# Idea

# Project Setup

Project Settings:

* Third Person template
* Blueprints
* Desktop/Console
* Maximum Quality
* With Starter Content
* Raytracing Disabled

The following packages were downloaded from the marketplace:

* Animation Starter Pack
* Custom Character Model

# Player Camera

The first thing is to setup the camera so that the player can toggle between the first person and third person camera perspective by pressing the ‘V’ key.

The first step is to change the third person perspective from the default view to a custom preferred view by changing the transform location and rotation to achieve the preferred view. The settings are as follows:

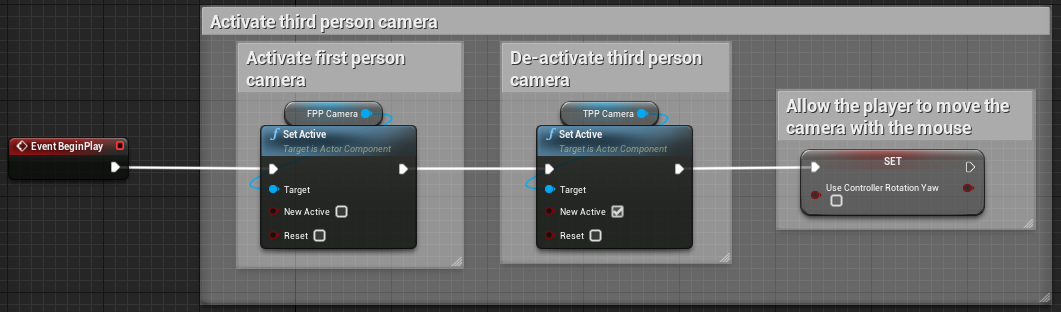
* Transform Location: {-118, 1.285714, 121}
* Transform Rotation: {0, -10, 0} (in degrees)

The second step is to add a new camera for the first-person perspective. The camera is positioned in front of the head of the player at a desired height and then made a child to the mesh. Then the camera view has been attached to the player’s head socket using the parent socket attribute under the camera details and the camera uses pawn control rotation that is checked under the camera options.

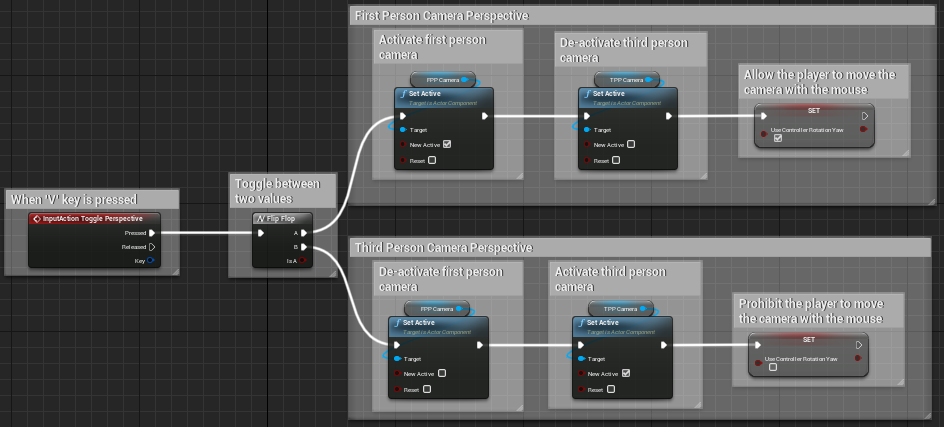
After the cameras have been set-up, the next step is to allow the player to transition between two perspectives when the player presses the ‘V’ key.

To allow the ‘V’ functionality an axis mapping of “Camera Perspective” was added in the input section of the project settings. Now, the last part is to create the blueprint for toggling between the two cameras under the character blueprint.

* The game must start in third person perspective.



* When the player hits the ‘V’ key for the first time, the game must first switch to the first-person perspective and then followed by the third person perspective. Then keep toggling between the two accordingly. Hence using the flip-flop node, the A execution path is attached to the first-person activation and the B path to the third person.



This completes the camera setup for the player.

# Health and Armor

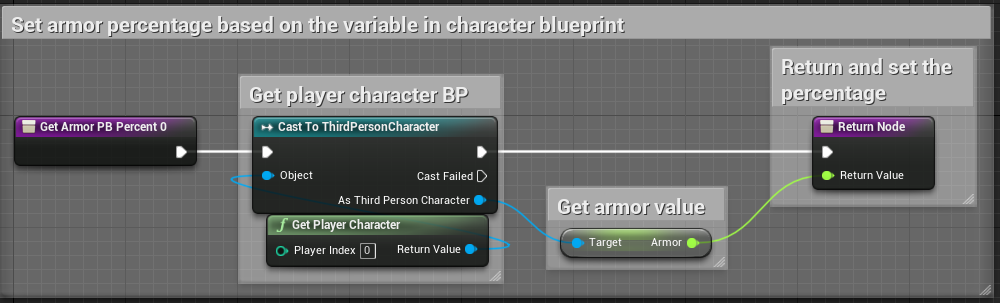
The first step is to create the HUD that contains the health and armor progress bar using the widget blueprint.

To create the widget a vertical box was created under which 2 progress bars were added. The top progress bar represents the armor (light yellow), and the bottom represents the health (light red).

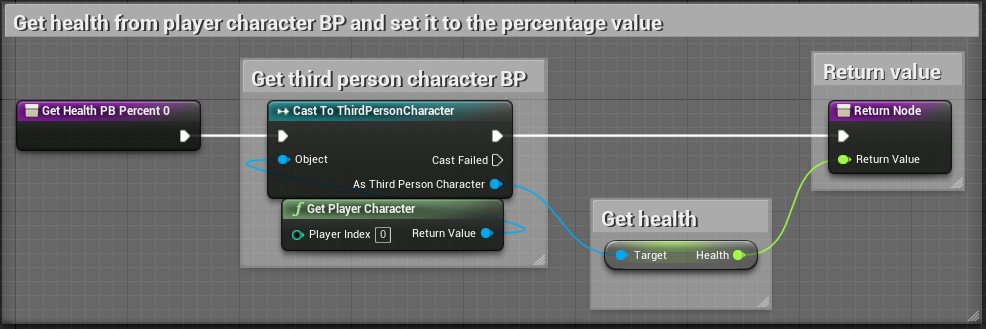


Then under the character blueprint, two float variables called “Armor” and “Health” were created and then the next step was to bind the progress bar in the widget with the variables in the character blueprint.

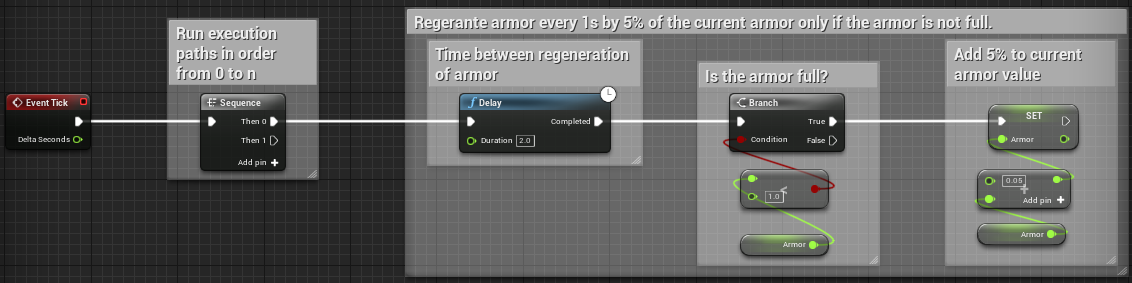
In the details panel for the armor progress bar, the create bind function is called under the percent attribute:



Similarly, for the health progress bar:



After the variables from the character blueprint were successfully linked to the progress bars, the next step was to regenerate the armor after every 2 seconds.



This completes the Health and Armor setup.

# Damaging Player

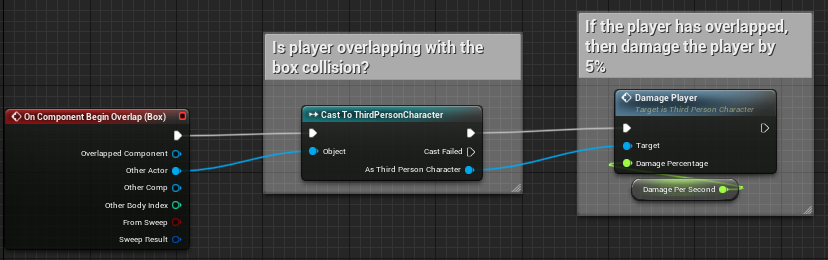
The next step was to damage the player. Hence, a new function was created with a float parameter input called “Damage Percentage”.

The first step is to damage the armor regardless of its value by getting the current armor and then subtracting it by the damage percentage. Then we need to check whether the armor has gone below ‘0’ or not. If the armor is below 0, then subtract the value of armor from the health and set the armor value back to 0. Since, the armor value will be in negative, the health variable will be added to the armor variable.

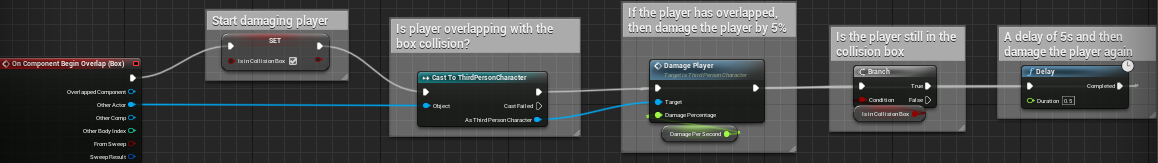


To test whether the damage function is correctly working a new blueprint actor called “Pain Volume” with a box collision is created to deal 5% damage to the player.

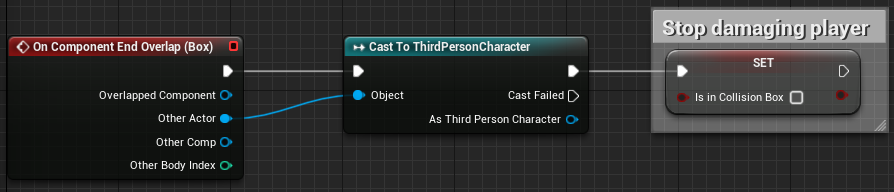
Therefore, firstly the box collision is added to detect whether the player is standing inside the box collision volume to damage the player or not.



Hence, whenever the player enters the box collision it will damage the player health. However, while damaging the player’s health the armor is also being regenerated at the same time. The idea is to keep damaging the armor until the player is inside the box collision without regenerating the armor. Therefore, a Boolean variable was created to detect whether the player is being damaged currently or not.

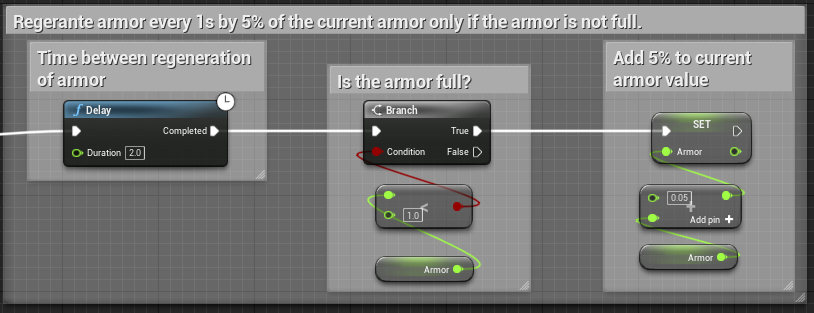


Once the player has left the collision box, we stop damaging the player:



Back to the character blueprint’s event graph we need to check whether the player is being damaged before regenerating the armor:

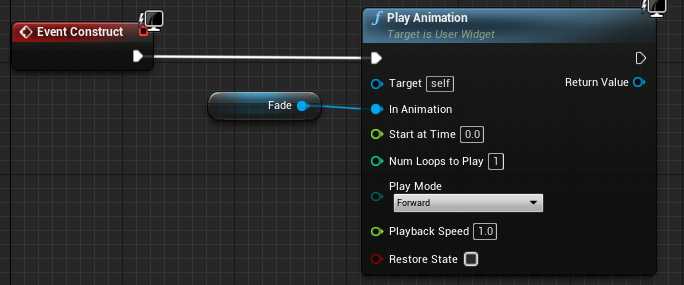




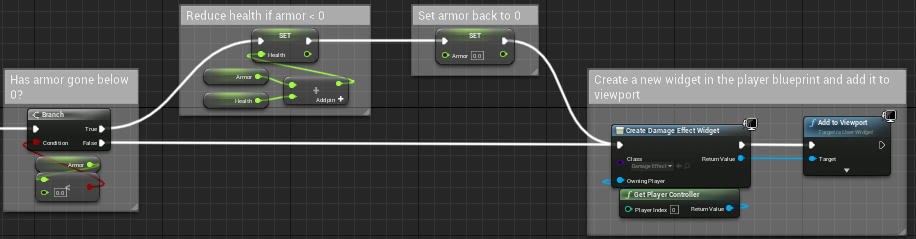
The next step is to fill the screen with blood effect when the player is taking damage. Therefore a “Red Vignette” image file was imported under the character blueprints folder.

A new widget with the blood effect is created and an image object is added onto the widget spanning the entire screen. Then the image was added to the image object in the widget. Moreover, in the beginning we do not want to display the blood effect and hence the alpha value under the color and opacity section is initially set to 0.

Then we create a new animation called “Fade”, add a new track called “Red Vignette”, and within the track, add another track called “Color and opacity”. After that the slider is dragged onto 0.5s and the opacity value is set to 1.0 and then back to 0 when the slider reaches 1s. Then we need to play the animation when the widget is constructed using the blueprint:



Then in the damage function under the character blueprint, we need to create this widget and add it to the viewport at the end after reducing the health/armor regardless whether the branch goes to true or false:



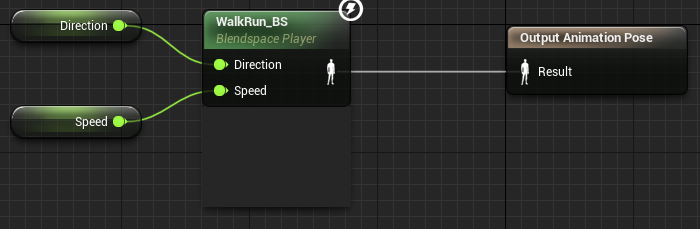
# Setting Custom Character

The first part is to organize the content browser as we will be adding a custom character skeleton mesh and custom character animations. Therefore, under the third person character blueprint we create a folder named “Character” and another folder inside the character folder called “Animations”. Then, we import the custom character skeletal mesh and the animation into the respective folders. These assets are available under the shooter game assets folder.

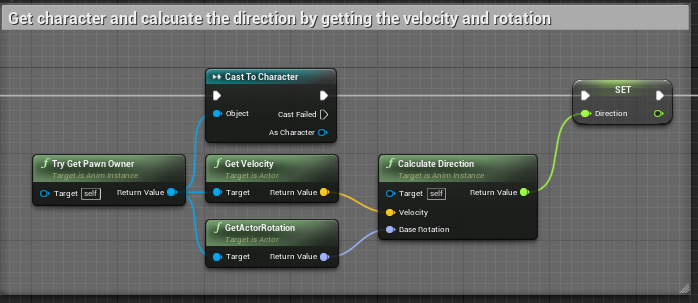
Then the next step is to create an animation blueprint that targets the Swat skeleton. However, the problem is in its default state it will only run one animation. Hence, we need a state machine to tell which animation to run and then further send it to the result to play the animation where and when.

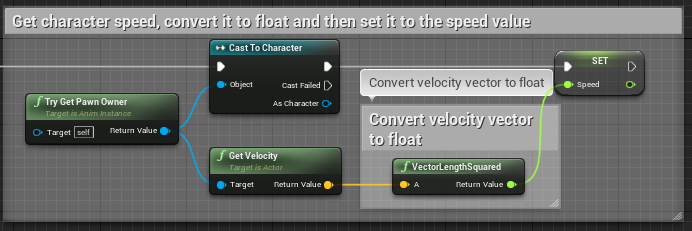
Creating the first state machine called “Wal Run” that allows the player to transition between the idle state, to walk state and to run state. By double-clicking on the walk run state we see an entry node. From the entry node we initiate the idle animation and from the idle animation we create another state called “Walk Run” again. However, we need to firstly create the animation blend space before attaching it to the Walk Run state. Therefore, in the content browser under the character folder we create a walk run animation blend space with the name “Walk Run BS”. [YouTube tutorial 1, video number 7](#_References) helps to setup the blend space for the walk run animation.

After setting up the blend space, the next step is to determine which animation to play based on the speed and direction. Therefore, in the walk run state we create 2 float variables – speed and direction and based on that we send the animation to the output pose.

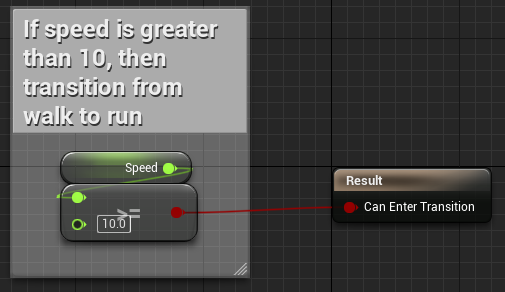


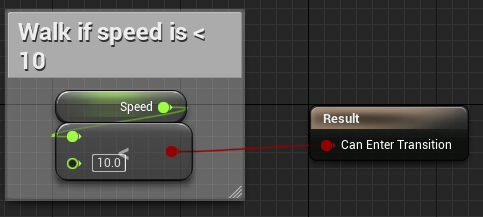
The next step is to get the speed and direction from the event graph:





Later, we need to specify the condition under what speed would the player walk and after what speed would the player run:





# References

* Toggle between first person and third person perspective: <https://www.youtube.com/watch?v=PwhGk5fSIjg>
* Shooter Game Assets: <https://www.devsquadacademy.com/resources>
* YouTube Tutorial 1: <https://www.youtube.com/watch?v=DywBqQtTHMo&list=PLL0cLF8gjBprG6487lxqSq-aEo6ZXLDLg&index=1>