starting wave, are random spawn points). It tracks which target points it has used by the increment int to "spawn point picker", which is why it has to be set to 0 at the start of each wave spawn, meaning 2 enemies cannot spawn in the same place. The number of enemies alive is then set at the end and then the target points are shuffled, ready for the next wave spawn. This system is extremely effective, working every time and allows for a high amount of control with enemy spawn because of the use of structures and data tables.

This system also allows for far easier testing of systems in the game. Creating a new data table with only a single enemy, or however many are needed for specific tests, and changing which table is called on in the wave spawns makes it easy to test mechanics, such as the win mechanic. This testing would usually take a long time as you would have to play through the entire game but with this method it is as easy as killing a single enemy. This also helped with many other issues, such as the shooting animation issues and behaviour trees, as it allows for less clutter in the level and just 1 single enemy to see how it functions.



Rc	EnemyInfos
1	(("EnemyClass": "BlueprintGeneratedClass'/Game/Blueprints/AI/EQSEne

1

EnemyInfos — 1 Array elements

Index [0] — 2 members

EnemyClass — EQSEnemy_BP

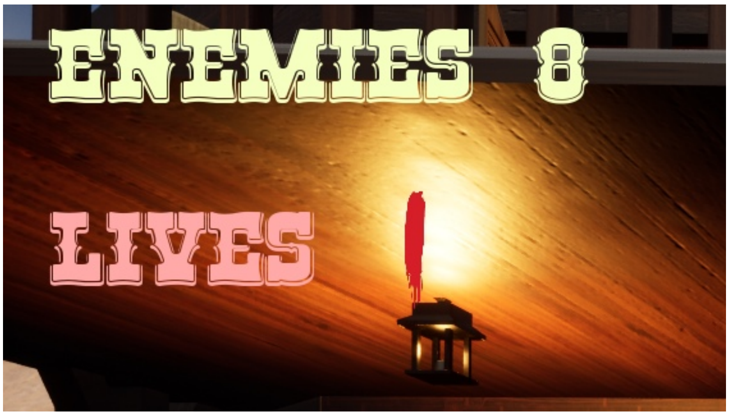
EnemyNumber — 1

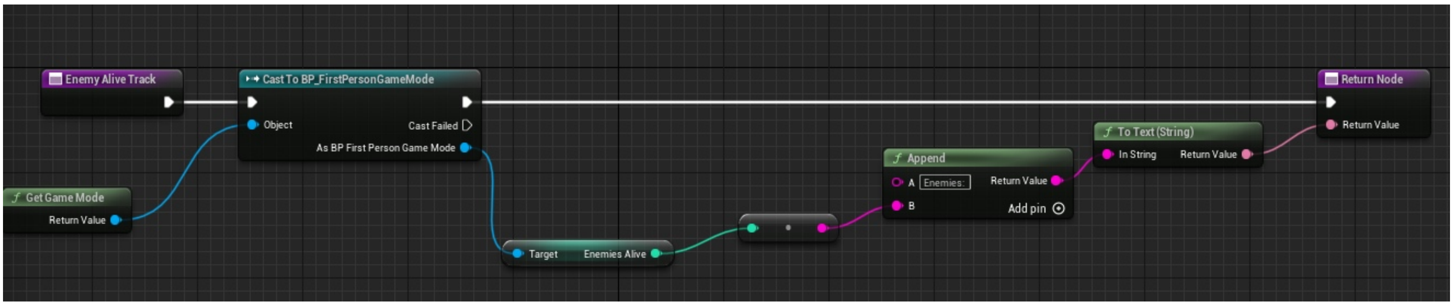
Enemy Spawn UI

Widgets

Widgets are customisable UI that you can add to a players viewport to display information. This can range from the crosshair, to health, to menus, to scopes and so on. The are extremely useful and allow for a high level of polish in a game.

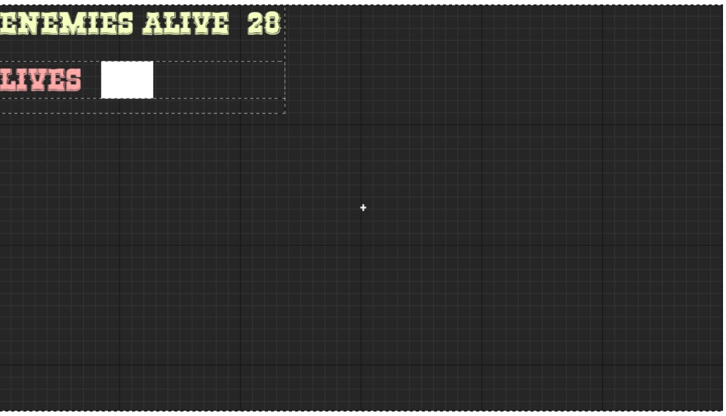
The widget that controls the number of enemies is also the widget with the crosshair and lives in. The enemy spawns text is bound to a function that gets the alive variable, from the game mode, and appends it with the text "Enemies".





Crosshair UI

Widgets



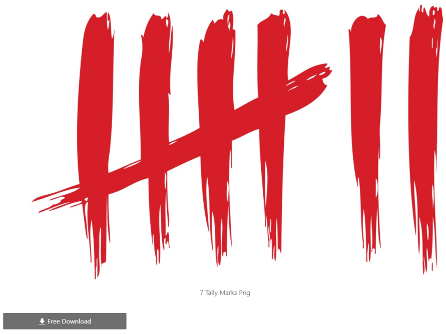
The crosshair is text in that appears in the centre of the screen, however, it is actually not in the centre of the screen. It has been adjusted to be slightly higher because the line trace that the first person character creates is not from the direct centre of the screen. Without adjusting the crosshair upwards slightly it would seem as though players are missing shots they should be hitting, which does not feel good.

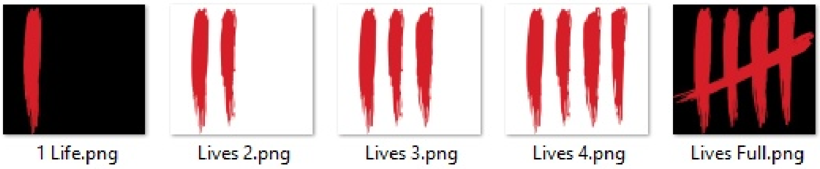
Lives UI

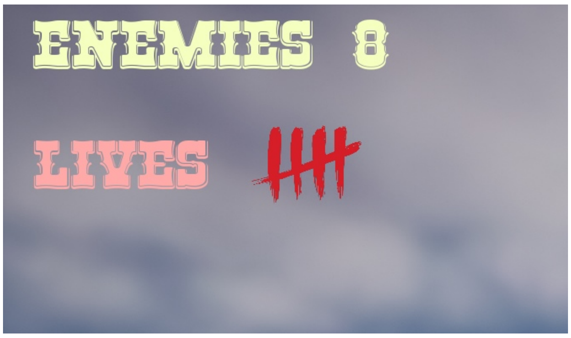
Widgets, Image V Text

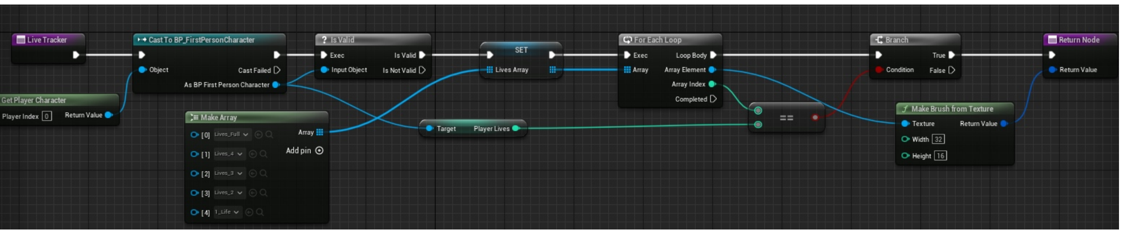
https://www.seekpng.com/ima/u2w7t4q8q8i1t4u2/

I found this asset on a free use site that I cropped the end 2 marks off. Then, in photoshop, I hid selected and hid each mark 1 by 1 until I had a set of 5 lives.







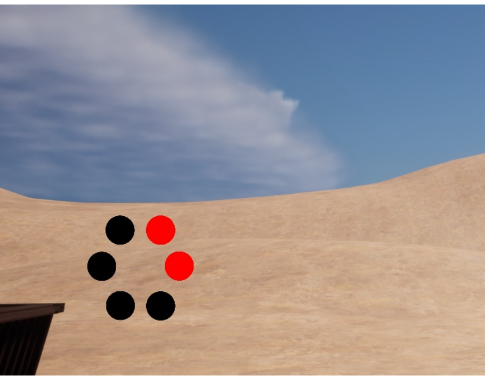


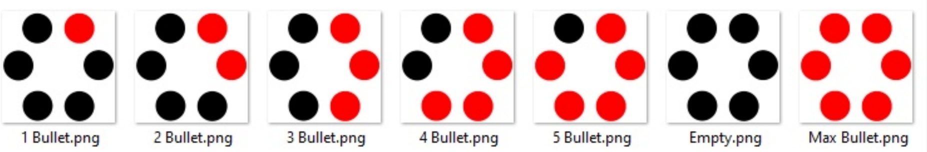
In the crosshair widget, I made an array of the images and bound the image next to the "lives text" to this array. I then get the players lives variable from the first person character and checked if the array index was equal to the number of lives and set the array element to the brush. I used an image, rather than text, because it looks more visually appealing and stands out far more than just text.

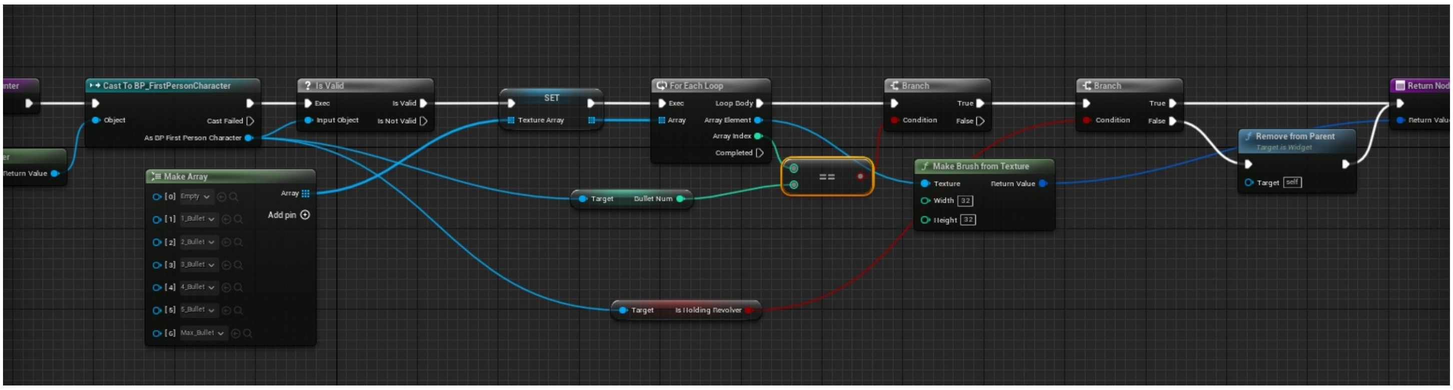
Ammo UI

Widgets, Text V Image

Using photoshop I made 6 circles that resemble a revolver magazine, fitting the theme. I then coloured each circle 1 by 1 until there was a set of 7 images that can display every amount possible. These images were used, rather than text, again because it fits the aesthetic of the game far more than boring text.





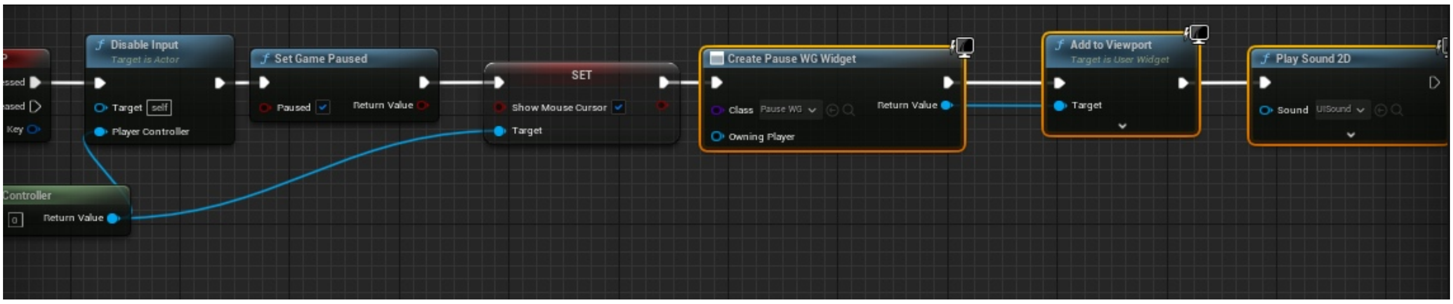


The crosshair and ammo widget needed to be separated because of the throwing revolver mechanic and having 2 separate widgets made it far easier to remove the ammo count from the screen, when the player didn't have a gun. The blueprint works in the same way of the lives, except it checks the ammo count from the character and, importantly, checks if the player is holding a gun or not. If the player is not holding a gun then the widget removes the counter from the parent, hiding it from the viewport.

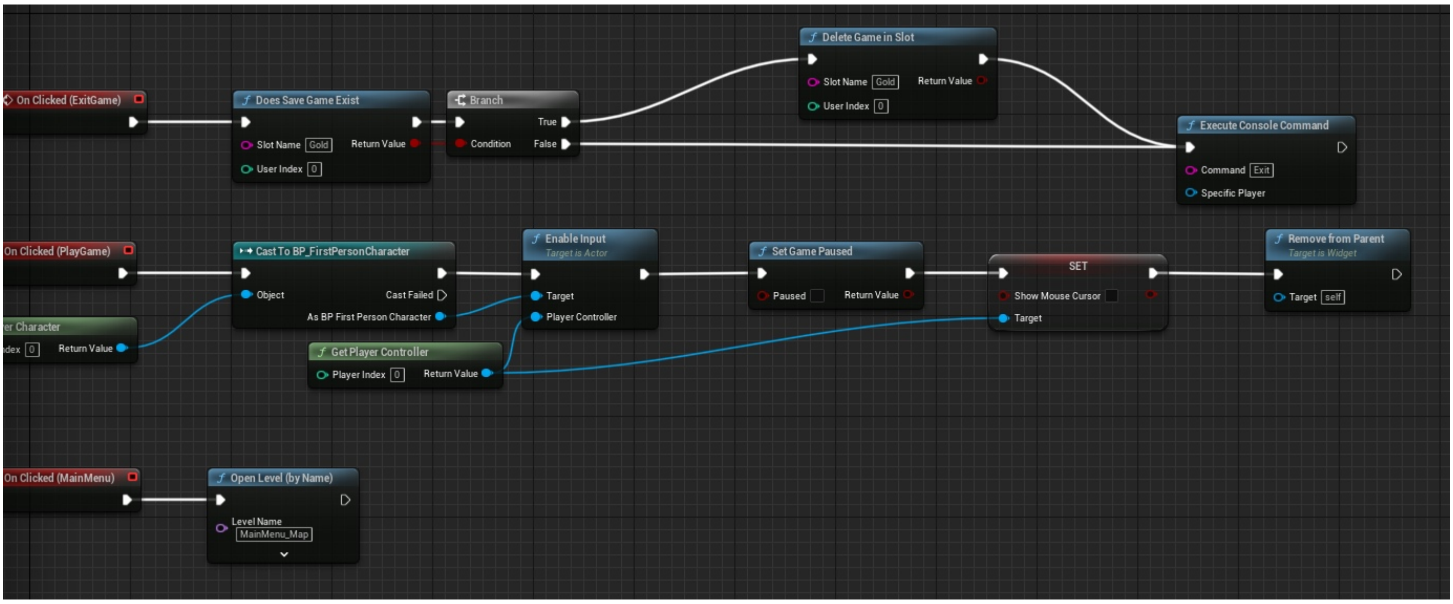
Pause Menu

Widgets

It is not possible to export **pauseing.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\pauseing.mp4**



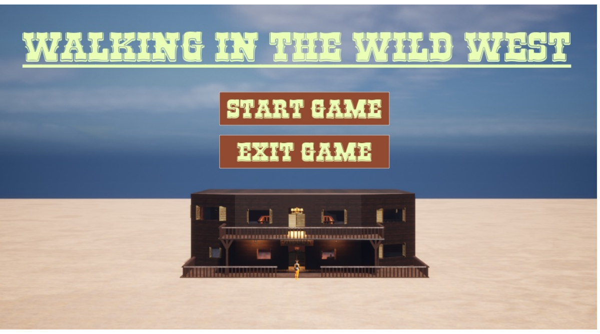
The pause menu is a fairly essential thing to have in the game to allow players to have a break mid game. It works by, when P is pressed the game disables input, sets the game to paused and shows the mouse cursor, so players can interact with the widget that is added to viewport. It also plays a 2D UI sound, to give plays some audible feedback.



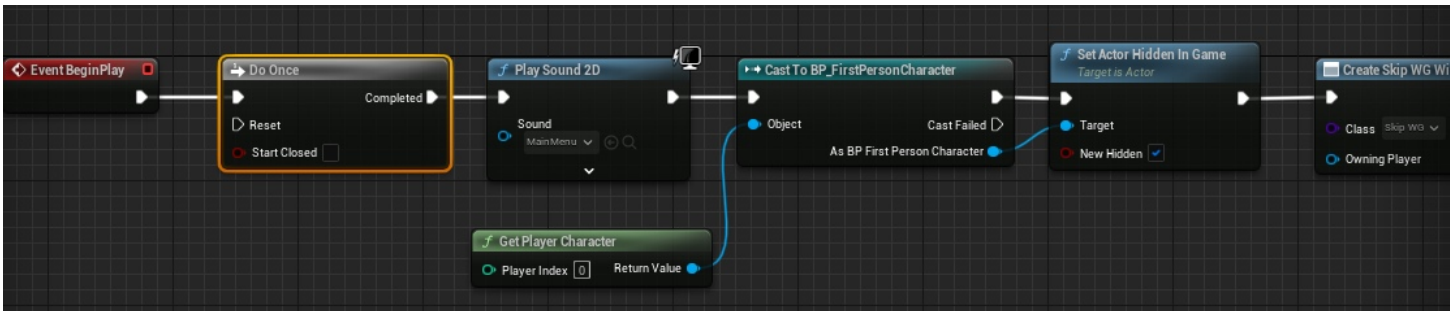
The menu itself allows players to exit the game, resume the level and exit the game. Resuming the level does the inverse of what pressing P originally did.

Main Menu

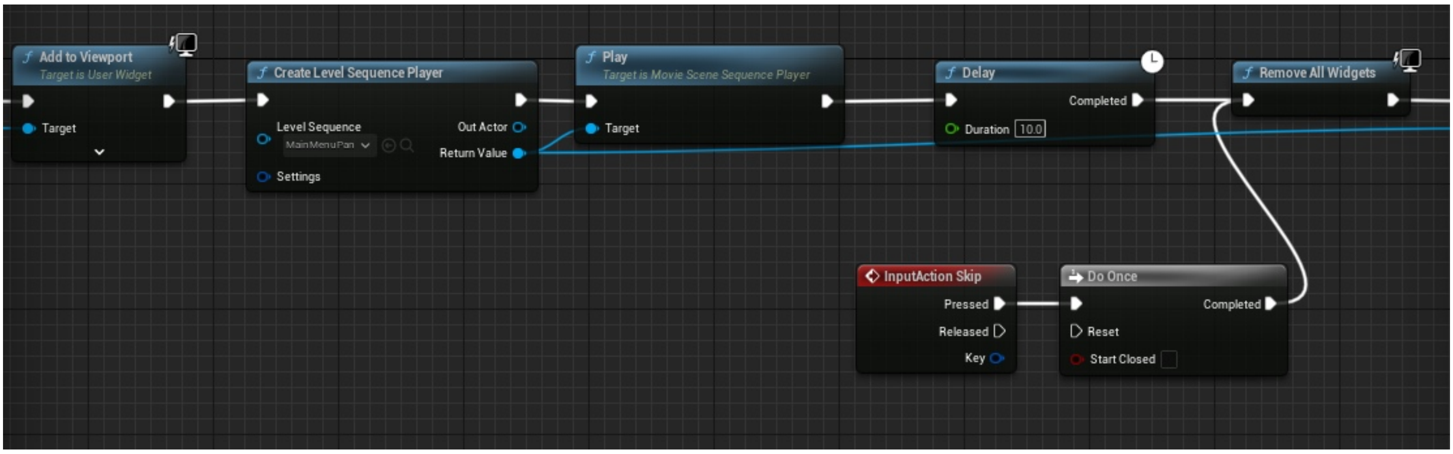
Widgets, Levels, Level Sequencers



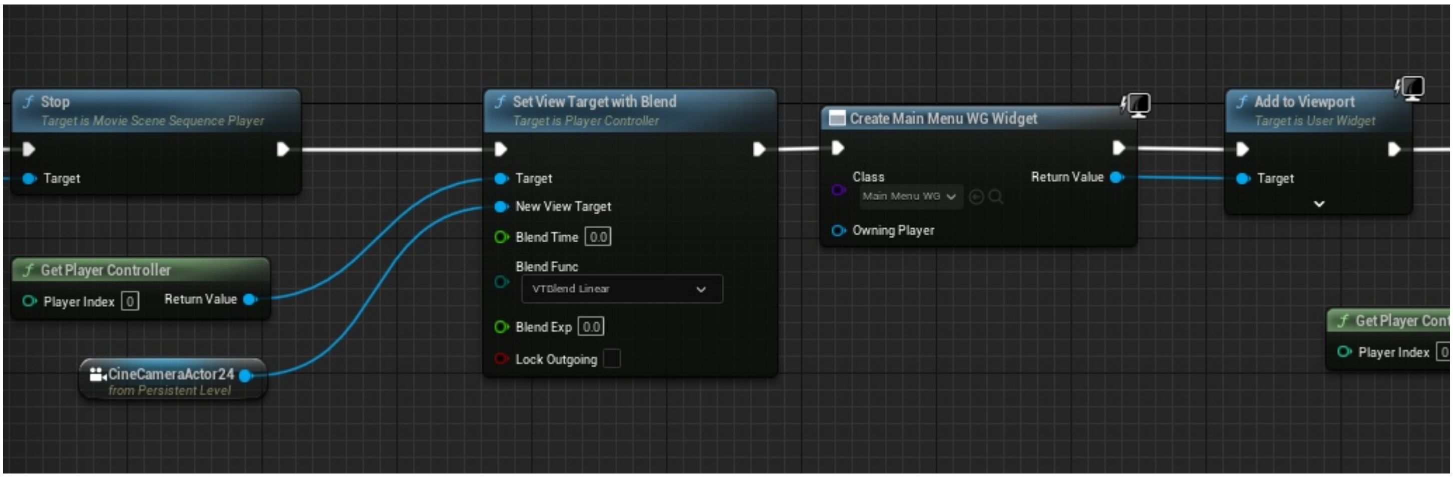
It is not possible to export **mainmunvid.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\mainmunvid.mp4**



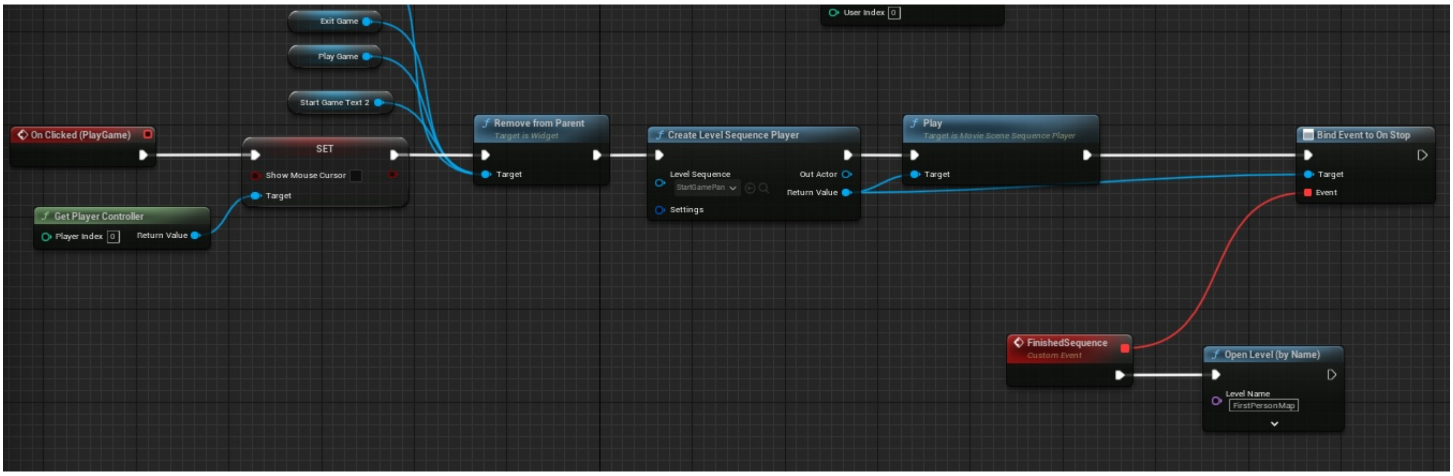
The main menu is a new level to the actual game level. This is because it is easy to restart a game by restarting a level and players dont want the main menu every time they want to play again. The main menu level blueprint handles how the main menu runs and it starts by playing the menu theme when opened. It then hides the player character because there was a repeated issue with the character spawning, this has fixed that issue. Then it loads a skip widget. The skip widget is there because of the level sequencers and allows for players who do not want the opening cutscene every time to skip it.



The opening cutscene is then played and a delay the length of the sequence ensures the entirety of the sequence can play. The input action for skip allows for the player to bypass this delay and removes the skip widget from viewport and stops the sequence.



There is a camera actor that is set to exactly where the sequence ends to give a seamless transition to the actual menu. The menu widget is then added to the viewport.



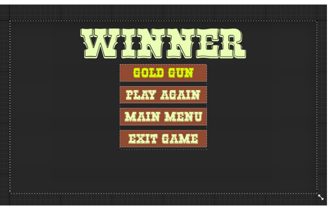
The play level button starts a sequence and removes all the text from the text and then runs the level once it finishes. The sequence ends right behind what the players are lead to believe is the player character and adds a sense of depth to the game as they can see who they are playing as.

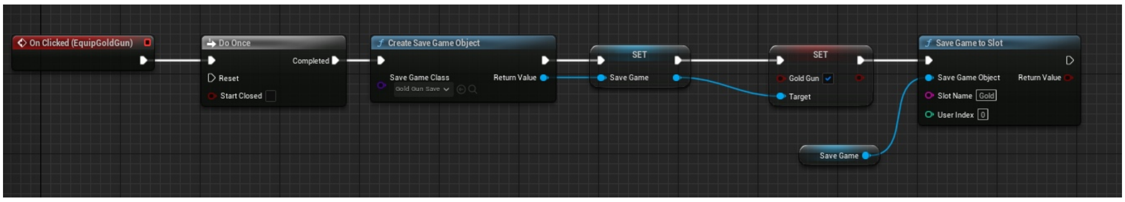
Winning

Widgets, Materials, Save games

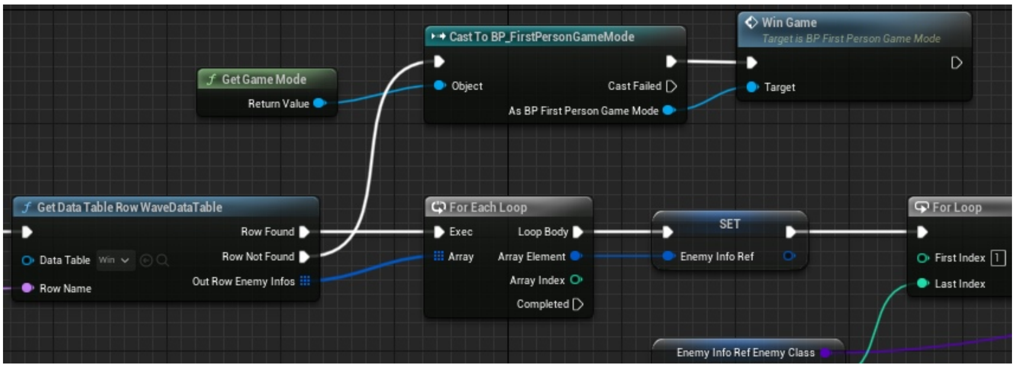
SaveGame class is preconfigured to save and load the data from the variables you add. Players computer then stores the variables set in a file. The variables can then be accessed in your game after this has been read. Save games have massive use cases in all games because it allows for progress to be saved and is used almost universally used across every game. In my game I simply use it to store a boolean but the same system can be used to store any type of variable for many different use cases.

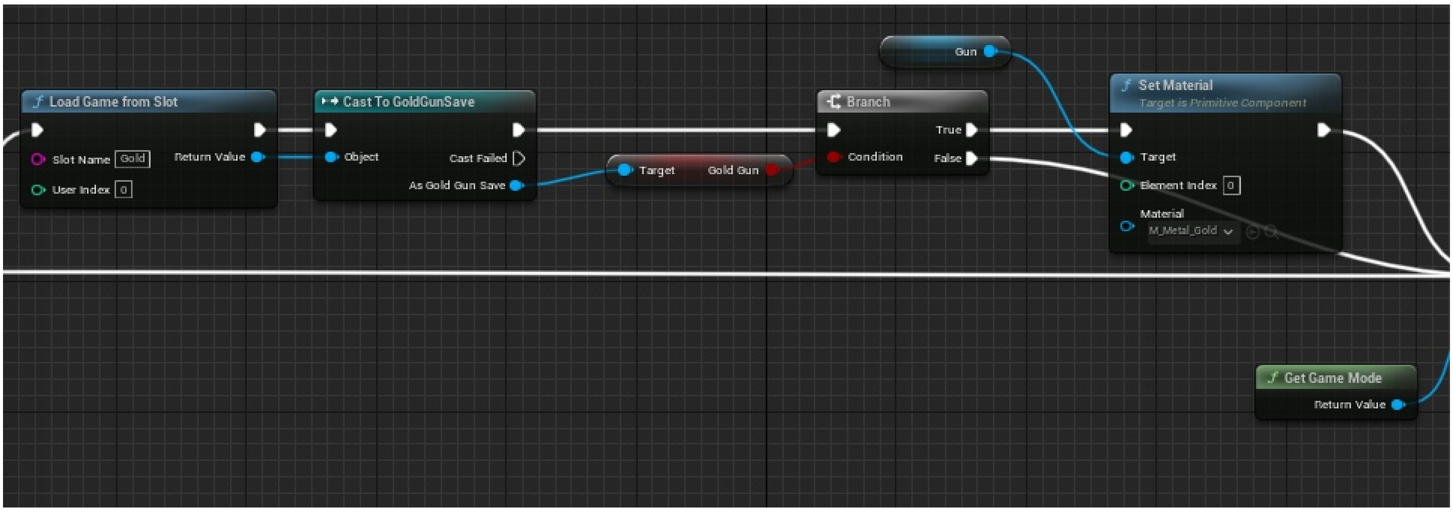
It is not possible to export **goldgunvid.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\goldgunvid.mp4**

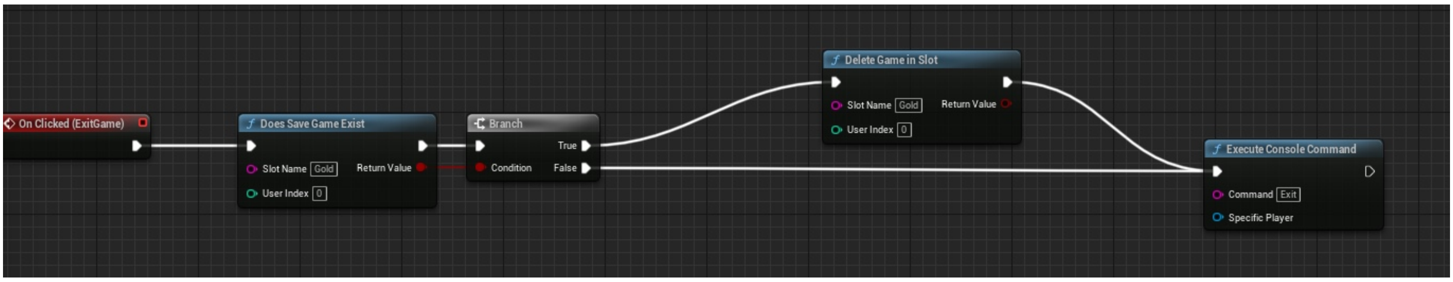




The winner function is called when the spawn enemies bp cannot find a row in the data table (after wave 15). The function does the same as the pause menu except its for the winner screen. The winner screen has a button that can be pressed called "gold gun". When pressed it creates a save game to store the variable "Gold Gun", which checks if the player has pressed the button.







On event begin play in the first person character, its checks if the save game exists and then checks if the boolean is true or false, on true it sets the players gun material to the starter content gold. I then have it set up so that when a player presses the exit button (on any version of the exit button) deletes the save game. This is so the player does not get to keep the gold gun forever, removing the replayability.

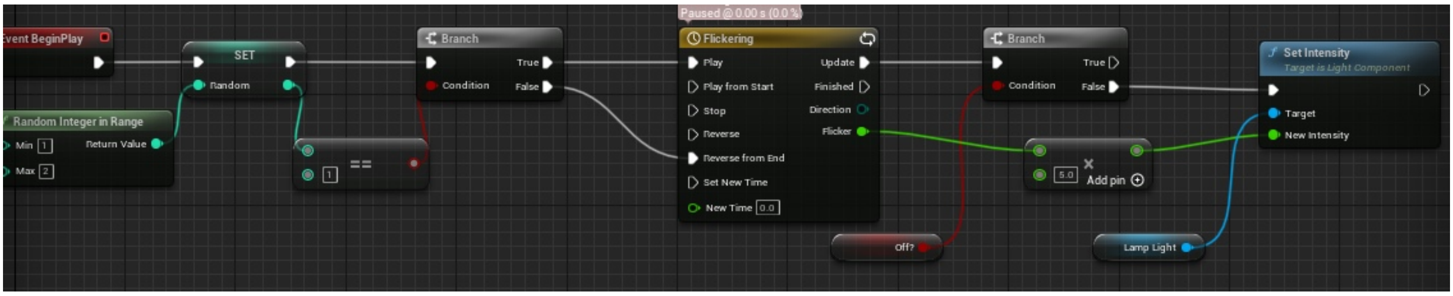
Chandelier/Lamp

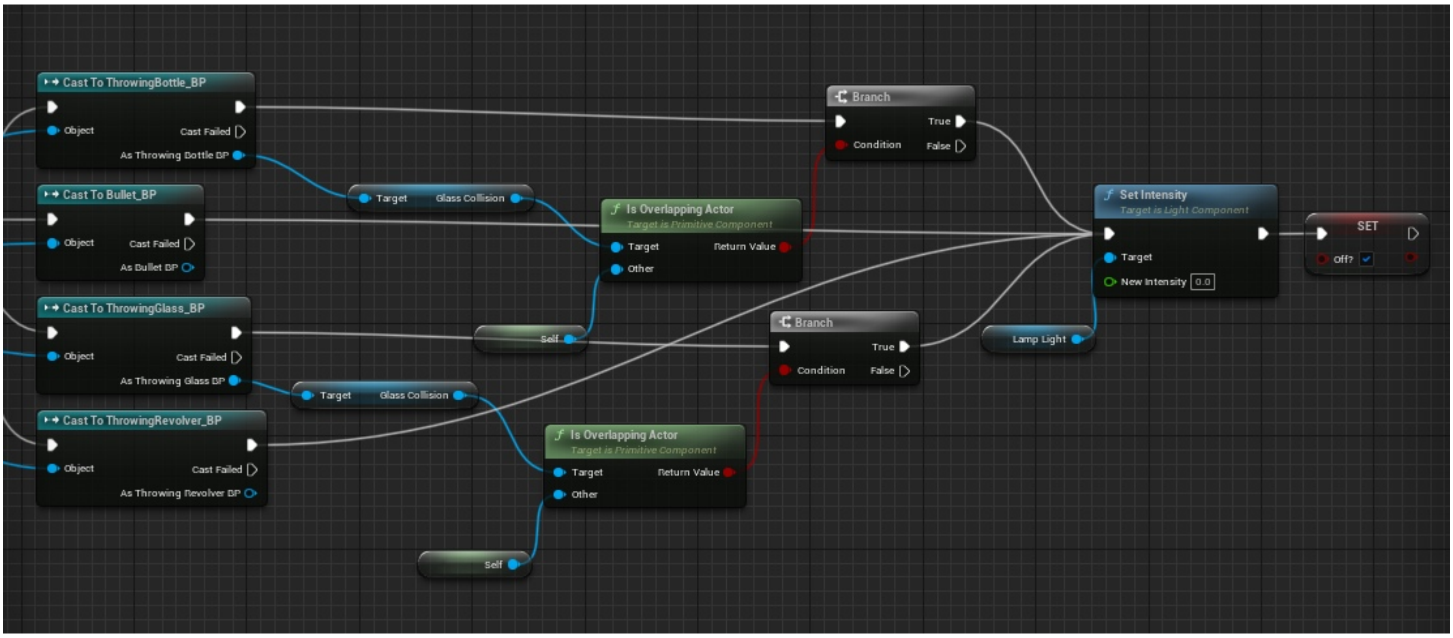
Flickering, lighting

It is not possible to export **chandelier.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\chandelier.mp4**

It is not possible to export **lamp.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\lamp.mp4**

The chandelier and lamp both have timelines that set the intensity of the lights, giving the impression that they are flickering. The random integer sets whether the timeline should play forward or backwards, meaning they dont all flicker at the same rate. The lights also have a the ability to turn off when hit by a throwable, by setting the intensity to 0. The chandelier has set simulate physics too when hit by a throwable and then has a life span set, much like the enemy death.

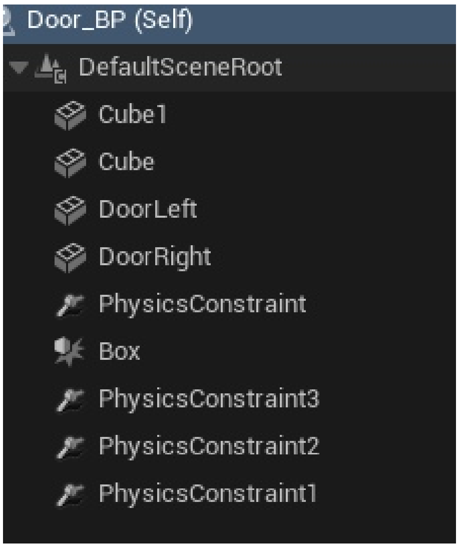




Door Physics

Physics constraints

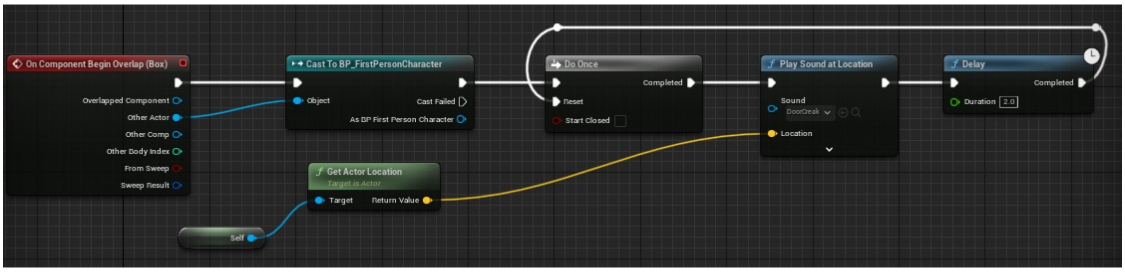
I wanted my saloon style door to open much like the classic saloon doors that you see in a western movie. To do so they needed to simulate physics, however, without constraints, they would spin unpredictably and uncontrollably. The physics constraints are set up on either side of the side and prevent it from completing a full rotation. The constraints also prevent gravity from locking them to the floor.

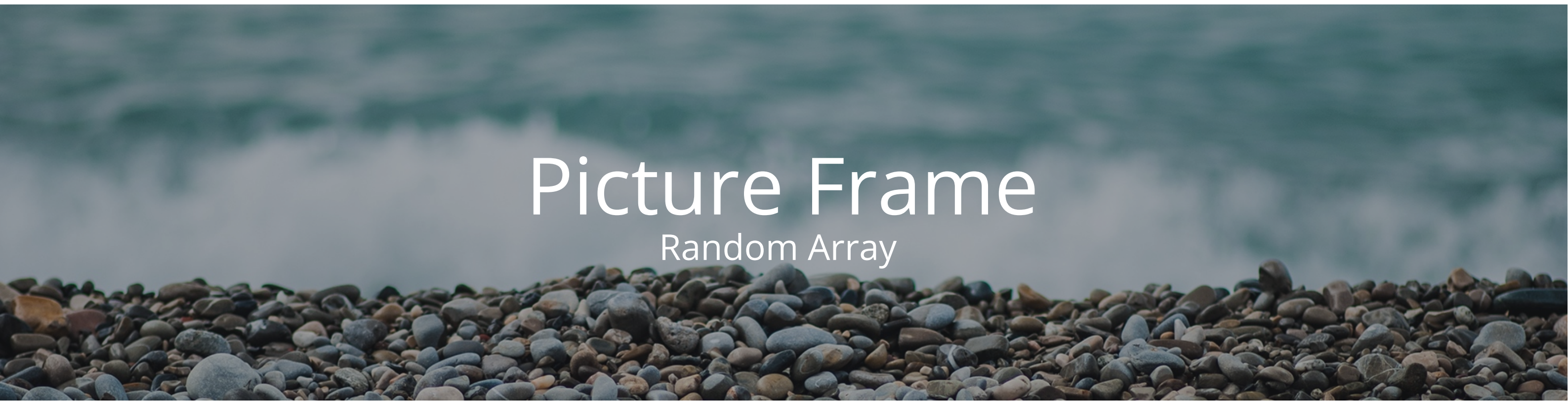


Door_BP (Self)
- DefaultSceneRoot
  - Cube1
  - Cube
  - DoorLeft
  - DoorRight
  - PhysicsConstraint
  - Box
  - PhysicsConstraint3
  - PhysicsConstraint2
  - PhysicsConstraint1

It is not possible to export **door squakvid.mp4** here. The file type is not supported by the pdf. The file has been made available alongside this PDF at **Files\door squakvid.mp4**

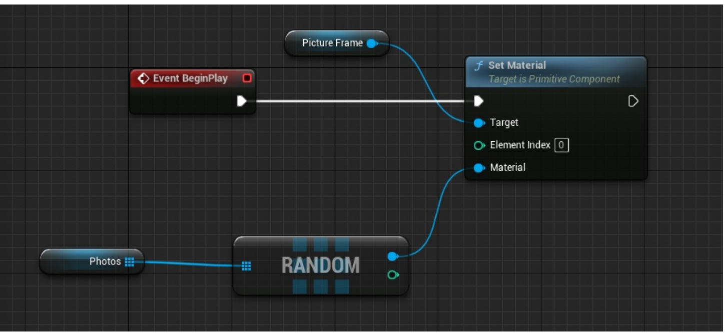
The door squeak is controlled by an overlap box and just plays a sound at its location. The do once node is to stop it from repeatedly squeaking and is reset by a delay.

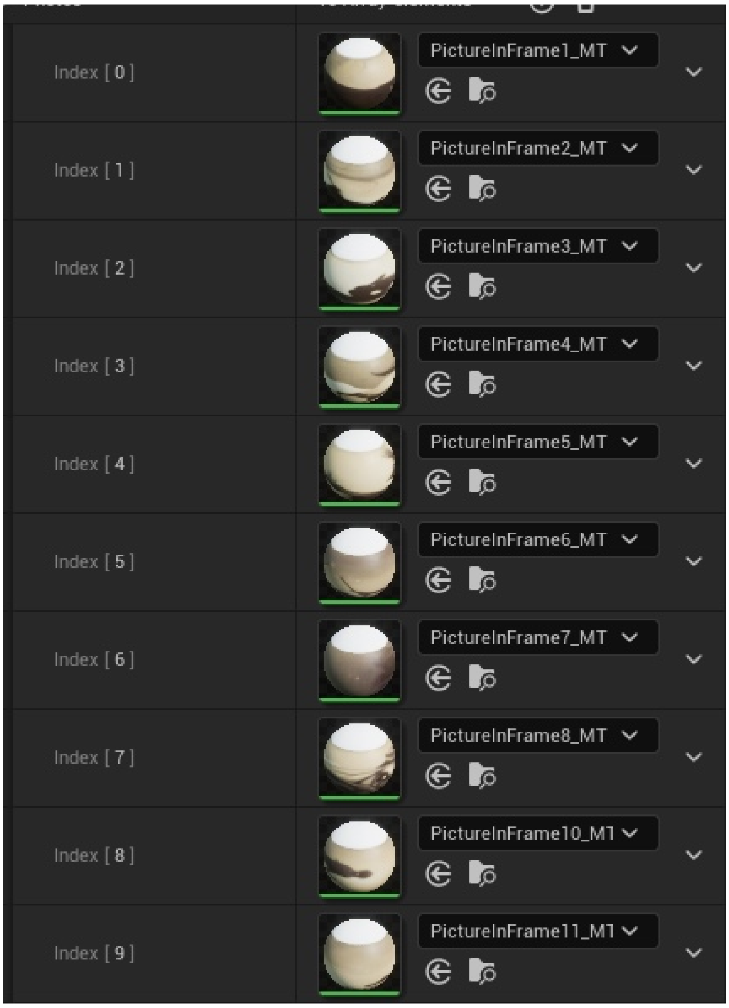




Picture Frame

Random Array





Index [0] PictureInFrame1_MT

Index [1] PictureInFrame2_MT

Index [2] PictureInFrame3_MT

Index [3] PictureInFrame4_MT

Index [4] PictureInFrame5_MT

Index [5] PictureInFrame6_MT

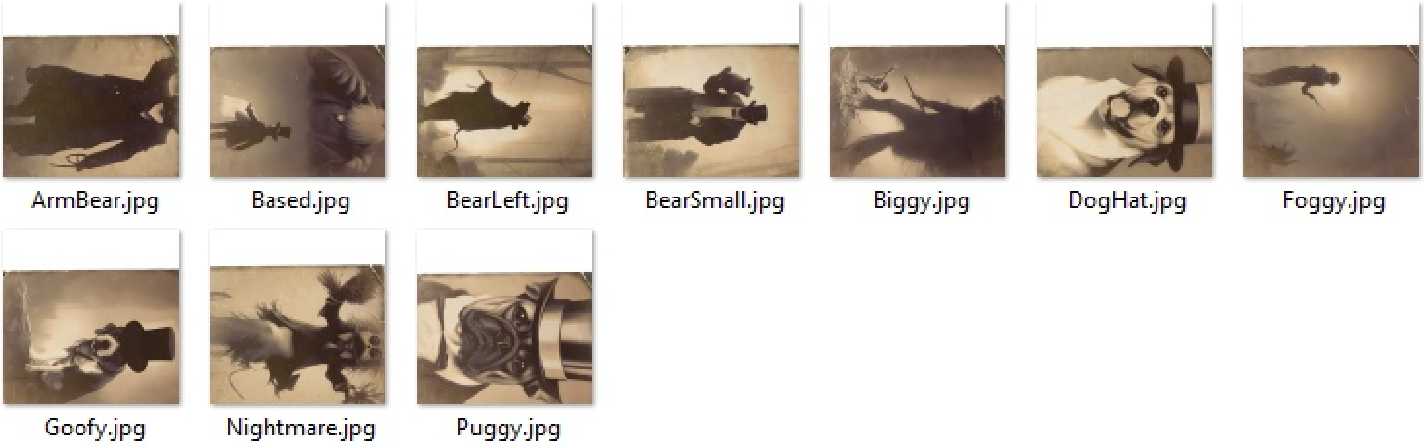
Index [6] PictureInFrame7_MT

Index [7] PictureInFrame8_MT

Index [8] PictureInFrame10_M1

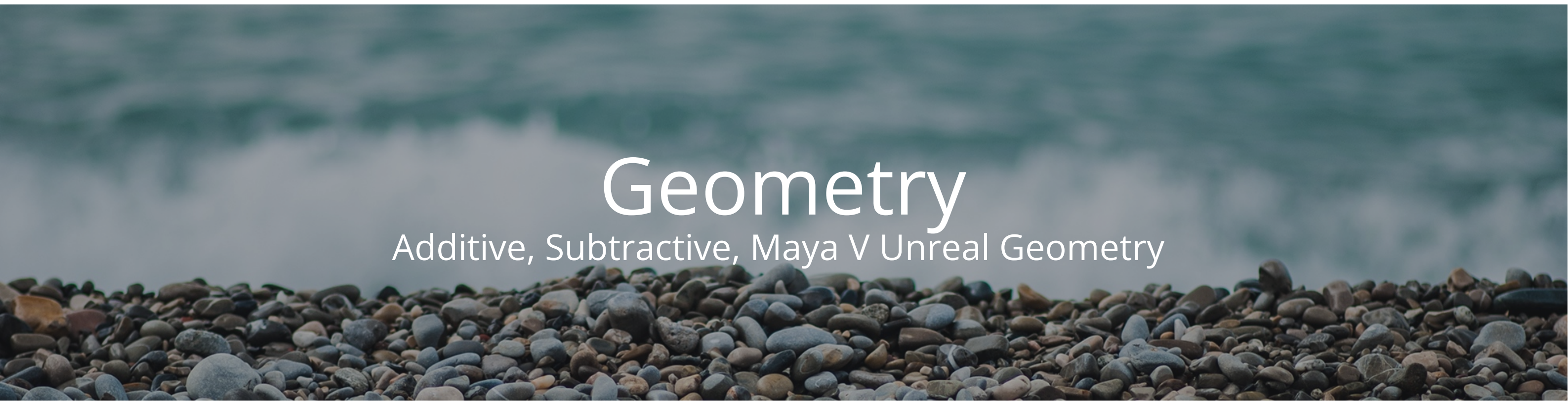
Index [9] PictureInFrame11_M1

The photo frames around the level are set to have a different picture in them every time, to prevent having just 1 boring picture frame. This was done by making an array of all the pictures and setting the material to a random texture from this array.



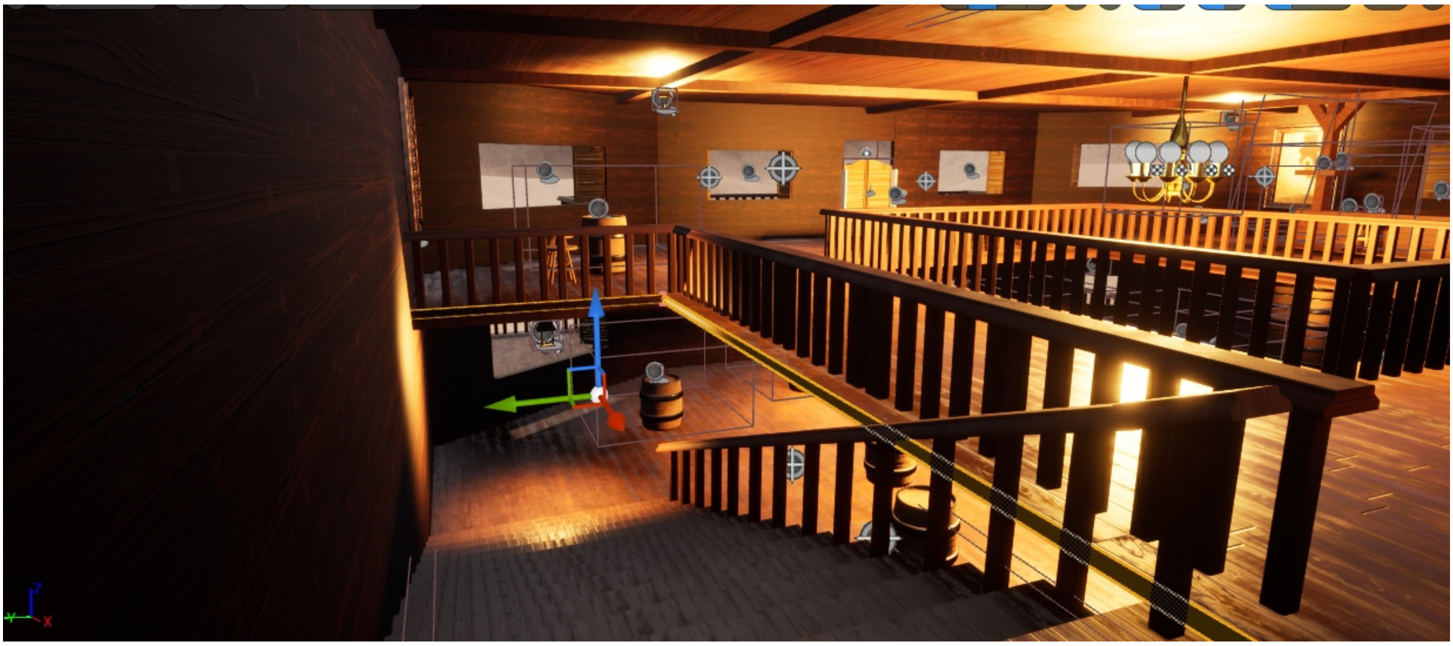
The sites I used to generate the are was nightcafe and then upscaled in upscaler.
https://upscaler.stockphotos.com/signup

https://creator.nightcafe.studio/



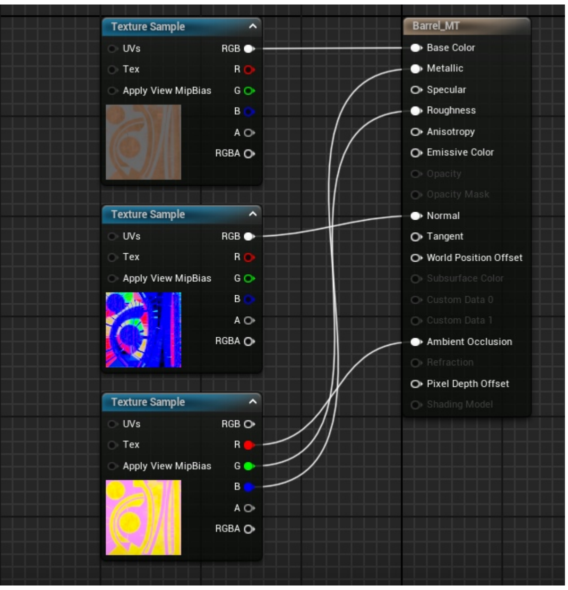
Geometry

Additive, Subtractive, Maya V Unreal Geometry

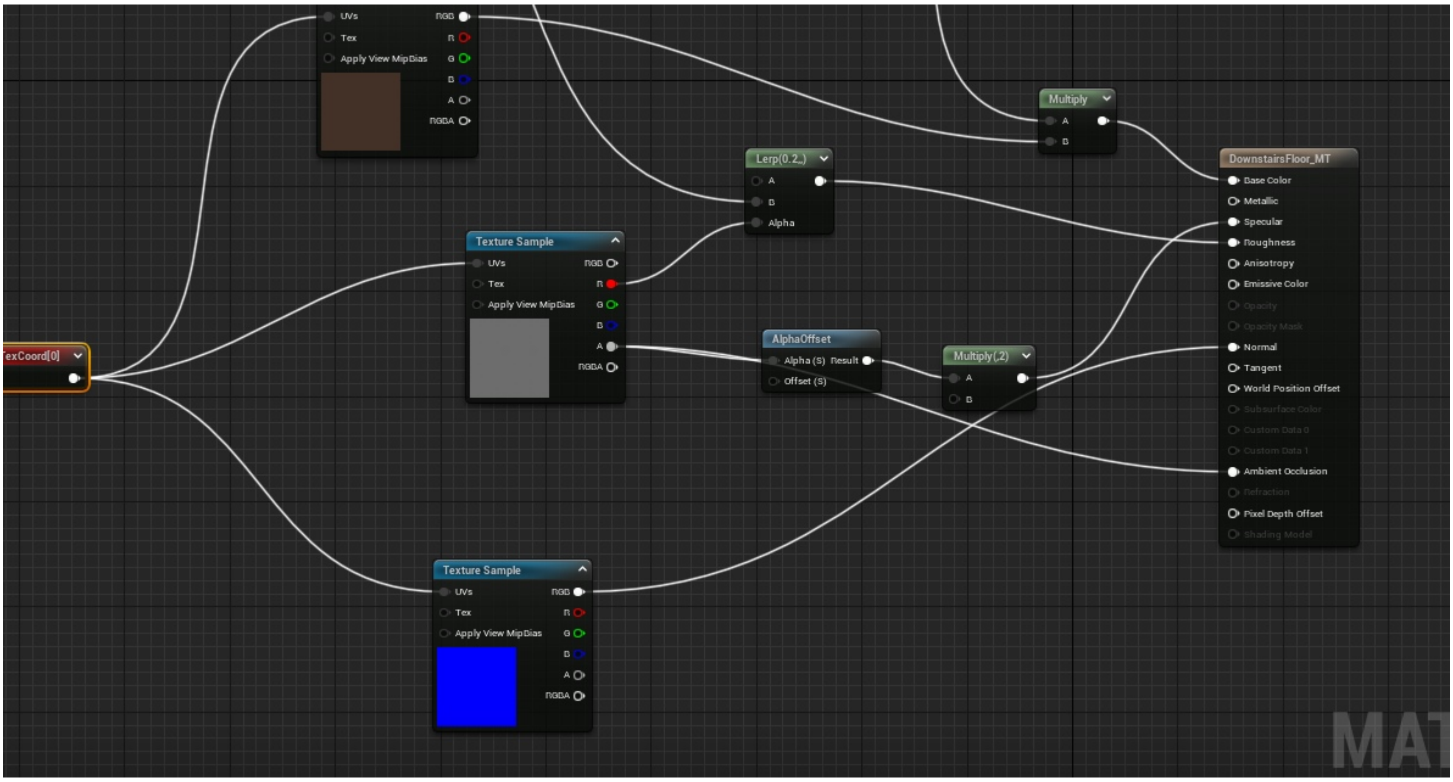


I made my level using the geometry features in Unreal. Given the chance to do this project again, I would use maya to make the level as I have ran into multiple issues with using Unreal's geometry tools. The issues mainly revolver around the use of subtractive boxes and that messing with the AI's pathing, particularly on the stairs. The other issue was the texturing because everything was stretched. The texturing for the building is done with a material I made in substance and then using the starter content material "M_Wood_Floor_Walnut_Worn" for the floor with adjustments to the scale so that it looked appropriate for the level, fixing the stretching issues.

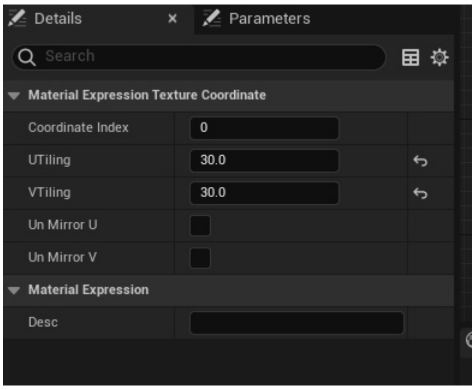
Materials



I made my textures in substance painter and then exported them into unreal. In unreal the material editor allows for a lot of dynamic changes to textures, which is why I used it.



To fix the issues with stretching textures I researched into tiling and discovered that by using a UV coordinate node you can set the UV tiling options allowing, otherwise stretched materials, to appear far more natural and in proportion.

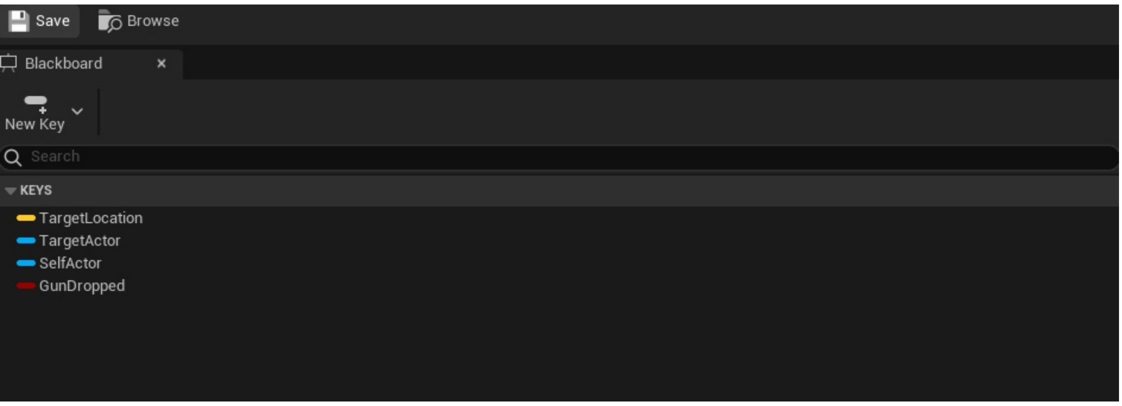


Material Expression Texture Coordinate

Coordinate Index	0
UTiling	30.0
VTiling	30.0
Un Mirror U	□
Un Mirror V	□

Material Expression

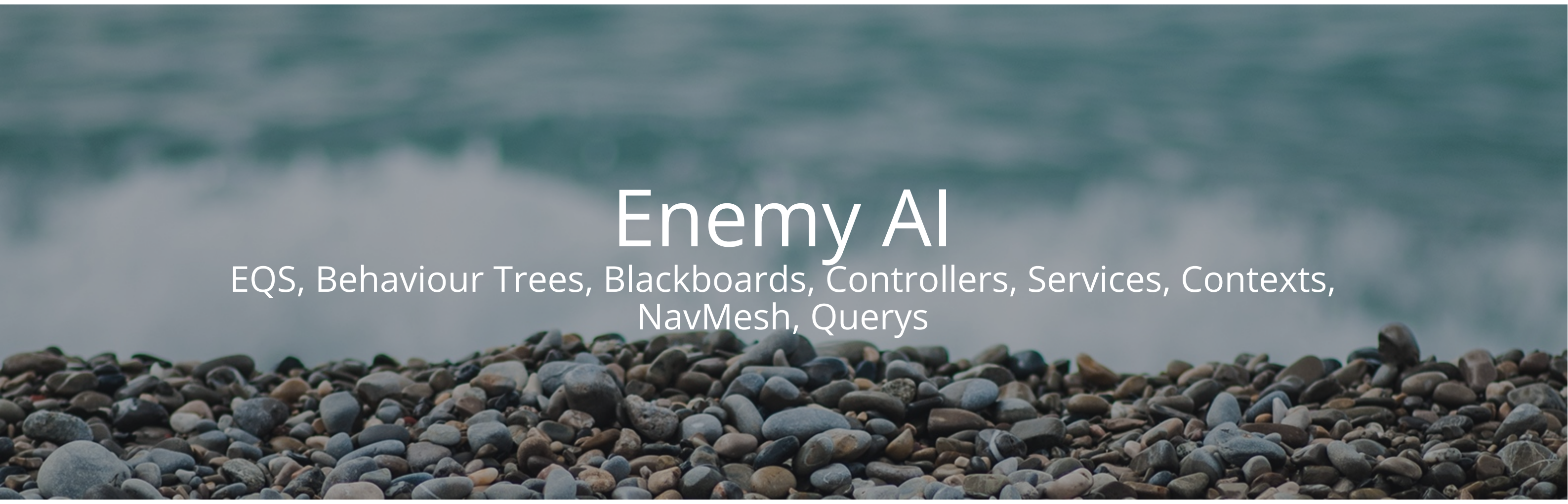
Desc	



KEYS

- TargetLocation
- TargetActor
- SelfActor
- GunDropped

Blackboards a way that behaviour trees access variables. The keys are set within tasks and services and are used in generators. Generators are what are used to apply conditions on whether a branch of the behaviour tree such run or not. They also let you abort a branch, meaning if a variable changes it will cancel what is below it, there are also customisable to abort specific branches.

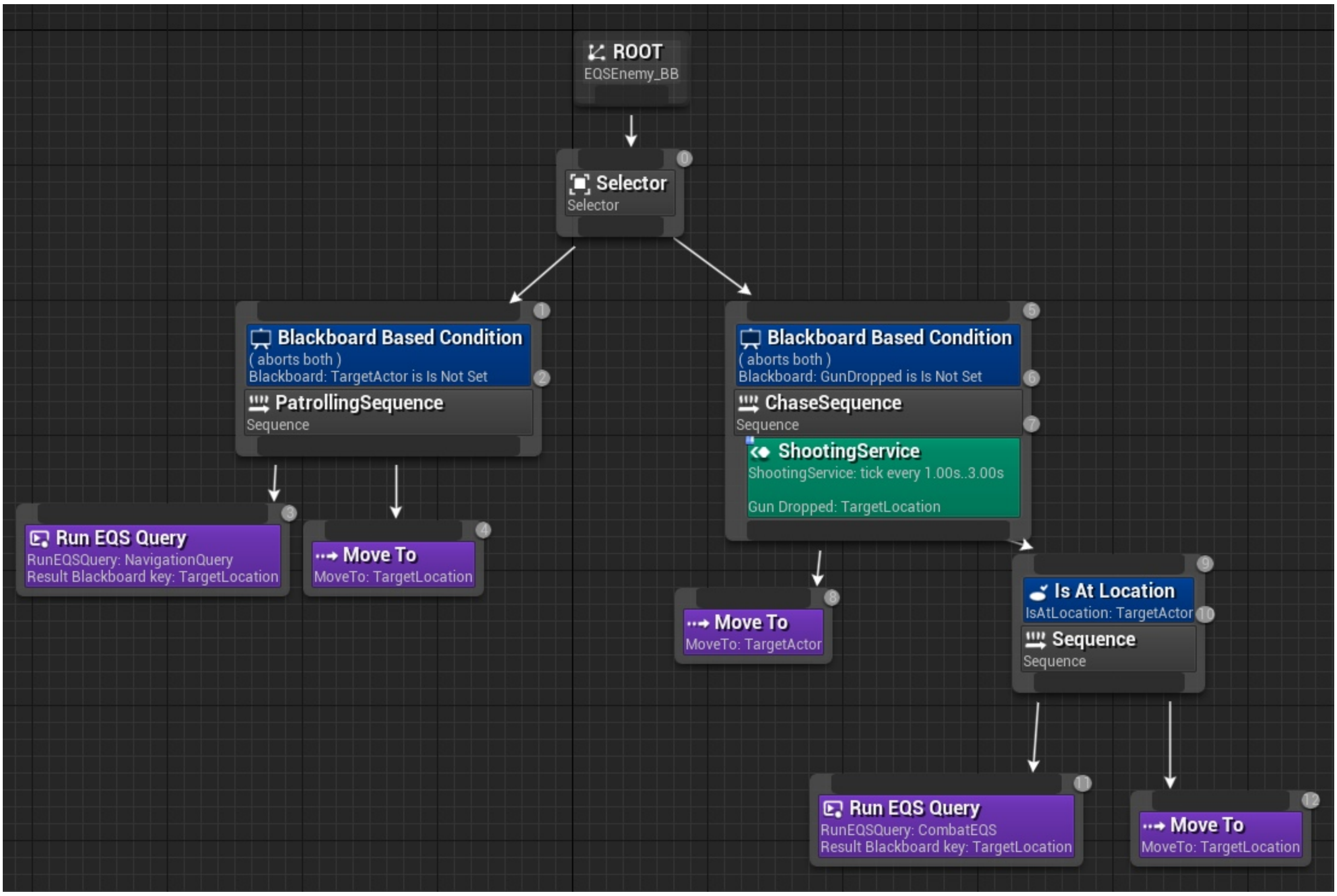


Enemy AI

EQS, Behaviour Trees, Blackboards, Controllers, Services, Contexts, NavMesh, Querys

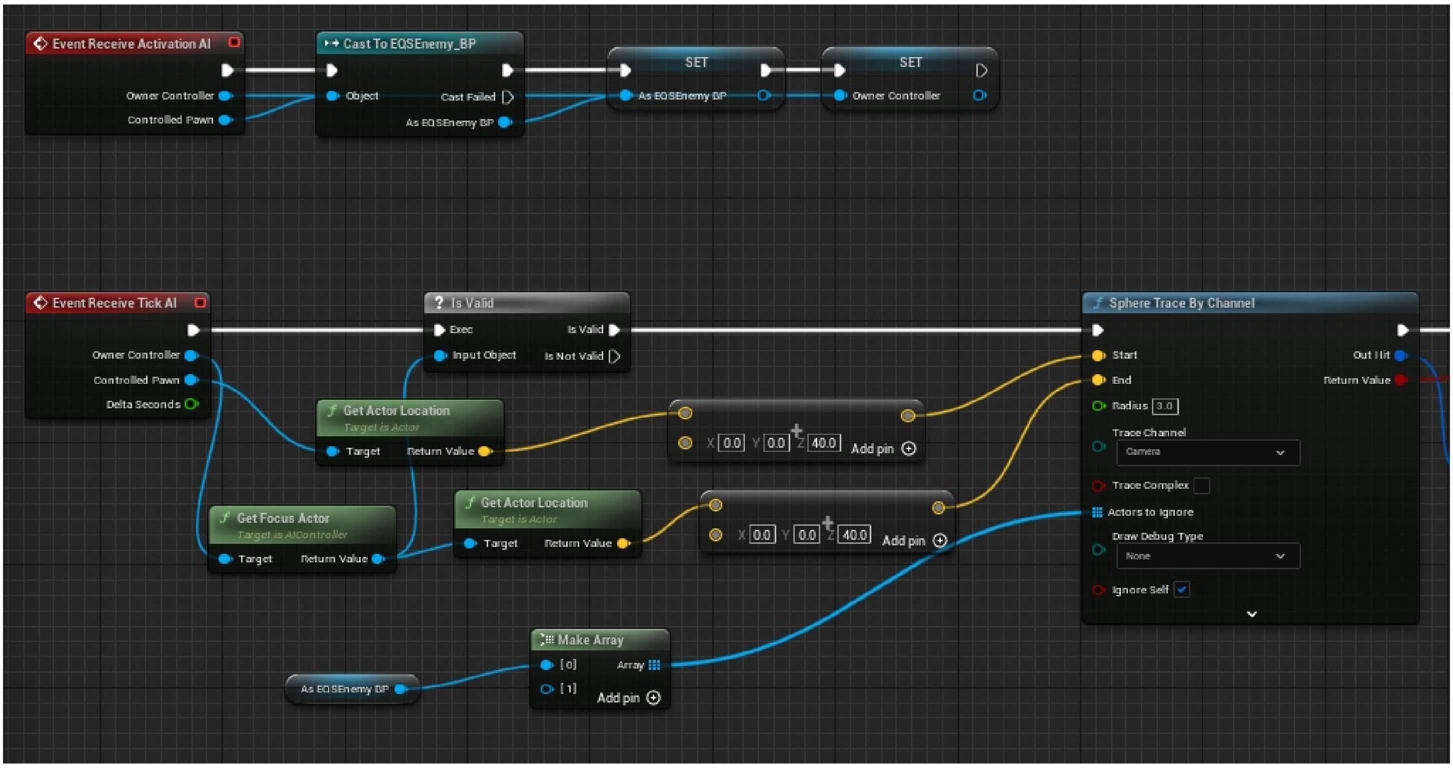
Blackboard and Shooting Service

Conditions, Keys, Services

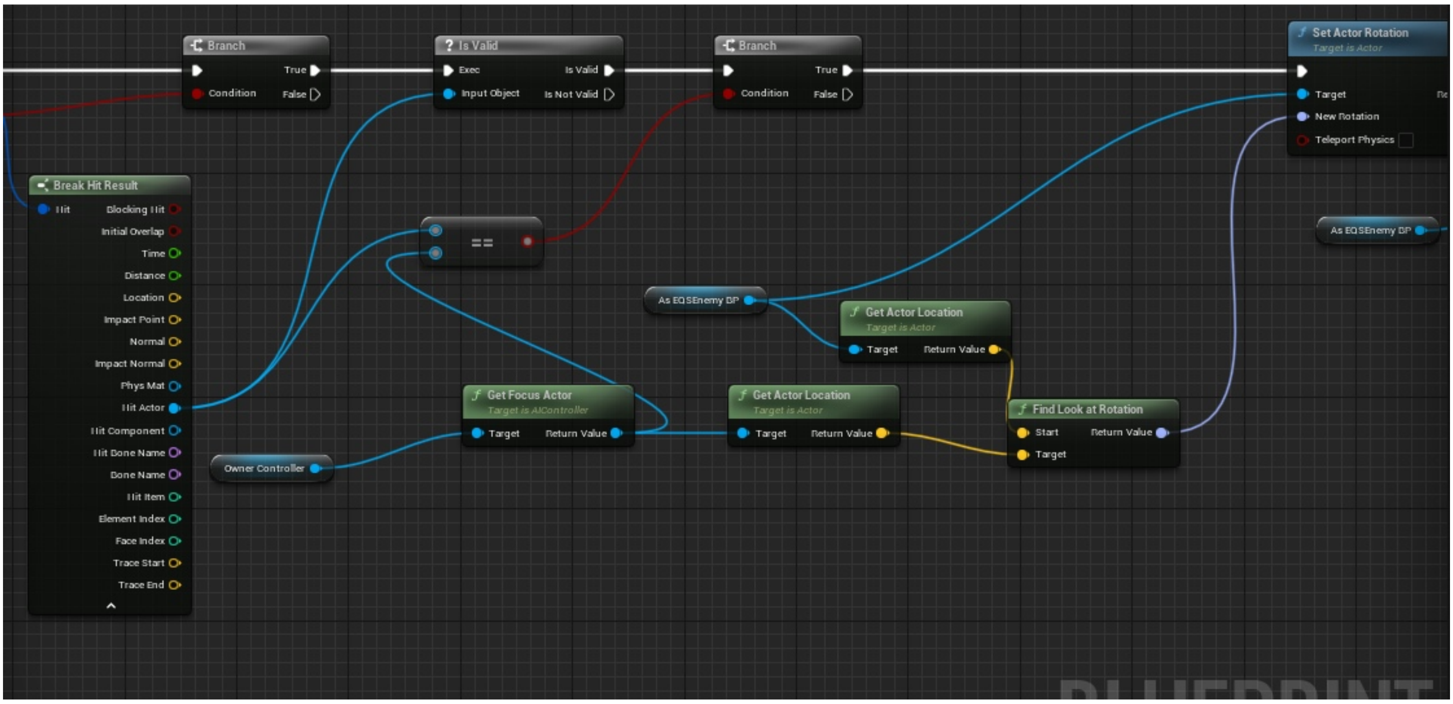

ROOT
EQSEnemy_BB
Selector
Selector
0
Blackboard Based Condition
(aborts both)
Blackboard: TargetActor is Is Not Set
1
2
PatrollingSequence
Sequence
Run EQS Query
RunEQSQuery: NavigationQuery
Result Blackboard key: TargetLocation
3
Move To
MoveTo: TargetLocation
4
Blackboard Based Condition
(aborts both)
Blackboard: GunDropped is Is Not Set
5
6
ChaseSequence
Sequence
7
ShootingService
ShootingService: tick every 1.00s..3.00s
Gun Dropped: TargetLocation
Move To
MoveTo: TargetActor
8
Is At Location
IsAtLocation: TargetActor
9
10
Sequence
Sequence
Run EQS Query
RunEQSQuery: CombatEQS
Result Blackboard key: TargetLocation
11
Move To
MoveTo: TargetLocation
12

The conditions that I have used is to check if the enemies can see the player, set the shooting service, and also the gun dropped condition. The gun dropped condition allows the enemies to carry out their standard patrolling sequence all the time if they have no gun, rather than following the player in any way.

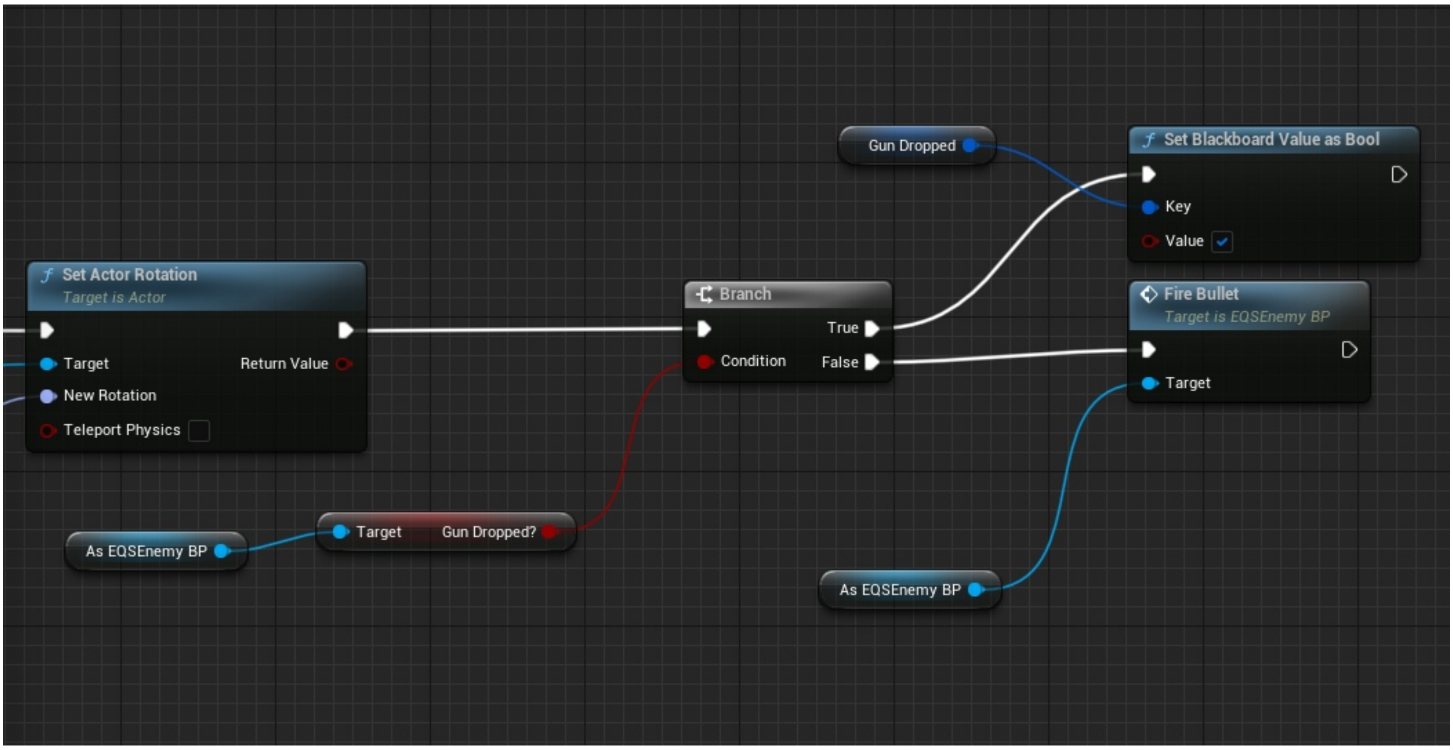
As long as their branch is being executed, services attach to Composite or Task nodes and run at the frequency they have set. These are frequently used to update the Blackboard and perform checks. Other Behaviour Tree systems use these in place of conventional Parallel nodes. These are what I have utilised to run my enemy shooting.



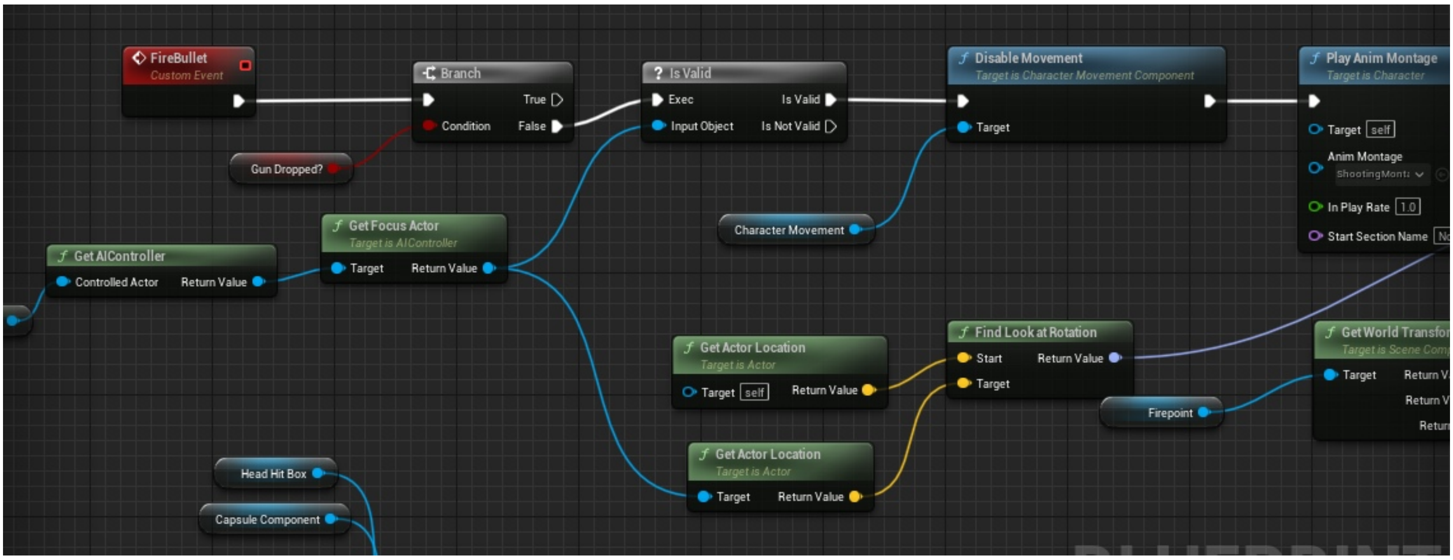
The shooting service first sets references to the enemy and controller, this is to cut down on the taxing performance hit of using the "cast to" node, which be running frequently because it is in a service. The service runs on a slow custom tick of 4 seconds that fires a sphere trace by channel, to see if it can see the focus actor (set in the AI controller). The sphere trace by channel is used, rather than line by channel, so that players can hide from enemies in a more gameplay focused way; for example if the player hides around a corner and peaks around it, the enemy should be able to see them (if using a line trace they would not be able to see them).

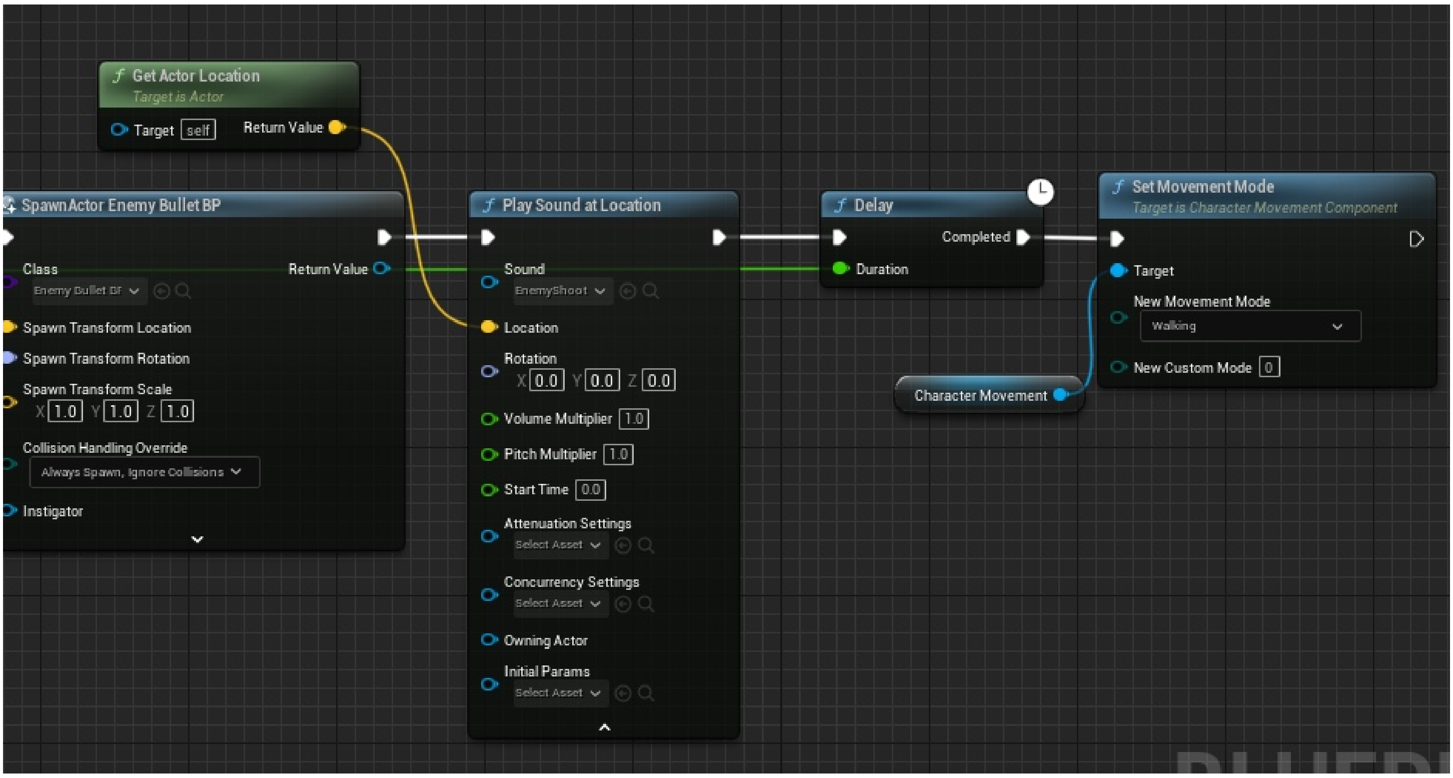


On hit, it checks if it it the focus actor and then sets the rotation to the focus actor. This is so that the enemy looks at the enemy when they shoot. This then triggers the shooting custom event in the enemy bp.



Above demonstrates how a blackboard key is set. It is setting the gun dropped key so that the AI knows what to do when it is not holding a weapon.

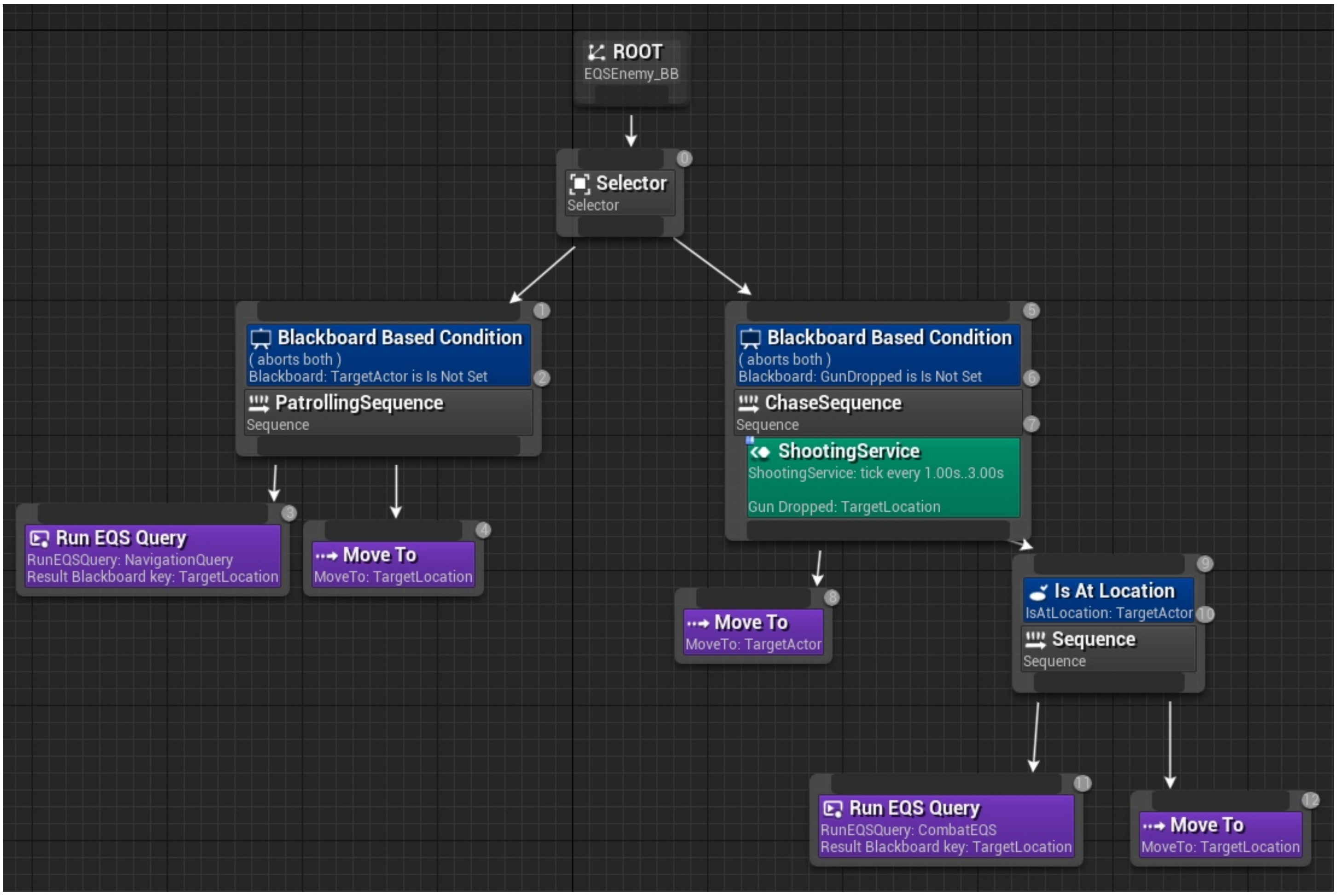




These are what occurs when the custom event is triggered. See the "Enemy Shooting Animation" section for a breakdown of the animation. After the animation occurs an enemy bullet is spawn (the same blueprints code as the player bullet but needed to be separate for overlap events) and spawns a sound effect. Then returns to its default movement mode.

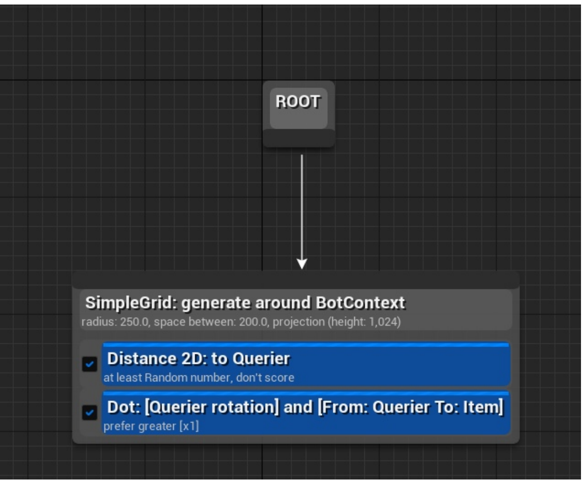
Environmental Query System

EQS



ROOT
EQSEnemy_BB
0
Selector
Selector
1
Blackboard Based Condition
(aborts both)
Blackboard: TargetActor is Is Not Set
2
PatrollingSequence
Sequence
3
Run EQS Query
RunEQSQuery: NavigationQuery
Result Blackboard key: TargetLocation
4
Move To
MoveTo: TargetLocation
5
Blackboard Based Condition
(aborts both)
Blackboard: GunDropped is Is Not Set
6
ChaseSequence
Sequence
7
ShootingService
ShootingService: tick every 1.00s..3.00s
Gun Dropped: TargetLocation
8
Move To
MoveTo: TargetActor
9
Is At Location
IsAtLocation: TargetActor
10
Sequence
Sequence
11
Run EQS Query
RunEQSQuery: CombatEQS
Result Blackboard key: TargetLocation
12
Move To
MoveTo: TargetLocation

Utilised to gather environmental data, the system can ask questions about data using a variety of user-defined tests and a generator, returning the best item that matches the questions being posed. Examples of possible uses for EQS include determining which enemy poses the greatest threat, enemy pathing, and determining the player's line of sight/enemies line of sight with player. In my game I have used EQS in my AI enemies to make the much smarter and interact with the environment in a more interesting, dynamic way. I used EQS over just basic AI behaviour trees because I tried the normal way and the AI struggled to interact with both floors well and were not smart enough to interact with the player in an interesting way.
https://www.youtube.com/playlist?list=PL4G2bSPE_8ukuajpXPlAE47Yez7EAyKMu) This playlist by Ryan Laley, about a 1v1 bot, is what I used for the bases of my AI, however, I have expanded on his concept and customised it so the AI works how I think is better suited to my environment and gameplay.



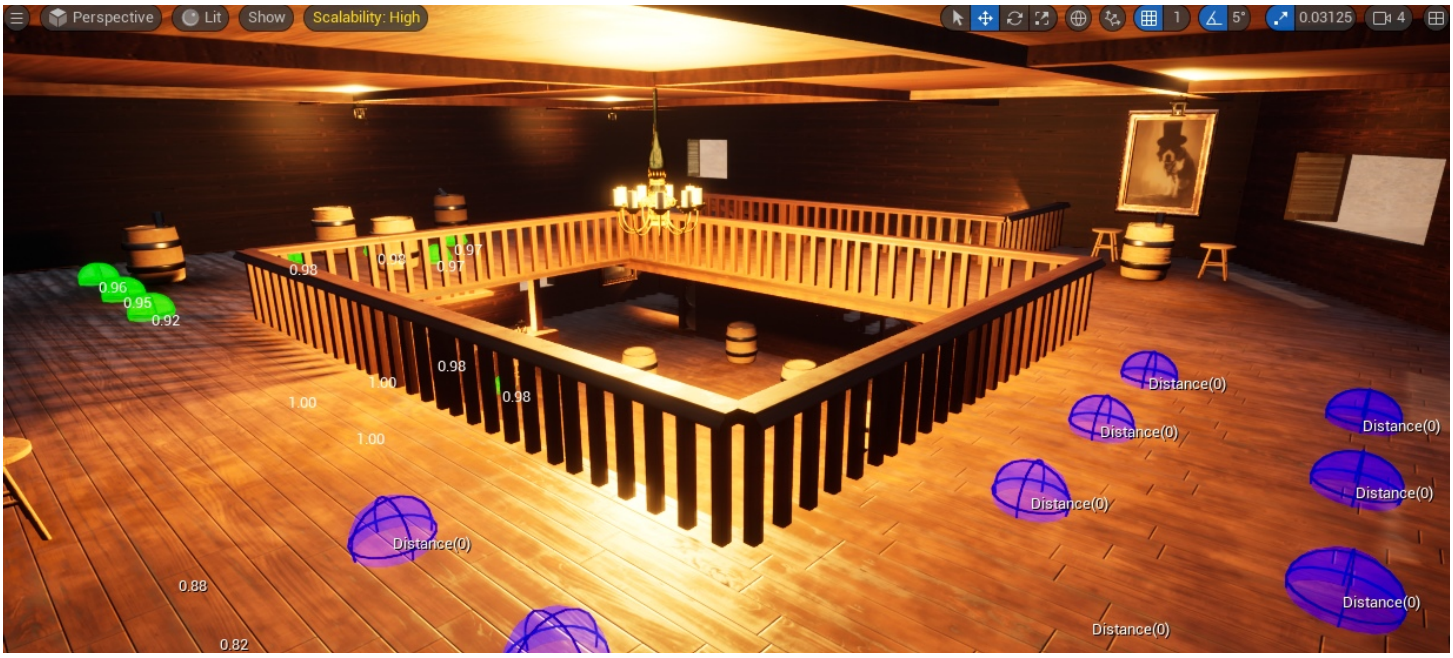
ROOT

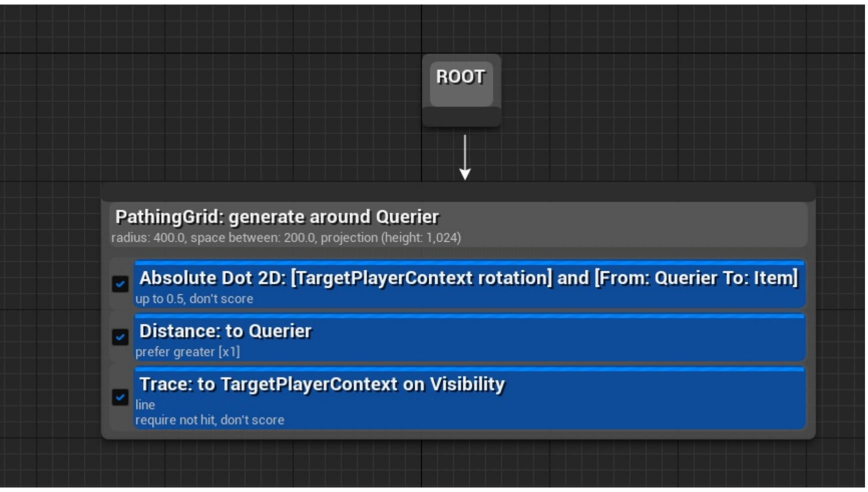
SimpleGrid: generate around BotContext
radius: 250.0, space between: 200.0, projection (height: 1,024)

☑ Distance 2D: to Querier
at least Random number, don't score

☑ Dot: [Querier rotation] and [From: Querier To: Item]
prefer greater [x1]

The Patrolling Sequence branch is what determines the default state that the AI are in. In summary, they find a location in the NavMesh and run to it. However, the location that they choose is based on an EQS query that determines the best spot for them to run to. The context that they base it off is points of interest (blank bp with a billboard in it) that have been laid out around the map. These spots have been carefully placed to suit my environment because incorrect placement leads to strange interactions with the AI (see the bug report for the stairs issues, these are what caused that). The points of interest are gathered in a context and then have a set of dots generated around them. The amount of dots and spacing was tuned specifically for this environment too because if too spaced or too many there were, again, strange interactions with the AI. The parameters that the query checks is from the points of interest and the distance to AI, so it isnt too close to them (otherwise they would constantly be changing directions and would look stupid). The dot test is to check rotation. The AI shouldnt really make a full 180 degree turn, because it would look out of place, so it scores them 0 (meaning they cannot be picked as a target location). Based on these parameters a target location is chosen and the AI will move to one of the top 5% chosen, giving some variety in the AI movement.




ROOT

PathingGrid: generate around Querier
radius: 400.0, space between: 200.0, projection (height: 1,024)

☑ Absolute Dot 2D: [TargetPlayerContext rotation] and [From: Querier To: Item]
up to 0.5, don't score

☑ Distance: to Querier
prefer greater [x1]

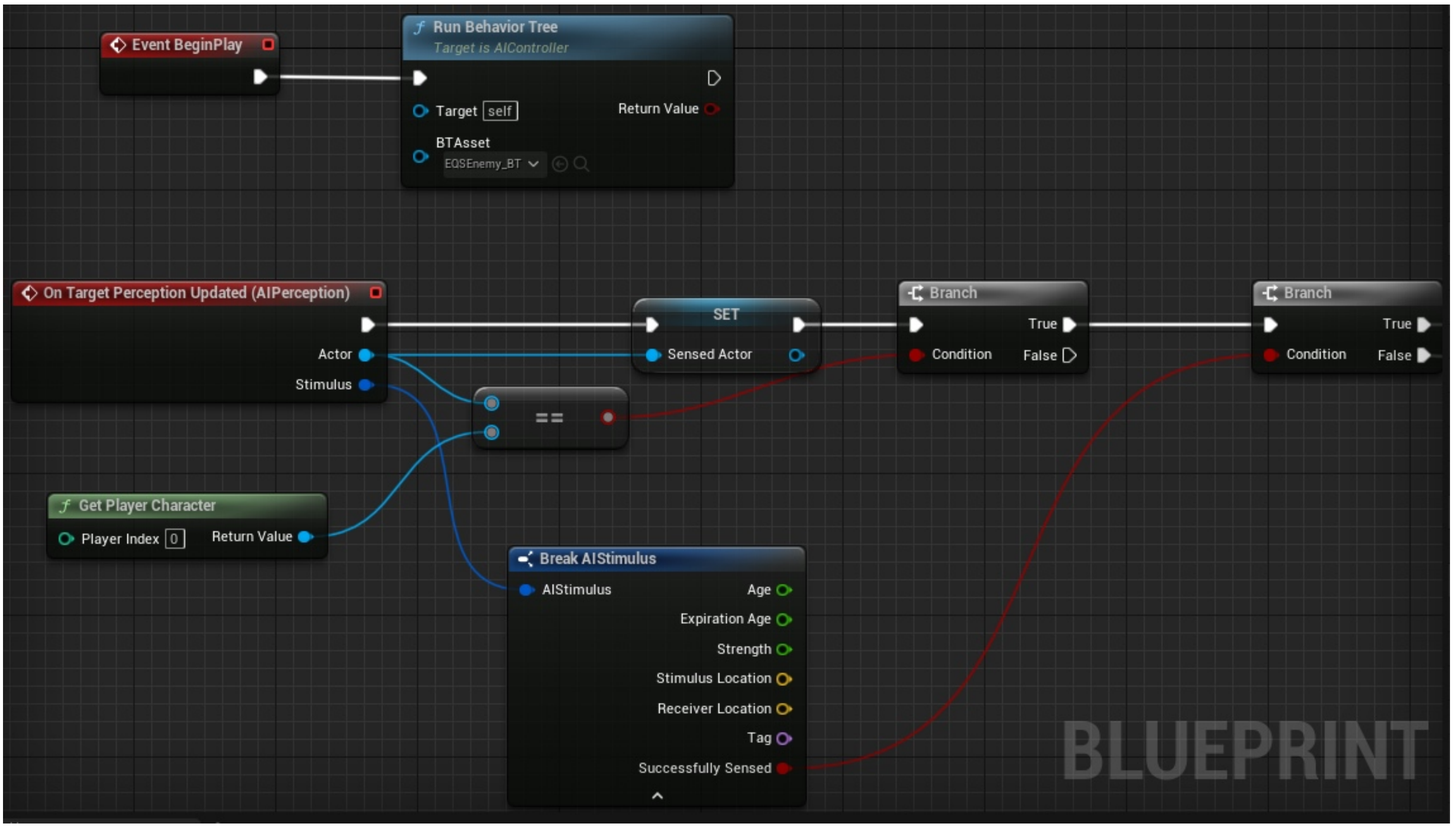
☑ Trace: to TargetPlayerContext on Visibility
line
require not hit, don't score

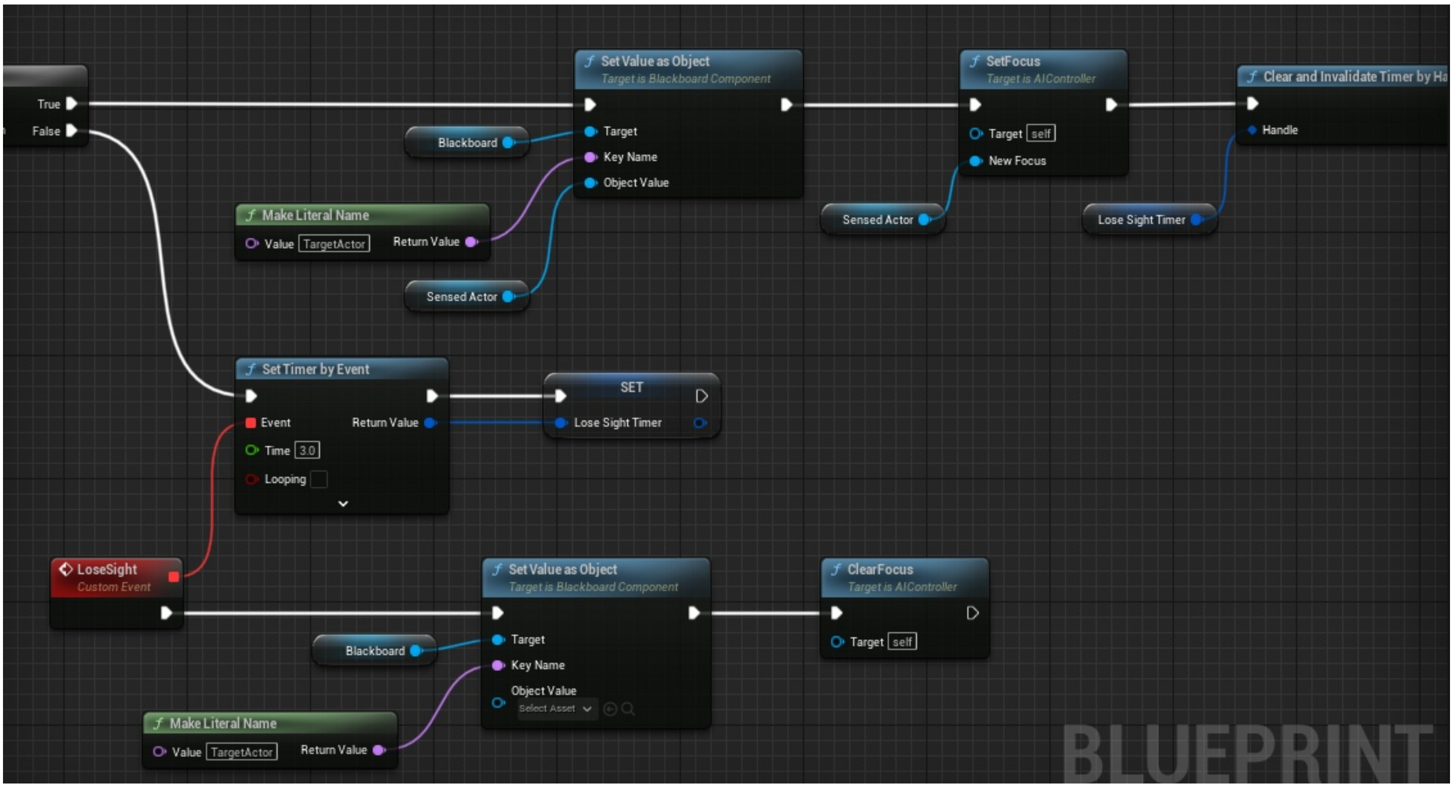
The chase sequence branch is what happens when the enemies can see the player character (who set as the focus actor). The shooting service is broken down in the section above. The first thing that the branch does is move the enemy towards the player character, to try not to break line of sight and engage in combat. Once towards the player character it runs an EQS query that determines where it should move next, all whilst running the shooting service in the background. The query checks: the rotation of the of the player character and tries to avoid it, the distance to the player character (to make sure it doesnt get too close) and the visibility of the player character to try and keep line of sight on the player. After running this query, it moves the enemy to the best result.

AI Controller

Sense Perception

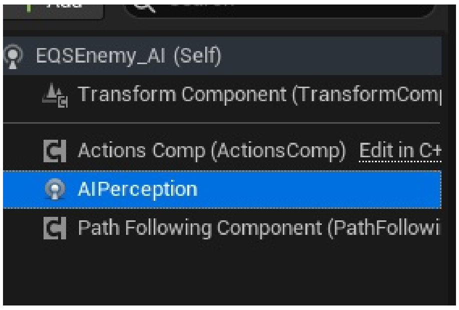
The AI controller is the brain of the AI and is used to run the behaviour tree. To set up the controller you must connect all the parts together. So the AI class default controller has to be set to this specific controller, the blackboard needs to be connected to the behaviour tree and AI needs to run the correct behaviour tree.



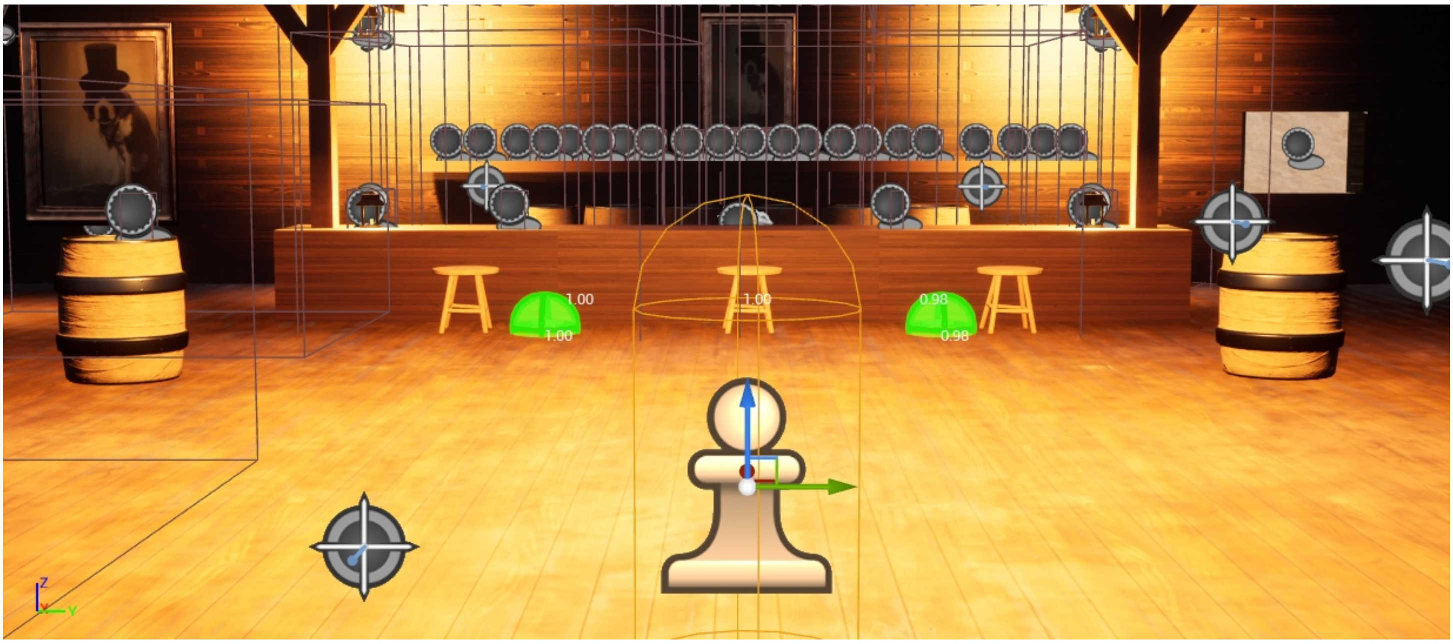


EQS Testing

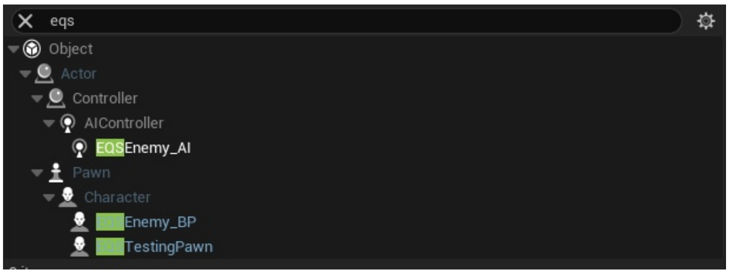
Testing Pawn


EQSEnemy_AI (Self)
Transform Component (TransformCom
Actions Comp (ActionsComp) Edit in C+
AIPerception
Path Following Component (PathFollowi

The controller has an AI perception component in it. This allows the AI to do things such as hear, see, sense damage and other sense related stuff. Sight and damage are enabled on my controller and the code for setting the focus actor relies on this. When the AI senses the player character it is set as a variable and then runs into a branch to check if it can still sense it. If it can sense it still then the blackboard key "target actor" is set and the focus actor is set. If it cannot sense the player character anymore then a timer is started that, when it runs out, clears focus and resets the blackboard key. This timer is cleared once it can sense the player character again.



Unreal has a specific blueprint class for testing EQS. This class is "EQSTestingPawn" and is a character with a capsule collision and can be used to view what an EQS query will result in. This is vital for testing how AI will respond to queries and whether things need to be adjusted. Without this, manual trial and error would have to occur which would be very inefficient.


eqs
Object
Actor
Controller
AIController
EQSEnemy_AI
Pawn
Character
Enemy_BP
TestingPawn