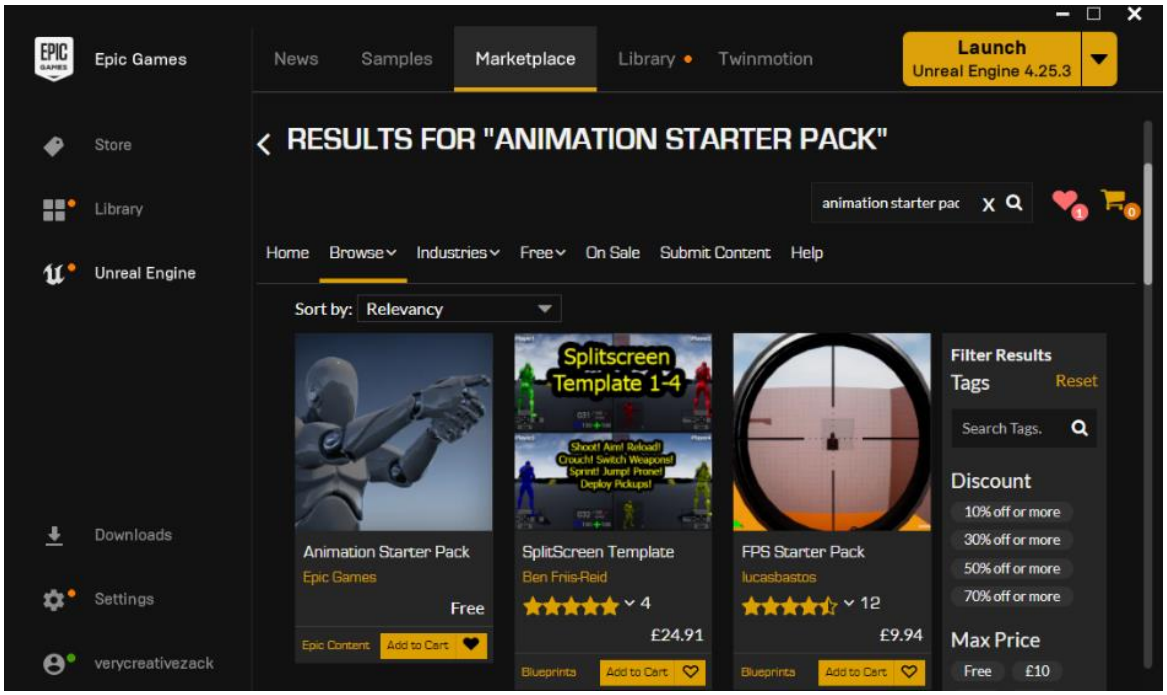
Bradzorg (documentation of my process)

Unreal engine packs:

Animation starter pack



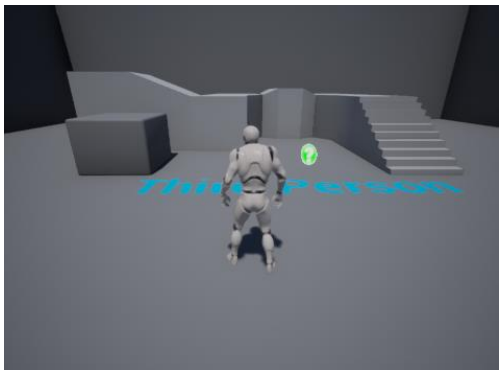
Setting up the project:

The blueprint project template is a 3rd person template.

Building a Third Person Shooter Character

Setting up the camera:

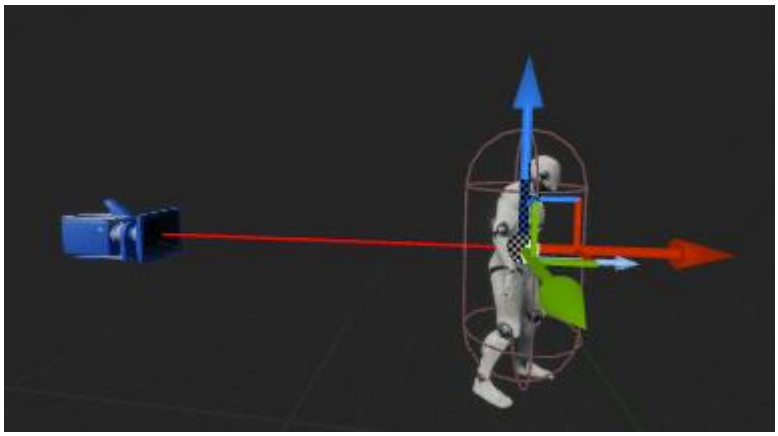
By default, the 3rd person camera blueprint is set-up for an adventure or mmo type game.



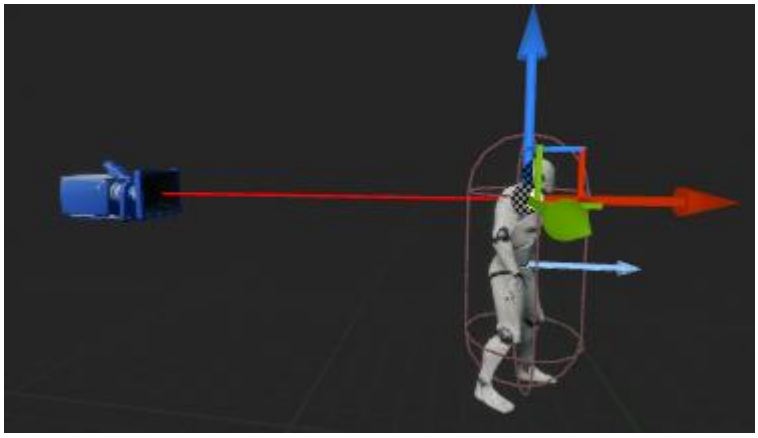
I need a 3rd person shooter camera set-up. Having this base will make development smoother and as a result the product also.

Moving the camera. If I just moved the camera manually it would clip through walls and objects.

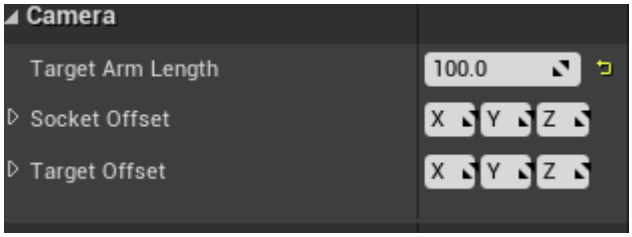
So, in order to prevent this from happening I needed to keep the **camera boom** in the centre of my characters mesh because this is the point that I want the camera to rotate around.

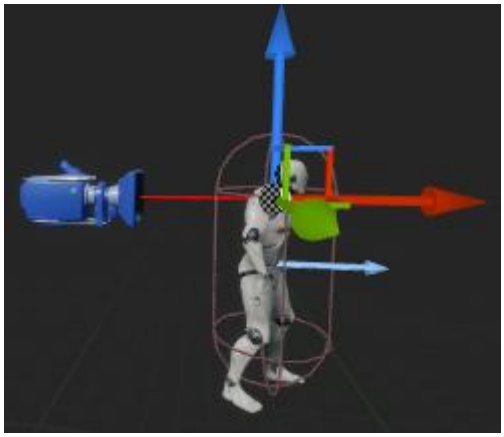


However, I needed to rotate the camera around the upper portion of my character. So, I have brought the camera up to the centre of my characters neck (58.49° Z-axis).

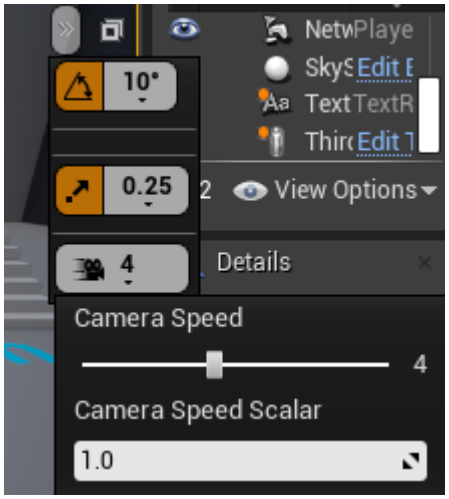


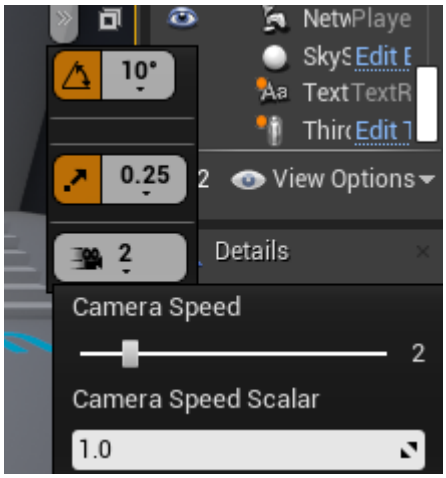
I then changed the **target arm length** variable to 100. I did this to bring the camera closer to my character.





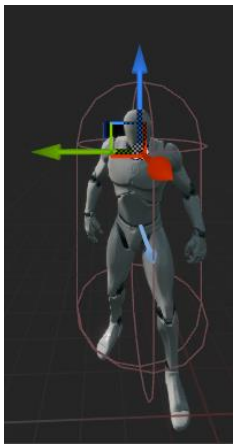
At this point the editor camera speed was proving to be way too fast for getting screen shots and over all traversing unreal engine, so I changed the camera speed from four to two.





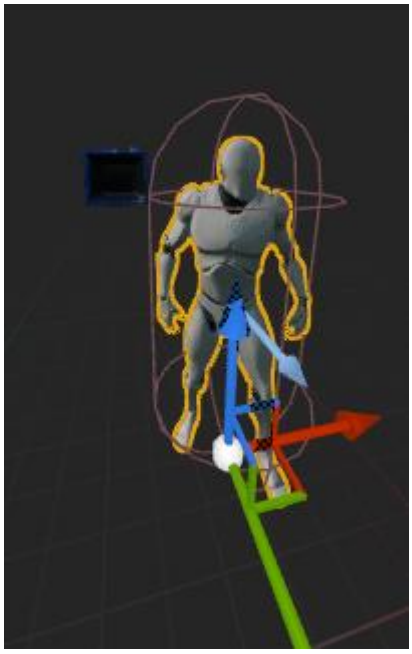
This was a well needed adjustment as now traversing unreal engine was much simpler.

I now needed to position the camera over the characters shoulder as at this point the character was in the way.



So, instead of manually moving the camera and causing the clipping issue again I changed the socket offset Y variable to 55.0

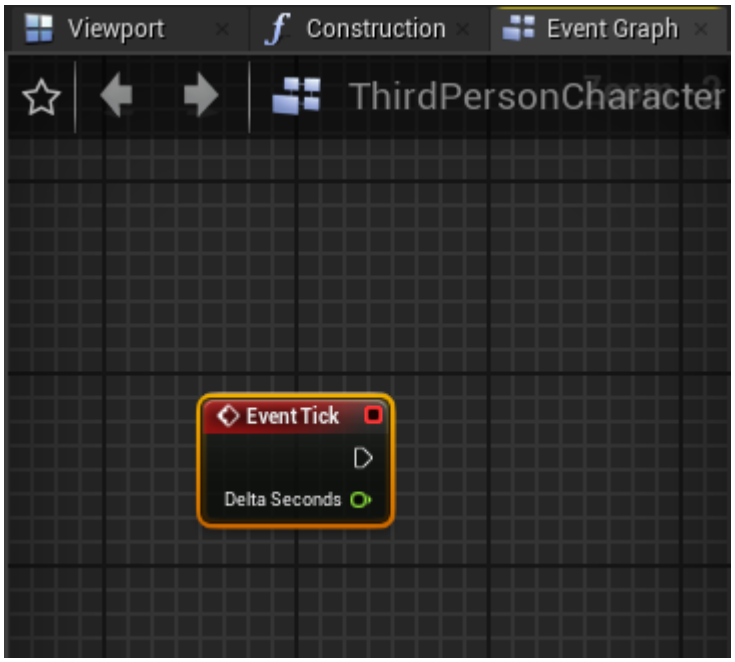
This positioned the camera around the character.



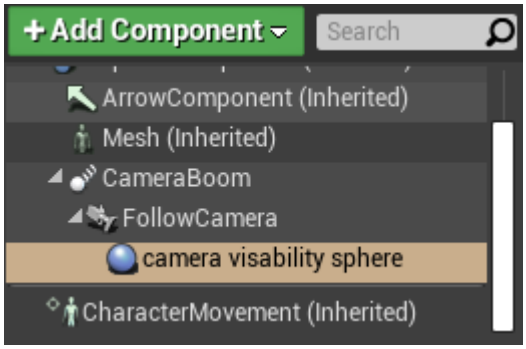
This has now prevented the camera from being able to clip through objects as it will get pushed back in instead. The camera now will interact with the environment as it should.

However, this has now caused a new problem. The camera will now clip through the character showing the inside of the model.

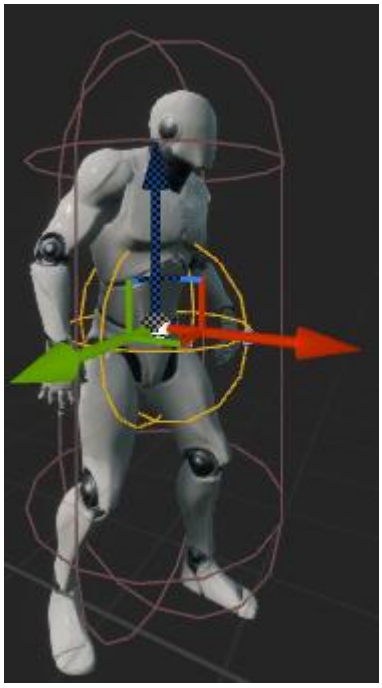
To fix this, if the camera entered the characters mesh, I hid the character and as the camera exited the mesh, I made the character reappear. In order to do the I need to use an **event tick** in the vent graph. Event ticks should be used as little as possible as they constantly run. Other event ticks are things such as input and turn rate. These need to be event ticks because the player is going to be constantly looking around the environment.



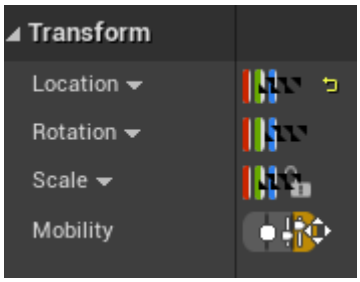
I then added a collision sphere **component.** A collision shape (box, capsule or sphere) is added to detail if this object collides or does not collide with something then something does or does not happen. This is going on the camera, so I named it ‘camera visibility sphere’. It is added to ‘follow camera’ as it needs to stay with the camera.



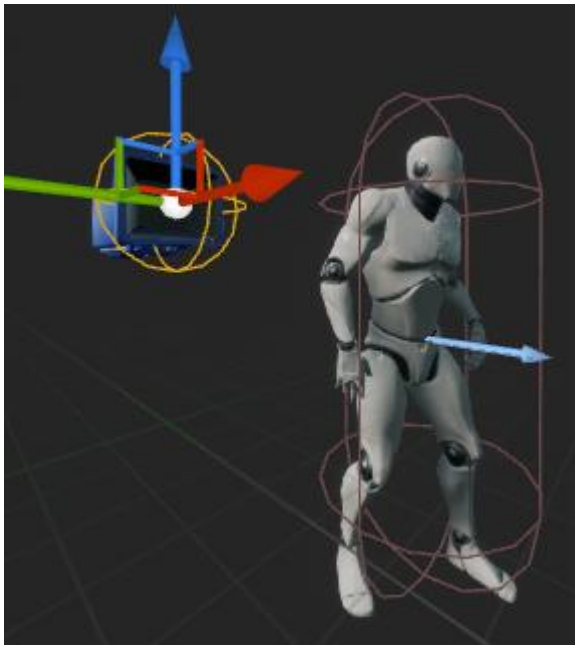
However, it is added to our character by default.



The easiest way to add this to the camera instead of the character is to go to transform and reset the ‘location’ to default by using the yellow arrow symbol.



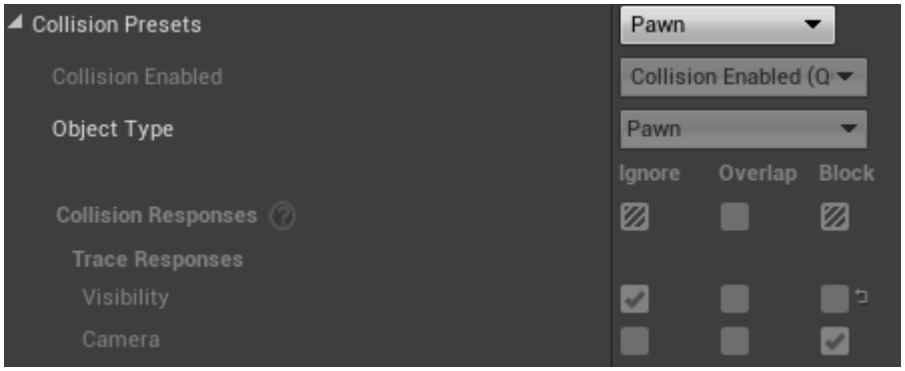
The sphere will then be visible on the camera.



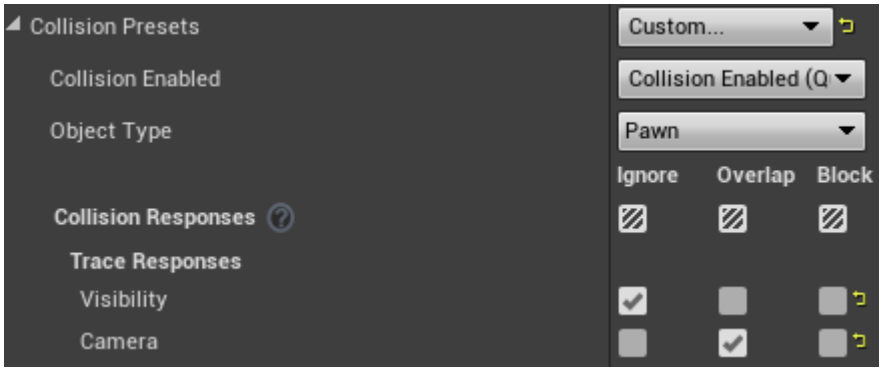
At this point the sphere was too big and as a result it could cause the character to disappear when the camera is an acceptable distance away from the character. So, to fix this I went to shape and changed the sphere radius variable to 12. I also moved the collision sphere back into the camera lens to ensure a fix.

I then needed to ensure that the collision sphere would not have any issues with the capsule over the character. To do this I went to collision settings and made sure the collision presets were set to ‘overlapalldynamic’. I did this because I only need the sphere to read events not to do any simulating in the game or interfering with the character at all.

I now needed to prevent the camera from bouncing off the character capsule. To do this I selected the character capsule and changed the collision preset from ‘pawn’.

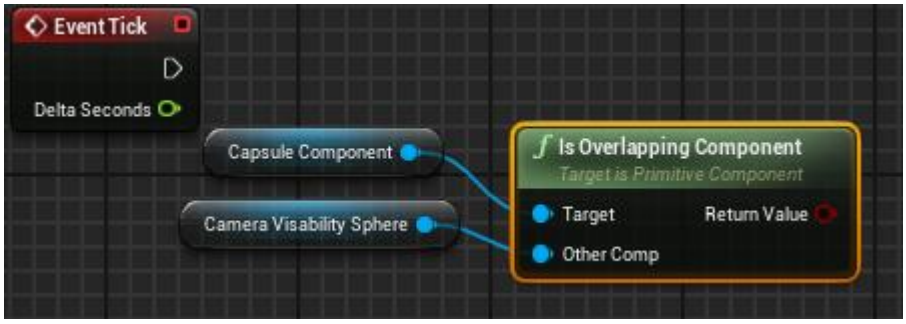


To ‘custom’ to allow me to change the camera from ‘blocked’ to ‘overlap’.



The next step was to write the code to drag and drop the capsule component and the camera visibility sphere from the component tab into the event graph.

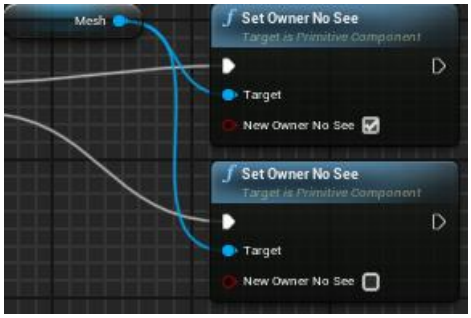
I then needed to add an event to do this I dragged from ‘capsule component’ and added a ‘Is overlapping component’ event. I also attached ‘camera visibility sphere’ to the ‘overlapping’ event.



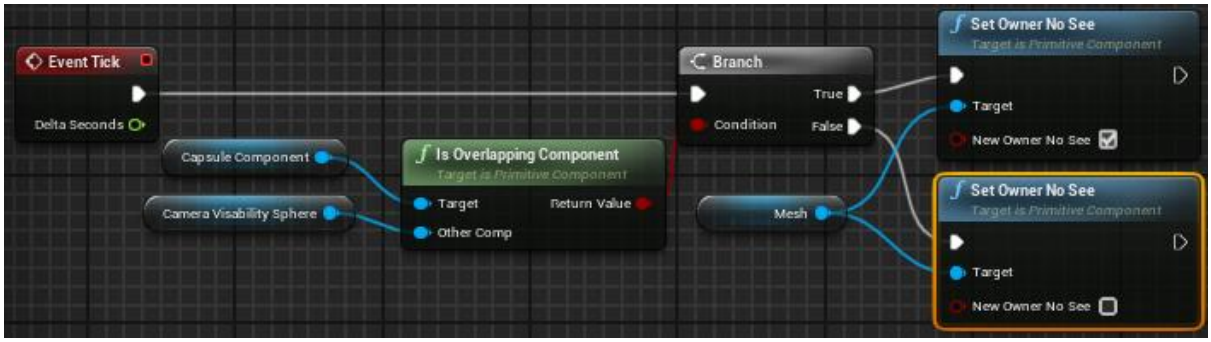
Next thing needed was a ‘Branch’. I connected the ‘event tick’ and the ‘is overlapping component’ to this ‘Branch’. This is because it will tell unreal that it should constantly be watching for these two components overlapping.

Now the mesh needed to be dragged in with a ‘set owner no see’ event. This is because the mesh is what is getting affected by this line of code.

The ‘set owner no see’ was then copied and pasted as I needed two of these with one ticked if the event is considered true to hide the character (this is the first one added) and one unticked if false so make the character reappear (the pasted one). The mesh was then connected to both.



The whole line of code looks like this.



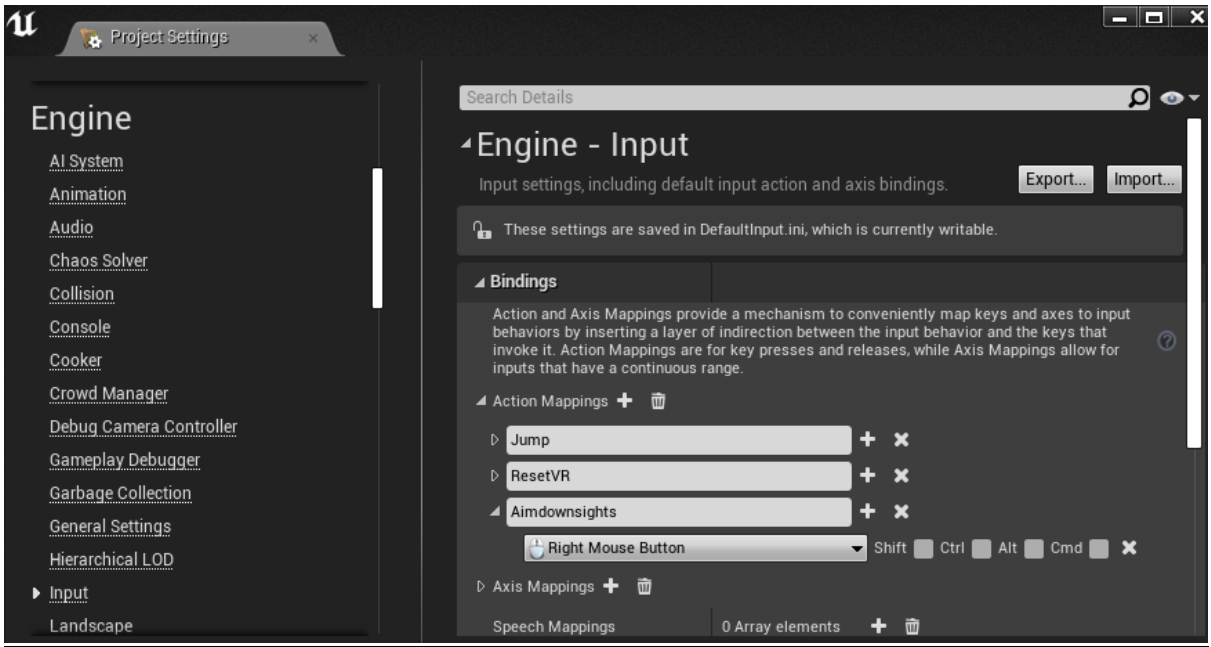
To get to this point I have used my secondary research videos 'Third person shooter build introduction' and 'TPS Build Part 1 – Camera Setup' (K,Dail 2015)

Reflect on the section!

Setting up the controls for aiming:

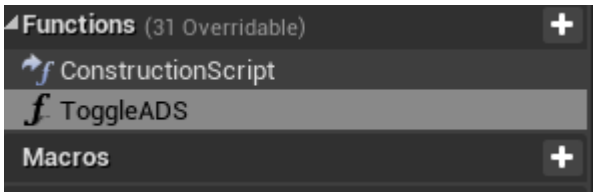
This next section shows how I set up the controls for the player to aim down sights.

The first thing that I had to do was add an action map for aiming down sights this is done in the 'project settings' under 'engine' then 'input'. The new action map was called 'Aimdownsights' because this is what it is in relation to. The input to aim down sights is the right mouse button.

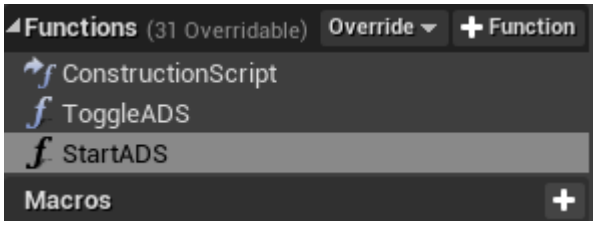


I was then able to add this to the events graph.

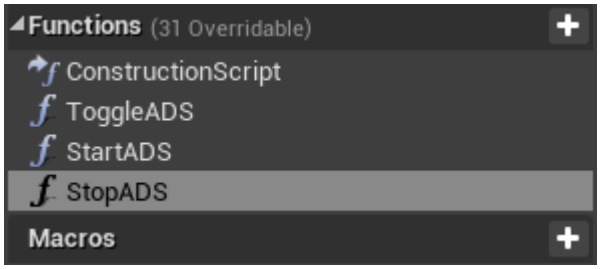
The next step was to set up a system that must run though a server as its primary. This means one person is hosting the game from his end with people joining his server using his bandwidth and sending messages to his server to allow them to perform actions in the server environment. So, to do this I had to set up a couple of functions to be able to handle this. The first function added was to toggle clicking and unclicking the aiming system, this was named 'ToggleADS'.



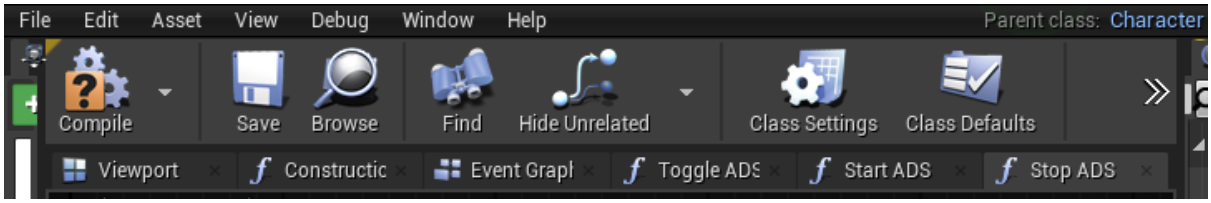
The second function added was named 'StartADS'.



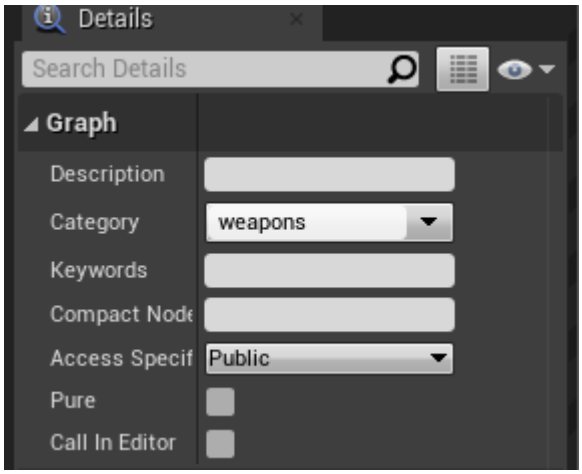
And lastly a function named 'StopADS'.



These were now visible in the bar up top.

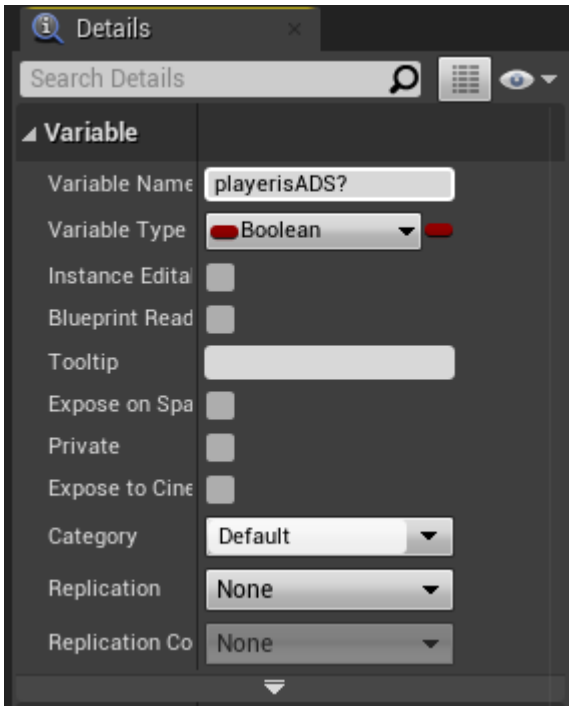


These then needed to be added to a category for handling the weapons. To do this I needed to add a new category called ‘weapons’ and add it to each function.

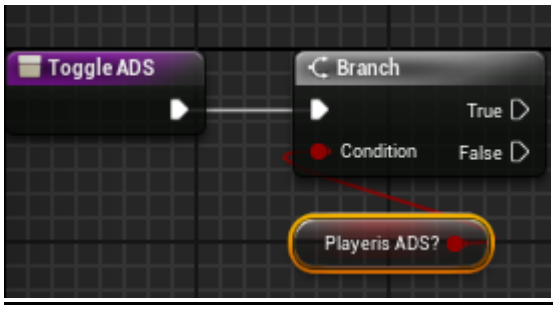


Now it was back to the events graph to ‘ToggleADS’ and connect the input to it.

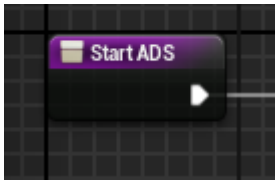
Now I needed to set up a Boolean, this is a variable that is true or false. To do this I added a new variable and named it ‘playerisADS?’ because this is what the game will be checking for.



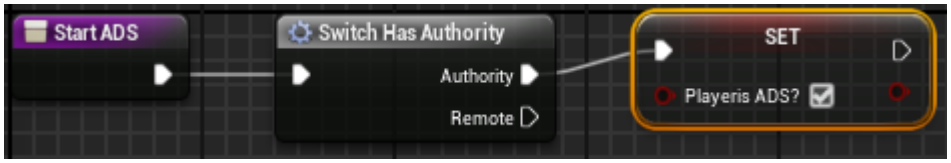
Now I needed to tell unreal to do something when the aiming button is pressed or realised. To do this a added a flip system which tells unreal if true do, if false don’t do.



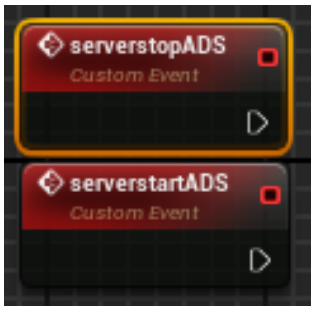
This is the base for this line of code. I now needed to tell unreal what to do when aim down sights is pressed and realised. I first needed to tell unreal to find out who is the server (host) and who is the client (guest). This is done in the start ADS function.



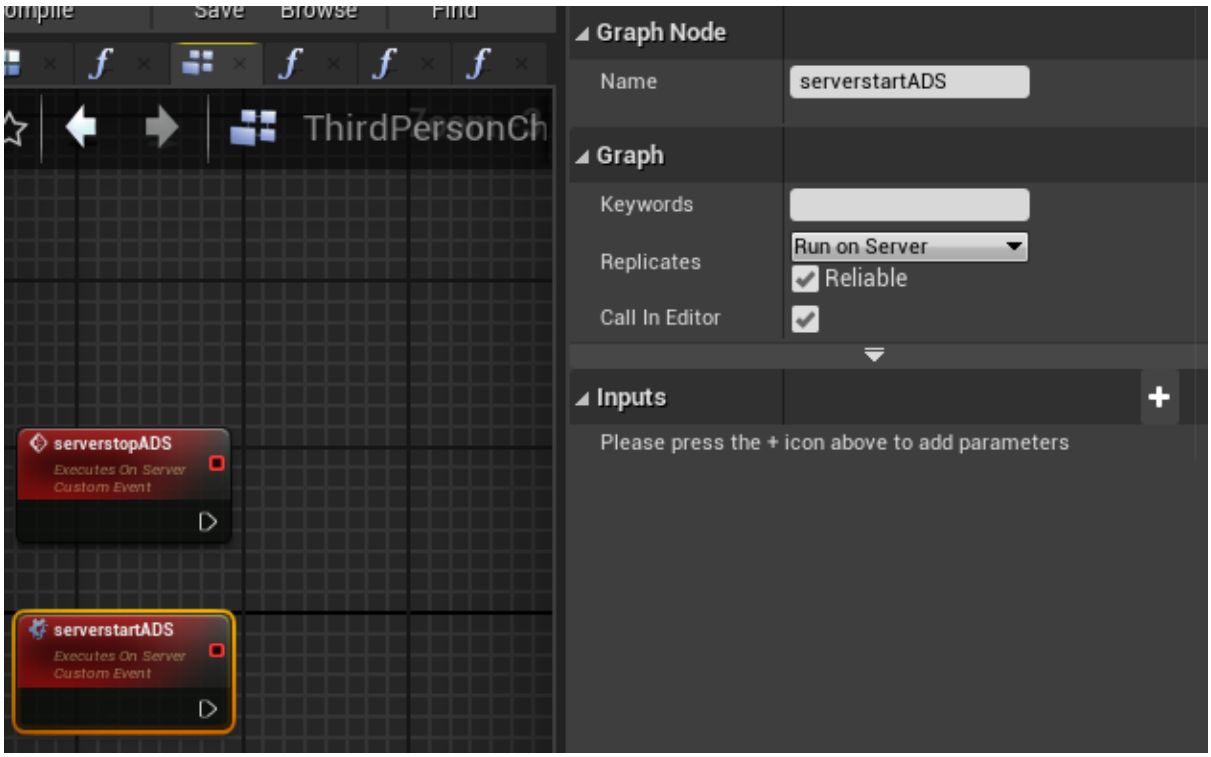
Now, ‘switch has authority’ needs to automatically set the event as true if input is made by the server.



However, if the client makes the input the client needs to talk to the server to pass the event on to tell unreal that the player has made this input. To do this I needed to make two custom events in the events graph, one named ‘serverstartADS’ and another called ‘serverstopADS’.



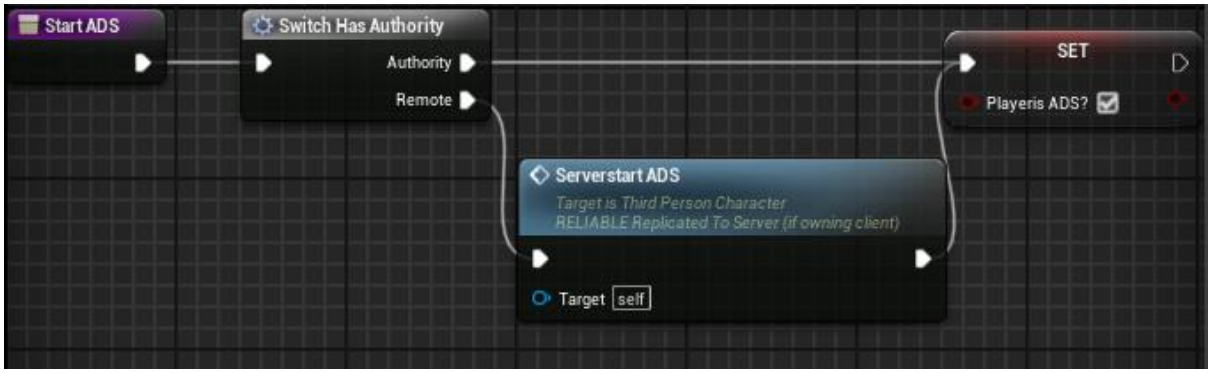
These both needed to be set as ‘run on server’, ‘reliable’ and ‘call in editor’. Call in editor made it so I could test it in the editor.



And from there I needed unreal to be able to toggle something.

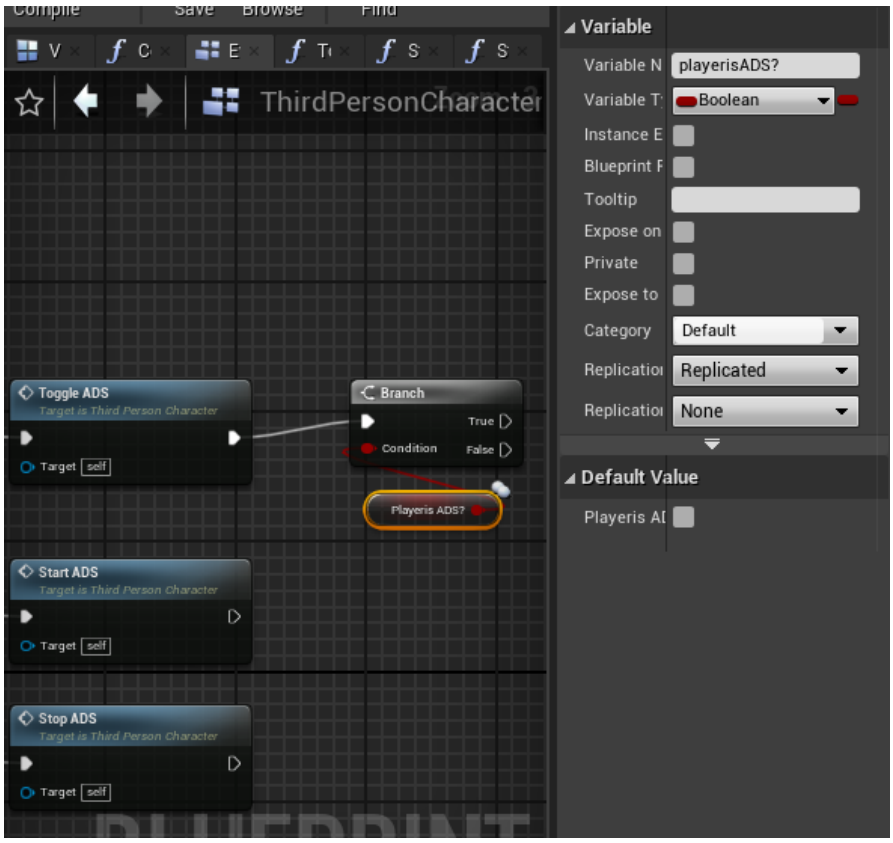
To do this I needed to add ‘startADS’ and ‘stopADS’ as we have now told unreal to send a message to the server. Which is represented by the connection line between functions/variables.

This will now tell the server that a client has made an input to aim down sights and give authority for them to set the ADS event to true.

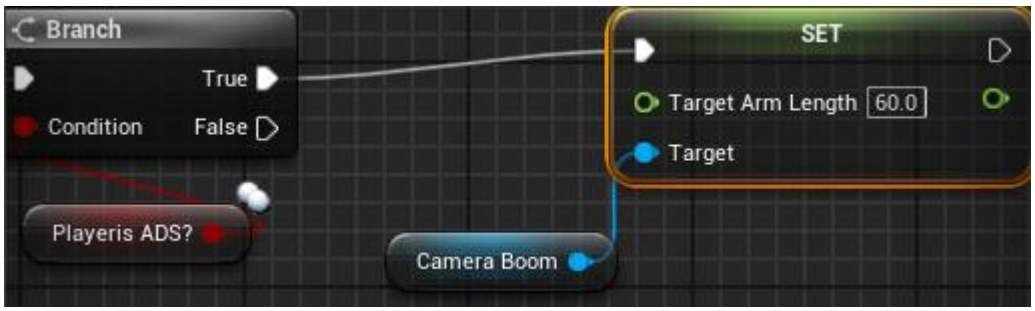


I then made the same code for the stop event but for stopADS instead of startADS and set the event to false.

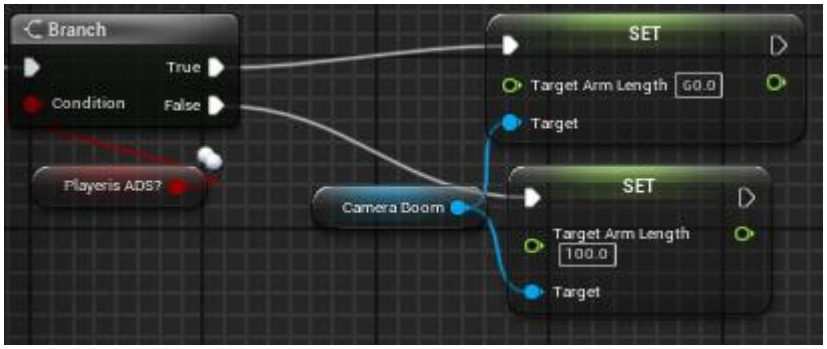
Now that ‘ToggleADS’ was passed through these two events that were just made I needed to set up a branch to handle the flip situation for the camera to be able to adjust the camera boom, field of view and the character rotation. So, I connected ‘playerADS’ to the branch. For the Boolean to be read on multiplayer it must be set to ‘replicated’.



Now that this is being read correctly. I needed unreal to take the camera and put it up closer to the character. The easiest way to do this was to drag the camera boom into the events graph and set the target arm length to 60 by adding a ‘set target arm length’ event in the graph and input the needed arm length in there and connect this to the branch.



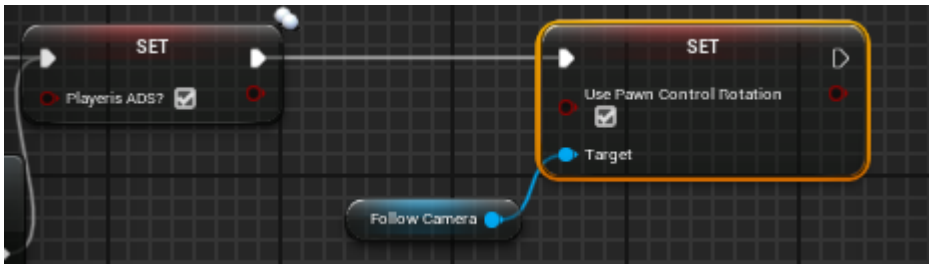
now I needed to add another ‘set target length’ with the default length and attach it to the false branch.



This was the result up to this point. This is not aiming down sights

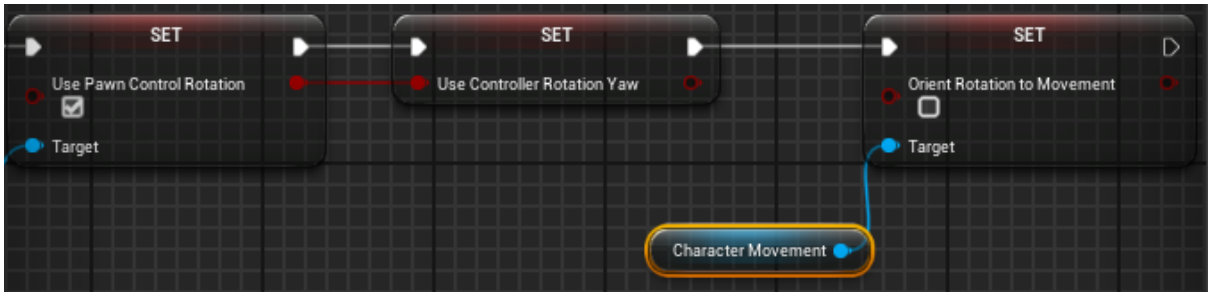
This is aiming down sights.

I now needed to set the character to rotate with the camera while the player is aiming down sights. To do this more code needed to be added for after the player toggle's ADS. Firstly, it needs to be pasted through both the server and the client. So, to do this I needed to add code into the 'StartADS' and 'StopADS'. To do this I added 'followcamera' to the 'StartADS' function and set the 'pawn' (character) to a 'control rotation' and to 'true'.

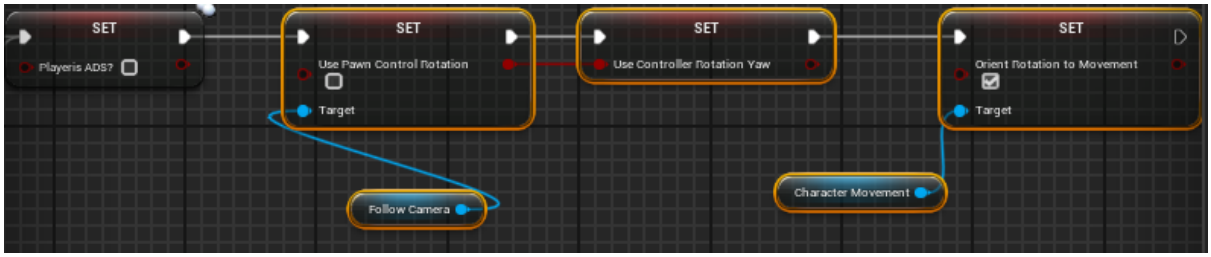


I then needed this rotation to use a 'yawn' (side to side) rotation. To do this I just needed to pass the Boolean along.

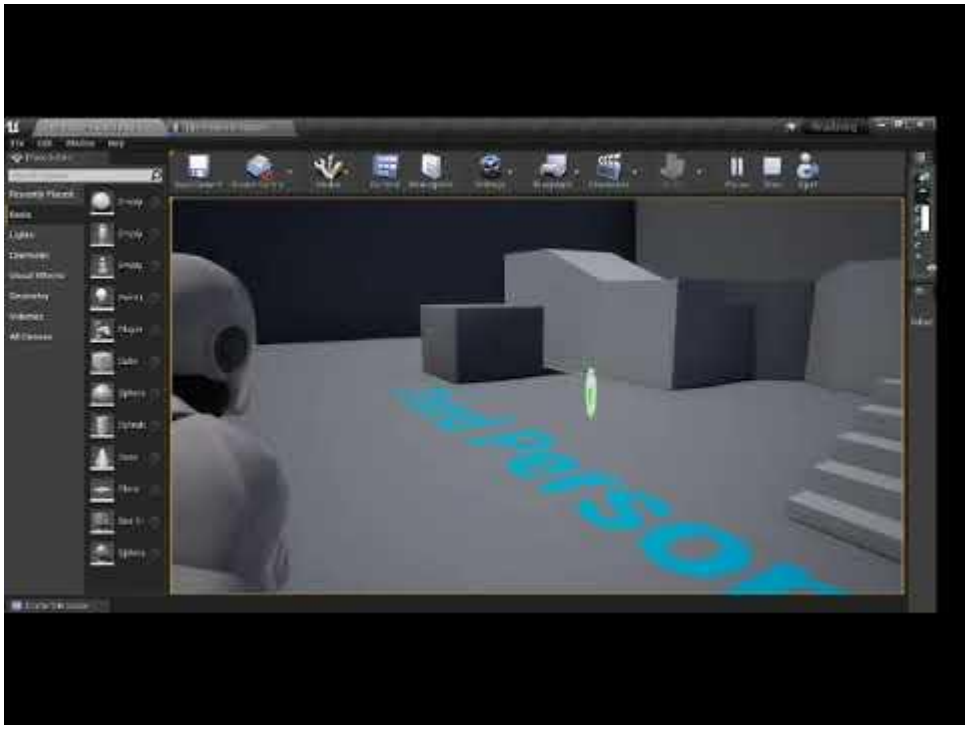
I now needed to manipulate the movement to where the character if facing by adding 'charactermovement' to the 'StartADS' function and orient the character to a direction. This is left as 'false' because the default for the 'charactermovement' is 'true'.



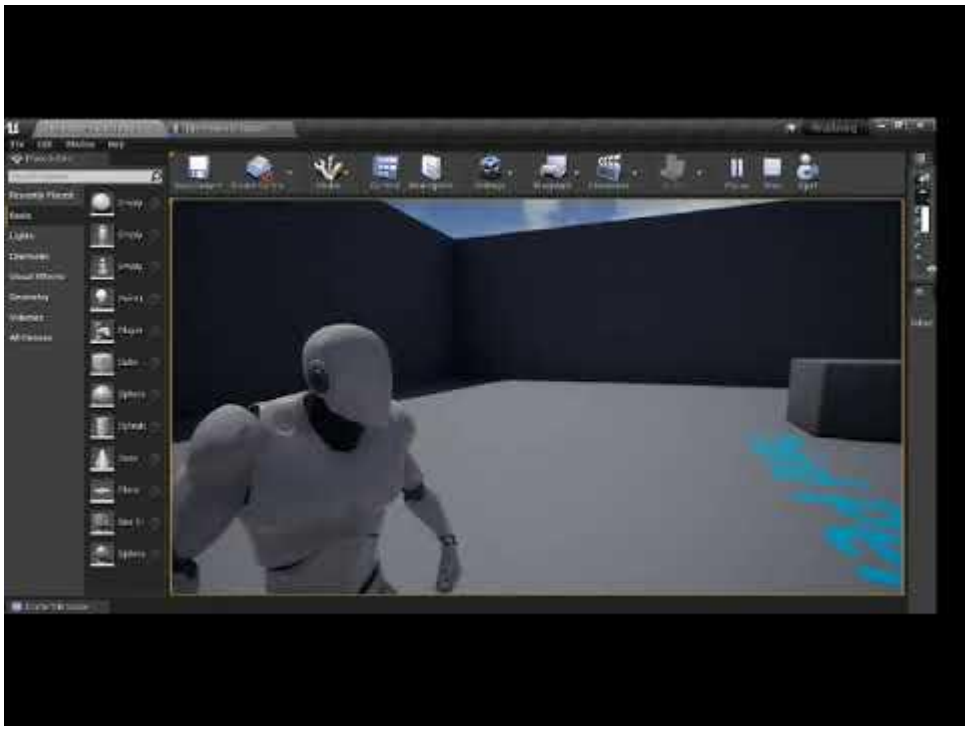
Now I copied this from the 'StartADS' and pated it to the 'StopADS' function. The 'true' and 'false' variables then got flipped.



At this point the camera will follow the aim down sights state

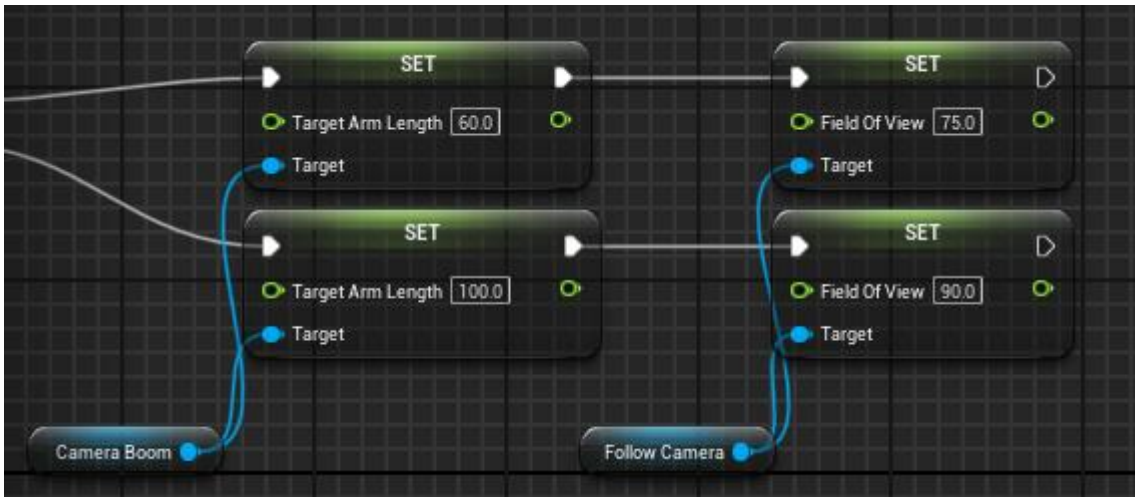


but rotate around the character if not aiming down sights.

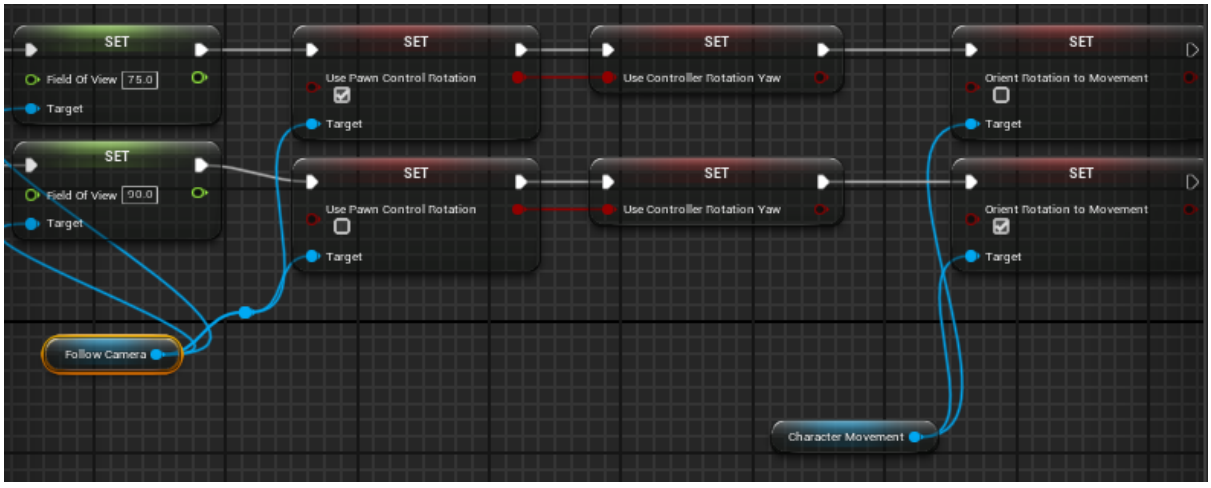


This also allows the player to snap to a point of interest by just simply aiming down sights.

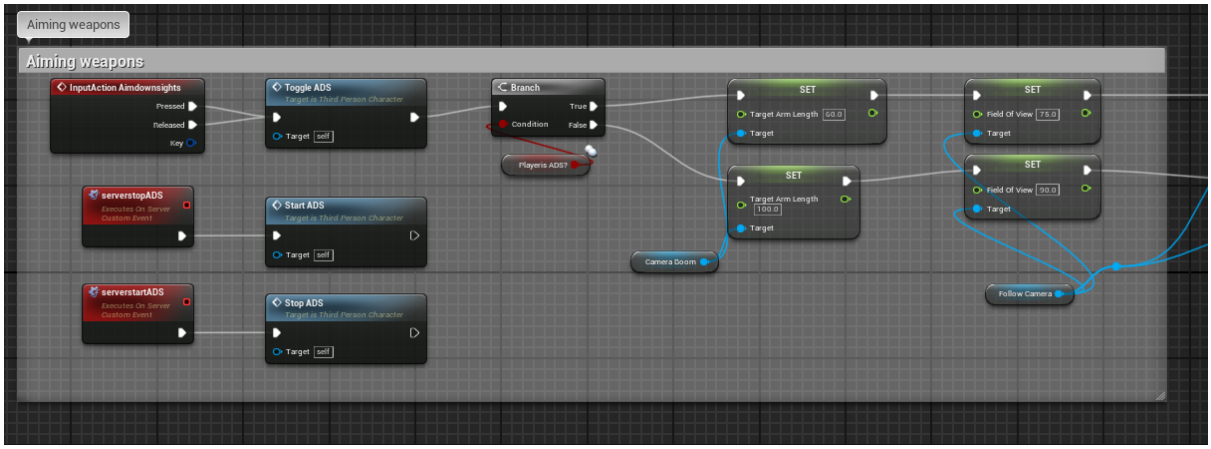
now I needed to do additional camera motions. This is done in the ‘events graph’. This is because the other code needed to get passed to everyone in the server as it had to do with character positioning. however, this next big of code is just for the players personal camera so it does not need to get sent to the other players. To add the additional camera motions, I first dragged ‘followcamera’ into the event graph and set the ‘field of view’ to a smaller variable (75) while the player is aiming down sights. I also needed a copy of this for the default camera ‘field of view’ (90). The variable is only set to ‘75’ as it is a base zoom for normal guns, if I were to add snipers and things like that this number would be changed accordingly.



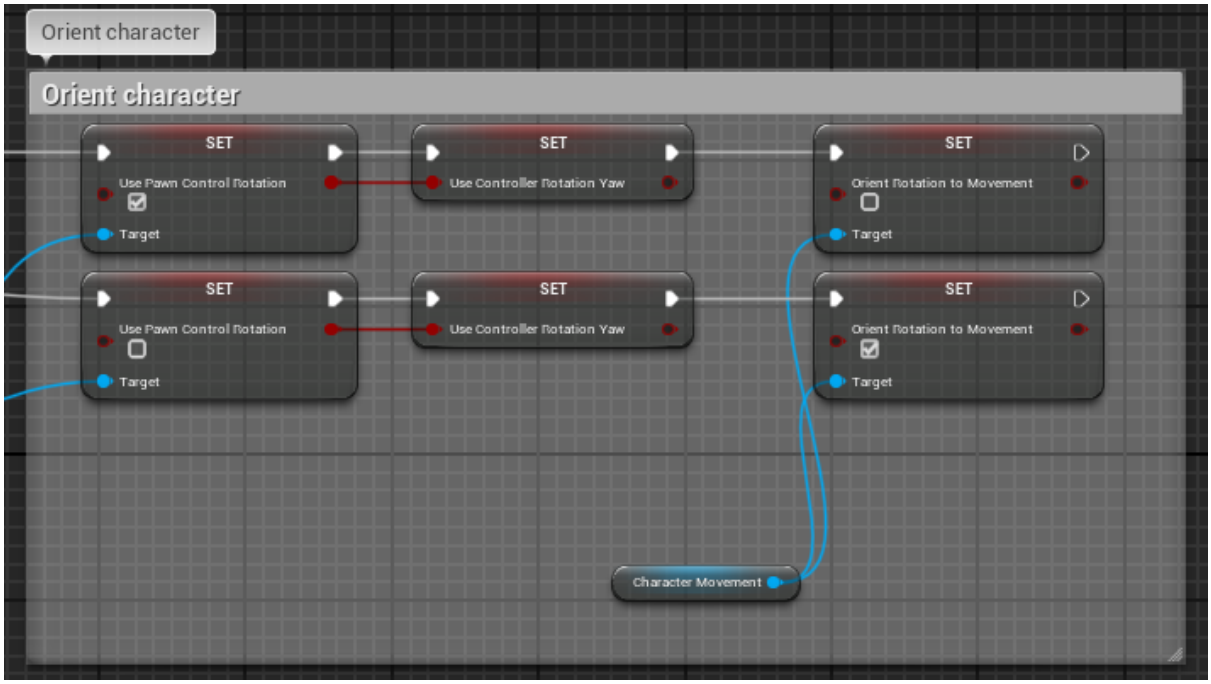
Now I needed to set the character rotation to be on the side of the players. So, I copy some code from ‘StartADS’ and paste it into the ‘event graph’ and made another copy of the top line of code because they need to be the opposite of each other. Everything is then connected to the rest of the code.



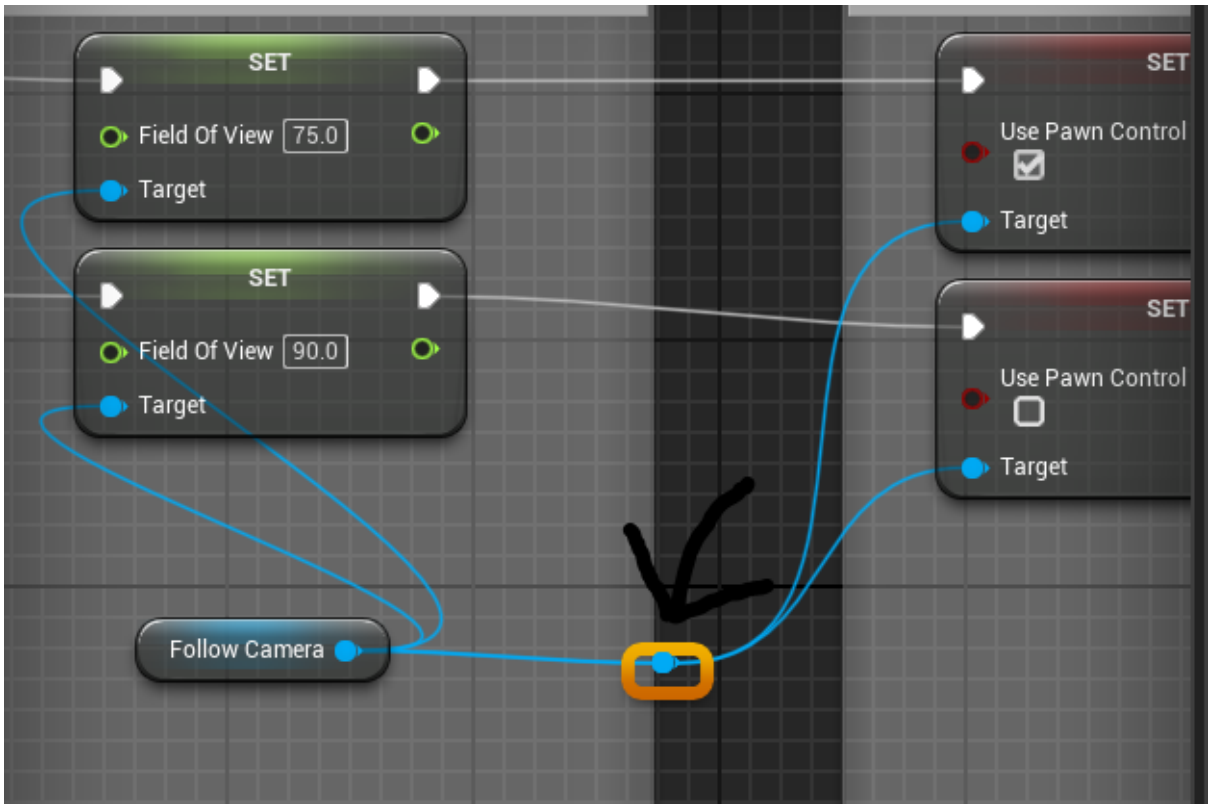
I now needed to start organising lines of code, so it wasn't a mess and make sure everything was easy to traverse in the 'Eventsgraph'. I set the code into two parts, by using the 'comment' system which creates a nameable white background. the first 'Aiming weapons'



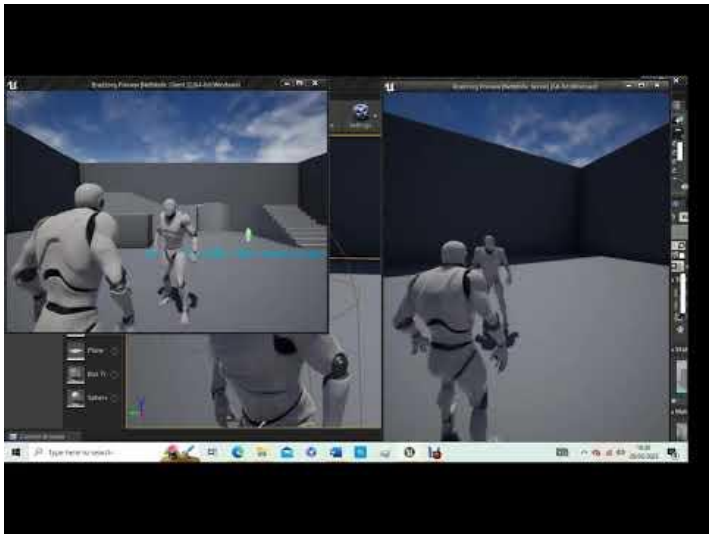
And the second 'Orient character'.



A helpful side note was the use of ‘reroute nodes’, These are points in the graph used to redirect a line for cleaner looking code.



I had trouble showcasing what was achieved at first as the tutorial was made on a slightly different build of unreal engine 4, this resulted in when I had the game run two players it was not working as intended and would not show the second player. however, I found that I had to set the ‘netmode’ to ‘play as listen server’. This fixed the issue and now made it possible to showcase today’s progress.



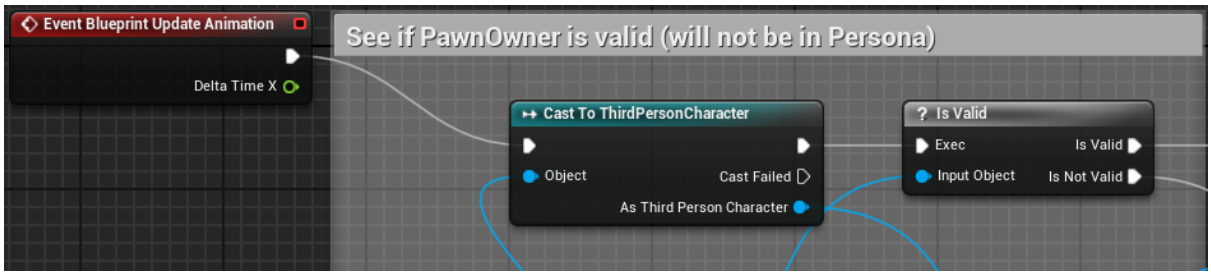
To get to this point I have used my secondary research ‘TPS Build Part 2 – Character_BP Aiming’ (K, Dail. 2015)

From this point I will be saving a day’s progress as a different save in unreal engine as if I lose one, I don’t lose the whole project. I am doing this as a precaution. This day’s version is ‘Bradzorg2’.

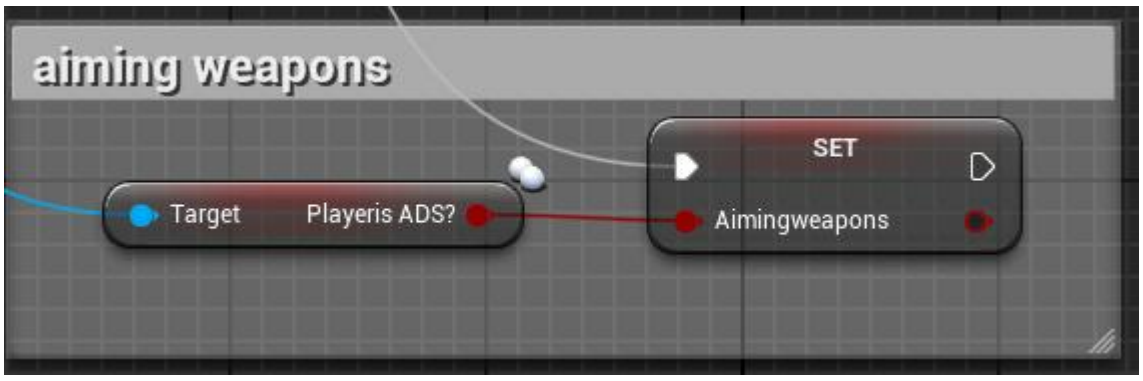
Now I will make an animation blending offset for when players are aiming weapons.

To do this I needed to go into the ‘mesh’ animation blueprint and edit it there. This can be easily found by selecting ‘mesh’ then going to the ‘animation’ detail and using the search button on the ‘anim class’.

I then added a ‘cast to Thirdpersoncharacter’ just after the ‘Event Blueprint Update Animation’ and fed it into the ‘Is Valid’ on the ‘Event Graph’.



Next step was to make a new bool by adding a new ‘variable’ named ‘Aimingweapons’ and dragging it into the ‘Events Graph’. I also needed a ‘variable’ that controls if the character is aiming or not this was already made with ‘playerisADS’. So, this was added by creating a new line from ‘cast to Thirdpersoncharacter’. A new ‘comment’ was then created.

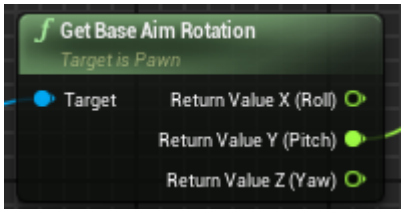


This will essentially drive the characters spine when he breaks in half.

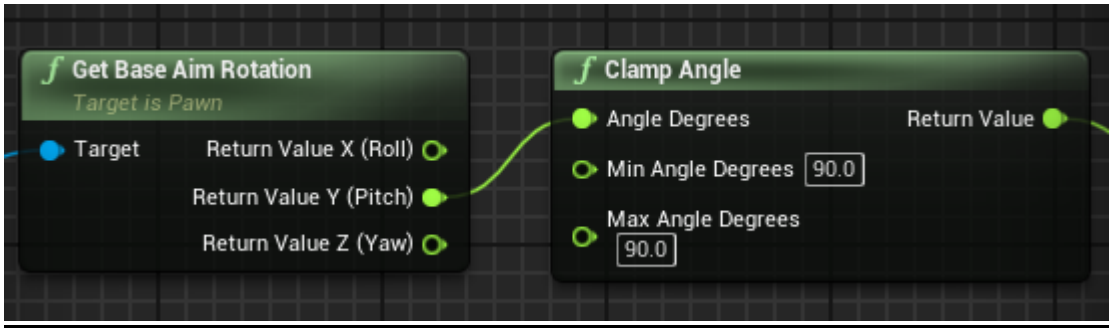
In this next part I needed to have something in place to drive the characters bones so what I needed to do was copy and paste ‘try get pawn owner’ and ‘cast to Thirdpersoncharacter’ to the end of the code to further extend it. The ‘cast to Thirdpersoncharacter’ was then ‘converted to a pure cast’ by right clicking on it.

From ‘cast to Thirdperson character’ a ‘Get Base Aim rotation’ was then added.

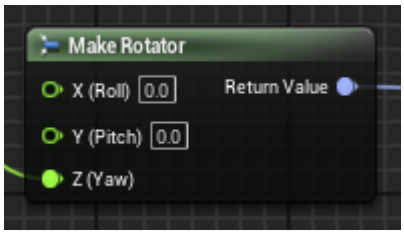
However, I needed the spine to not exceed an angle of 90° or 75°. So, I ‘split’ the ‘Get Base Aim Rotation’



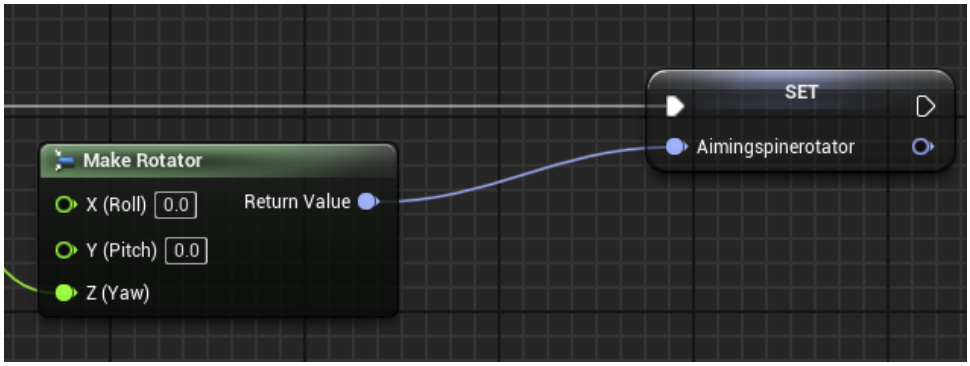
This was to set angle parameters with the use of ‘clamp angle’ with a ‘min angle degrees’ of 90 and a ‘max angle degrees’ of 90.



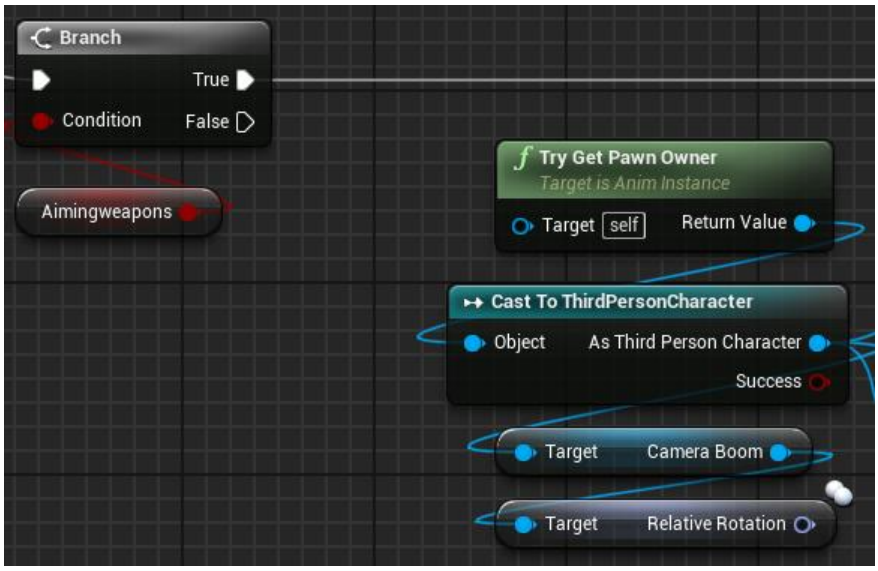
Now this needed to be set to ‘make Rotator’ Z(yaw).



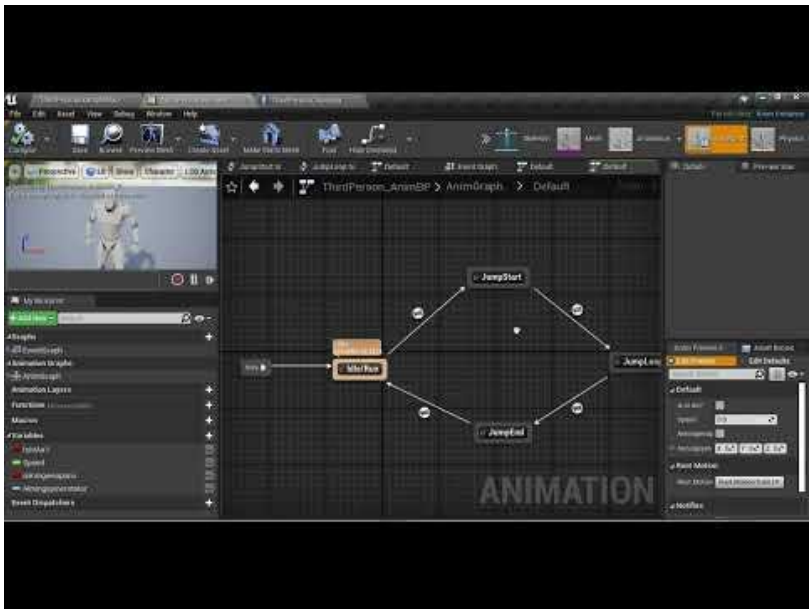
I then added a new variable named ‘Aimingspinerotator’, if this was to be connected now the value being read would be inverted.



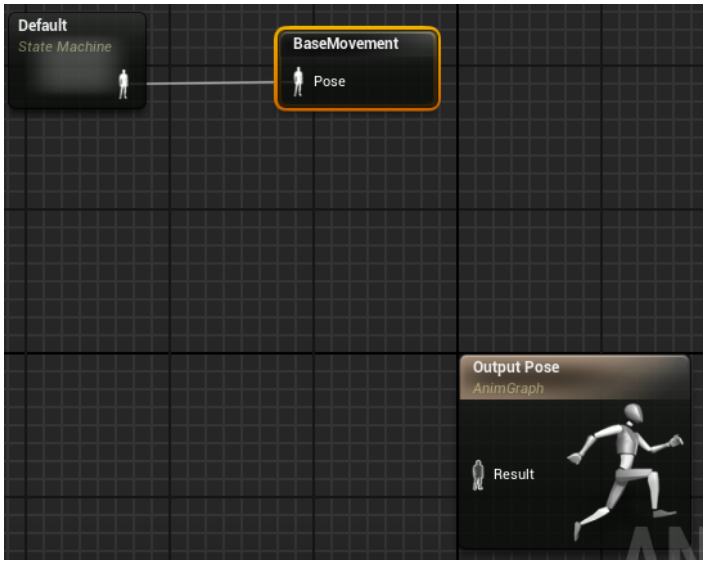
Next step was to add a ‘branch’ to tell unreal to play a set of animations when the player is aiming and a different set of animations when the player is not aiming. All ‘Branch’s’ have a ‘condition’ that needs to be set and in this case its ‘Aimingweapons’. As a result, if the ‘branch’ ‘condition’ is ‘true’ then the spine will be influenced however, if the ‘condition’ is ‘false’ the spine is not influenced.



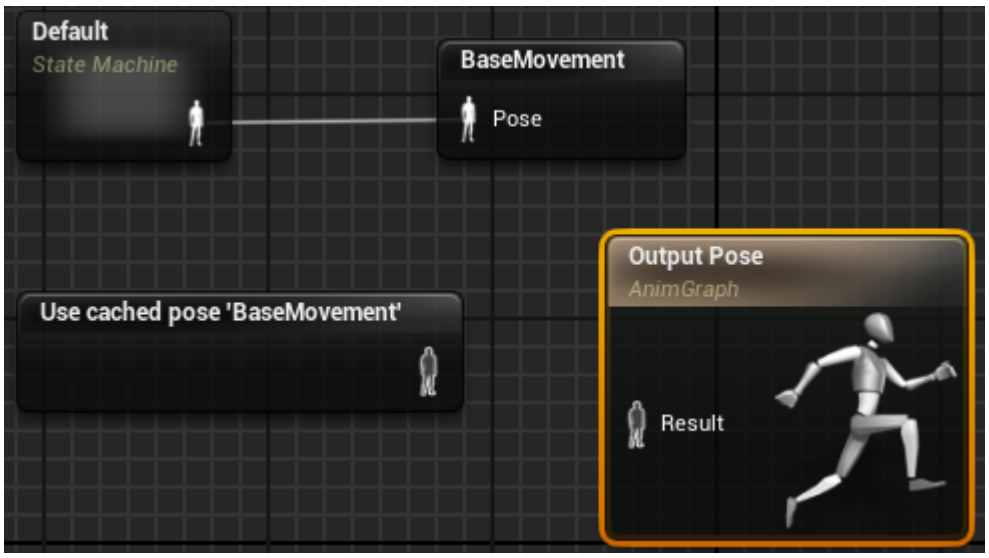
The next few steps were then done in the ‘animgraph’, This is in ‘myblueprint’. By default, there is an ‘idol’, ‘walk’, ‘run’ and ‘jump cycle’.



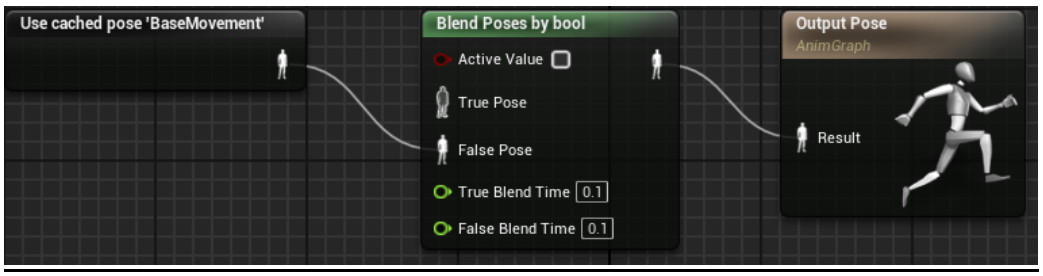
The first step here was to separate the default ‘state machine’ and add a ‘new save cached pose’ named ‘BaseMovement’.



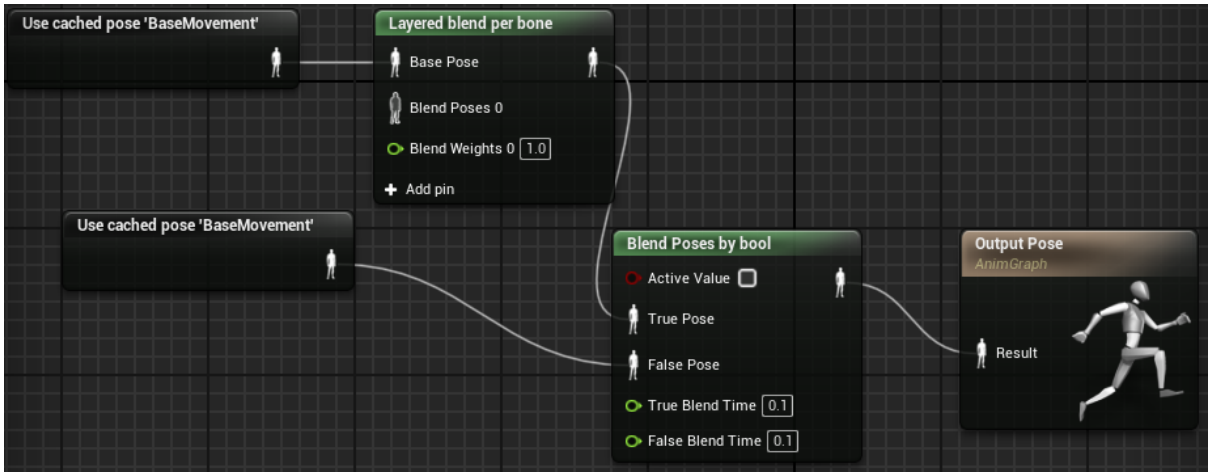
The second thing added was a ‘use cached pose’ and that pose was/ is ‘BaseMovement’. This is basically just a copy of the default animations. Its beneficial to have this because I can duplicate this one tab instead of the two above it.



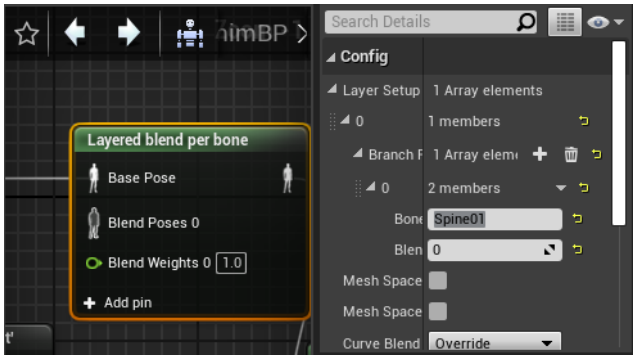
The third thing needed was to start breaking the character up so to do this a ‘Blend Poses by bool’ needed to be added this acts as a ‘switch’ and or a ‘flip flop system’. This is set to false and is connected to the base motion movements. this is for when the player is not aiming.



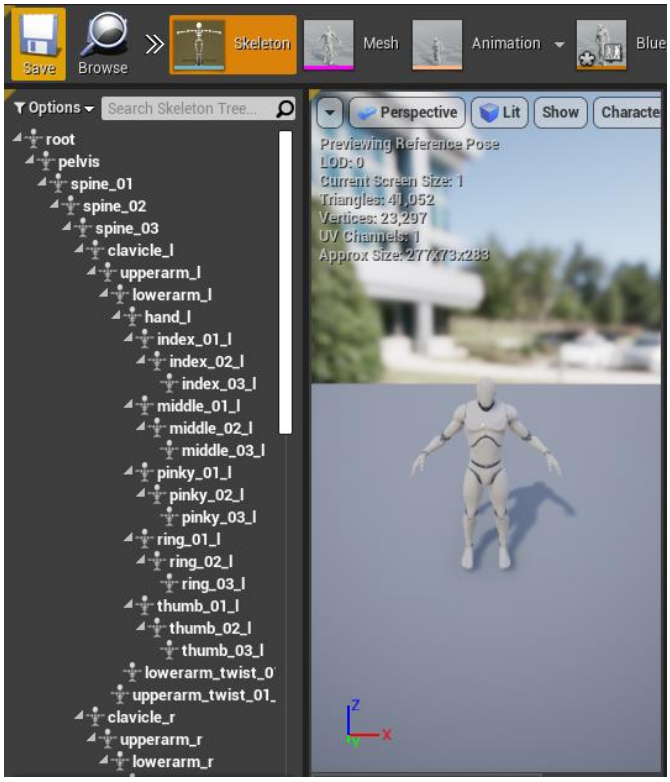
However, I needed some animations for when the player is aiming to do this, I needed to do a brake up of the characters body. To do this I needed to use something called ‘layer blend per bone’. Another copy of ‘use cached pose’ needed to be added and connected to the ‘base pose’ as well as being connected to ‘false’ on the ‘Blend Poses by bool’.



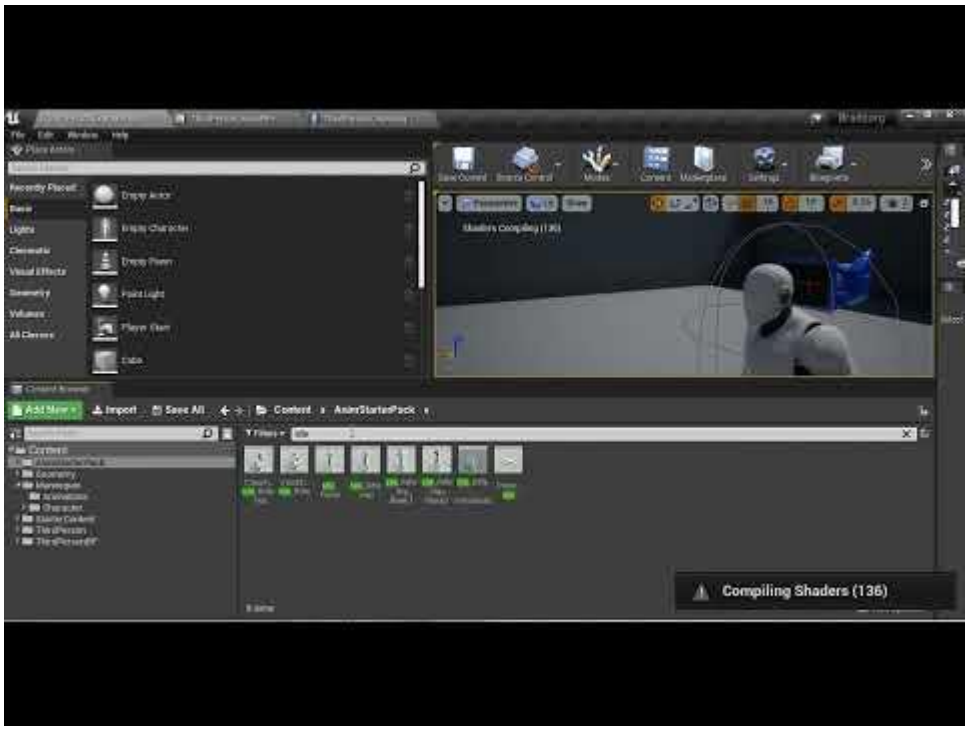
Then I needed a bone in the upperpart of the body that this can start to manipulate. So, I needed to go to ‘layer set up’ to add another branch and named the bone ‘spine01’ because this is the main bone that will be manipulated.



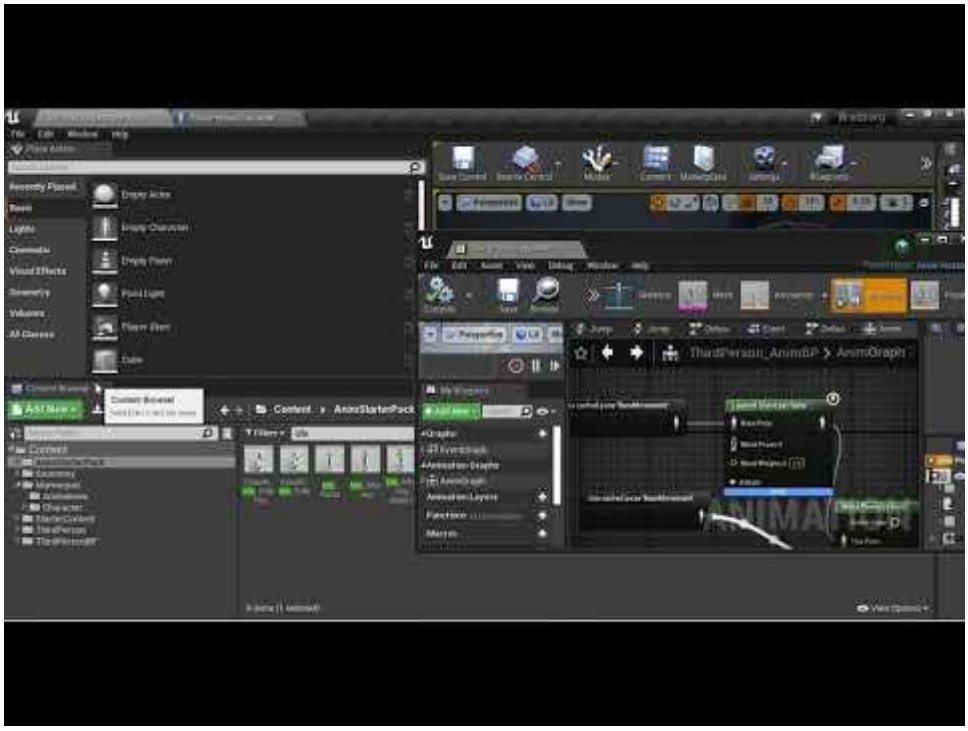
All the bones are in the ‘skeleton’ tab.



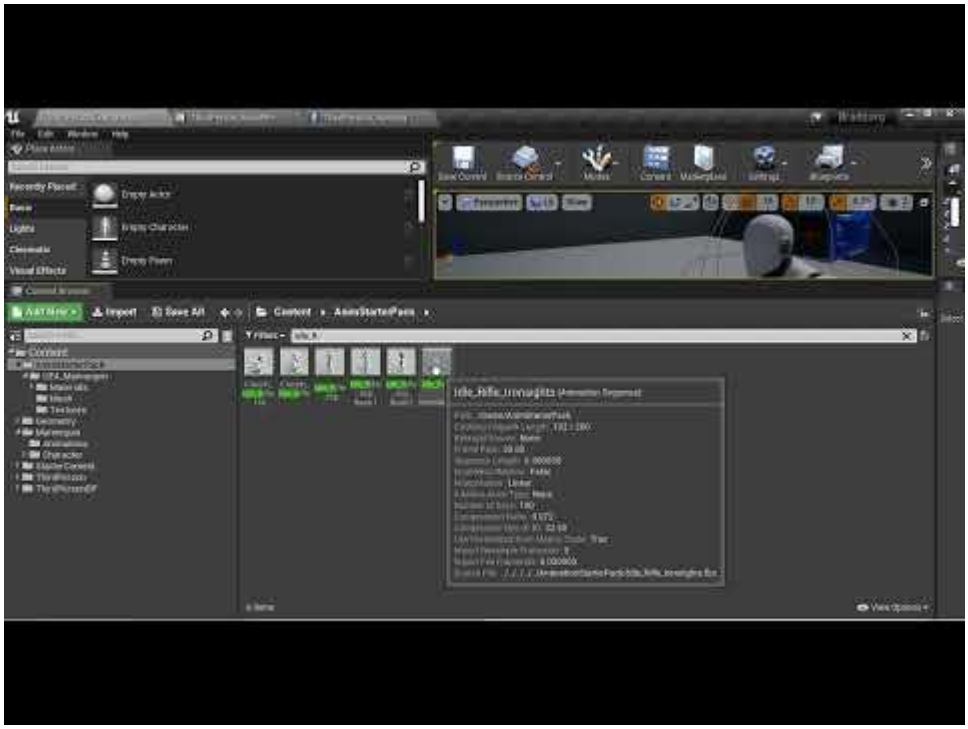
From this I needed something to drive the spine done while aiming. However, I needed an animation that will allow the character to aim a weapon when the player aims. This is where the ‘animation starter pack’ comes in handy. The animation I needed from these downloaded animations was named ‘Idle_Rifle_Ironsights’.



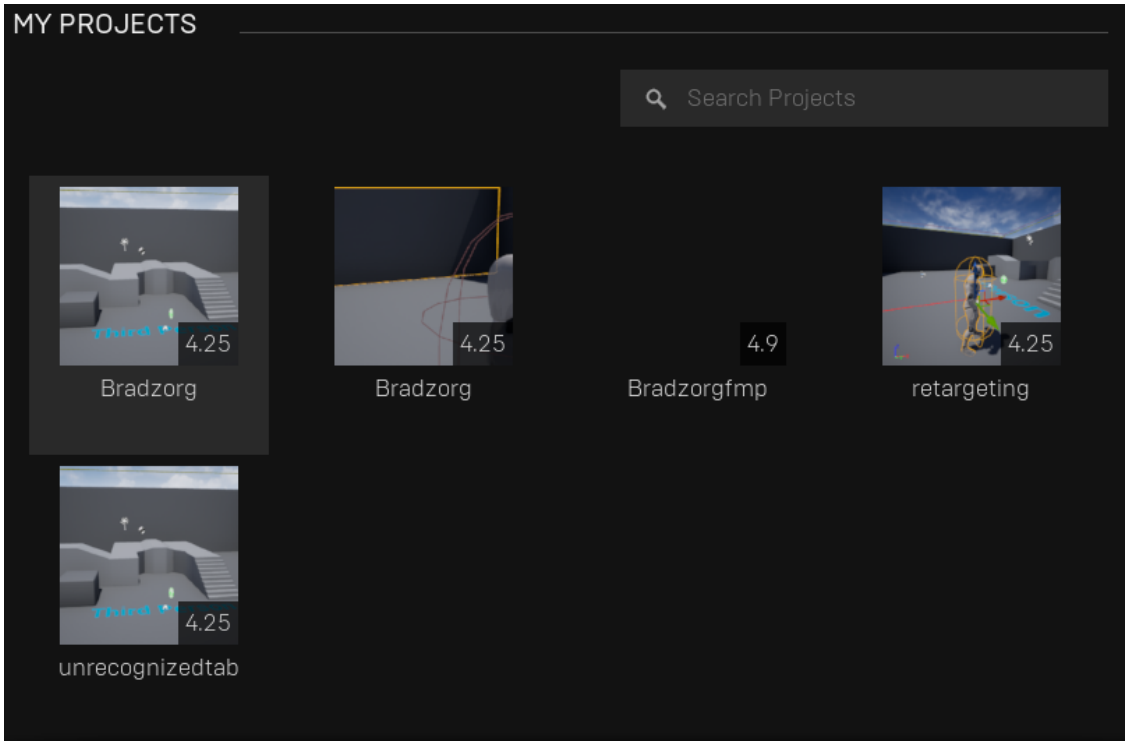
However, more steps were necessary to be able to add this animation into the ‘animgraph’ as it’s not possible just to just drag and drop it in.



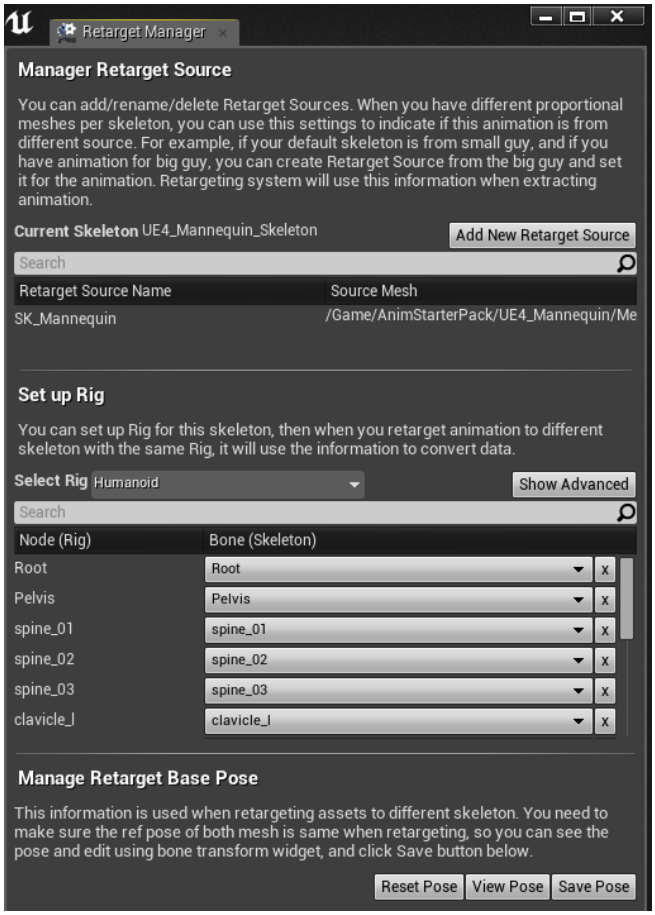
This is because the ‘skeletons are not compatible’. So, to fix this I needed to make a copy of the character. So, to do this I found the skeleton of the ‘mesh’ then retarget the ‘Idle_Rifle_Ironsights’ animation. However, there were no skeletons to choose from.



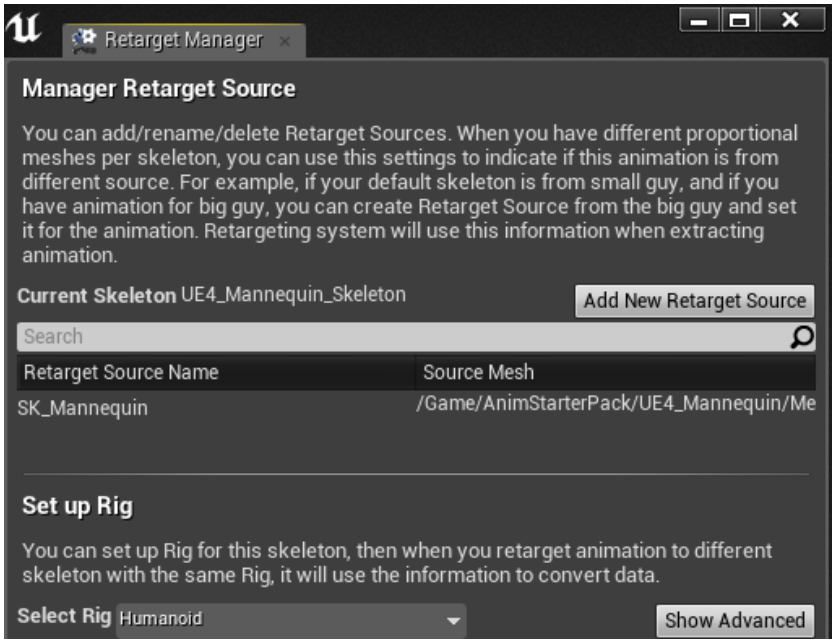
So, at this point to fix this I needed to do ‘retargeting’ to get the animation over to the other skeleton. However, I ran into a huge bug with unreal engine version 4.25.4 (the version I started on). I had the ‘retargeting’ tab open as a window and I clicked else ware on the screen, resulting in it closing and refusing to reopen apart from the times it showed up as an ‘Unrecognized tab’. I searched for around eight hours for a fix. I searched on Unreals official website (Epic Developer Community Forums (unrealengine.com) , as well as YouTube and eventually I asked my tutor if he knew how to fix it but after resetting the layout, reinstalling Unreal Engine and searching around the software just in case It was there and I was missing it, I was forced to restart. This was not all bad though because now I downloaded the same version of Unreal Engine as the tutorial was using (4.9.2). I also had the opportunity to reassure myself that I was learning how to use the software and not just copying the tutorial as I was able to catch back up in just around 3 hours. At which point I found the fix for the previous build because as I opened ‘retargeting’ on my new version of the game it was then miraculously able to be reopened on my previous version. However, in my attempts to find a workaround when I couldn’t find a fix, I had messed up some mesh blueprints, so restarting was necessary regardless. I now have one old version, three copies I made to try fix’s safely where the ‘retargeting’ tab was still showing as an ‘unrecognized tab’ and a new one named ‘Bradzorgfmp’ this will be the version I will be continuing to develop.



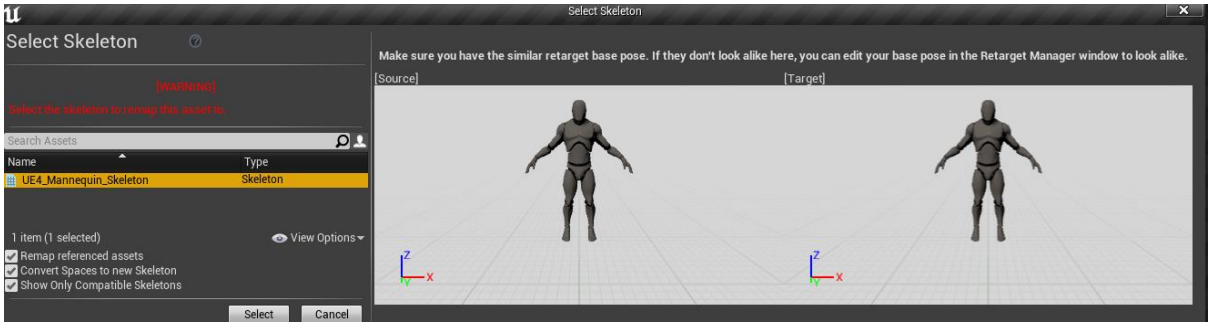
Now that I was able to access the ‘retargeting manager’ again I continued by setting the character into a ‘reference pose’ so that both the skeletons are as close as possible adding a ‘new retargeting source’ this being ‘SK_Mannequin’ and setting the ‘rig’ to ‘humanoid’.



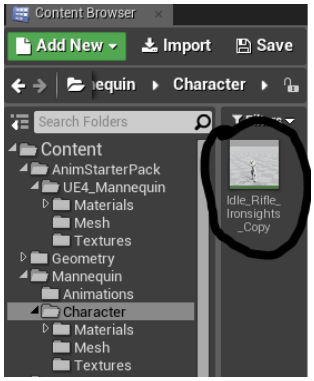
This was then replicated with the ‘Animpack’ character.



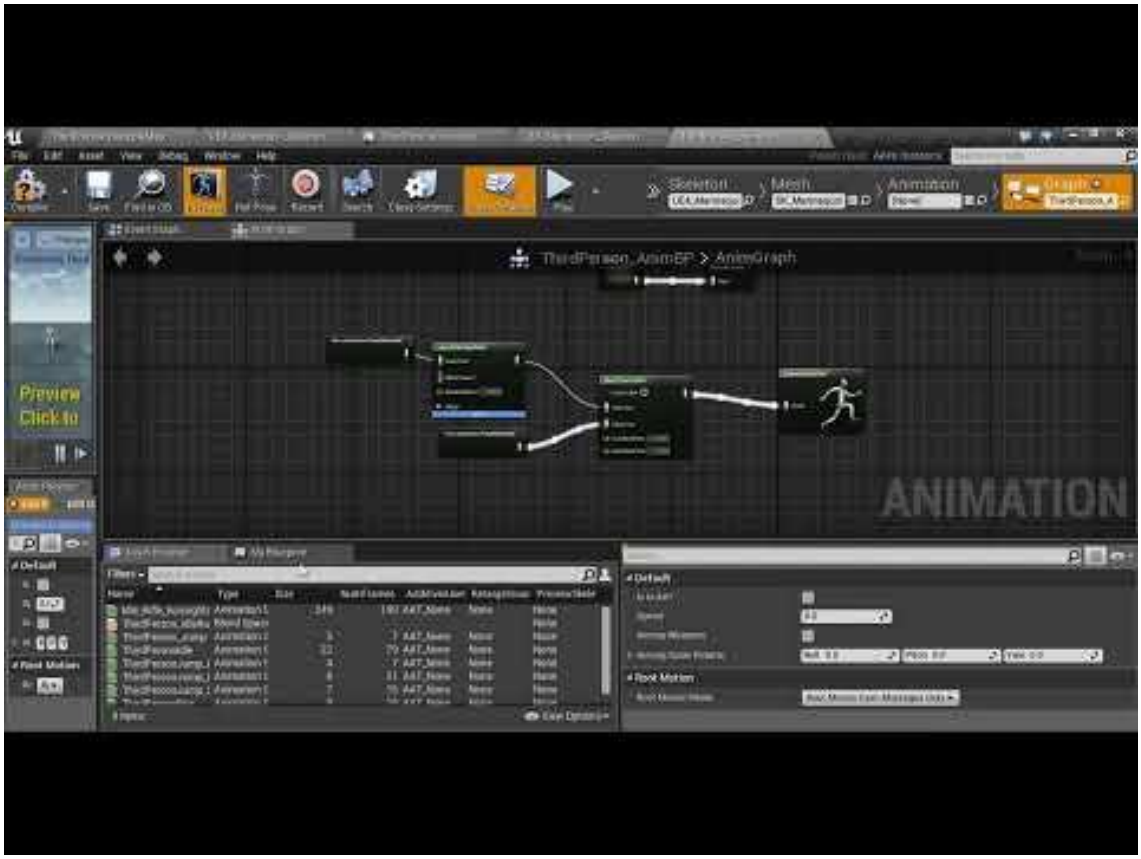
I then found the ‘Idle_Rifle_Ironsights’ animation and ‘duplicated’ it to the other skeleton.



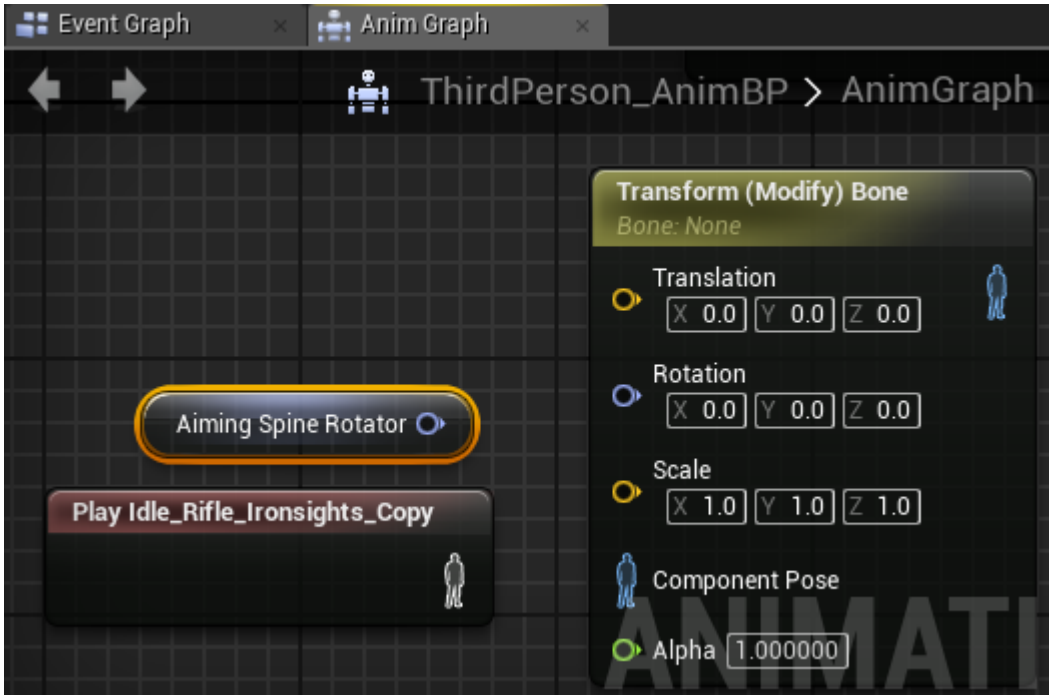
This animation can now be found in mannequin – character.



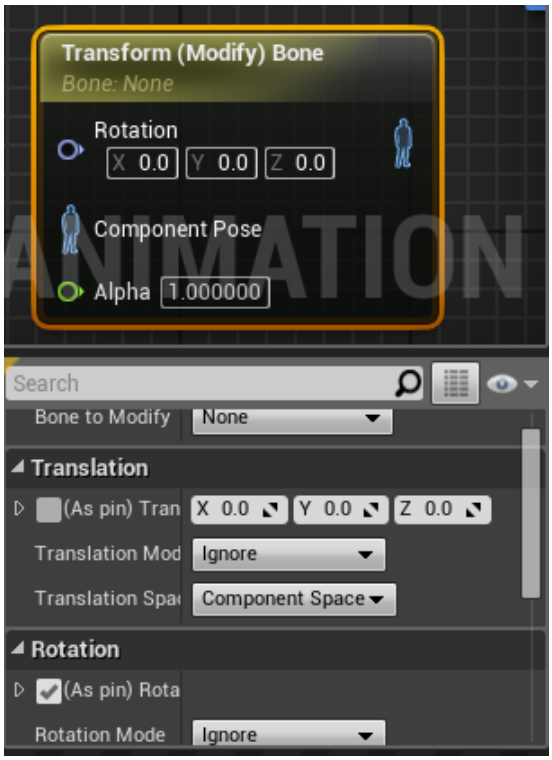
This new animation is then added to the ‘Animgraph’ from the ‘Asset Browser’.



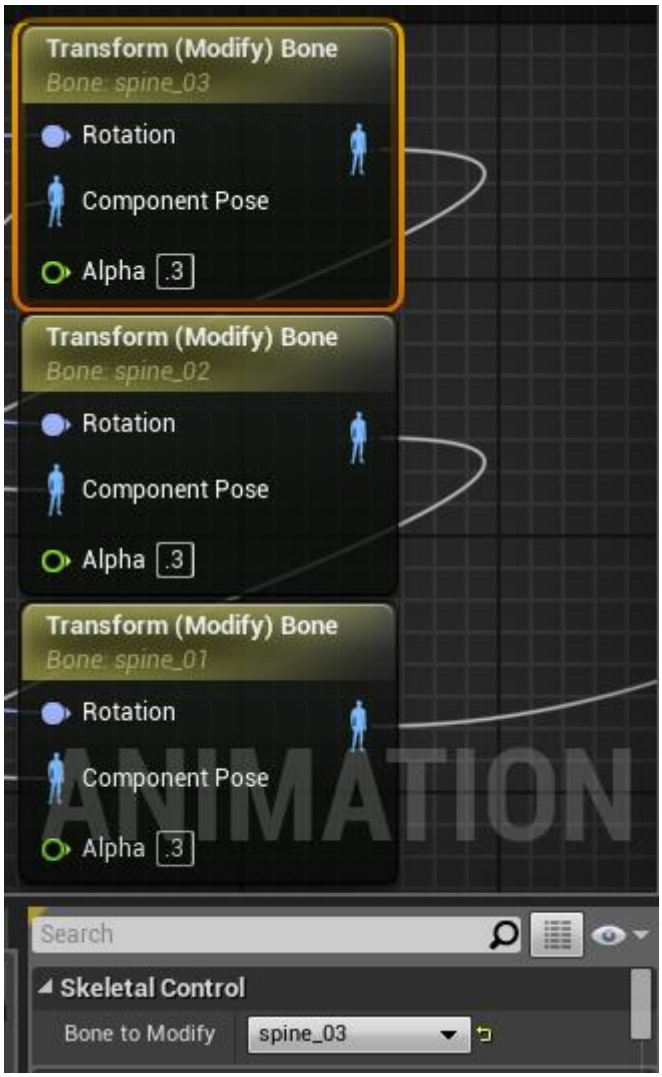
The next step was to add the ‘AimingSpineRotation’ variable to the ‘animgraph’ alongside a ‘Transform (modify) Bone’. By default, this allows you to modify the ‘scale’, ‘Rotation’ and ‘Translation’.



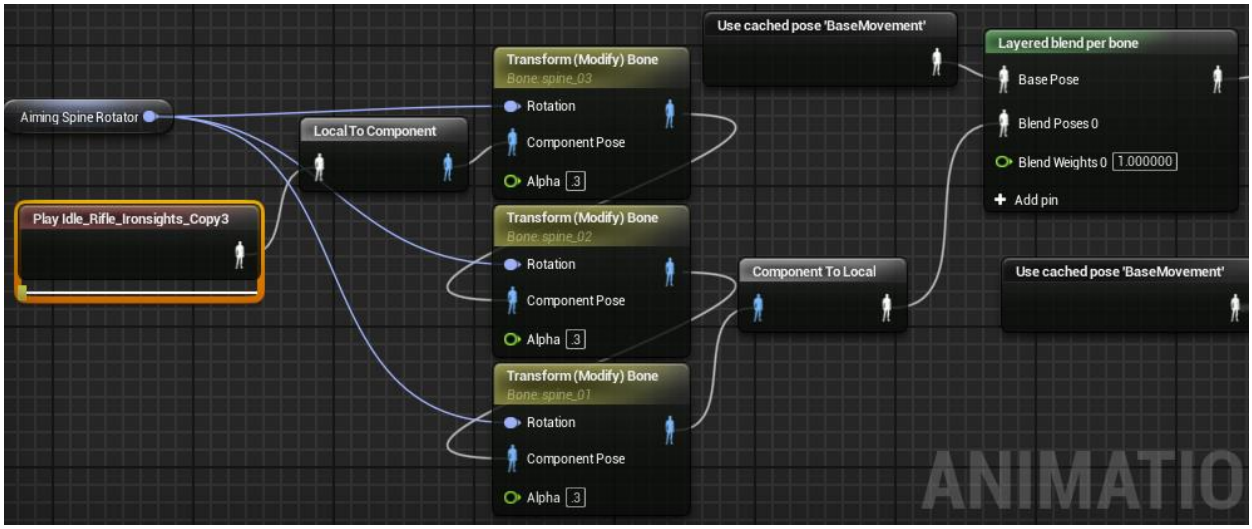
However, I only needed ‘rotation’, so I took ‘scale’ and ‘Translation’ off by using the tick boxes. I did this to minimise any potential mistakes by mis-clicking.



At this point the ‘alpha’ needed to be set to an even blend for two more copies of this too, the value is set to ‘.3’. The ‘Alpha’ value then equals ‘.9’. this is equal to 90% manipulation of the characters back. I needed to set a ‘skeleton control’ bone for each of these one being ‘spine_01’ another being ‘spine_02’ and another being ‘spine_03’. These are the bones that are being manipulated or ‘modified’.

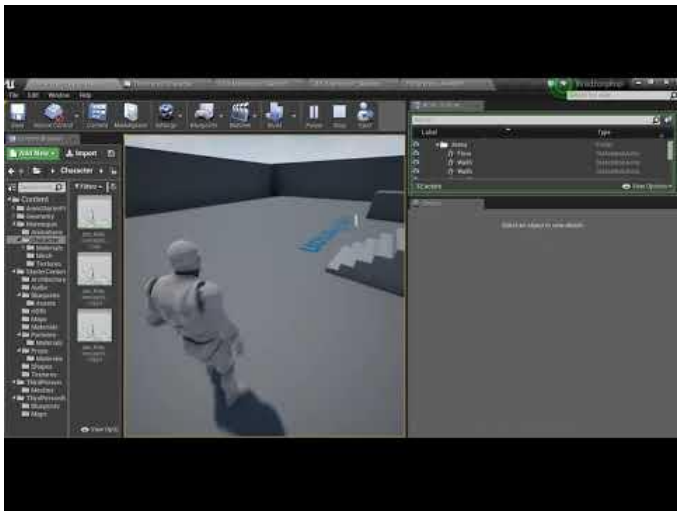


These are then all connected to the rest of the ‘code’. When the main animation for the character ‘Play Idel_Rifle_Ironsights’ is connected to the top ‘Transform (modify) bone’ a ‘local to component’. These allow unreal to make something happen in the character and not to the character its self. These are commonly used for montages or animations. Another is then created when the bottom one is connected into ‘Layer blend per bone’.

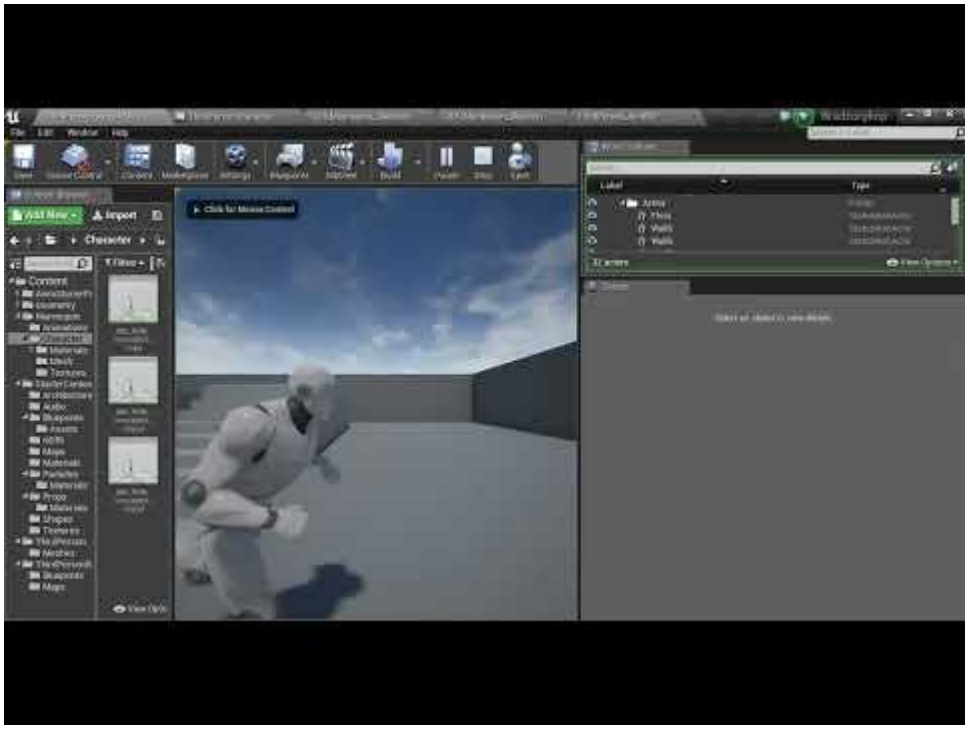


The final step for this was to set the ‘AimingWeapons’ variable to the ‘Blend Poses by bool’.

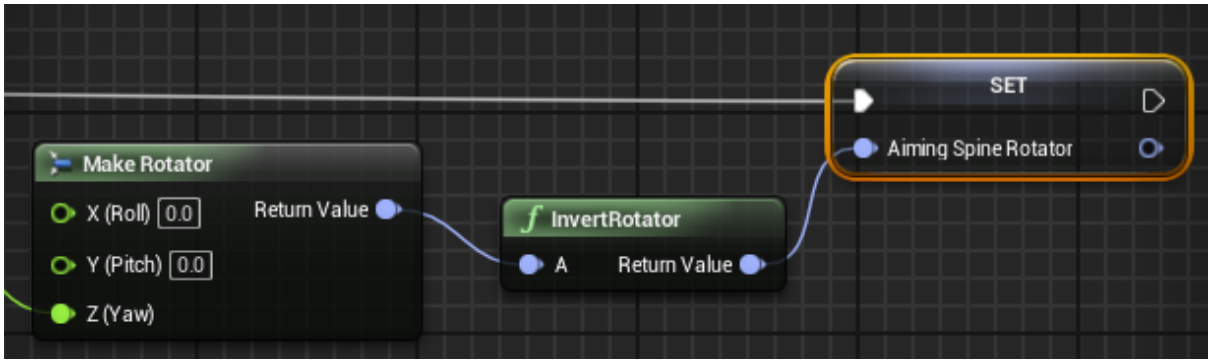
This should have been the end of this part however, when I play tested it all the characters movement animations were not playing at all.



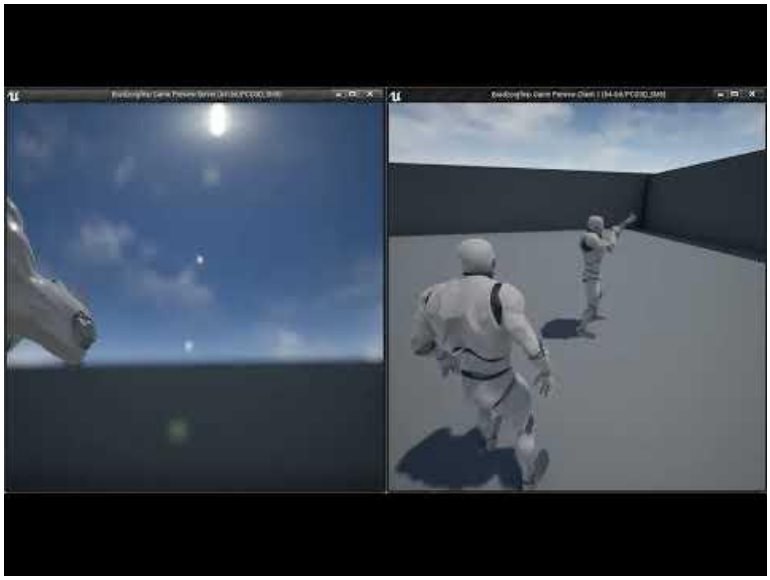
To fix this I rewatched the video following every step thoroughly. I found that I just missed to reconnect the ‘Event Blueprint Update Animation’ to the rest of the code at the start of this section.



This brought the animations back but, towards the end of the video I realised as I aimed the camera up my characters animation would play backwards. This was I forgot to fix it earlier. So, to do this I only needed to add a ‘InvertRotator’ in the events graph, connect it to the ‘set AimingSpineRotator’ and connect ‘make rotator’ to the inverter.



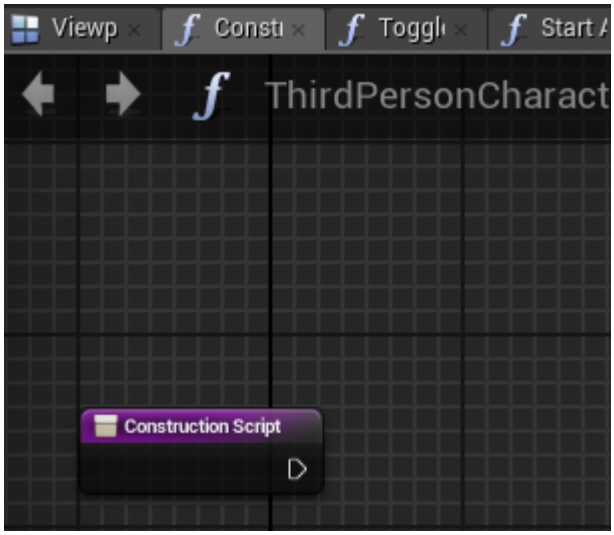
This was the end result of using my reference video ‘TPS Build Part 3 Character AnimBP Aiming’ (K, Dail. 2015)



Asset migration

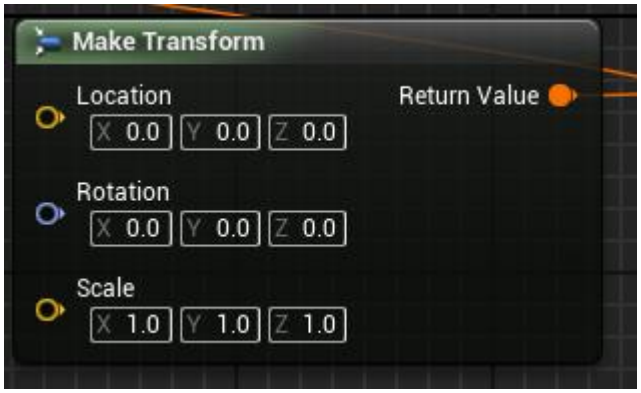
The next natural step in the process was to add a weapon. I did this using ‘Asset migration’, I took the 3D model from unreal engines ‘firstpersonshooter’ blueprint. This will act as a place holder just in case I run out of time and cant create my own weapon asset. It is a simple process of just copying the file over from that blue print over to my Bradzorg folder under the ‘content’ file.

Now that I had the weapon asset, I needed to add it to my characters hand. This is done in the ‘ConstructionScript’, these are only called once as the character is spawned into the level.



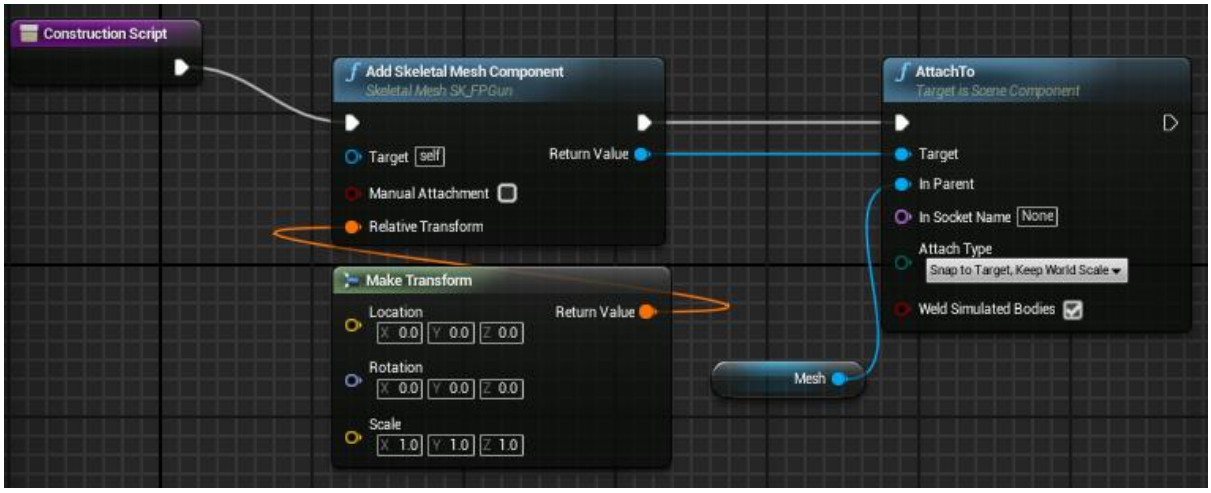
The first thing added is the prototype assault rifle by just simply dragging the mesh over from the ‘content browser’.

The second is a ‘make Transform’, this tells unreal to spawn it in the game world to a ‘scale’ of ‘1.0’ or the ‘scale of the character model.

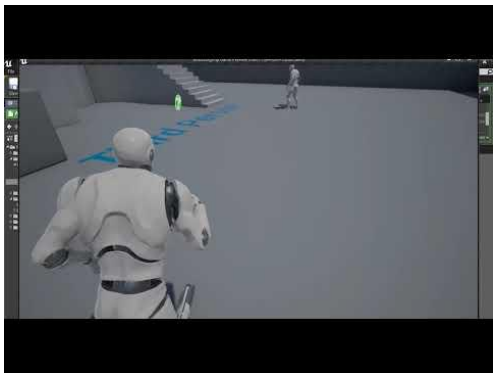


The third step is to drag the ‘Mesh (Skeletal Mesh Component) in as well.

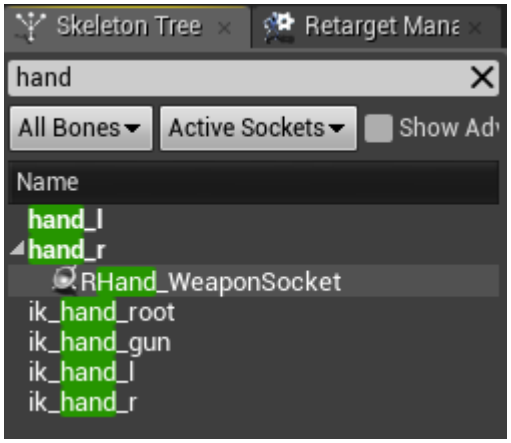
The fourth step is to add an ‘Attach to’ and change the ‘Attach type’ to ‘Snap to Target, Keep World Scale’.



however, if this was to be played now it would be at the characters feet.

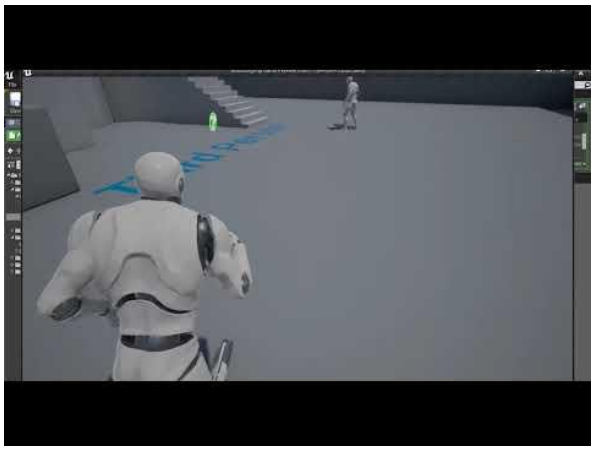


So, to fix this I needed a place to attach the weapon model to, this is done in the ‘skeletal mesh’. I added a ‘socket’ named ‘RHand_WeaponSocket’ to my characters right hand, this is done in the ‘skeletal tree’.



The name of this socket was then copied into the ‘In Socket Name’ back on the ‘attach to’.

When I then play tested it again the weapon was not facing where I needed it to be.



This was fixed by simply adjusting the position of the model in the ‘skeleton’ map. In order to see the model, I needed to ‘preview’ the asset. However, I needed to do this on the paused animation ‘Idle_Rifle_Ironsights’, so that I could place it in my characters hand’s easier.

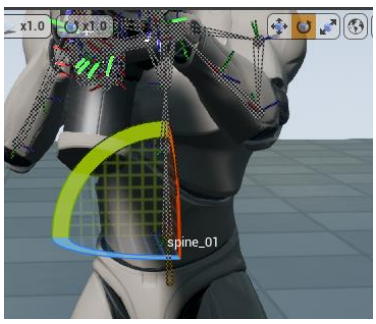
This was the result of my reference video ‘TPS Build Part 4 – Sockets and Migrate Assets’ (K, Dail. 2015)

Setting up aim offset

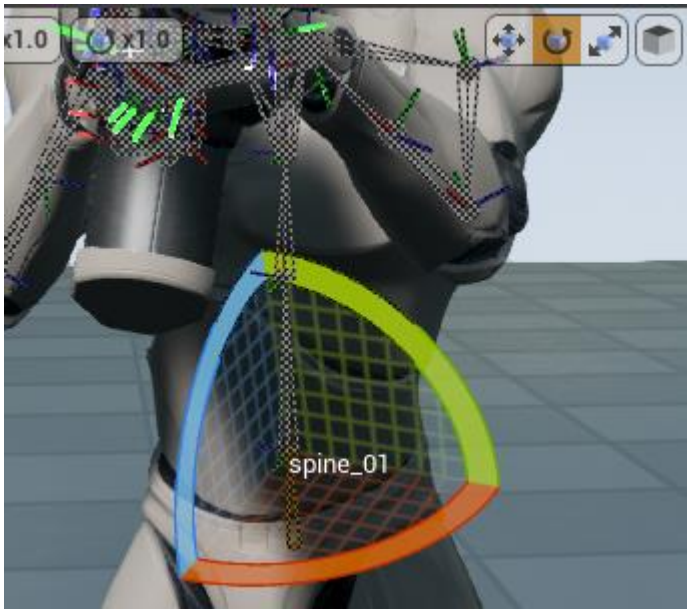
The first step for setting up my ‘aim offset’ is to manipulate the spine to get it ready for additional animations. To do this I needed to do an ‘animation layer’ and ‘keyframes’ to change the actual animation in unreal. The starting point with this was to bring up the animation that I wanted to copy, ‘Idle_Rifle_Ironsights’.

The problem with this animation was that the character would lean left and right as the camera faces up or down.

To fix this the spine bones needed to be facing straight. They look fine in the ‘world view’.

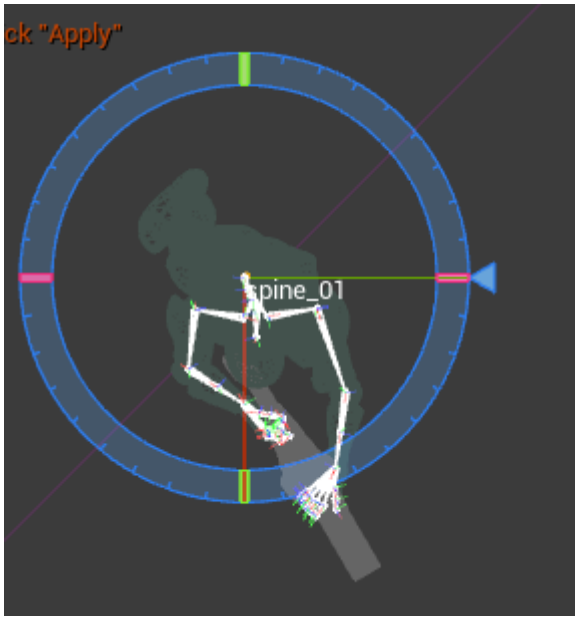


However, in ‘local view’ the spine bones are shown to be around 25% off.

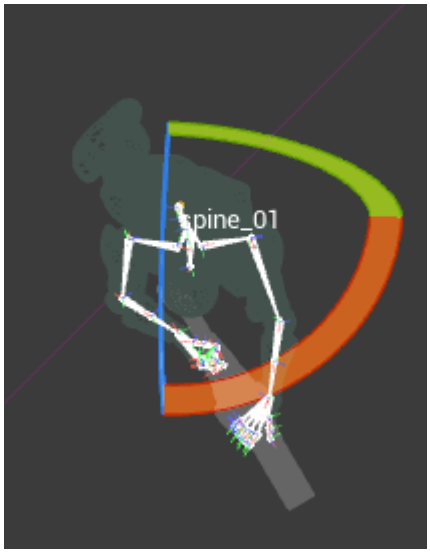


In order to centre these bones a ‘key’ had to be made.

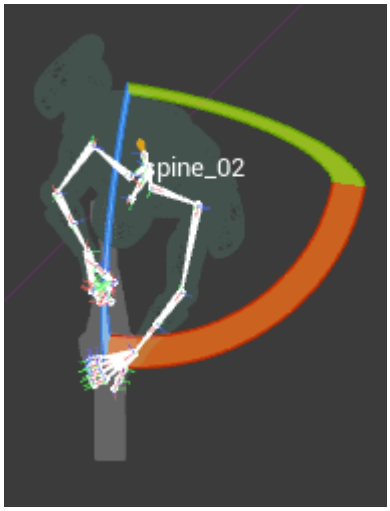
I then had to adjust these in the ‘world’ position by using ALT+J.



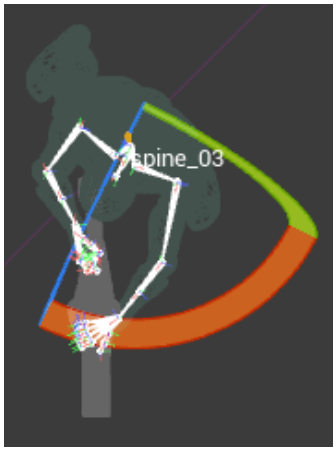
Then view if they were adjusted correctly in the ‘local’ position. The idea is to get the blue curve facing the front of the character. This bone ‘Spine_01’ now had an ‘additive’ animation applied to it.



The same was then done to ‘Spine_02’.

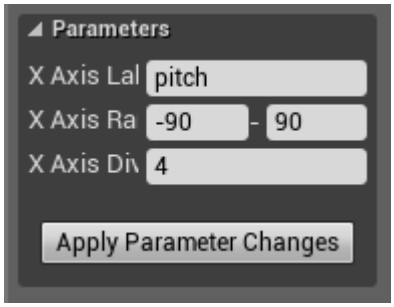


With ‘spine_03’ the gun had to be facing straight forward.



To finish these animations, I had to ‘retarget’ them to the ‘mannequin’ skeleton.

Another way to do this is through the use of animation ‘blend shapes’. To do this I needed to first set the parameters of the ‘X Axis Range’ to a minimum of (-90) and a maximum of (90) like I had already done to the spine bone. This label was named ‘pitch’ because that Is up and down in unreal.

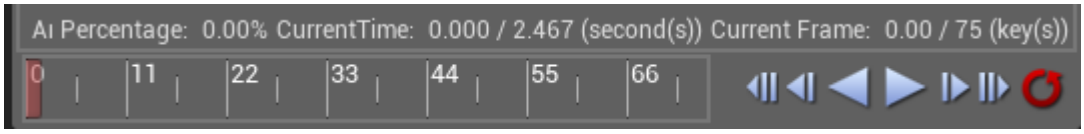


The next step was to record 3 different animation frames from the ‘Idle_Rifle_Ironsights’ animation, one being the first frame (1) and named ‘Rifle_AO_CC’. (AO - Aim offset CC – Centre Centre)

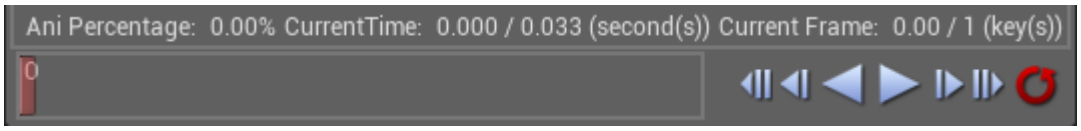
The second being frame (10) named ‘Rifle_AO_CU’ (CU – Centre up)

And the third being frame (20), named ‘Rifle_AO_CD’ (CD – Centre Down).

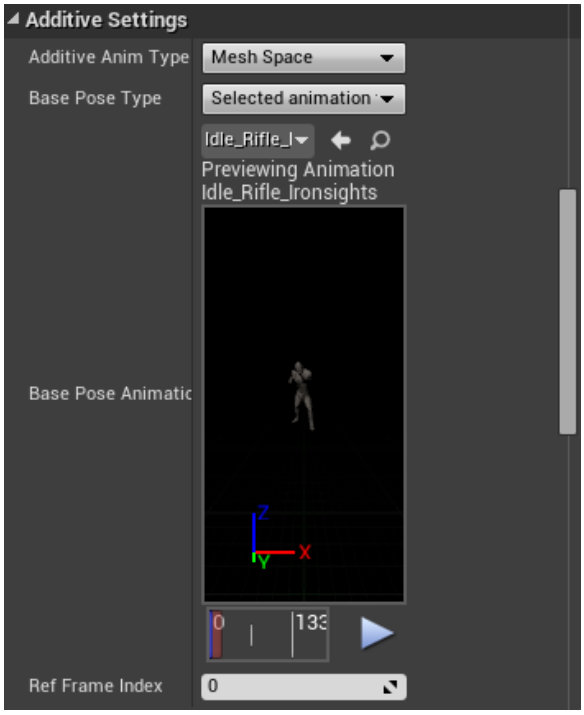
However, because I had to record the animation to get these three frames I was left with hundreds of the same frame.



To delete the unwanted frames, I just had to right click on the red animation curser and select the ‘remove’ option. This was done to all three recordings.



The next step was to change the ‘Additive Anim Type’ to ‘Mesh Space’, the Base Pose Type to ‘Selected animation and select the animation I wanted to us it for which is ‘Idle_Rifle_Ironsights’.



These could now be added to ‘Rifle_AO’. This is the result of my reference video ‘TPS Build Part 5 - Retarget, Adjusting and Creating Animations/ Setting up Aim Offset’ (K, Dail. 2015)

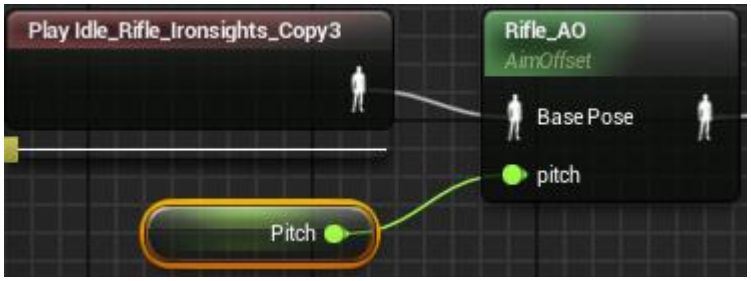
Finishing Aim offsets

The first thing to do was to in the ‘AnimGraph’ was to add the ‘Rifle_AO’.

This needed a ‘BasePose’ animation, this being ‘Idle_Rifle_Ironsights’. So, I made another copy of it in the ‘AnimGraph’ and attached it to my ‘Rifle_AO’.

This needed a ‘Pitch value’ so in order to do that I made a new variable in the events graph and attached it to the pitch manipulation I made earlier under the ‘comment’ ‘Spine rotation for aiming weapons.

This variable was then added to ‘Rifle_AO’ in the ‘AnimGraph’.

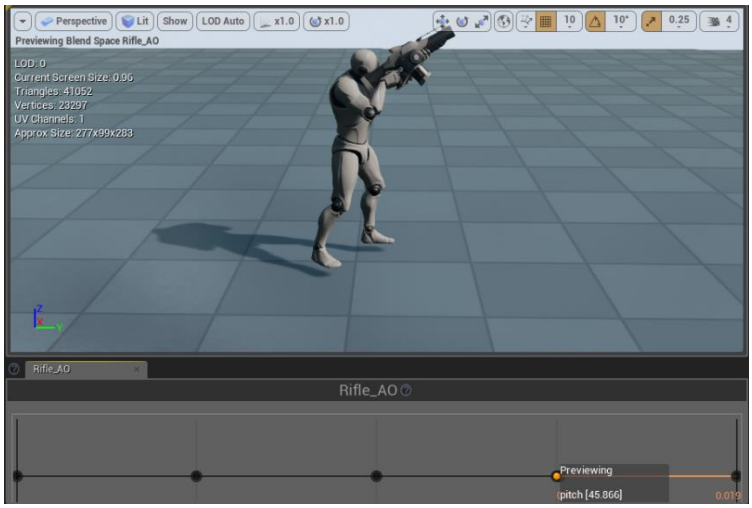


This now allowed for the character animation to look directly up and directly down smoother than before. However, the animation was not complete yet as the characters left hand would slightly drift off from where I needed it to be.

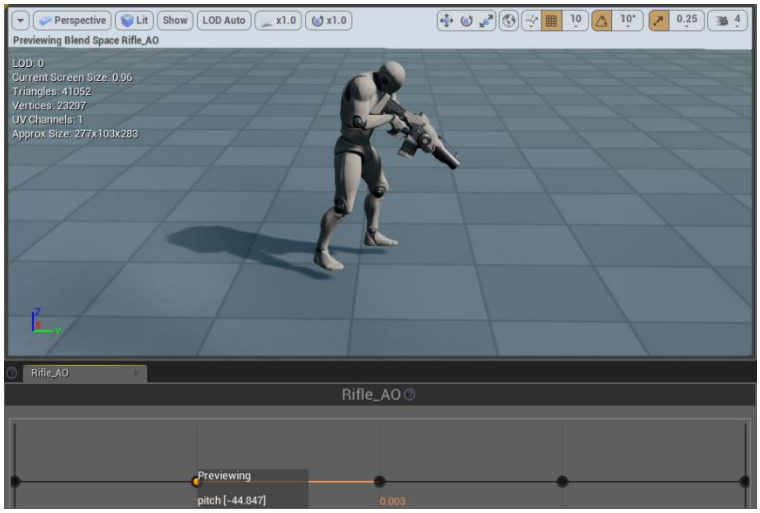
To fix up the animation I added two more recorded frames to my ‘Rifle_AO’ at ‘pitch’ (-45 and 45). The frames I needed to get were halfway between ‘CC and CD’.

and another frame from between ‘CC and CU’.

These are then added to the ‘Rifle_AO’. The frame ‘CU45’ is placed at ‘Pitch’ (45).

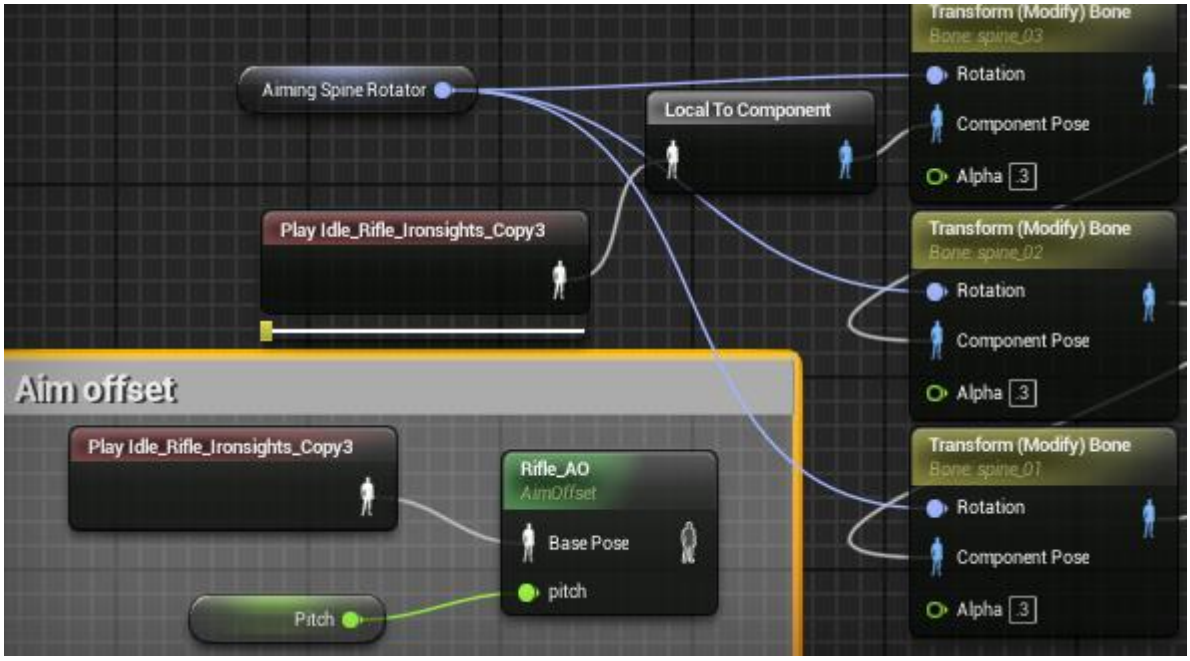


The frame ‘CD45’ was then placed at ‘Pitch’ (-45).



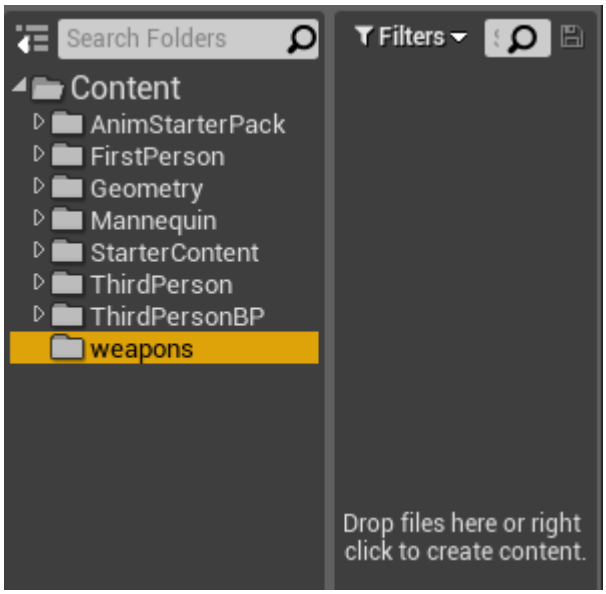
This is the result of my reference video ‘TPS Build Part 6 – Finishing up Aim Offset and Moving / Asset References’ (K, Dail. 2015).

However, I’m not going to be using this instead I will be using the ‘Transform’ spine bone I set up earlier.

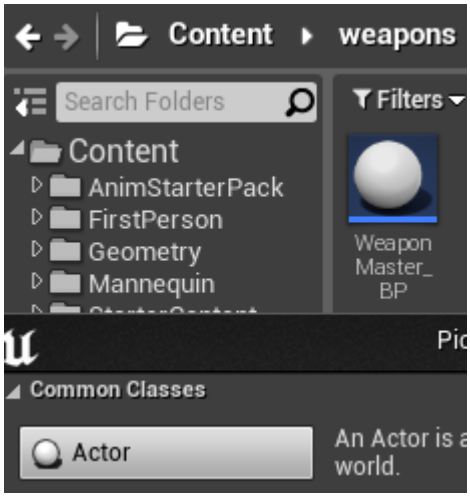


Starting the weapon blueprint

The first step is to make a new folder in the ‘Content Browser’ named ‘Weapons’.

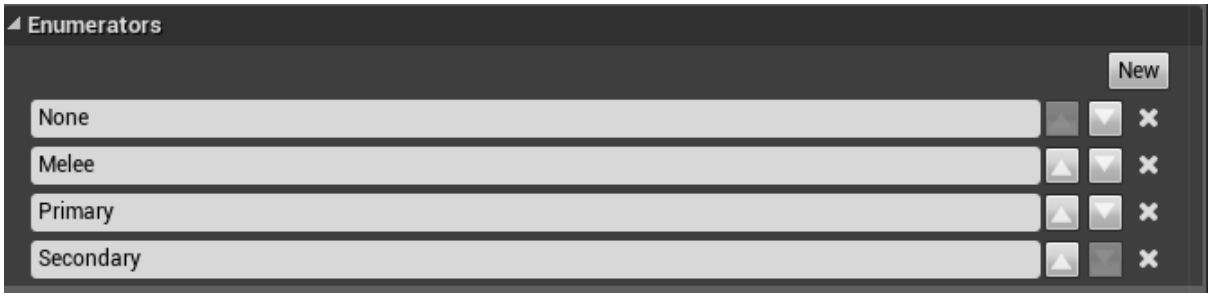


The next step was to add an ‘Actor-Blue Print class’ to the new folder and name it ‘WeaponMaster_BP’.

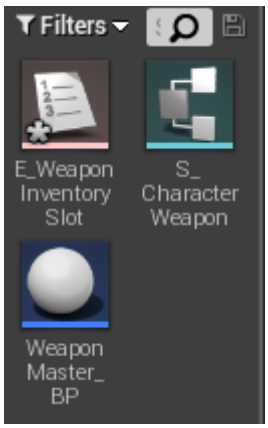


Now I needed to add an ‘Enumeration’ folder named ‘E_WeaponInventorySlot’,

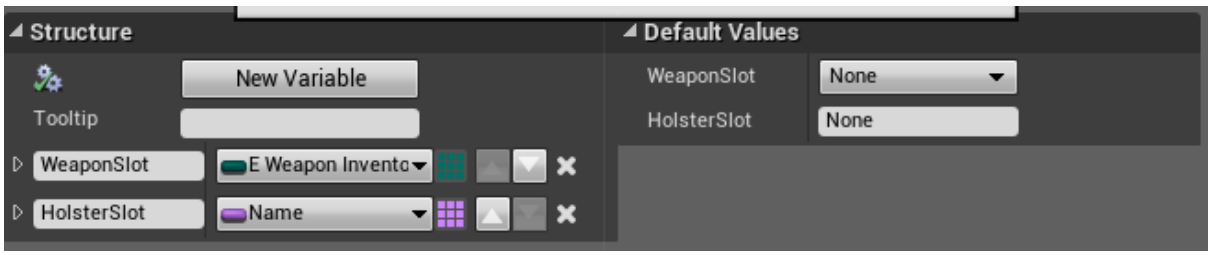
and make four new ‘Enumerators’ these will be my four weapon states. The first weapon state is named ‘none’ for when the character is not holding a weapon. The second is named ‘Melee’ for things such as swords and/or knives. The third named ‘Primary’ for weapons such as rifles and shotguns. Finally, the fourth is named ‘Secondary’ for things such as pistols. However, I will be focusing and completing the ‘Primary’ Enumerator, but the other ones are there for future use.



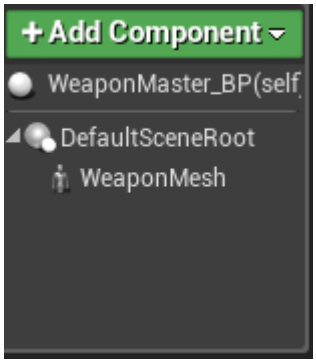
To set up the ‘primary- Enumerator’ I needed to add another component for the ‘holster socket’ and the ‘weapon slot’. To do this I needed to add a ‘structure’ named ‘S_CharacterWeaponSlotStruct’.



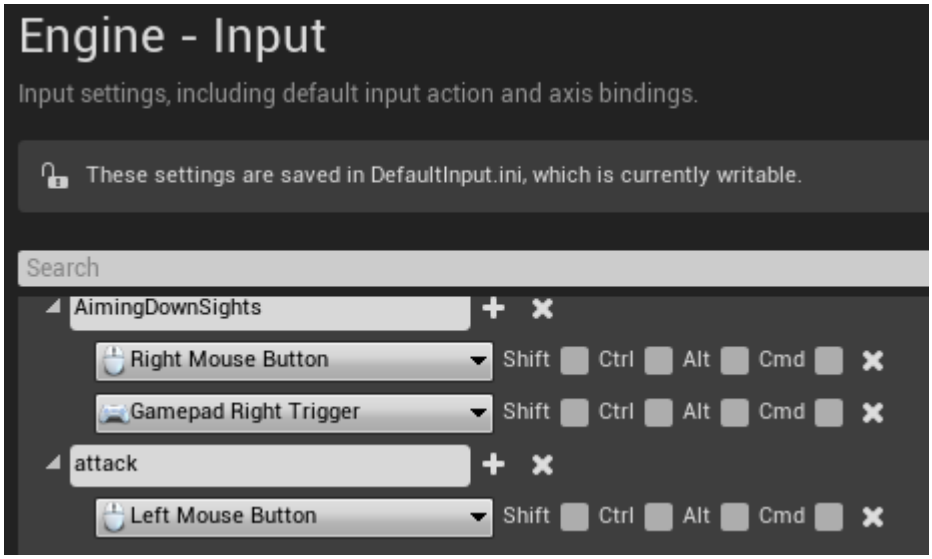
In this ‘Structure’ I needed to make the ‘variable pin type’ ‘E_WeaponInventorySlot’ and name this ‘WeaponSlot’. I also needed to add a new ‘variable’ named ‘HolsterSlot’ with a ‘name- variable pin type’.



Now that this was set up, the next thing I needed to do was set up the characters base weapon, this is done in ‘MasterWeapon_BP’. The first step was to add a ‘skeletal mesh’ component named ‘WeaponMesh’.

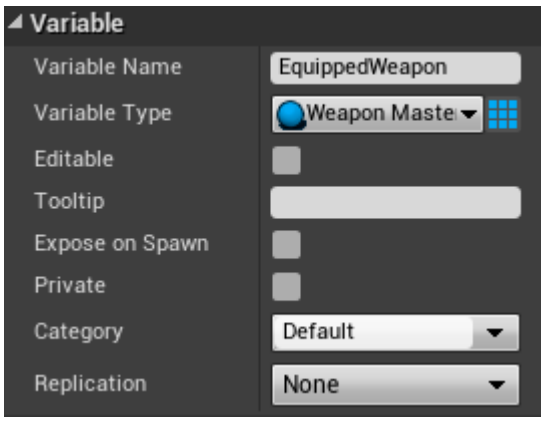


Next, in the events graph I needed something to tell unreal to fire the weapon. The third person character drives this. Before I could make any adjustments to the characters ‘EventGraph’ I needed to set up an ‘input’ named ‘Attack’ that I wanted the shoot button to be, ‘Left Mouse Button’. This is done in the ‘project settings.

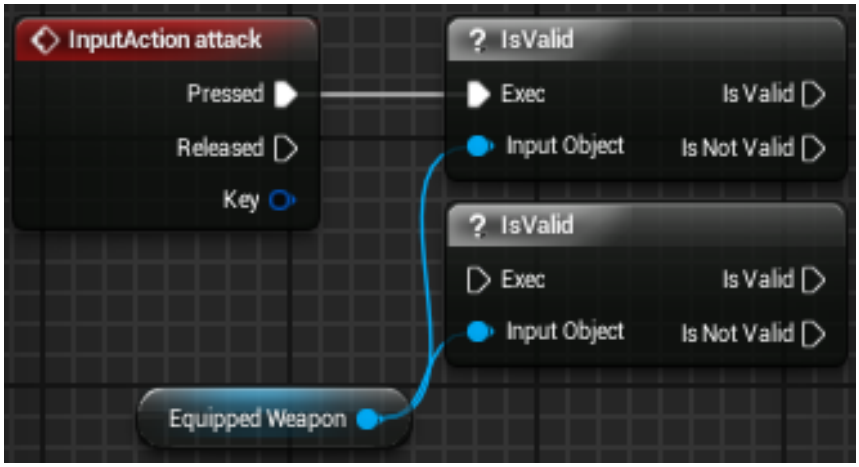


This could now be added to the ‘ThirdPersonCharacter- EventsGraph’ as an ‘Action Event’.

Furthermore, I needed this to check if my weapon ‘Actor’ Is valid. So, I needed to add an ‘IsValid’ after the ‘InputAction’. This will dictate the weapon in use. I already had an ‘Actor’ blueprint made for this however, I needed a new Variable to tell the ‘IsValid’ what it is checking for. So, the new ‘Variable’ has the ‘Variable type- Weapon Master BP’ and is named ‘EquippedWeapon’.



A second ‘IsValid’ was then created and the EquppedWeapon variable was connected to both ‘IsValid’.

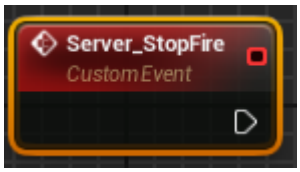


I then needed unreal to be able to notify who is using this weapon. To do this I needed to change the ‘Replication’ of ‘EquippedWeapon’ to ‘Repnotify’.

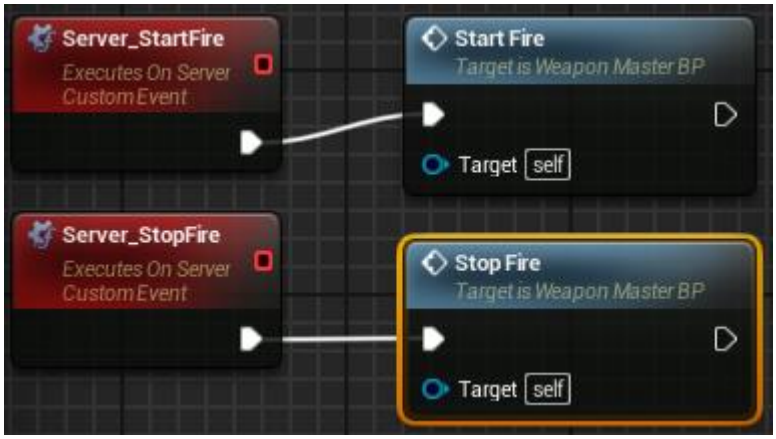
(This will be continued later in the project)

For now, I needed to create a state to drive the ‘Attack input’. So, back in ‘WeaponMaster_BP’ I needed to create a custom event named Server_StartFire’.

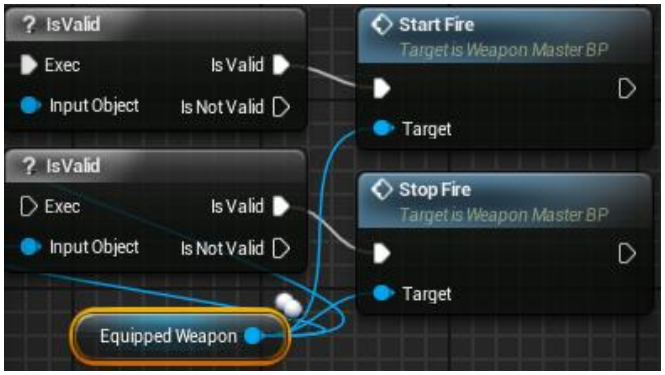
A second one was also created named ‘Server_StopFire’.



These both needed to be set to ‘run on server’, ‘Reliable’ and ‘Call in editor’. Two new functions then needed to be created one named ‘StartFire’ and another named ‘StopFire’ these were then added to the ‘CustomEvents’. As so…

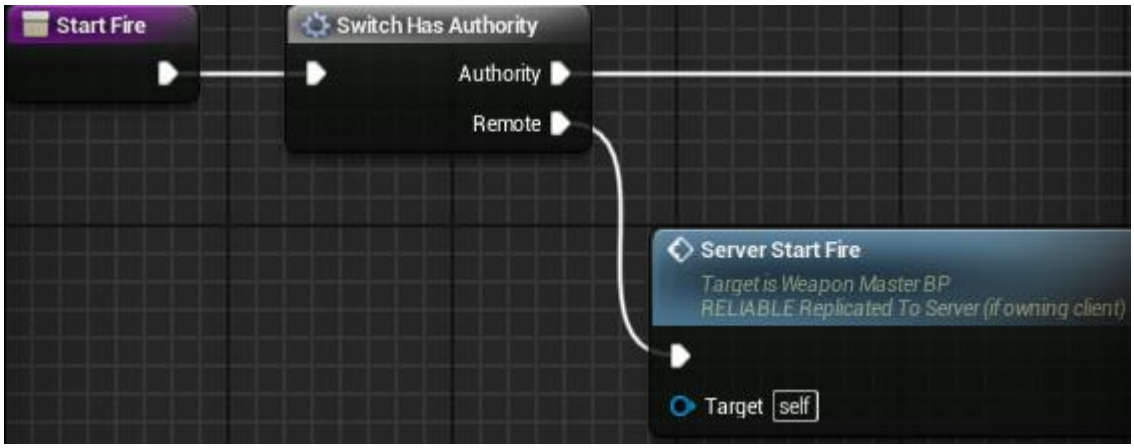


Before writing the ‘logic’ for these I needed to make sure the ‘ThirdPersonCharacter’ could call the logic when the input is made. To do this both ‘Functions’ are added to the ‘ThirdPersonCharacter- EventsGraph’ and connected as so…



To set up the logic for ‘StartFire’-

I first added a ‘Switch Has Authority’ and added ‘Server Start Fire’ to the ‘Remote’.

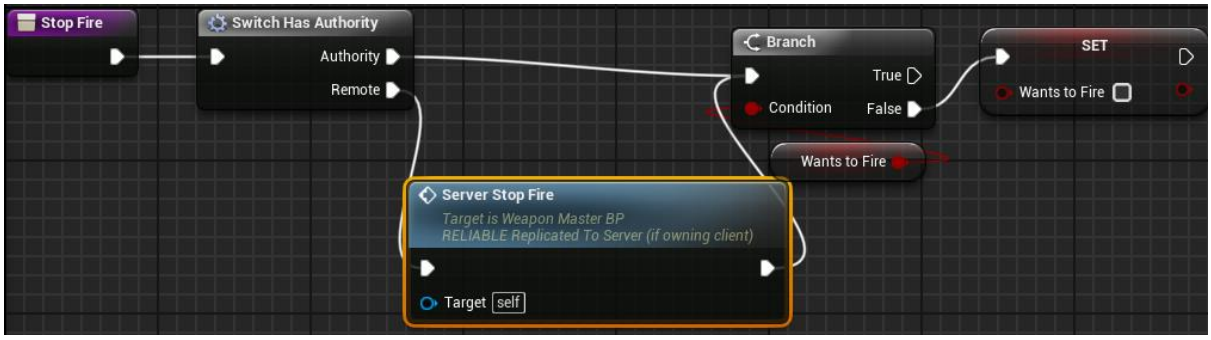


I then needed to create a new ‘Variable’ named ‘WantsToFire’ for the condition on a ‘Branch’.

However, if the ‘WantsToFire’ variable is ‘false’ It needs to be set to ‘true’.

To set up logic for ‘StopFire’-

The same was then done to ‘StopFire’ however the ‘Server Start Fire’ is now a ‘Server Stop Fire’ and the ‘set’ is false.

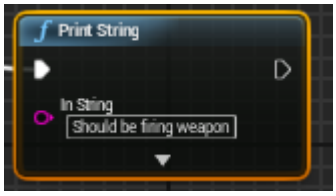


I then created a new 'function' named 'CamerAim' this is where the 'logic' for 'tracing projectiles' this will control where my 'projectiles (bullets)' travel.

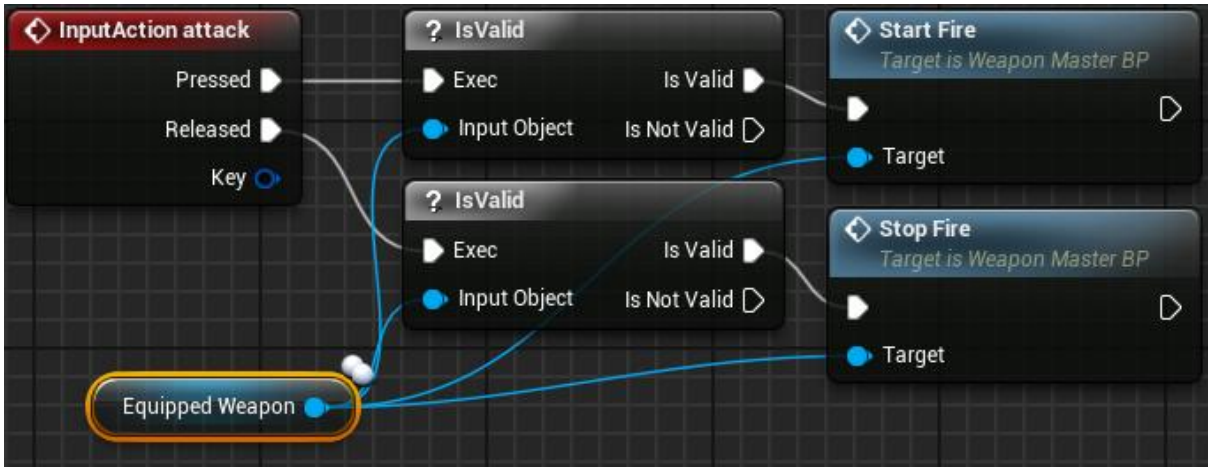
This is the end of my reference video 'TPS Build Part 7 Starting the Weapon Blueprint' (K, Dail. 2015).

Creating Child Actor Class for Primary Weapon

I needed to make the weapon understand which player owns it so it can fire in the game editor. First, I needed the player character to be able to 'talk' to the weapon in the game world. The first step to do this is to add a 'printstring' in the 'StartFire' function named 'Should be firing weapon'.



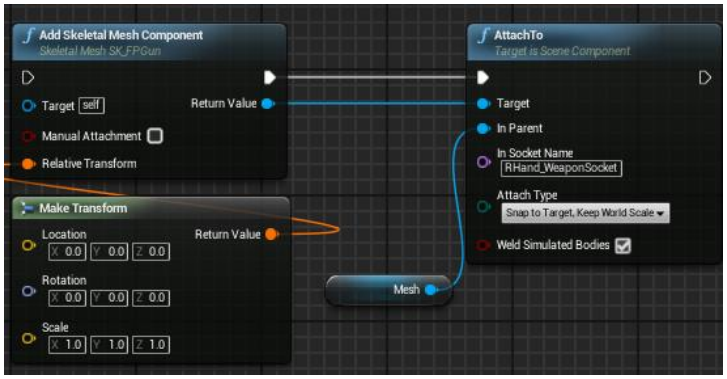
Now at this point the 'Input Action Attack' would never have been passed as 'valid' and that's because my 'Equipped Weapon' variable does not exist in the game world.



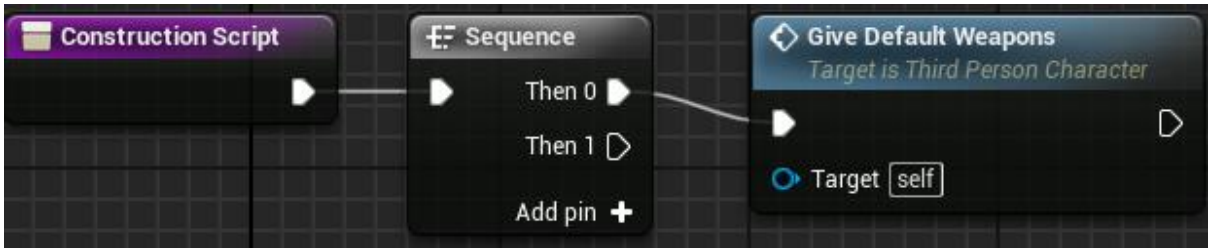
I needed to create a ‘child actor’ to replace this variable. To do this I needed to make a ‘child Blueprint class’ from the ‘WaponMaster_BP’ named ‘PrimaryWeapon_BP’. This ‘Child Blueprint class’ inherits the code from ‘WeaponMaster_BP’ however, extra unique code can be added to this for specific weapon/projectile types.

The primary weapon ‘SK_FPGun’ was then set as the ‘Skeletal Mesh’. This allowed me to separately drag this object into the game editor.

Now I needed the blueprint to spawn in the world. I already made some logic in the ‘ConstructionScript’ of ‘ThirdPersonCharacter’ to call the ‘static skeletal mesh’ to appear in the world and attach to the ‘skeletal mesh’ hand ‘socket’. However, that got detached for now.

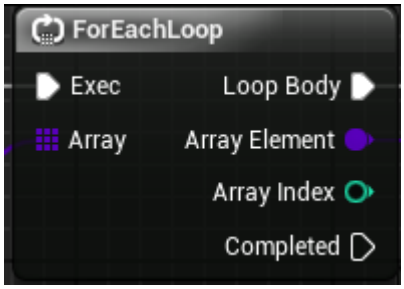


A ‘sequence’ was then added to the ‘Construct Script’ instead, this allows me to write other functions for when the character spawns into the world. A new function named ‘GiveDefaultWeapons’ is required. This function allows for pre-game ‘loadouts’ with the primary weapon attached to the characters main hand and a second weapon on there back.

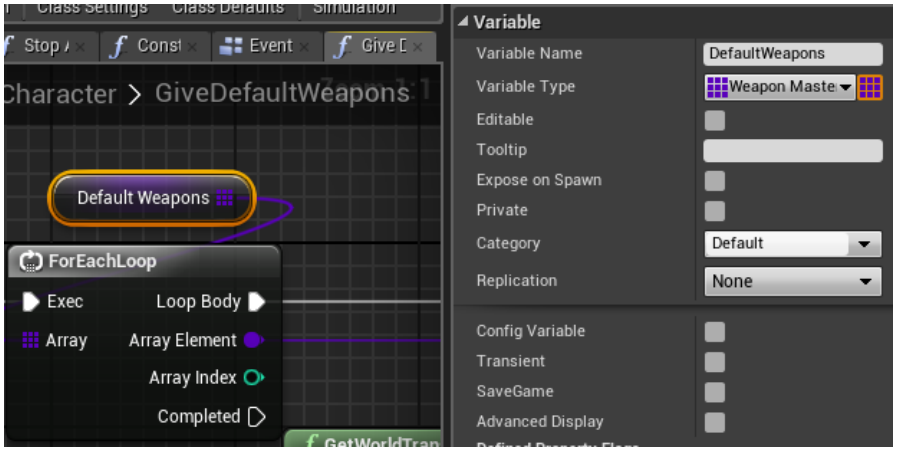


To do this I first needed to write some logic in ‘GiveDefaultWeapons’. The first thing I needed this to do was to check who has Authority with a ‘SwitchHasAuthority’.

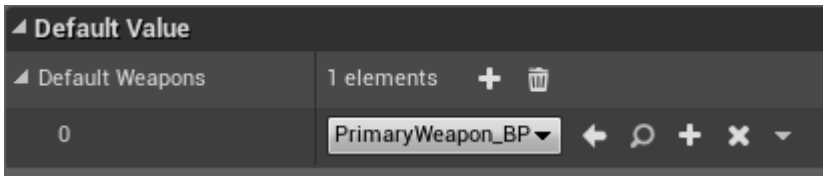
Secondly, I needed to add an ‘ForEachLoop’. This will find everything in an ‘Array’ and do something to everything in that ‘Array’. Once it has done all the logic in the ‘Array’. It will then do a ‘Completed’ And continue from there. An ‘Array’ is a storage space of variables for mechanics such as inventories.



At this point I needed to make a new ‘Array’. This was done by making a new ‘variable’ named ‘DefaultWeapons’ with ‘Weaponmaster_BP’ as a ‘class’ for the ‘variable type’. This was then attached to the ‘ForEachLoop’.



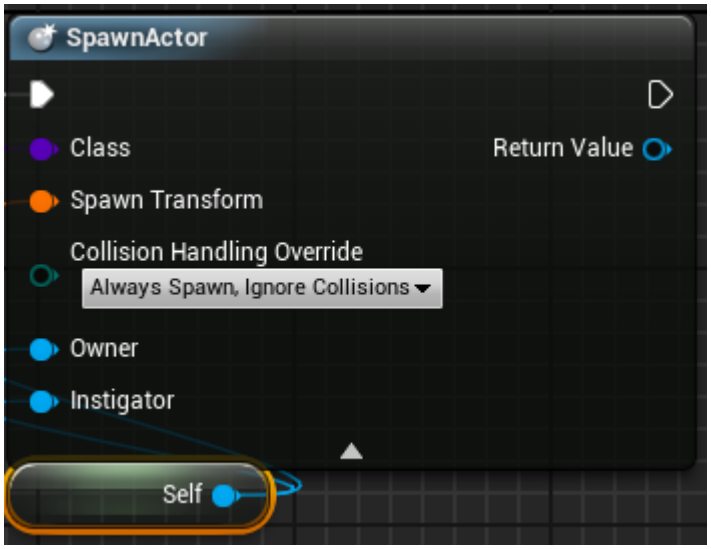
The ‘PrimaryWeapon_BP’ is its ‘Default Value’.



Now the ‘Actor’ needed a space to spawn in the world. First, I needed the ‘mesh component’ that I am using ‘Mesh’.

I then needed a ‘getWorldTransform’,

And from this I needed a ‘Spawn Actor From Class’ with the ‘collision Handling Overide’ set to ‘Always Spawn, Ignore Collision’ so that it is guaranteed to spawn into the world even if it is colliding with something because that could cause it not to spawn in the game world. The ‘SpawnActor’ also needed an ‘owner’ and an instigator, this allows what ever is attacked in the game world gets traced back to the ‘ThirdPersonCharacter’ that spawned it. This requires a ‘self’.

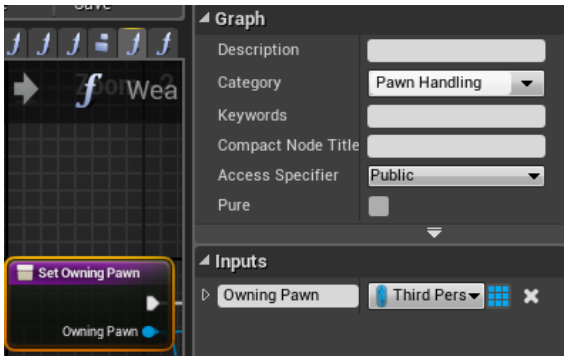


This is the end of my reference video ‘TPS Build Part 8 Creating Child Actor Class for Primary Weapon’ (K, Dail. 2015).

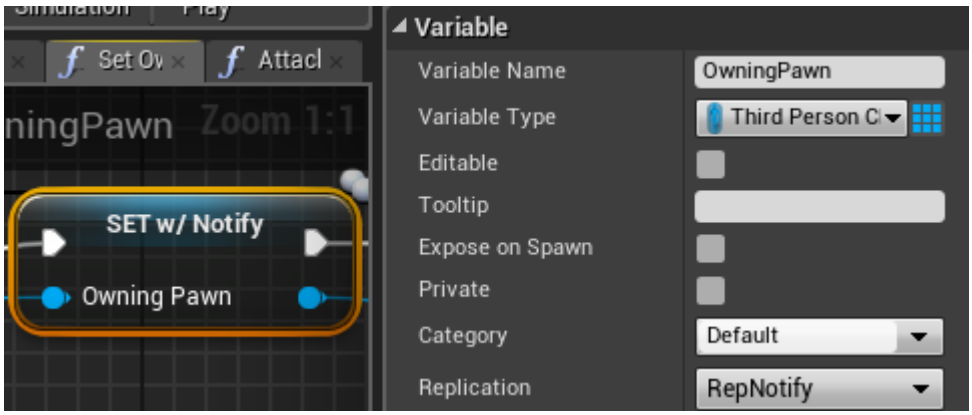
Setting ‘owner pawn’ and ‘attach owner holster’.

In this section I will be showing how I set up the player character to be the ‘owner’ for the ‘target Actor’ this is to pass through logic such as the player has shot an enemy who is the owner of that kill (for example). There are multiple ways to do this however I just stuck to my tutorial reference video to minimise any problems that could possibly occur as the tutorial is to set up the ‘ThirdPersonCharacter’ to play in both single and multiplayer. This was done in the ‘WeaponMaster_BP – EventGraph’.

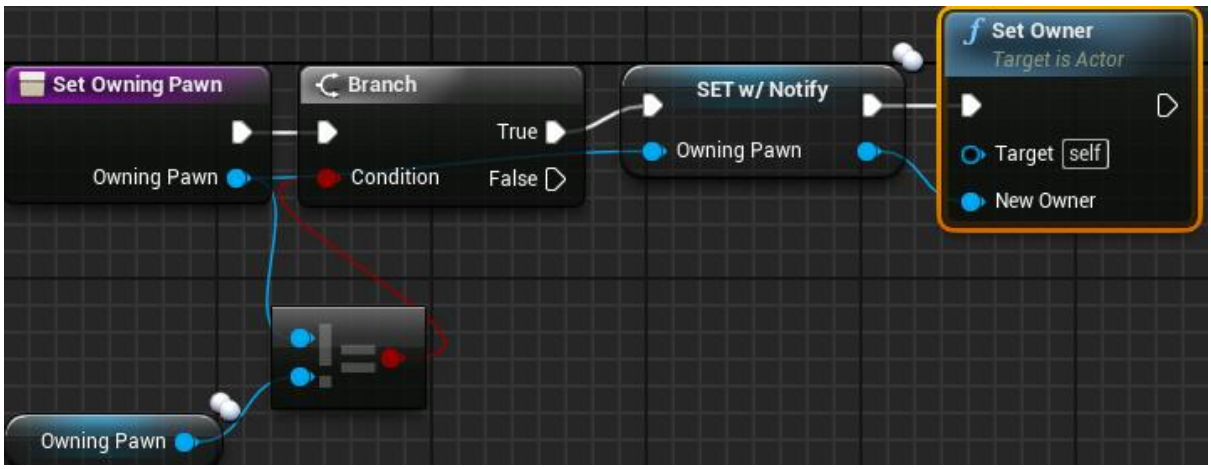
I needed to add a couple of functions that are going to handle the logic of assigning the ‘Actor’s new owner. The first new function was named ‘SetOwningPawn’. When the logic is called the first thing that needs to happen is to find out which character called the logic. The first thing I needed to do was to add an ‘Input’ to ‘SetOwningPawn’ named ‘Owning Pawn’ and set the reference to the ‘ThirdpersonCharacter’.



Next, I needed it to find a comparison. I also needed a variable for the ‘ThirdPersonCharacter_BP’ because it needed to know if it’s the same actor, if it’s not it must be a different actor that’s being assigned. So, I ‘promoted’ the ‘SetOwningPawn’ to a variable named ‘OwningPawn’ and set the ‘Replication’ to ‘RepNotify’.



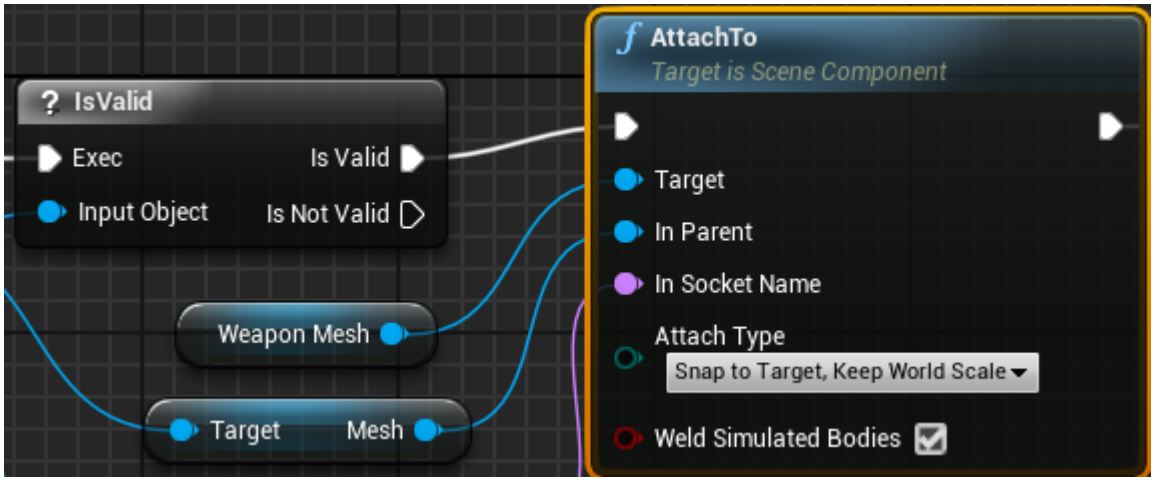
The next thing to do was to add a ‘Branch’ that will ask if the ‘owning pawn’ is not equal to the ‘OwningPawn’ variable and if the condition is true then it needs to set ‘Owning Pawn’ as the owner by using a ‘Set Owner’. This now made it so whoever the ‘Owning Pawn’ was now the owner of the ‘Actor’ Blueprint.



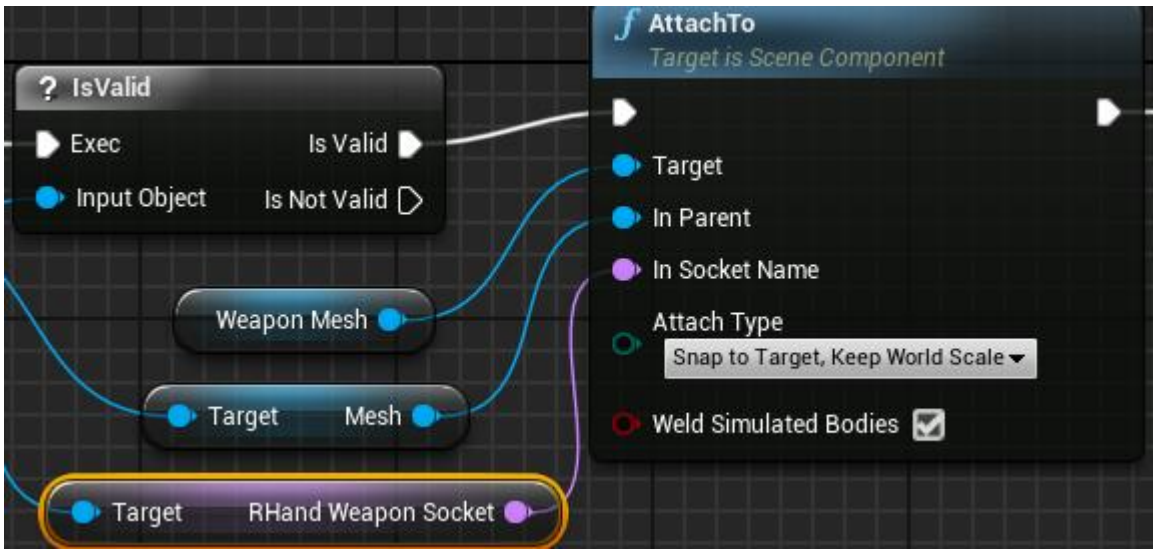
I needed to rewrite some of the Attachment logic, this was done in the second new function added named ‘AttachToOwnerHolster’. The first thing added here was an ‘IsValid’ with the ‘Owning Pawn’ as the ‘Input Object’.

And from the ‘IsValid’ I needed to add an ‘AttachTo(Weapon Mesh)’ with an ‘Attach Type’ of ‘Snap to Target, Keep World Scale’. This tells the ‘WeaponMesh’ to attach to the ‘OwningPawn’.

However, it couldn’t act as the ‘Parent’, so I had to add a ‘get Mesh’ instead.

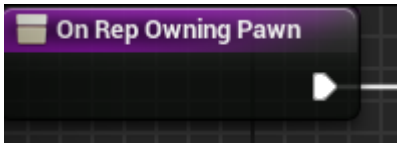


I then added the ‘name-RHand_WeaponSocket’ to the ‘Socket Name’.



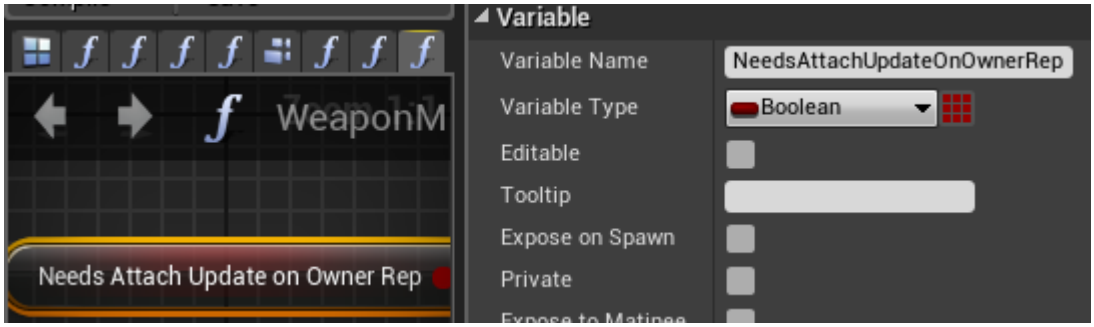
I now needed to make sure that the ‘Owning pawn’ knew that the ‘primary weapon’ is equipped, and the equipped weapon is the ‘target actor blueprint’. So, I added a ‘set Equipped Weapon’ with a reference to ‘Self’.

Now I needed something to happen when all this ‘Is Not Valid’. I needed to find out if the ‘owning pawn’ is ‘valid’ and if not, then I needed something to make the ‘variable’ turn to ‘valid’. This was done in the ‘OnRep_OwningPawn’.

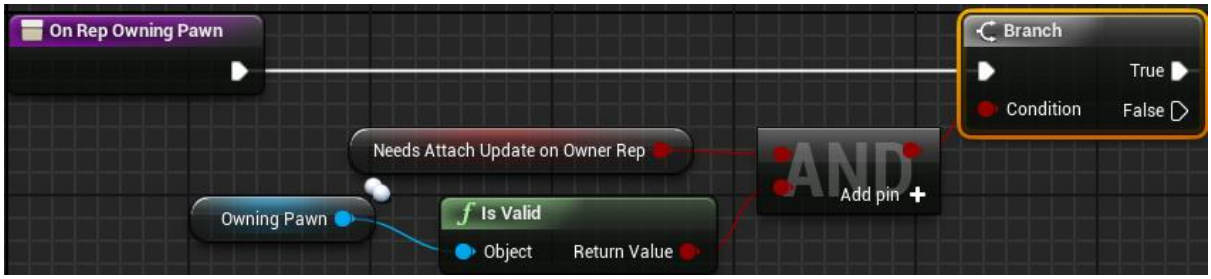


The first thing added here was a ‘Function Is Valid’ on the ‘OwningPawn’ Next, I needed another ‘Branch’ for if this is ‘False’

If it is ‘Valid’ I needed something in ‘RepNotify’ so it will call this again. So, to do this I added a new ‘boolean Variable’ named ‘NeedsAttachedUpdateOnRep’



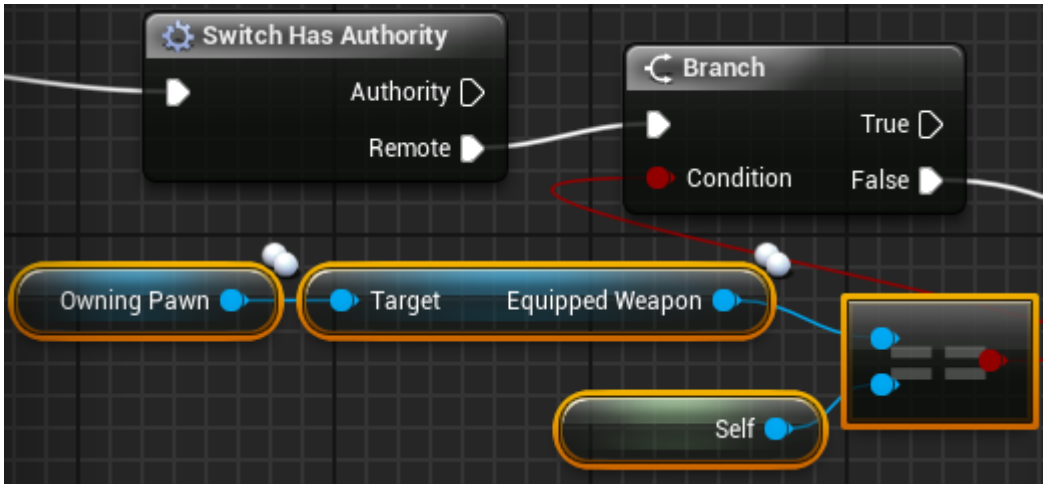
The next thing I needed to add was an ‘And Statement’ as a condition for a ‘branch’.



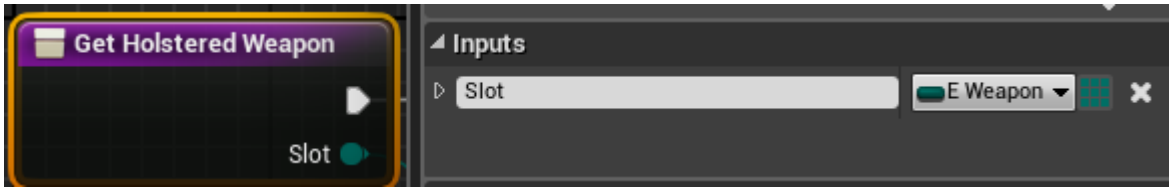
I then dragged in a ‘set’ for ‘NeedattachedUpdateOnOwnerRep’ from the ‘true’ on ‘branch’ and set it too ‘false’.

And from this I needed an ‘switch Has Authority’ with another ‘Branch’ from the ‘Remote’

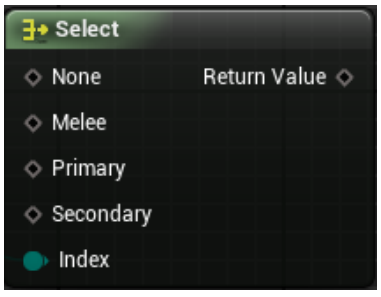
Now I needed the ‘Owning Pawn’ to find the ‘Equipped Weapon’ with an ‘Equal Statement’ for ‘Self’. This was then connected to the ‘Branch Condition’.



Next, I needed to write some more logic for getting the ‘Holstered Weapon’ In the ‘ThirdPersonCharacter_BP’. For this I needed a new ‘function’ named ‘GetHolsteredWeapons’ this is for any of the weapons that are not being used. It needed a new ‘Input’ named ‘Slot’ with the ‘Pin Type’ of E_Weapon Inventory Slot’.



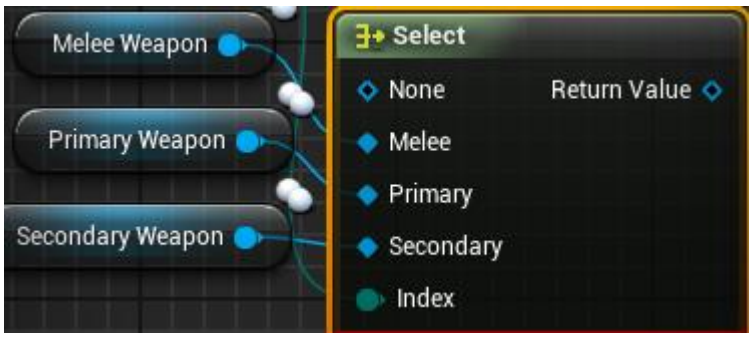
From this I needed to add a ‘Select’ This determines what the ‘Actors’ are going to be inheriting the ‘slots’ for the Weapons



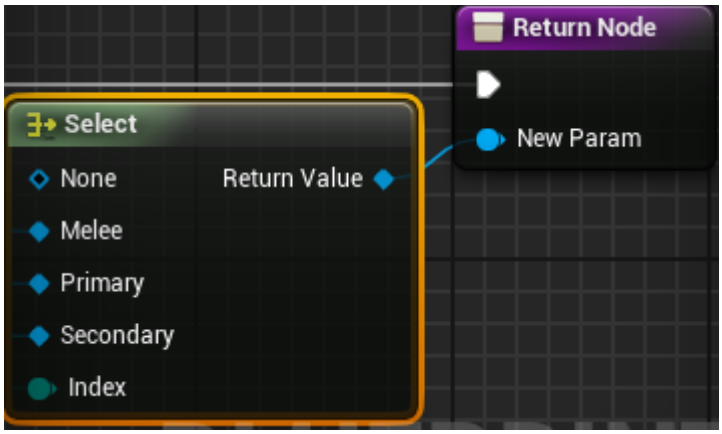
The first step I am going to have to make a new ‘Actor’ for each ‘slot’ but they are all based on the ‘BaseWeapon_BP’. The first step to do this was to make tree duplicates of the ‘Equipped Weapon Variable’, The first named ‘MeleeWeapons’,

The second named ‘PrimaryWeapon’

And the third named ‘SecondaryWeapon’

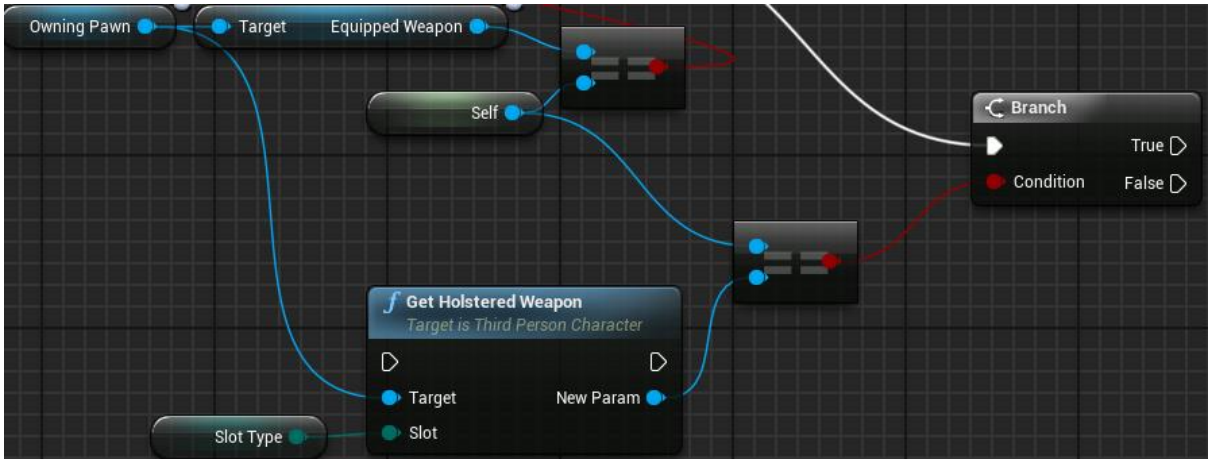


The ‘select’ now needed to be able to talk to the ‘WeaponMaster_BP’ this was done using a ‘ReturnNode’ set to ‘Pure’.



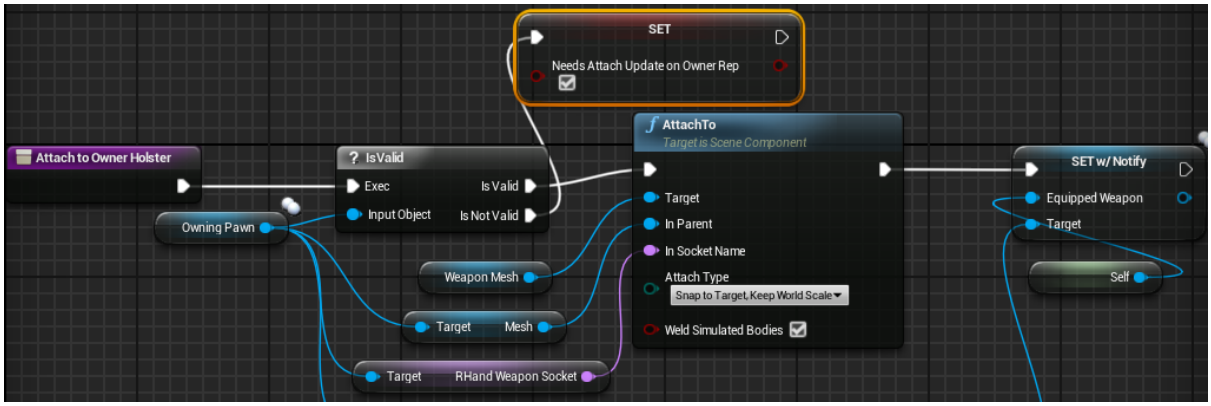
Now, back on the ‘WeaponMaster_BP’, from the ‘OwningPawn’ I could add a ‘Get Holstered Weapons’ with the ‘SlotType’ to drive it.

Then I needed the ‘Self’ Actor to be the same that is currently equipped in the weapon ‘Slot’. So, I added an ‘Equals’ from ‘Get Holstered Weapons’ and attached it to a new ‘Branch condition’.

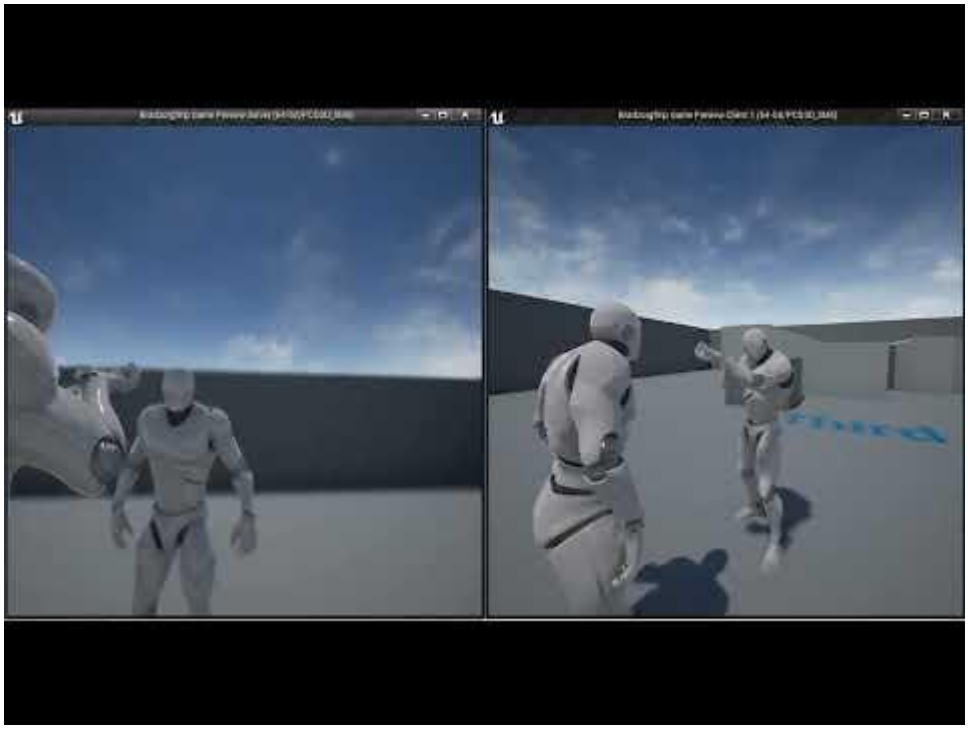


I then added and attached the ‘function’ named ‘AttachToOwnerHolster’.

Now, back in the ‘AttachToOwnerHolster’ function the ‘NeedsAttachUpdateOnOwnerRep’ variable was then added to the ‘Is Not Valid’ and set to ‘True’.



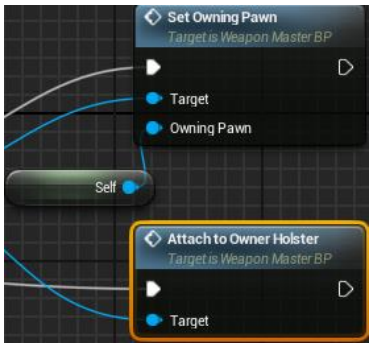
The character is now set up with a simple inventory. This is the result of my reference video there is no weapon as there isn't one assigned yet. 'TPS Build Part 9 Blueprint Owner Logic Between Character and Weapon Actor' (K, Dail. 2015).



Starting Player Camera and Weapon Actor Shooting

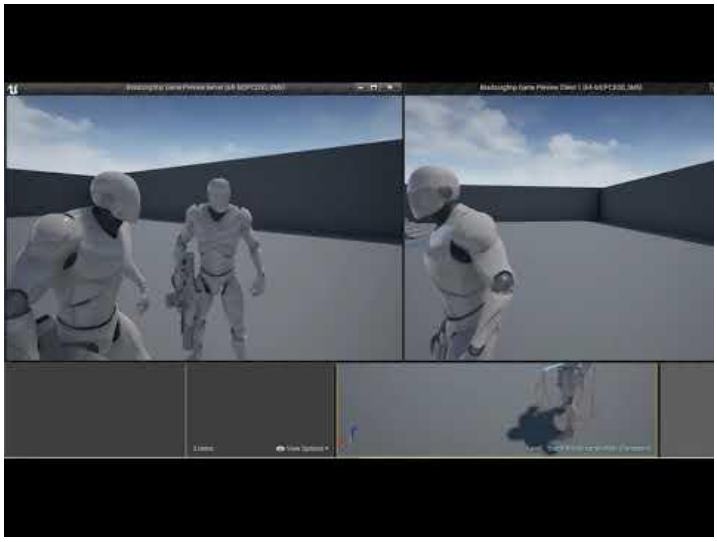
With the main 'blueprint' logic out of the way for the weapon 'Actor'. I needed to get the last bit of the 'character blueprint' done. I also had to set a couple more lines of logic in the 'player blueprint' which will control the 'line tracing' ability from the camera to the 'player characters' muzzle area. I also needed the 'character' to be able to shoot something out of his weapon.

The first thing to do here was to add a 'set Owning Pawn' set to 'self' in the 'ThirdPersonCharacter' function 'GiveDefaultWeapons' after the 'SpawnActor' alongside an 'AttachToOwnerHolster' connected from the 'ForEachLoop' 'completed'.

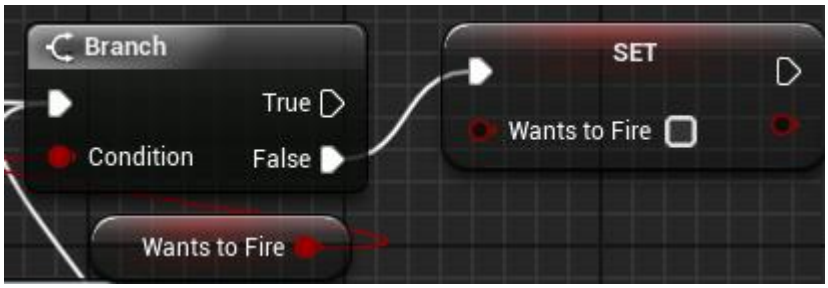


Now I needed to go into the ‘EventsGraph’ and add a ‘Event BeginPlay’ and connect the ‘GiveDefaultWeapons’ function to it.

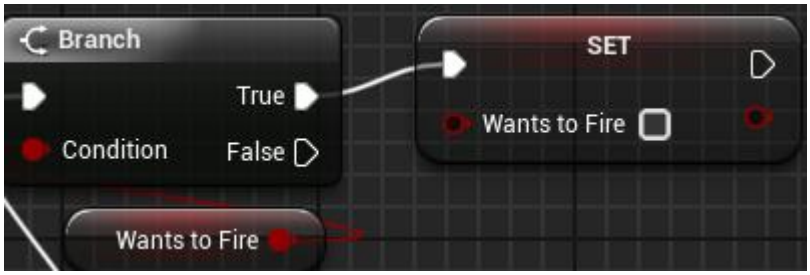
This will now allow the ‘Actor’ to spawn in the game world for all characters.



At this point I was ready to set up the weapons being able to fire. However, the ‘StopFire’ function was only allowing for the character to fire once this was because the logic was already reading its ‘branch’ as ‘false’.



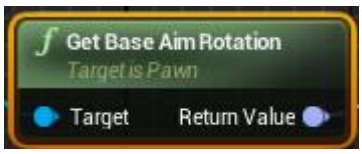
To fix this I just needed to ‘branch’ it from ‘true’ and not ‘false’ and this fixed the issue.



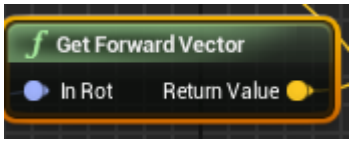
Now I started the process of allowing the player to start shooting some kind of ‘line tracer’. The logic that is going to drive the firing is the player is going to shoot the weapon ‘Actor’ and this is going to shoot the ‘logic’ In the ‘CameraAim’ function. The first thing added to the ‘CameraAim’ function is a new ‘variable’ with the ‘variable type’ set to ‘Vector’ named ‘AimVector’. this is the characters position in the world.

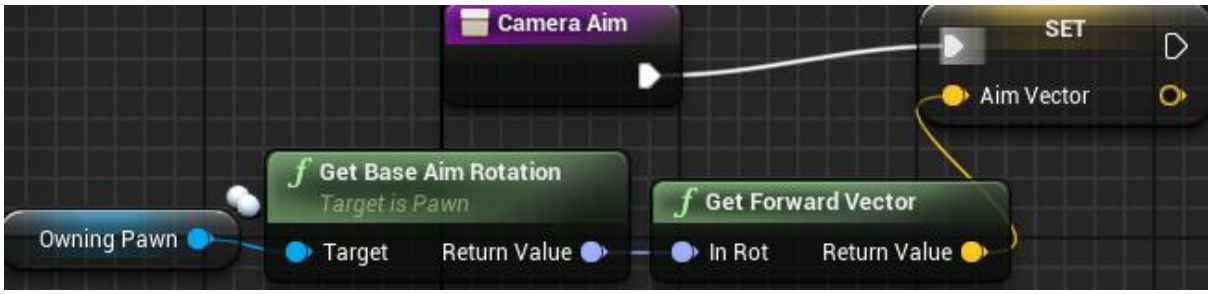
I now needed a starter Numerical value. This was done by dragging in the ‘OwningPawn’ variable,

then I added a ‘Get Base Rotation’ from this,

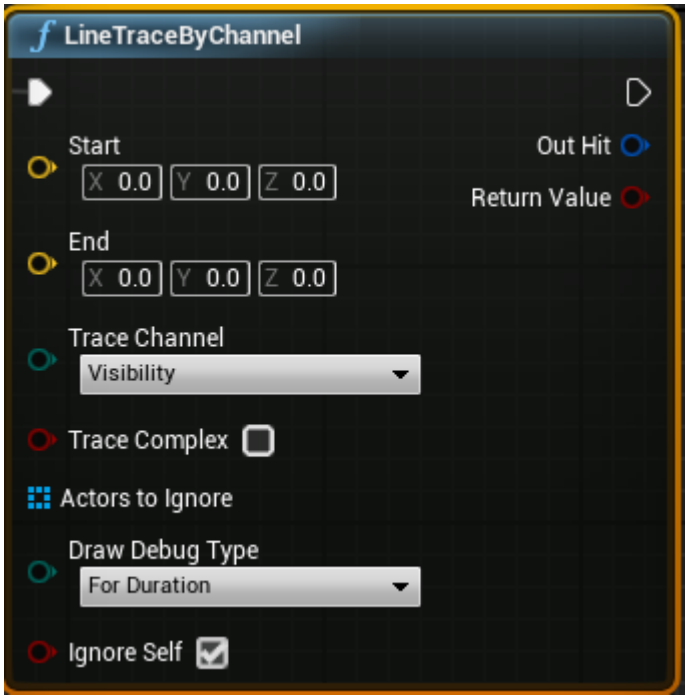


And lastly from that I added a ‘Get Forward Vector’. Once connected to the ‘Aim Vector’ this will tell unreal which way the character player is facing.

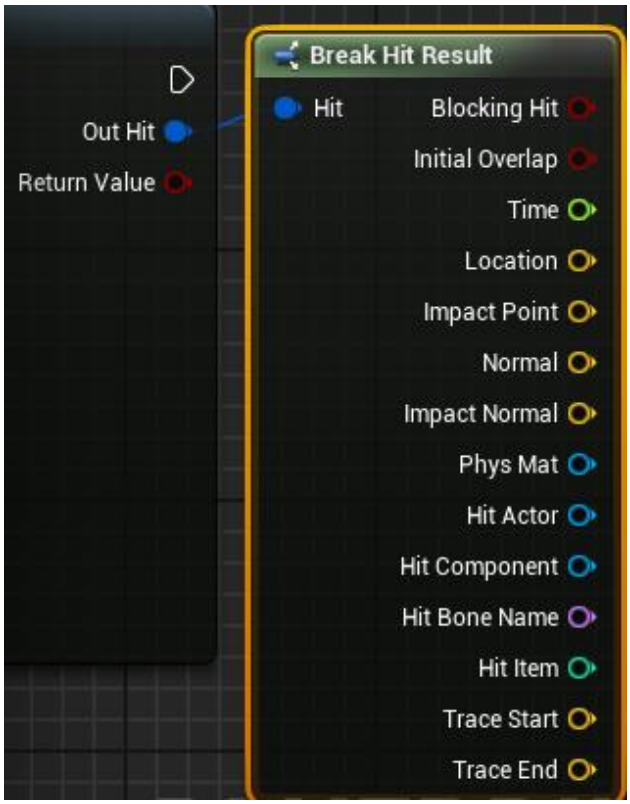




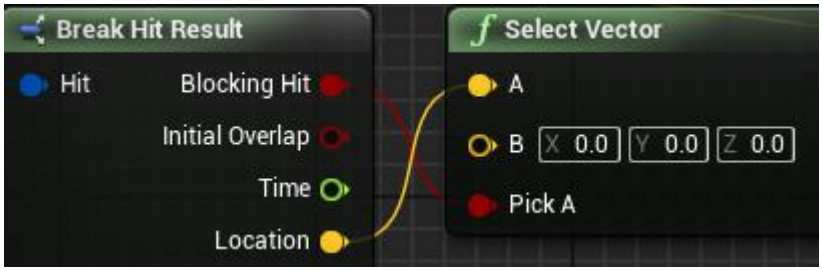
Next, I needed to set up the ‘debug tracing’. So, from the ‘Aim Vector’ I added a ‘LineTraceByChannel’ with the ‘Draw Debug Type’ set to ‘For Duration’ so I could see it in the world if I went to shoot it.



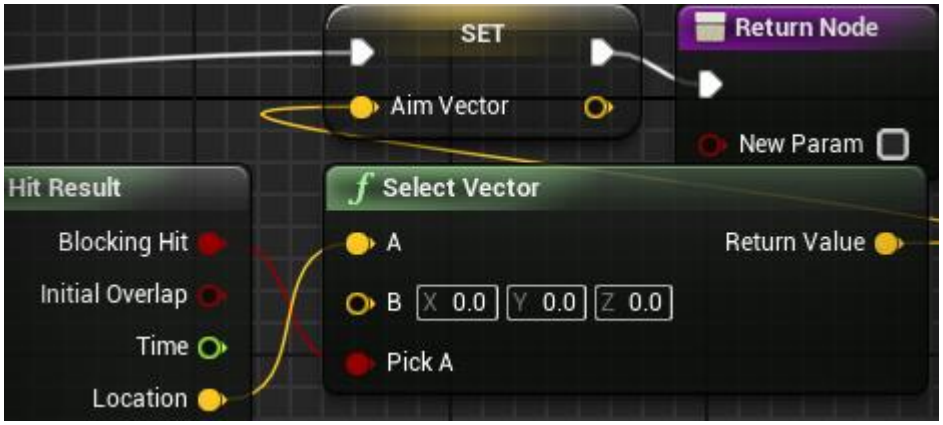
From this I needed to find out what the projectile would be hitting. So, from the ‘Out Value’ I added a ‘Break Hit Result’ to show all the results that its hitting something.



from this point I needed to find out what the character location is and compare it to another value, but I needed this to stop bullets traveling indefinitely. So, from ‘Blocking Hit’ I added a ‘select Vector’ function and connected the ‘location’ from the ‘Break Hit Result’ to ‘A’ on the ‘select Vector’.



From this I needed another ‘Set Vector’ and a ‘ReturnNode’.

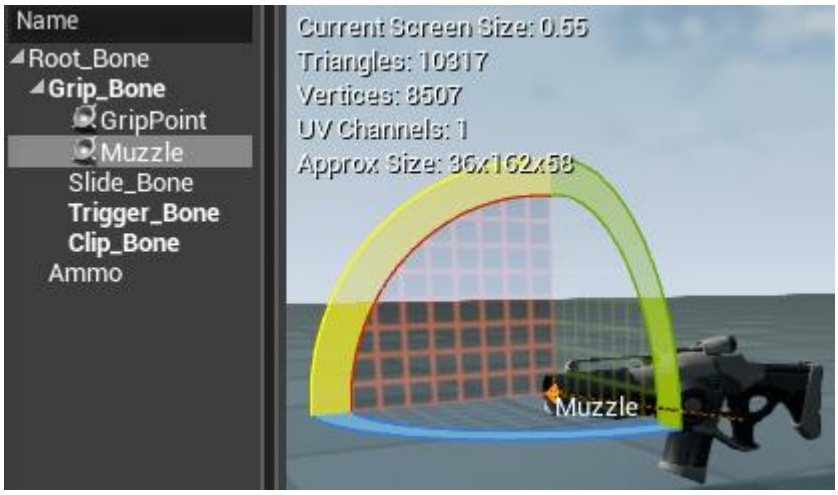


I now needed to create a new ‘Pure Function’ named ‘GetMuzzleLocation’ and the first thing I had to do was make a new ‘Name Variable’ named ‘MuzzleSocket’ and add this in the new function alongside the ‘WeaponMesh’.

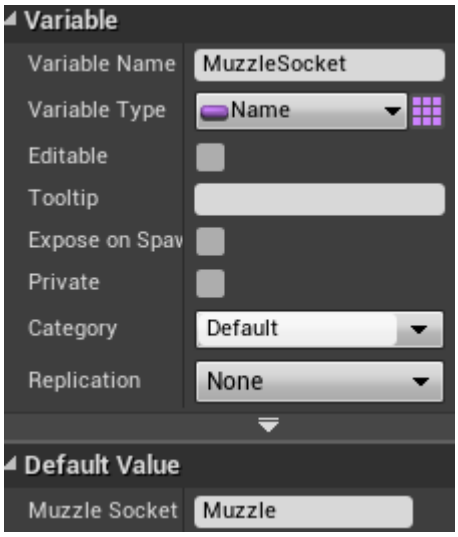
From the ‘WeaponMesh’ I then added a ‘Get Socket Location’ and attached the ‘MuzzleSocket’ variable to the ‘Socket Name’

I now needed to add a ‘Return Node’ as a ‘Vector’ named ‘Muzzlelocation’ and this will be driven by where the ‘Socket Location’ is.

Now I needed to find out what ‘socket’ I needed from the ‘WeaponMesh’, this is located in the ‘SK_FPGun_Skeleton’.

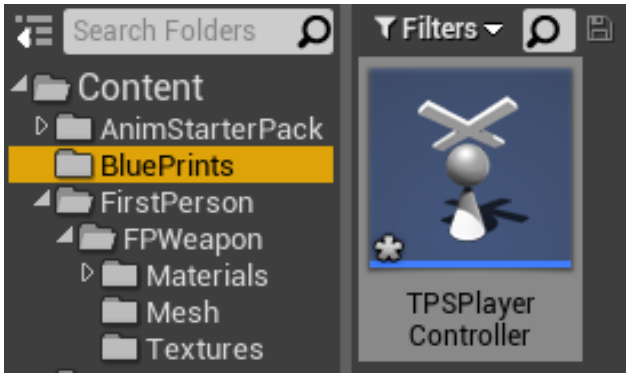


Once I found it, I then copied and pasted the name into the ‘MuzzleSocket’ variable ‘Default Value’.



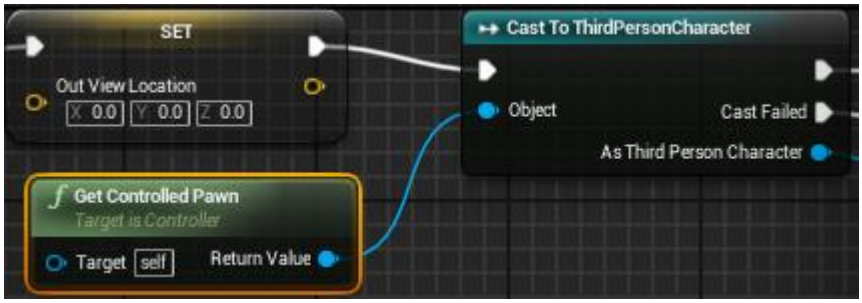
Now back in the ‘CameraAim function’ I could us a ‘get Muzzle Location’ as the ‘start’ on ‘LineTraceByChannel’.

The Next step was to build a new ‘BluePrint’ so, I made a new folder in the ‘content’ folder named ‘BluePrints’ and within this I made a new ‘BluePrint Class’ as a ‘Player Controller’ named ‘TPSPlayerController’.

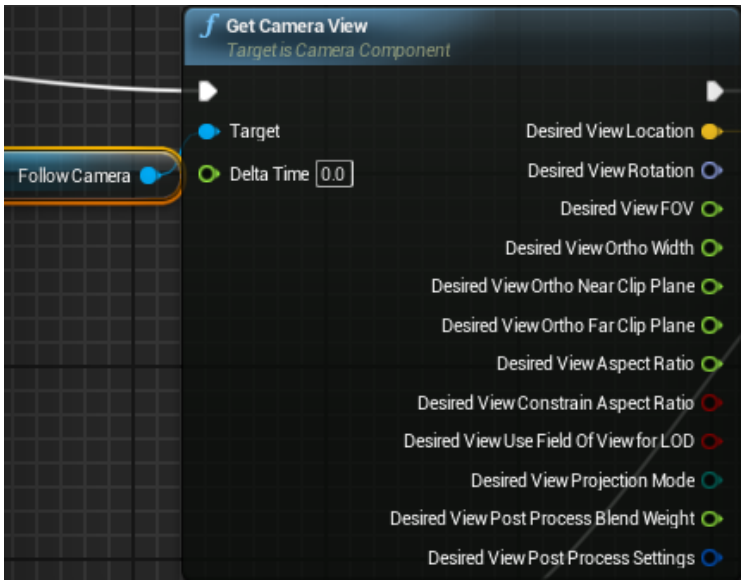


In here I need to establish the variables I needed to be able to call them in the Weapon BluePrint. For this I needed to add a new function named ‘ViewPoint’. I also needed a new ‘Vector variable’ named ‘OutViewLocation’ I first dragged this into the ‘viewPoint’ function as a ‘set’.

I then added a ‘Get Controlled Pawn’ separately and from this I needed to find out who is playing what character, either a ‘client’ or a ‘server’ so from this I added a ‘Cast To ThirdPeronCharacter’.



From this I needed the camera that is associated with the ‘ThirdPersonCharacter_BP’ which is the ‘Follow Camera’ and from this I added a ‘Get Camera View’



And the ‘desired View Location’ is connected to the ‘Set OutViewLocation’. This will now allow the ‘CameraView’ to set the ‘OutViewlocation’ for this ‘Vector’.

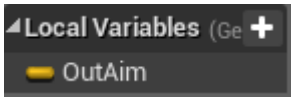
I now needed something in place to drive it alongside this as well just in case the ‘Cast’ fails. To do this, from ‘Cast Failed’ another ‘Set Viewlocation’ is added with a function of ‘Get Focal Location’.

And finally, a ‘Return Node’ was added at the end.

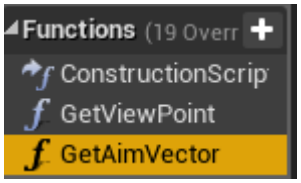
This was the end of my reference video ‘TPS Build Part 10 Starting Player Camera and Weapon Actor Shooting’ (K, Dail. 2015).

Finishing camera Aiming

So, the first steps to continuing the camera Aiming was to create a new ‘LocalVariable’ with the ‘type’ set to ‘Vector’ named ‘OutAim’

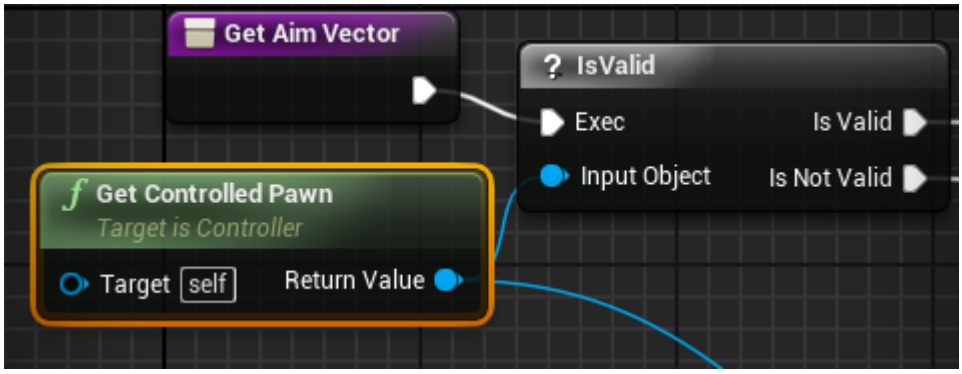


A new function was then added named ‘GetAimVector’.



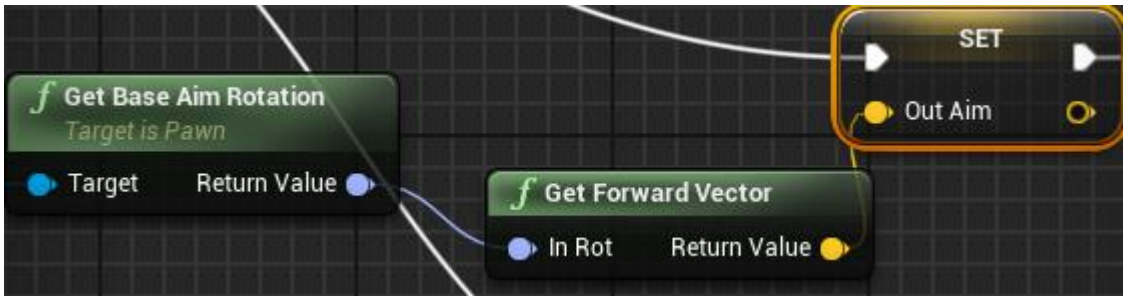
To start off the logic in this function an ‘IsValid’ was added.

The next thing added was a ‘GetControlledPawn’. This was then connected to the ‘Input Object’.



The ‘Local Variable – OutAim’ was then added in as a ‘set’

To set the number in this I needed to drag from the ‘GetControlledPawn’ to add a ‘GetBaseAimRotaion’ and from this I added a ‘Get Forward Vector’ this then gave the set value needed in ‘OutAim’.

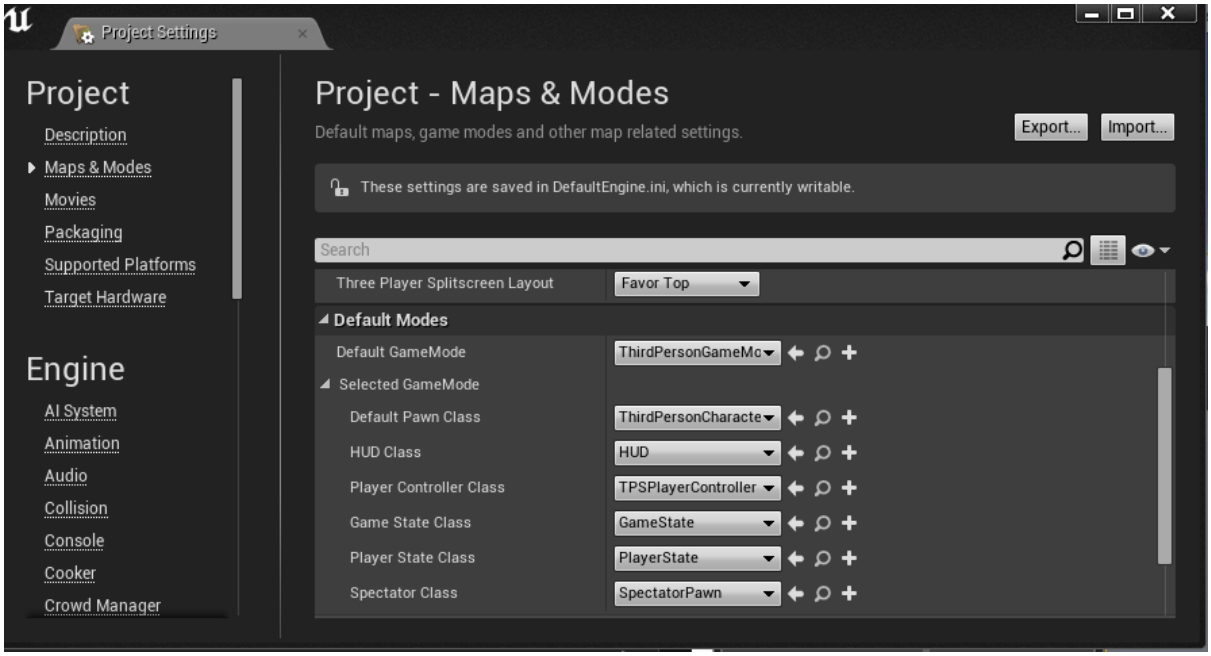


Now that the ‘IsValid’ was set, I needed to set what would happen if it was ‘IsNotValid’. So, to do this I added another ‘Set – OutAim’ and a ‘Get ControlledRotation’ alongside a ‘GetRotationXVector’.

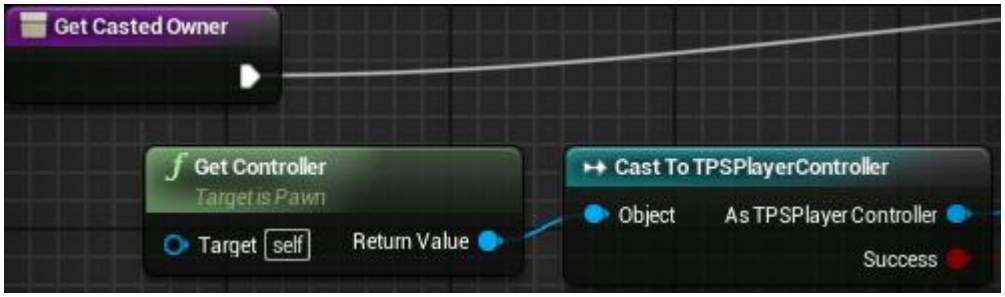
From here I created a ‘return node’ with the ‘OutAim’ variable connected.

The next step was to go back into the ‘GetViewPoint’ function and connect the variable ‘OutViewLocation’ to its ‘ReturnNode’. These two functions were then completed.

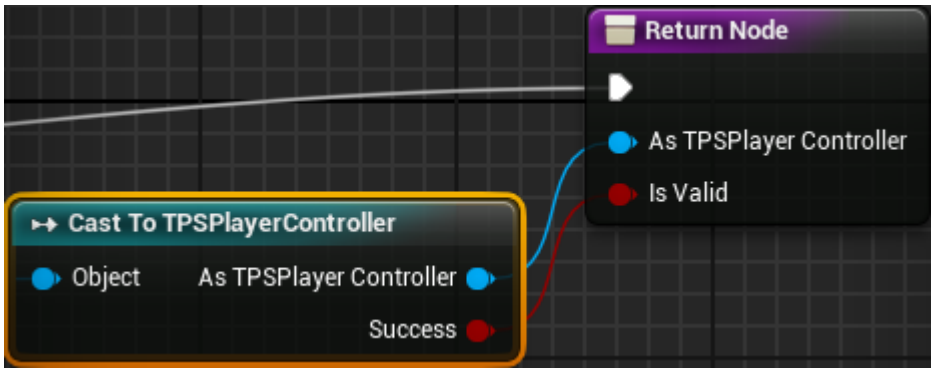
The next step in this aiming process was to go into the project settings under ‘Maps & Modes’ and change the ‘Player Controller Class’ to the folder that I made not long ago ‘TPSPlayerController’.



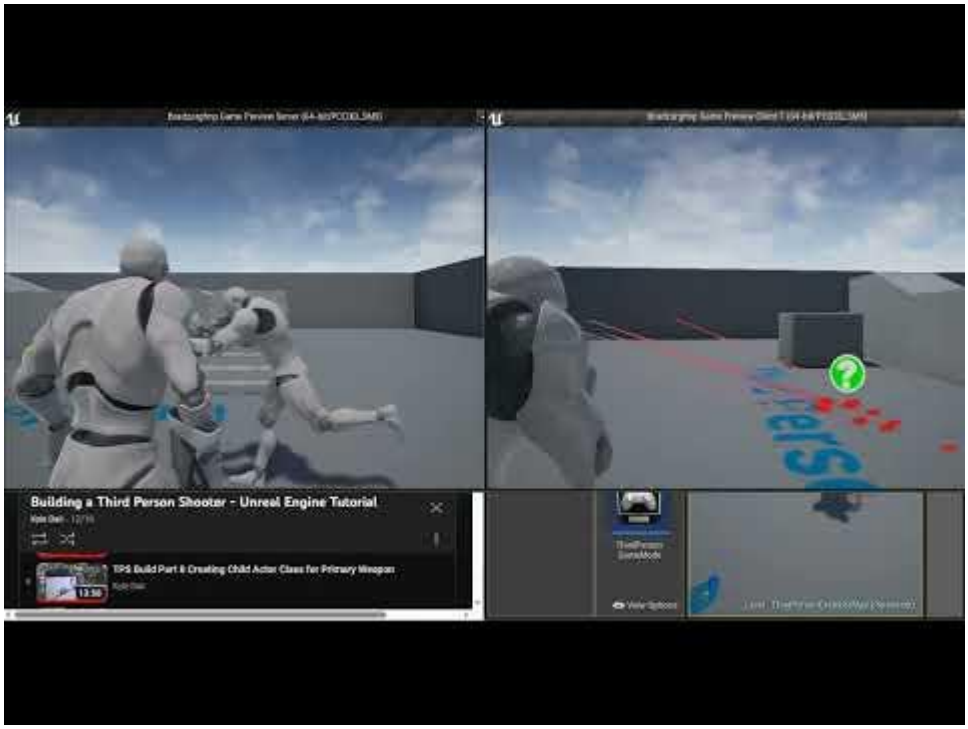
After doing this I created a new ‘Pure’ function named ‘GetCastedOwner’. Within this new function I first added a ‘Get controller’ with a ‘Cast To TPSPlayerController’ this was convered into a ‘Pure Cast’.



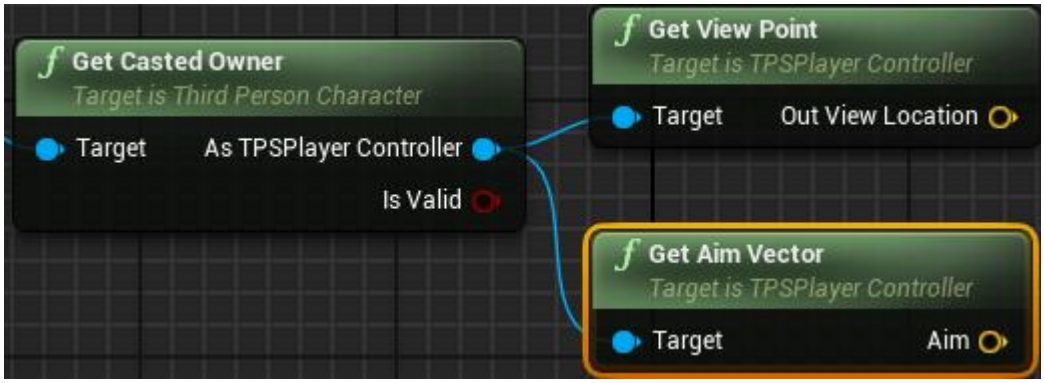
Both out puts were then conneceted to a newly made ‘ReturnNode’ with the ‘Success’ named ‘IsValid’.



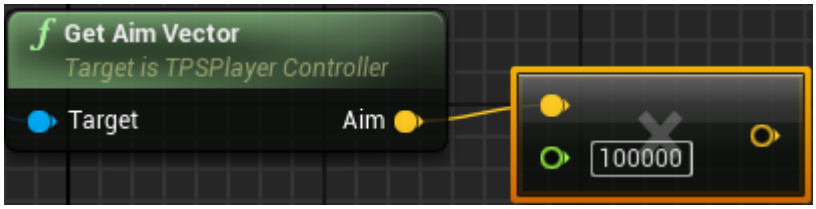
At this point when I ‘Preview’ the game I could aim and I could shoot however, all the ‘LineTracers’ go to world (0,0).



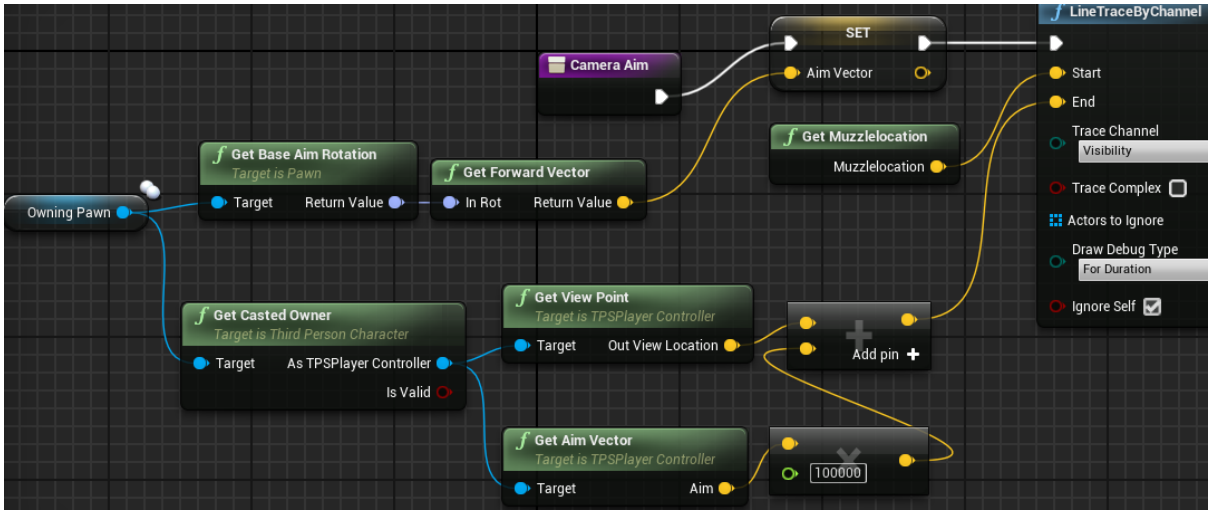
This meant more work was necessary in the function ‘CameraAim’. So the first step to fix this was to drag off of the ‘OwningPawn’ to add a ‘GetCastedOwner’ and from this I added a ‘GetViewPoint’ alongside the ‘GetAimVector’. These were both then set to ‘Pure’.



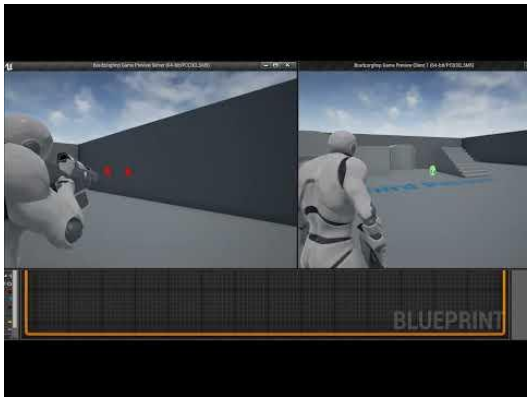
From ‘Get Aim Vector’ I then added a ‘Vector * Float’ the multiply value was the set to ‘100,000’ this adds a stopping point.



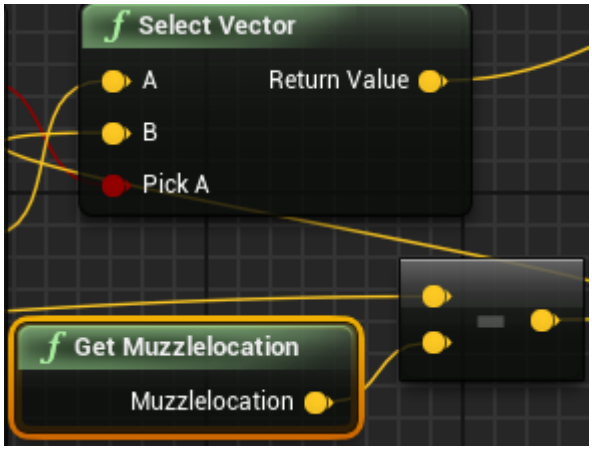
From this a ‘Vector + Vector’ was then added into the ‘GetViewPoint’ and this ended up becoming the end point.



Now at this stage when I ‘Previewed’ the game the ‘Line Tracers’ followed the Camera centre point.

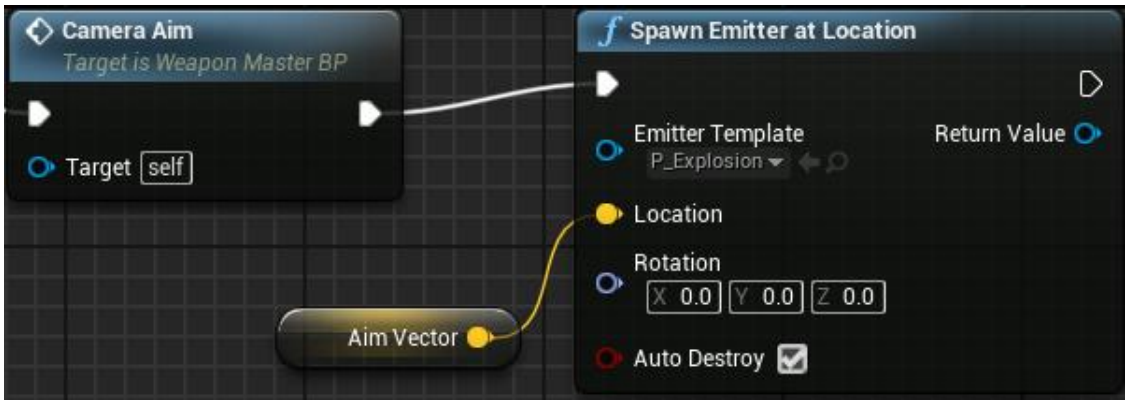


However, the Line Tracer would just go on forever so to fix this ‘Vector – Vector’ was simply added with the ‘GetMuzzleLocation’ and connected to the B value in ‘Select Vector’.

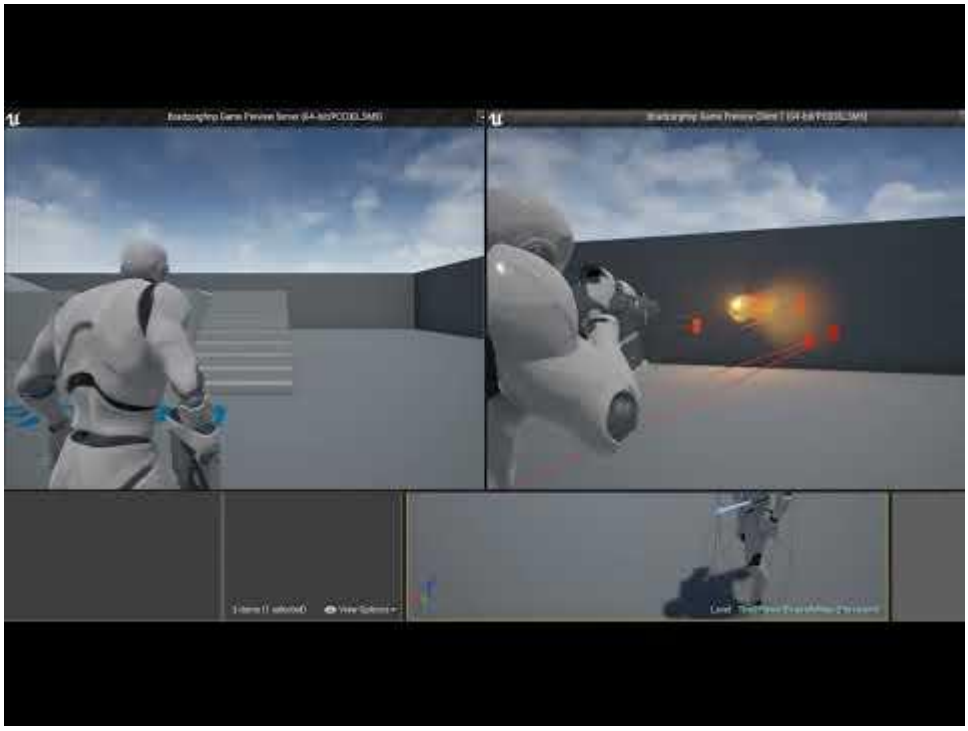


This concluded the logic for aiming.

The last thing to do in this section was to have something shoot out of the gun. This was done in the ‘WeaponMaster_BP’ under ‘StartFire’. A ‘Spawn Emitter at Location’ with the ‘AimVector’ set as its ‘location’ and the Emitter Template set to ‘P_Explosion’ was added to the end of the logic.



This was the end of my reference video ‘TPS Build Part 11 – Finishing Camera Aiming’ (K, Dail. 2015). This was the result.

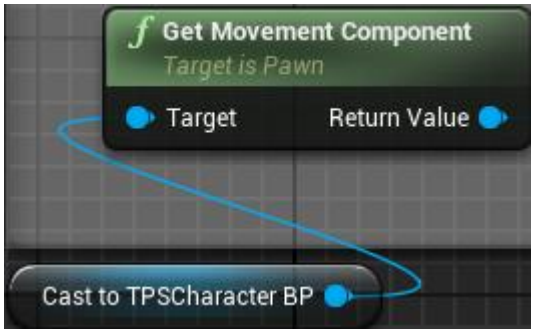


Making the Character Look In Direction of Camera

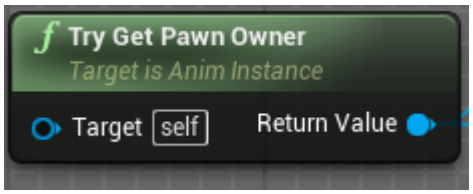
At this point most of the work was out of the way for getting the weapon established, being able to shoot, walking around and aim up and down. In this next section I will be making the characters head look towards where the camera is looking.

A slight future problem arose, that being associated with the ‘Cast To ThirdPersonCharacter’. If I was to use these and move them around, they could cause future errors. So, to prevent this I promoted it into a variable named ‘CastToTPSCharacter_BP’. This made it so I could just copy the variable and change the ‘Cast to’. This would make all these promoted variables around unreal change at the same time, saving a lot of hassle.

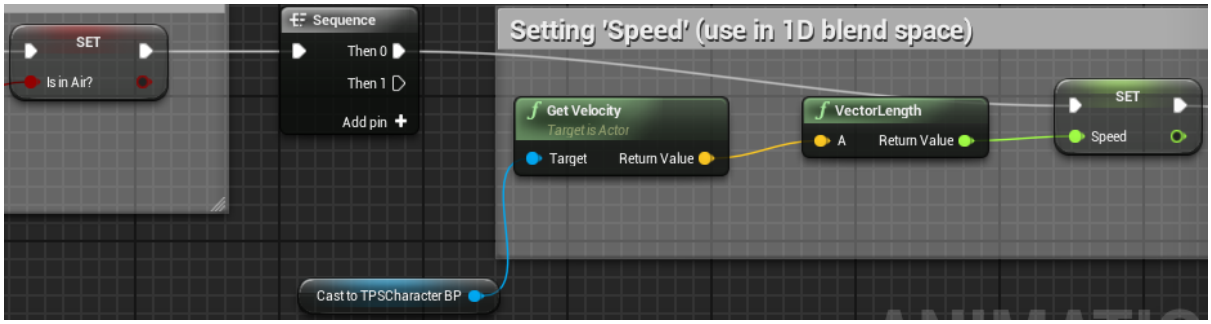
I then made a bunch of changes to the ‘EventsGraph’ in the ‘ThirdPerson_AnimBP’. The first change was to connect this new variable ‘Cast to TPSCharacter_BP’ to the ‘Get movement Component’ in the comment ‘Set 'IsInAir' (used in state machine)’



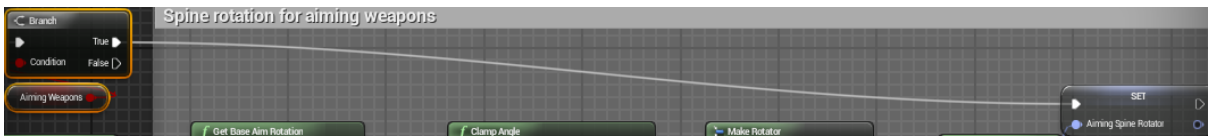
Instead of the ‘Try Get Pawn Owner’



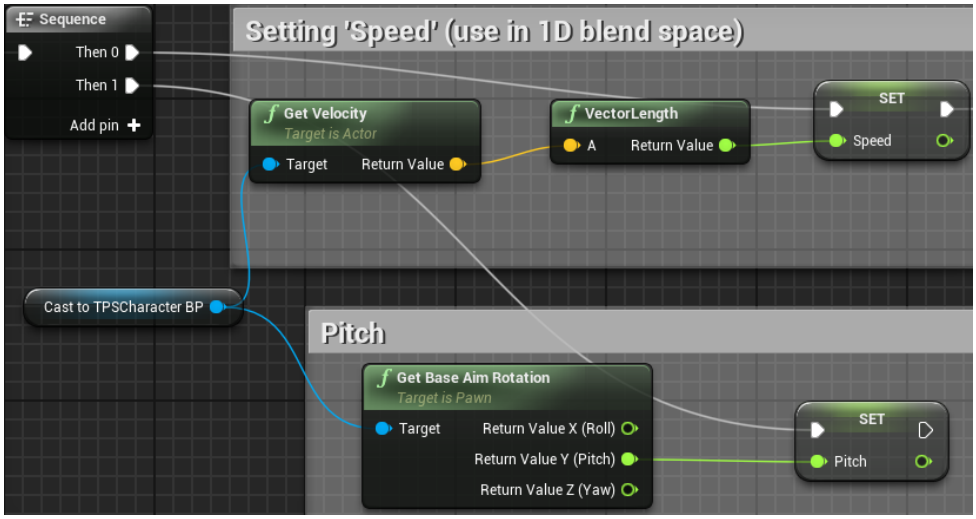
The second change was to add a ‘sequence’ at the end of the comment ‘Set 'IsInAir' (used in state machine) and connect this to the ‘set – Speed’ in the comment ‘Setting 'Speed' (use in 1D blend space)’ with the ‘Cast to TPSCharacter_BP’ variable connected to ‘Get Velocity’.



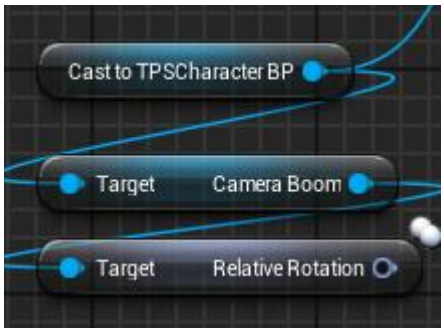
The third change was to delete the ‘Pitch’ in the comment ‘Spine rotator for aiming weapons’ and connect the branch directly into the ‘Set-AimingSpineRotator’ instead.



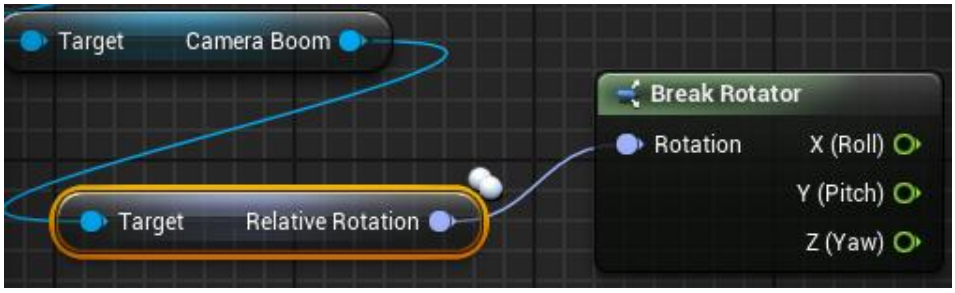
The fourth change was to copy and paste the ‘Get base aim rotation’ with the ‘Cast to TPSCharacter_BP’ connected to it and from there it was plugged into a ‘set- Pitch’ with the newly made ‘sequence’ connected to that.



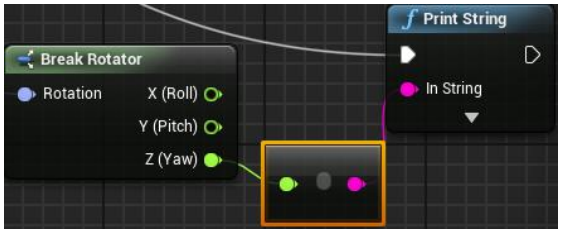
The final change was to delete ‘Try Get Pawn Owner’ and ‘Cast to ThirdPersonCharacter’ to replace it with the ‘Cast to TPSCharacter_BP’ variable.



Now that this was all reorganised, I could create a new logic section for the head manipulation. The first thing I needed to do was find the ‘node’ that’s going to feed a value to drive the head bone. This being the ‘CameraBoom’. So to do this I dragged from the ‘relative rotation’ and added a ‘Break Rotator’ because I only needed the rotation of ‘Z(Yaw)’.



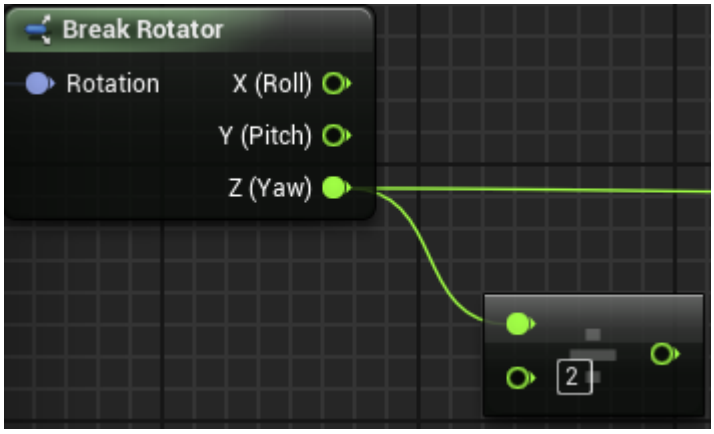
So, from that I needed to check what the ‘Z(yaw)’ value was doing so to do this I dragged from the ‘Branch- False’ into a ‘Print String’ I also connected the ‘Z(Yaw)’ to the ‘Print String’ aswell.



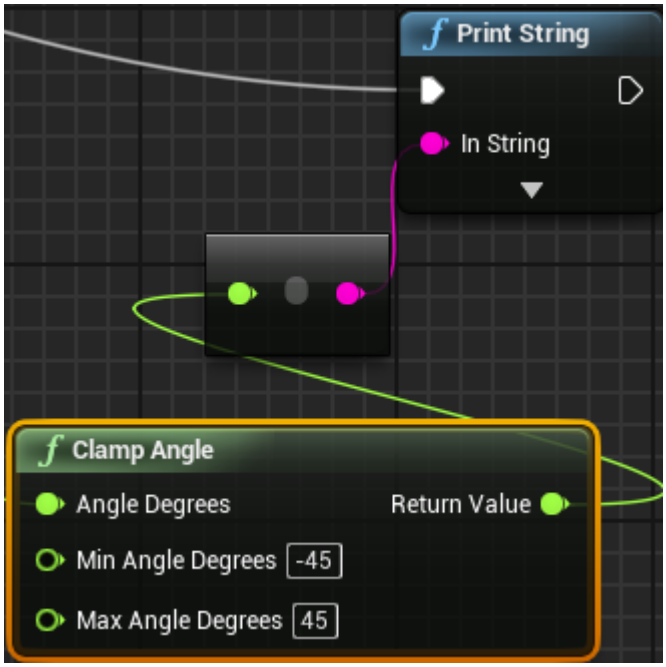
Now when I previewed this it would show facing left upto ‘-180°’

then when it was facing right it would go upto ‘180°’.

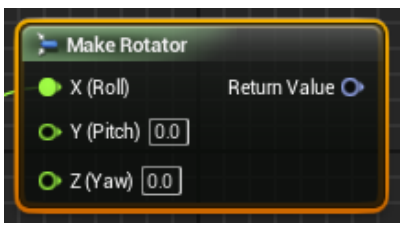
However, I didn’t need the full ‘180°’ either direction as this would twist the characters head all the way around as if he’s possessed. I needed it around ‘45°’ instead. So, to do this I grab from the ‘Z(Yaw)’ and added a ‘Float/Float’ with the value of ‘2’ this divides the ‘180°’ by 2 giving ‘90°’



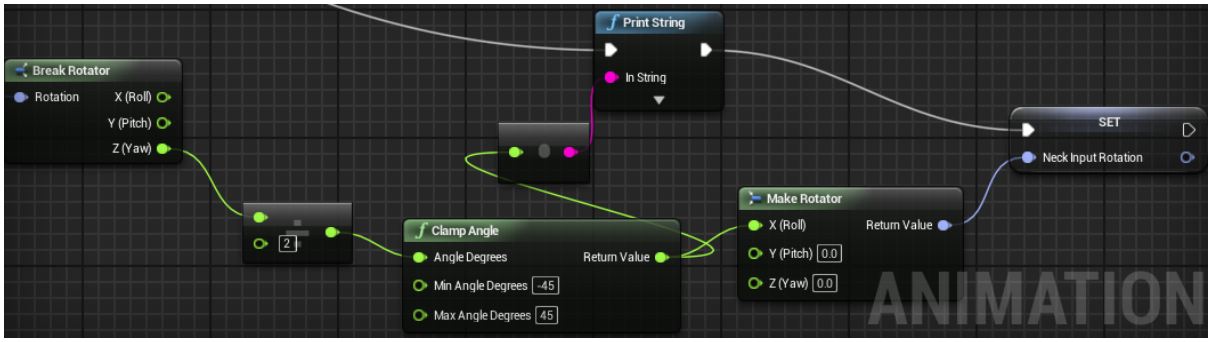
From this I added a ‘clamp angle’ with a ‘Min angle degrees’ of ‘-45°’ and a ‘Max Angle Degrees’ of ‘45°’. This makes it so the characters head will not rotate past a more realistic angle.



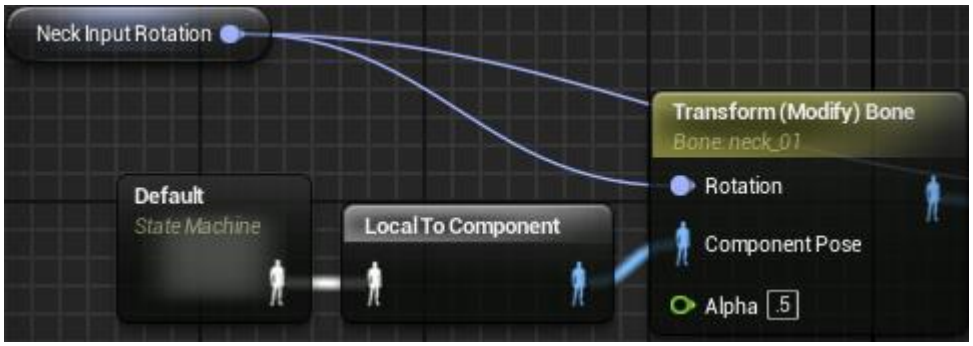
From this I added a ‘Make Rotator’, I needed this to set a degree that is going to manipulate the head bone.



The next step was to make a new ‘rotator variable’ named ‘NeckInputRotation’ and connected the ‘Print String’.

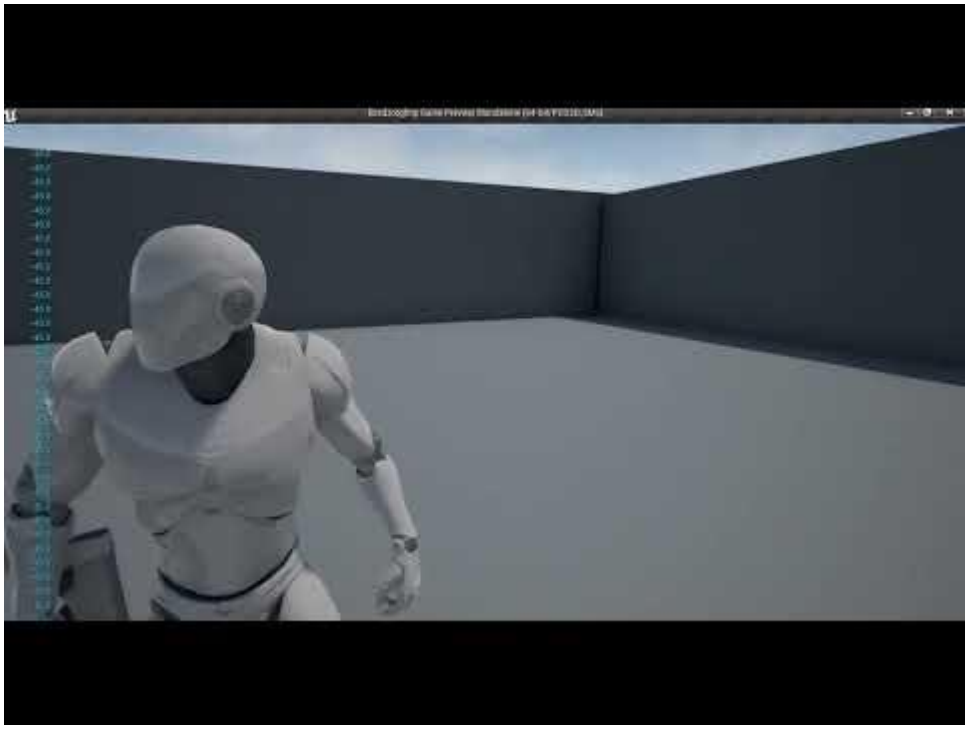


Now I needed to go over to the ‘ThirdPerson_AnimBP’ in the ‘AnimGraph’ and add a in the new variable ‘NeckInputRotation’. A ‘Transform (Modify) Bone’ with a ‘rotation mode’ of ‘Add to Existing’ and the ‘Rotation Space’ set to ‘Bone Space’. The ‘Bone to Modify’ was set to ‘neck_01’ at an ‘alpha value’ of ‘0.5’ so the manipulation wasn’t too severe.

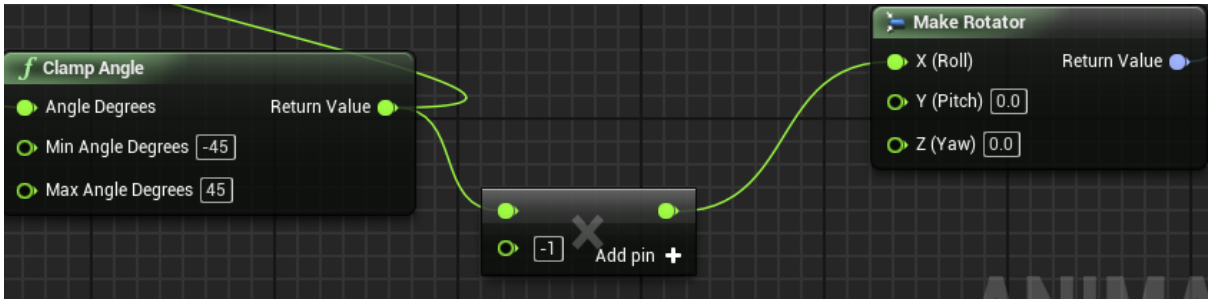


A copy of this was then made with the ‘Bone to Modify’ set to ‘head’ at an ‘Alpha value’ of ‘1.0’ as it needs full manipulation.

Now this allowed head movement however it was in the wrong direction.

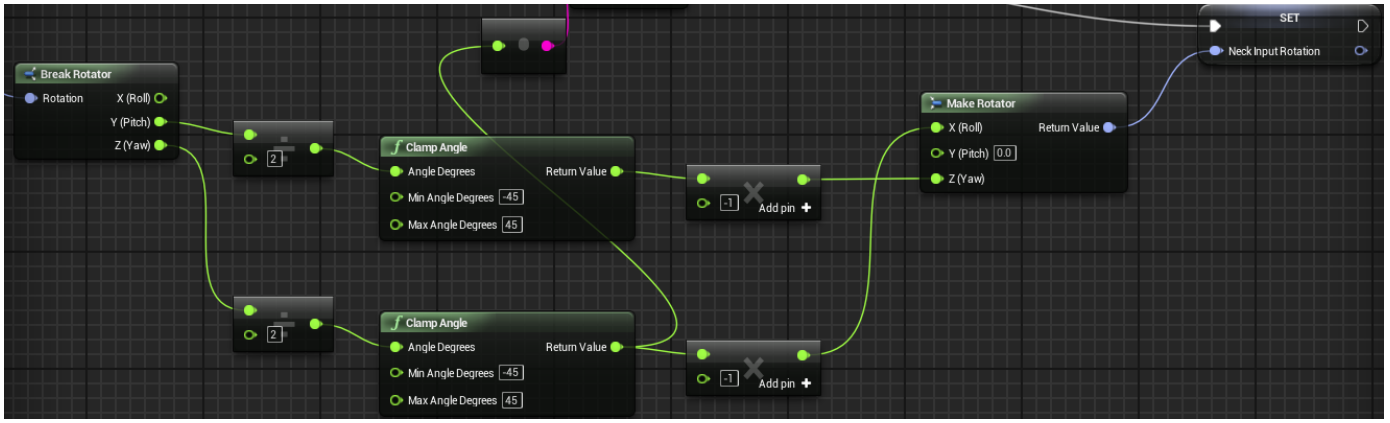


So, to fix this I added a ‘float * float’ with a value of ‘-1’ in the ‘ThirdPerson_AnimBP’.

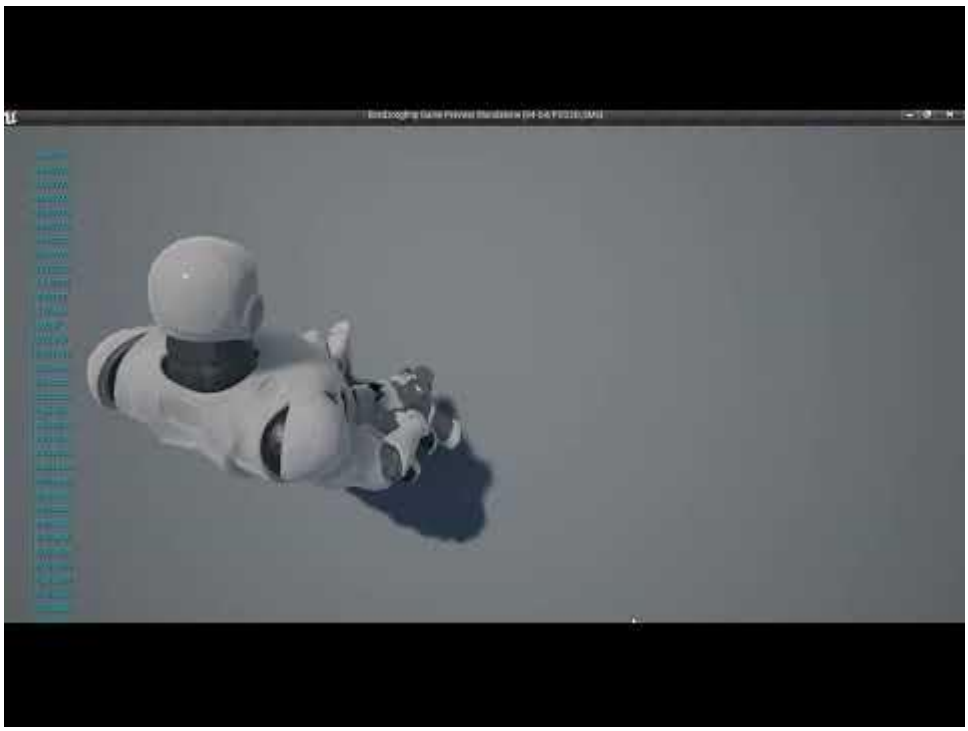


This fixed the issue.

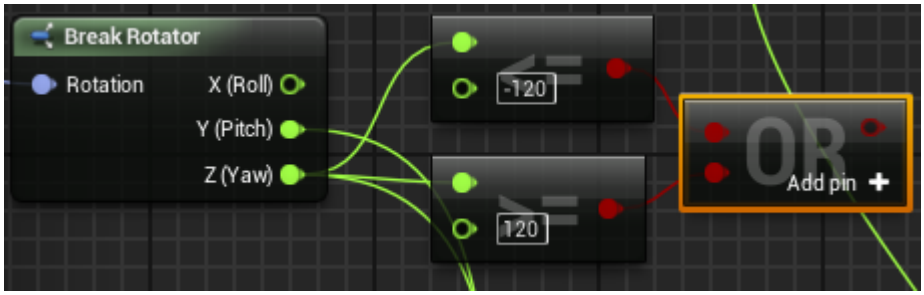
Now I needed to set this up again but for the ‘Pitch’. So, to do this I copied and pasted the line of logic for Z(Yaw) but connected it from the Y(Pitch) on the first ‘Break Rotator’ and into X(Roll) on the second ‘Make Rotator’.



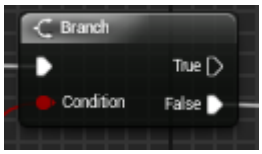
Now this allowed the characters head face up and down.



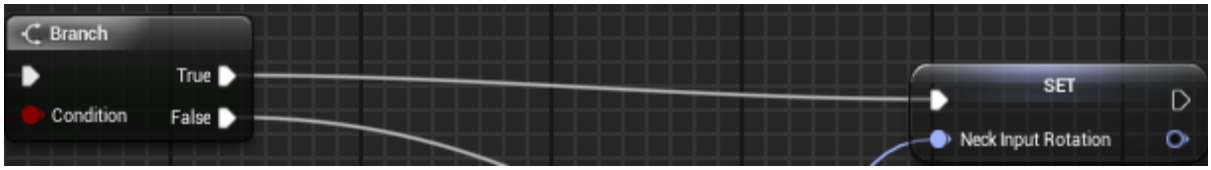
However, towards the end of the video it shows the characters head snapping side to side after it reaches its ‘clamped angle’ limitations. So, to fix this I added a ‘float <=’ set to a value of (-120) alongside a ‘float >=’ set to a value of (120). These were then plugged into a ‘OrBoolean’.



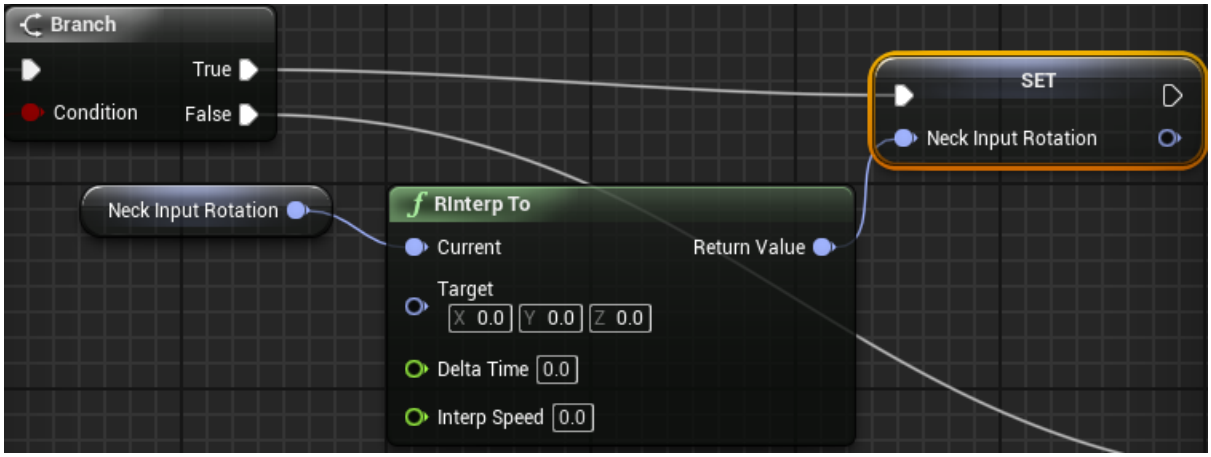
From this the ‘Print String’ was then deleted and replaced with a ‘branch’



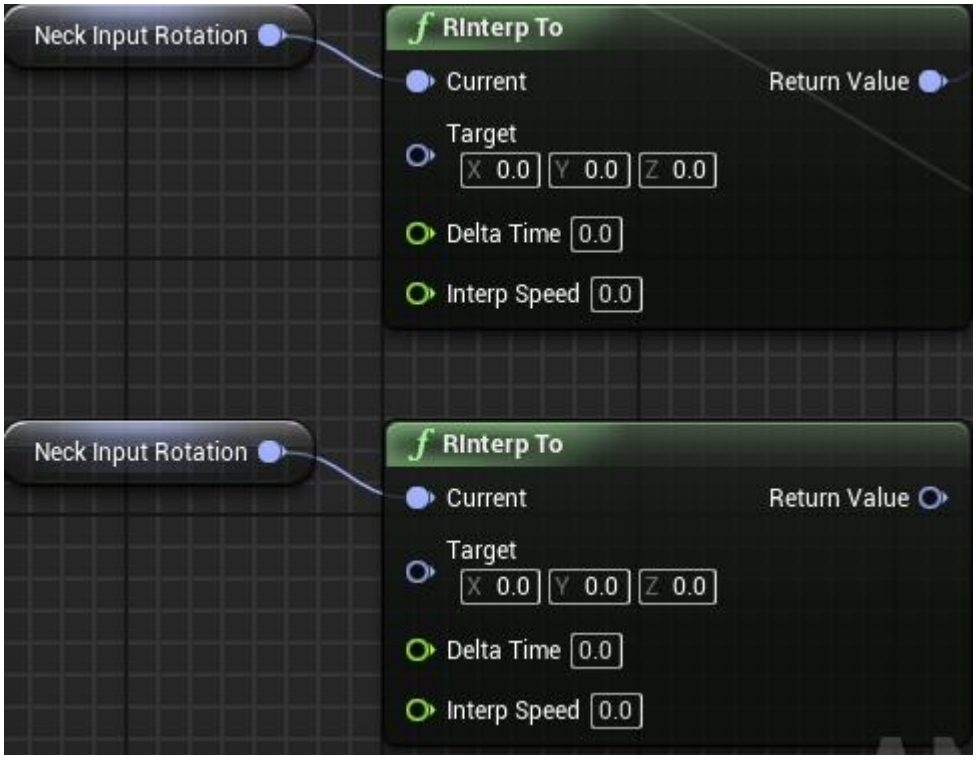
Now from the second ‘Make Rotator’ I needed to add a ‘Set - NeckInputRotation’ and connect it to the ‘Branch’ as ‘True’



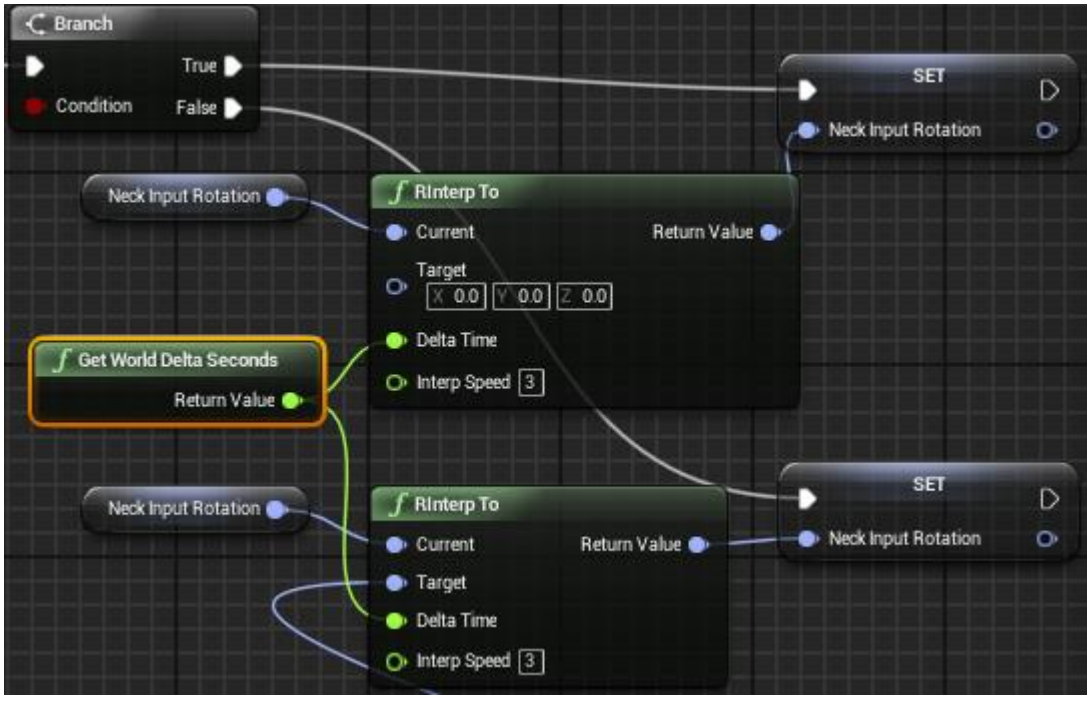
After that I added a ‘Get – NeckInputRotation’ and a ‘Rinterp To’ from that connected to the ‘Set – NeckInputRotation’.



Now that I had this set for the head moving forward, I needed a second one to control the head as the character looks at the camera. So to do this I copied and pasted the ‘Rinterp To’



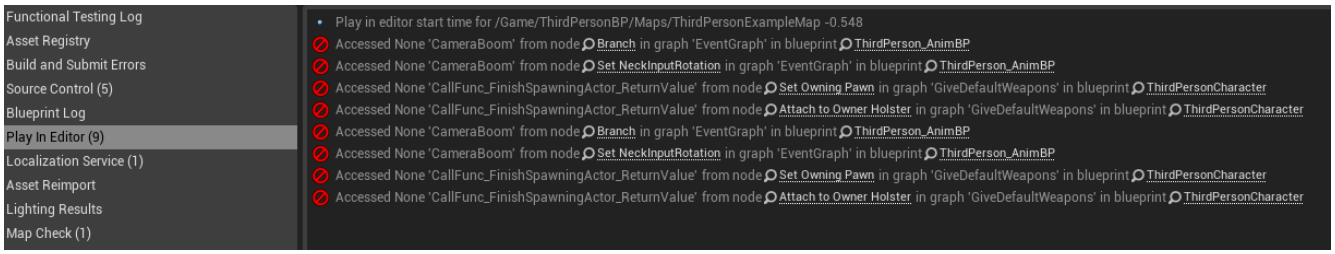
For both ‘Delta Time’ I connected a ‘Get World Delta Seconds’. I also set both the ‘Interp Speed’ to (3.0). This was then plugged into the bottom ‘Set – NeckInputRotation’ allowing this to set the value on its own.



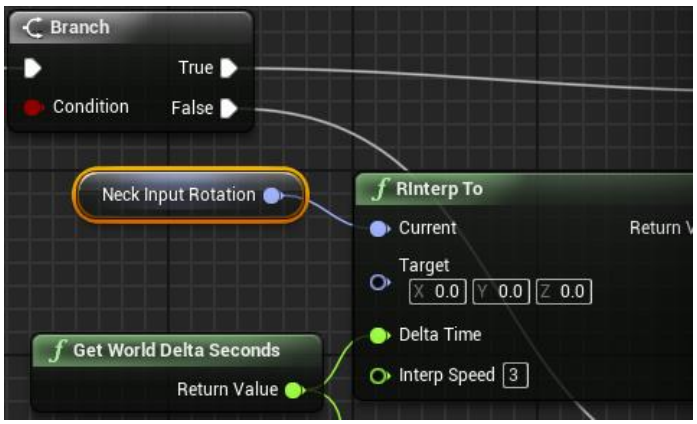
Now this will allow the characters head to look around smoothly wherever the player is looking, while standing still or running around.

This is the end of my reference video ‘TPS Build Part 12 – Making the Character Look In Direction of Camera’ (K, Dail. 2015).

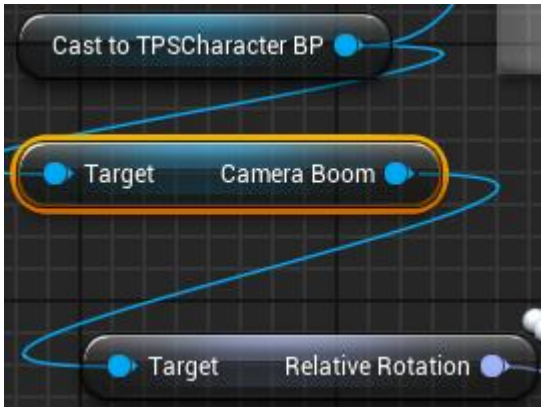
Fixing editor errors:



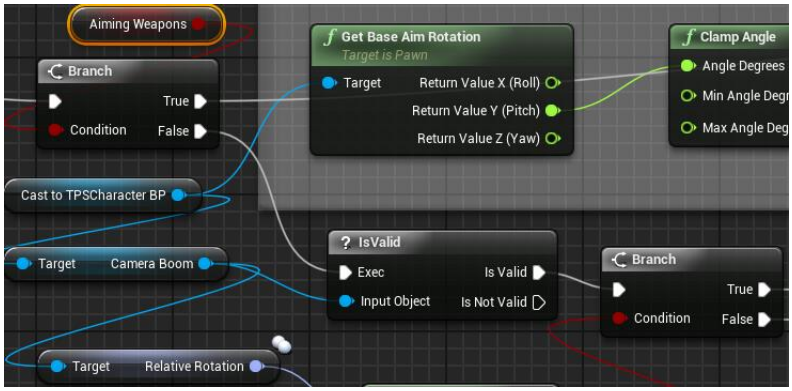
The first error was caused by unreal not knowing exactly where the ‘NeckInputRotation’ is coming from.



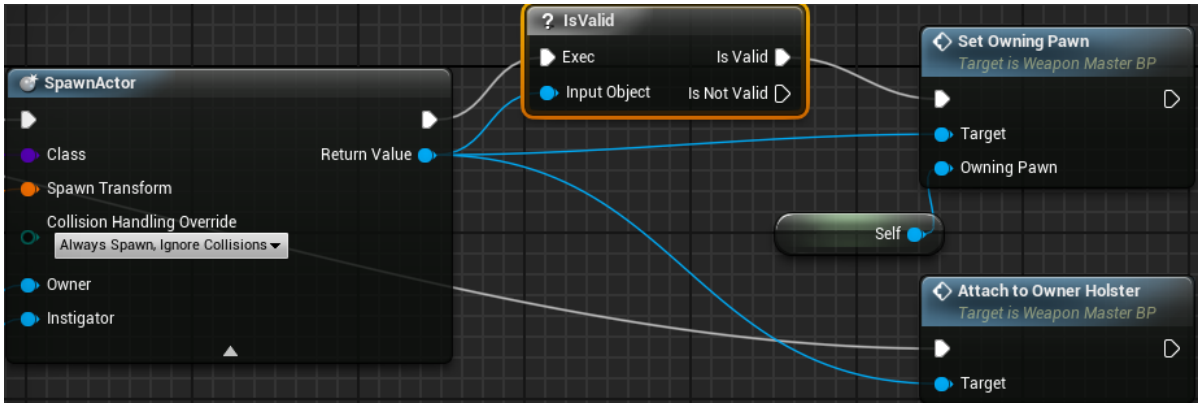
To fix this I needed to feed it from the ‘CameraBoom’



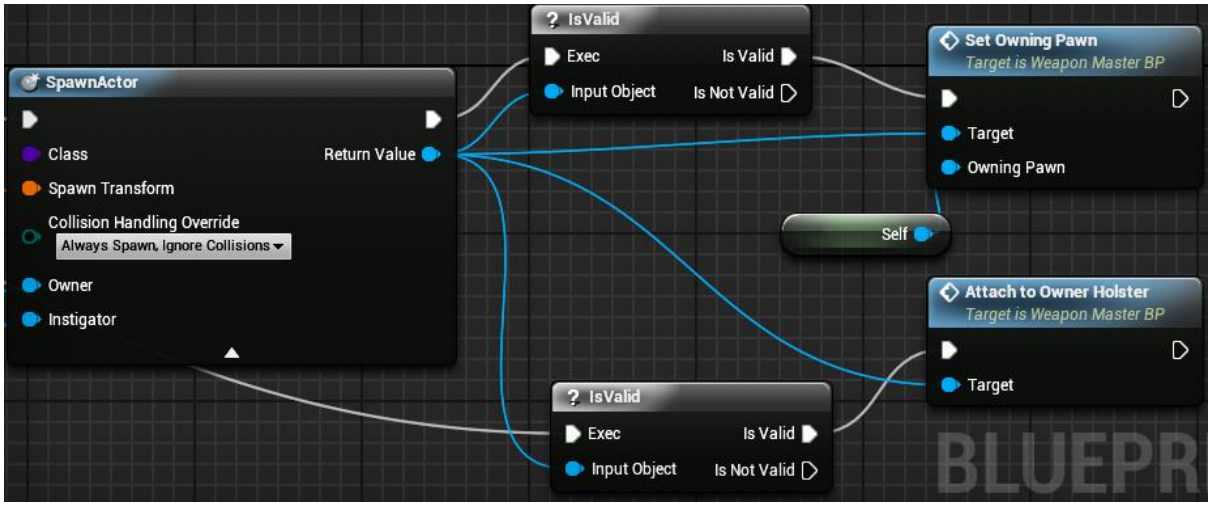
So, from the ‘Branch’ I connected an ‘IsValid’ and connected the ‘CameraBoom’ to it aswell as the ‘Input Object’. This was then fed into the second ‘Branch’. This will check if the ‘Cameraboom’ is active. This fixed the first ‘editor error’.



The second error to fix was in the ‘ThirdPersonCharacter – GiveDefaultWeapons’ function. Two things were happening here the first was when it was giving the default weapon to the character it was not realising what it was even though the ‘Actor’ was spawning in. so to fix this I added an ‘IsValid’ after the ‘SpawnActor’ that basically asks if there is a weapon it will spawn but if not, it won’t do anything.



The same was then done to the ‘Attach to Owner Holster’. This fixed the second ‘Editor Error’.



At this point there were no more errors, and I was happy with my character logic. This was the result of my ThirdPersonCharacter after the end of my reference video ‘TPS Build Part 17 Closing and Changes to Project File’ (K, Dail. 2015).