Cinematic Camera System Documentation

Dead Pedal - Working Title

Originally Created: 07/02/25

Last Updated: 07/02/25

Documentation by: Matthew Wojciechowski

Overview

The ‘Cinematic Camera System’ is for allowing the player to view their vehicle from alternate angles, enable a “GTA Style” cinematic follow mode, and spectate other racers.

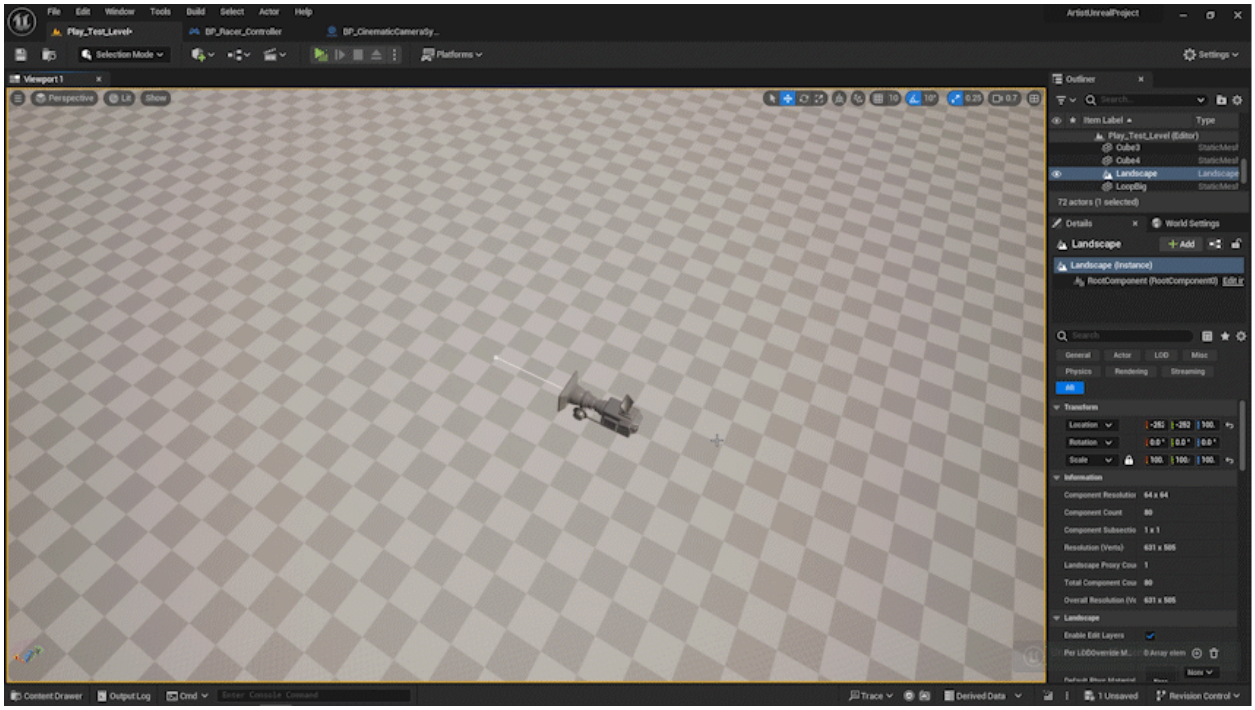
Cinematic Camera System Basics

The ‘BP_CinematicCameraSystem’ must first be MANUALLY PLACED within EACH LEVEL.

Following this, you will notice a spline protruding from the front of the camera, if you grab the end point of the spline you will be able to move it around with the gizmo. But if you ***HOLD LEFT ALT*** and *then* move the spline, you will create a new point that extends it.

These spline points are the locations the camera is moved to during the “GTA Style” ‘Follow Mode’. The camera is automatically moved to the point that is *nearest* to the ‘FayCar’ assigned to the ‘currentFayCarToFollow’ variable.

Now this part needs to be made very clear, ***THE LINE OF THE SPLINE DOES NOT MATTER, THE ORDER OF THE SPLINES DOES NOT MATTER, THE LOCATION OF A SPLINE POINT IS THE ONLY THING THAT MATTERS.*** This isn’t to say it isn’t good to build a spline path in a clean and organized way, but in terms of functionality, *only the location of each individual spline point matters for this system.*



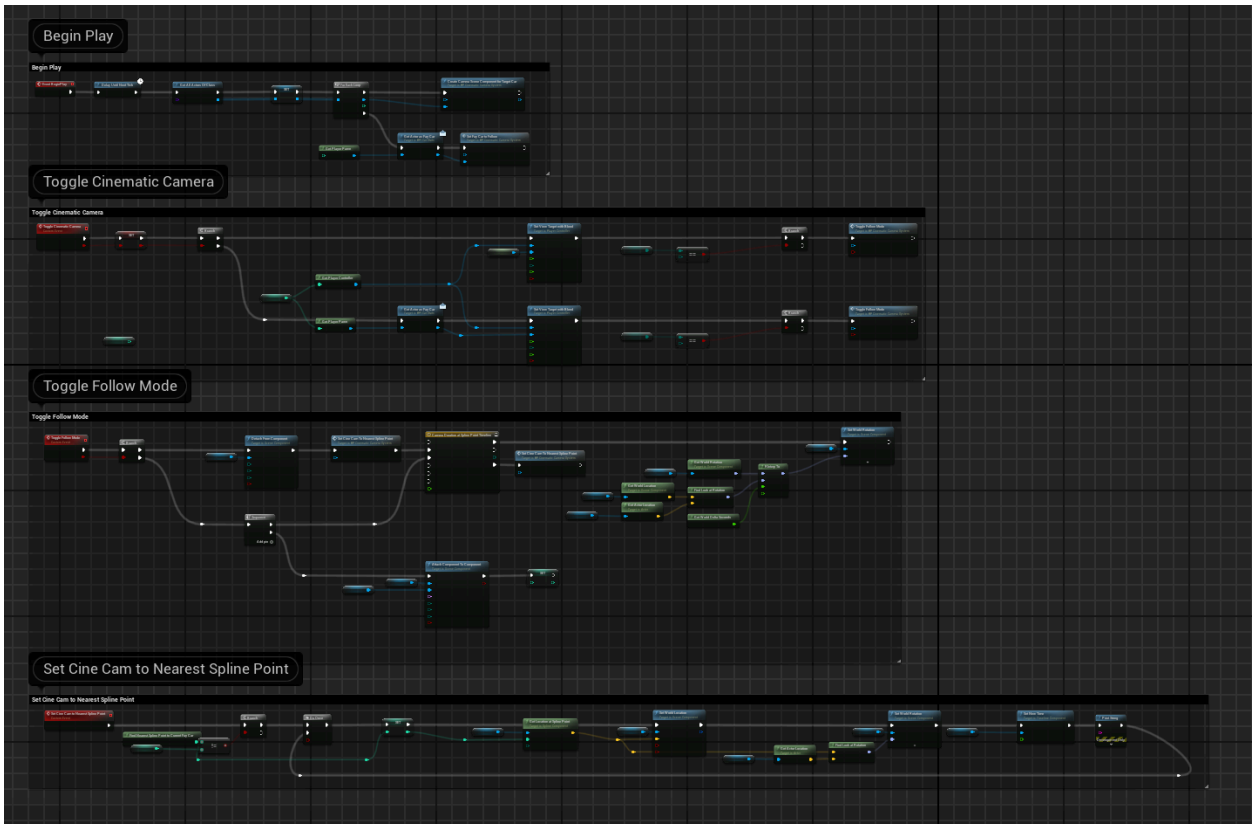
For the player to access the ‘BP_CinematicCameraSystem’ they currently need only to press the ‘M’ key, this will toggle them in and out of the cinematic camera.

To change camera angles you use the ‘LEFT’ and ‘RIGHT’ ARROW KEYS.

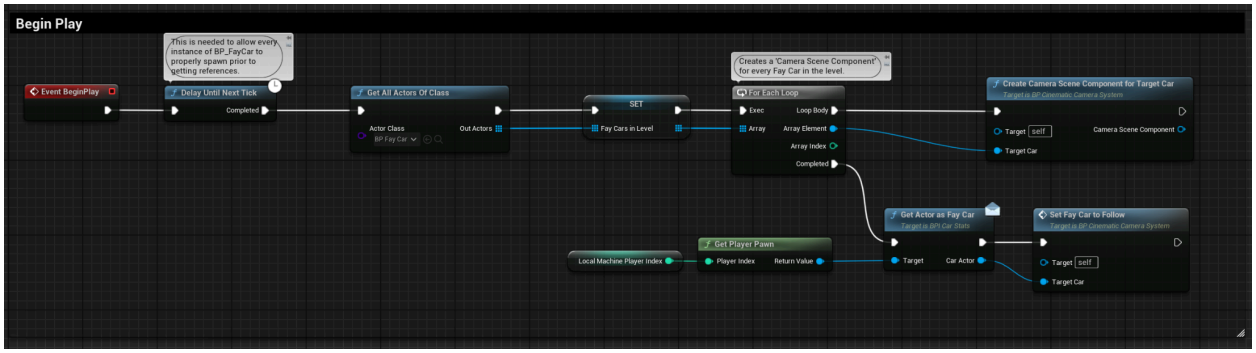
To change the car to spectate you use the ‘UP’ and ‘DOWN’ ARROW KEYS.

This system contains the majority of its code in ‘BP_CinematicCameraSystems’, but it also has a chunk within ‘BP_Racer_Controller’ for activating the inputs listed above.

BP_CinematicCameraSystem

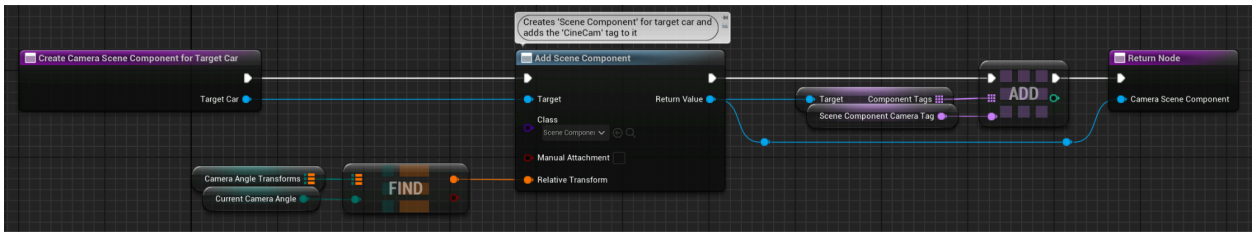


Begin Play



On ‘Begin Play’ there is an immediate ‘Delay Until Next Tick’ node, the purpose for this is to allow each racer to spawn within the level prior setting an array containing each one. Following the array, each actor within this ‘fayCarsInLevel’ array will be put through a ‘For Each Loop’ where they will run the ‘Create Camera Scene Component for Target Car’. Once each actor has been run, then the local player pawn will be input as the ‘targetCar’ for the function ‘Set Fay Car to Follow’.

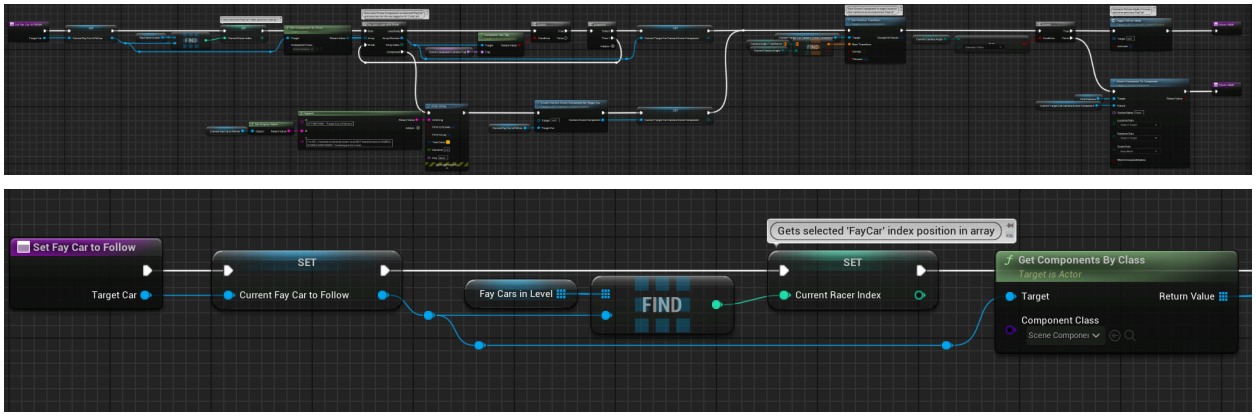
Create Camera Scene Component for Target Car (Function)



When ‘Create Scene Component for Target Car’ is run, it will take the supplied ‘targetCar’ actor and add a scene component to it. The relative transforms of which are set based on the ENUM value of the ‘currentCameraAngle’ and the transforms associated with that camera angle type within the ‘cameraAngleTransforms’ map.

Once created, that component will have the ‘sceneComponentCameraTag’ added to its list of ‘Component Tags’. This tag currently being the value of “CineCam”. More on this is explained in the ‘setFayCarToFollow’ function.

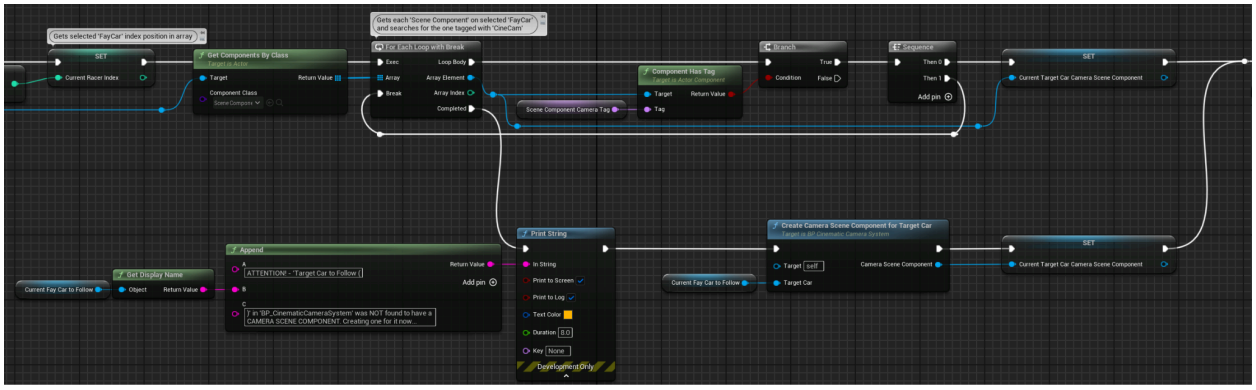
Set Fay Car to Follow (Function)



When ‘Set Fay Car to Follow’ begins, it will take the value supplied to its ‘targetCar’ input and set the ‘currentFayCarToFollow’ variable to it.

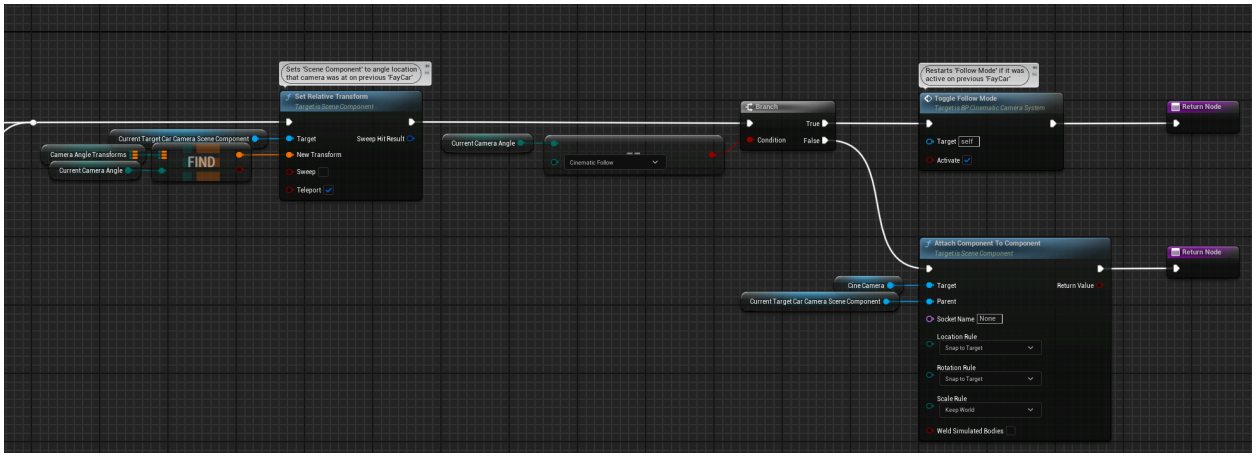
Next, it will retrieve the ‘fayCarsInLevel’ array and find the index where ‘currentFayCarToFollow’ is found at, this value will then be set as the integer variable of ‘currentRacerIndex’. More on ‘currentRacerIndex’ is explained in the ‘CycleToNextRacer’ function.

After this each ‘Scene Component’ found within ‘currentFayCarToFollow’ will be retrieved and run through a ‘For Each Loop with Break’.



Within this loop, each retrieved component will be checked to see if it contains the ‘sceneComponentCameraTag’, if it is found then, then that ‘Scene Component’ will be set as the ‘currentTargetCarCameraSceneComponent’.

If it is not found, and the ‘For Each Loop’ is able to complete, a print string will be displayed warning that the particular racer was not found to have the proper scene component. It will then run the ‘Create Camera Scene Component for Target Car’ function for the ‘currentFayCarToFollow’, before subsequently setting *that* as the ‘currentTargetCarCameraSceneComponent’.



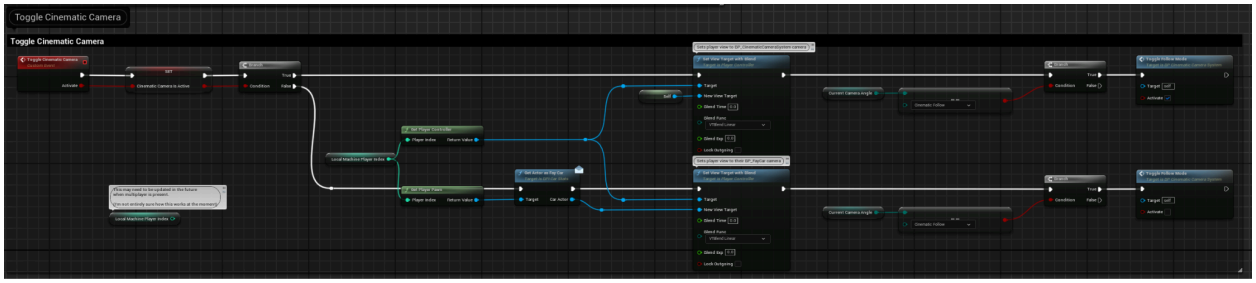
The ‘currentTargetCarCameraSceneComponent’ will then have its relative transform set to that which corresponds with the transforms found within the ‘cameraAngleTransforms’ for the value of ‘currentCameraAngle’.

After this there will be a check to see if the ‘currentCameraAngle’ is equal to ‘Cinematic Follow’

If TRUE, then the ‘Toggle Follow Mode’ function will be run with ‘Activate’ ticked.

If FALSE, then the ‘cineCamera’ component will be attached to the ‘currentTargetCarCameraSceneComponent’, snapping to both its location and rotation.

Toggle Cinematic Camera

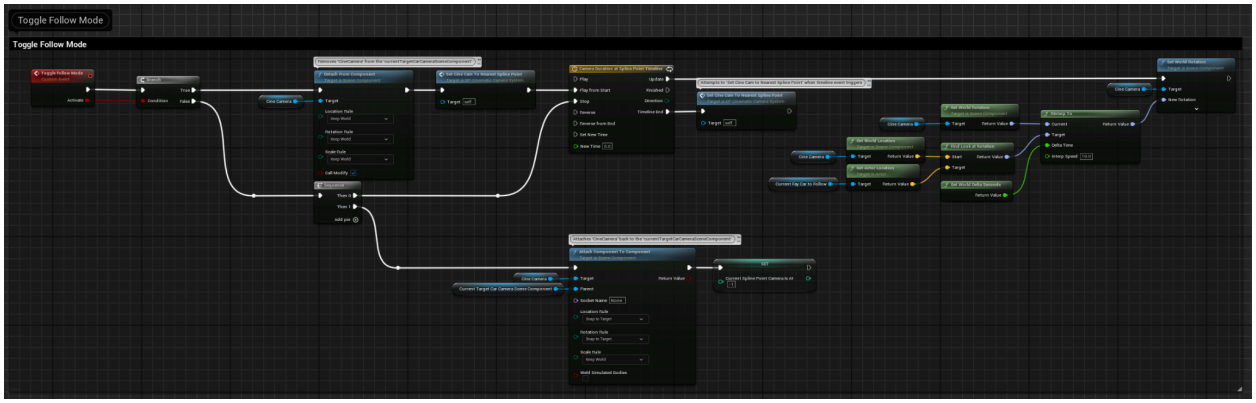


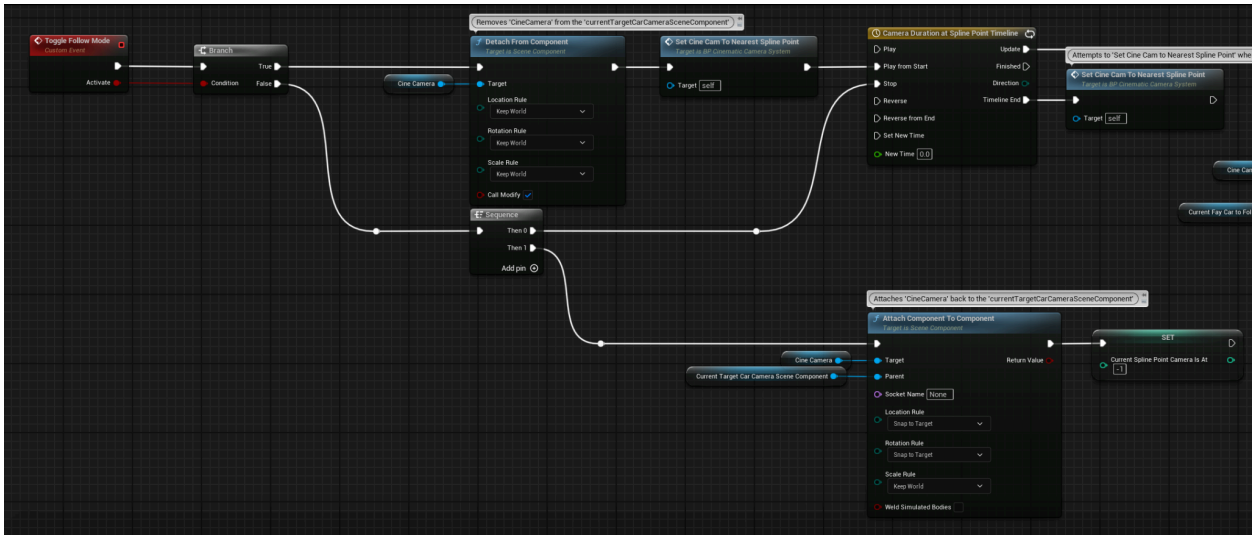
When the ‘Toggle Cinematic Camera’ event is run, it will take it’s supplied bool value of ‘Activate’ and use it to set the ‘cinematicCameraIsActive’ bool before running it through a ‘Branch’ node.

If the bool is TRUE, then the specified player controller will have a ‘Set View Target with Blend’ node run with the ‘BP_CinematicCameraSystem’ being the input of the ‘New View Target’. Next, the ‘currentCameraAngle’ will be checked to see if it’s equal to ‘Cinematic Follow’ and if it is, then it will run the ‘Toggle Follow Mode’ event with the ‘Activate’ bool ticked.

If the bool is FALSE, then the specified player controller will have a ‘Set View Target with Blend’ node run with specified player pawns instance of ‘BP_FayCar’ set as the ‘New View Target’, Next, the ‘currentCameraAngle’ will be checked to see if it’s equal to ‘Cinematic Follow’ and if it is, then it will run the ‘Toggle Follow Mode’ event with the ‘Activate’ bool unticked.

Toggle Follow Mode





When the ‘Toggle Follow Mode’ event begins, it will take its supplied ‘Activate’ bool value and run it through a ‘Branch’ node.

If TRUE, then the ‘cineCamera’ will have a ‘Detach From Component’ node run for it. This will remove it from the ‘currentTargetCarCameraSceneComponent’ it was previously attached to. Next it will run the ‘Set Cine Cam To Nearest Spline Point’ event before telling the ‘Camera Duration at Spline Point Timeline’ to ‘Play from Start’. The purpose of running the ‘Set Cine Cam To Nearest Spline Point’ prior to the timeline, is to immediately move it to the best possible spline point immediately instead of having to wait until ‘Timeline End’ is activated.

If FALSE, then it will first tell the ‘Camera Duration at Spline Point Timeline’ to ‘Stop’ before setting the ‘cineCamera’ to attach to the ‘currentTargetCarCameraSceneComponent’ and setting the ‘currentSplinePointCameraIsAt’ integer to -1.

Camera Duration at Spline Point Timeline

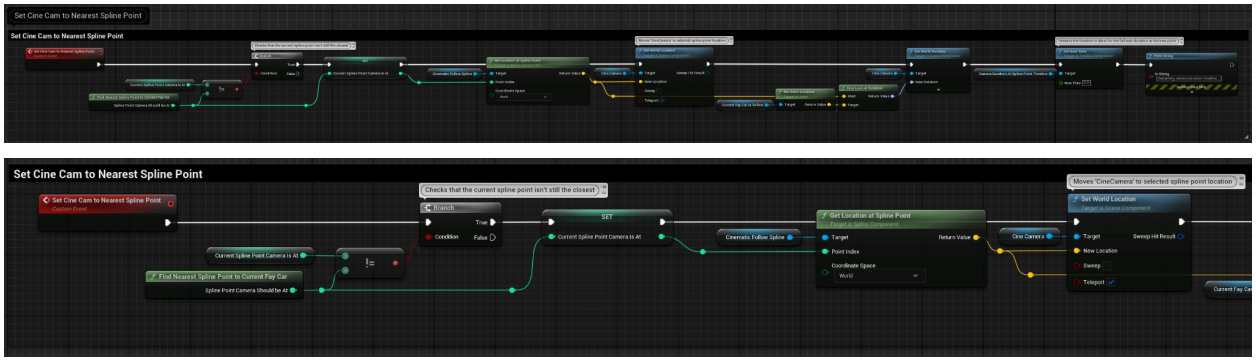
This timeline is simply a timeline with a track length of 4.0 and a ‘Timeline End’ event that is set to trigger at the end of the track's duration. ***The timeline length is responsible for how long the ‘cineCamera’ will stay at a spline point for changing locations while in ‘Cinematic Follow’.***

As the timeline updates, it will get the world location of both the ‘cineCamera’ and the ‘currentFayCarToFollow’ and utilize their locations to determine the ‘Find Look at Rotation'. This value will then be imputed into the ‘target’ rotation of an ‘RInterp To’ node with the ‘cineCamera’ world rotation being set to ‘current’ rotation. The return value of this node is fed into a ‘Set World Rotation’ node for ‘cineCamera’.

What this is doing is providing a smooth blend for the camera to follow ‘currentFayCarToFollow’ with. ***You can edit how fast the camera tracks the ‘currentFayCarToFollow’ by editing the ‘Interp Speed’ of the ‘RInterp To’ node.***

When the event ‘Timeline End’ has been triggered within the timeline, it will run the ‘Set Cine Cam To Nearest Spline Point’

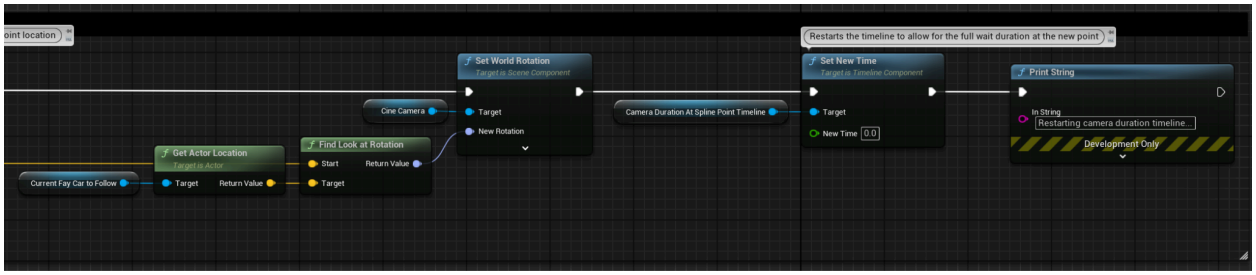
Set Cine Cam to Nearest Spline Point



When ‘Set Cine Cam to Nearest Spline Point’ is run it will run the function ‘Find Nearest Spline Point to Current Fay Car’, but it will check that the return value of that function is not equal to the ‘currentSplinePointCameraIsAt’. The reason for this is to continuously set the camera's location to the location that it’s already at.

So long as the previous branch is true, the code will continue through a ‘Do Once’ node and set the value of ‘Find Nearest Spline Point to Current Fay Car’ to the integer variable of ‘currentSplinePointCameraIsAt’.

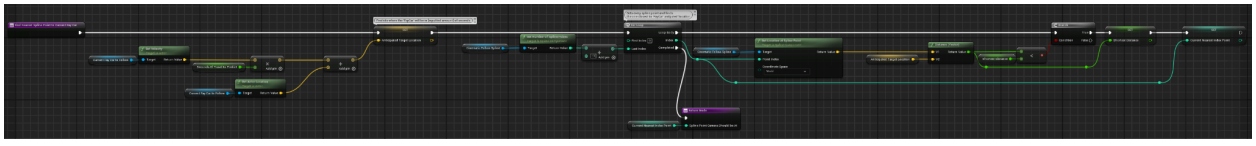
Next, that same value will be used as the ‘Point Index’ input for finding its world location along the ‘cinematicFollowSpline’ component before setting the ‘cineCamera’ to that world location.

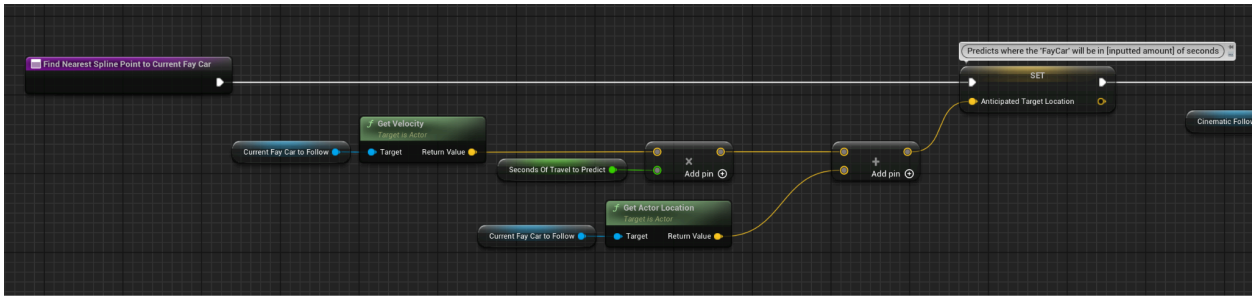


After this, that world location will be input as the ‘start’ for a ‘Find Look at Rotation’ node, with the target being ‘currentFayCarToFollow’ actor location. The return value of this will be set as the ‘cineCamera’ world rotation. While this same process is being done within the previously stated timeline, it’s done here without the blend to immediately make the camera point towards the car, otherwise there would be a camera pan as the camera rotation blends towards it.

Finally, the ‘Camera Duration At Spline Point Timeline’ will be restarted. This ensures the camera will remain at the new world location for the specified duration of the timeline. Otherwise the timeline may be in the process of already counting down, so when the camera jumps to the new location, 2 seconds later it jumps to another point that was determined to be closer, skipping the 4 second waiting period.

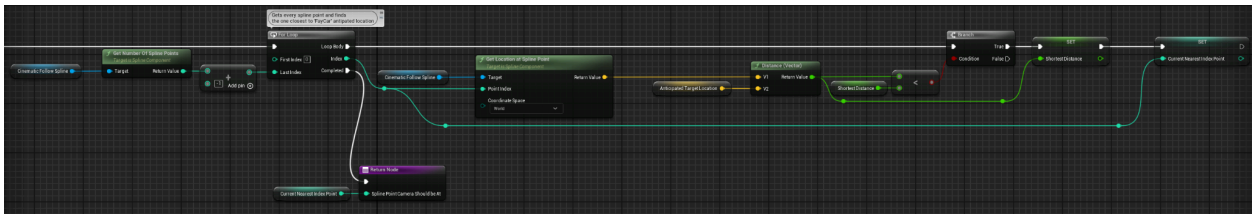
Find Nearest Spline Point to Current Fay Car





When ‘Find Nearest Spline Point to Current Fay Car’ begins, it first calculates the estimated location the ‘currentFayCarToFollow’ will be in the next ‘x’ amount of seconds given its current location and velocity.

it first retrieves the velocity of ‘currentFayCarToFollow’ and multiplies it by the float value of ‘secondsOfTravelToPredict’ (Default value is 2.0), it then adds the result of this to the actor location of ‘currentFayCarToFollow’ and sets the final vector to the ‘anticipatedTargetLocation’ variable.



Next, the number of spline points found within the ‘cinematicFollowSpline’ will be retrieved, that value is then “subtracted” by 1 and fed into the ‘Last Index’ of a ‘For Loop’.

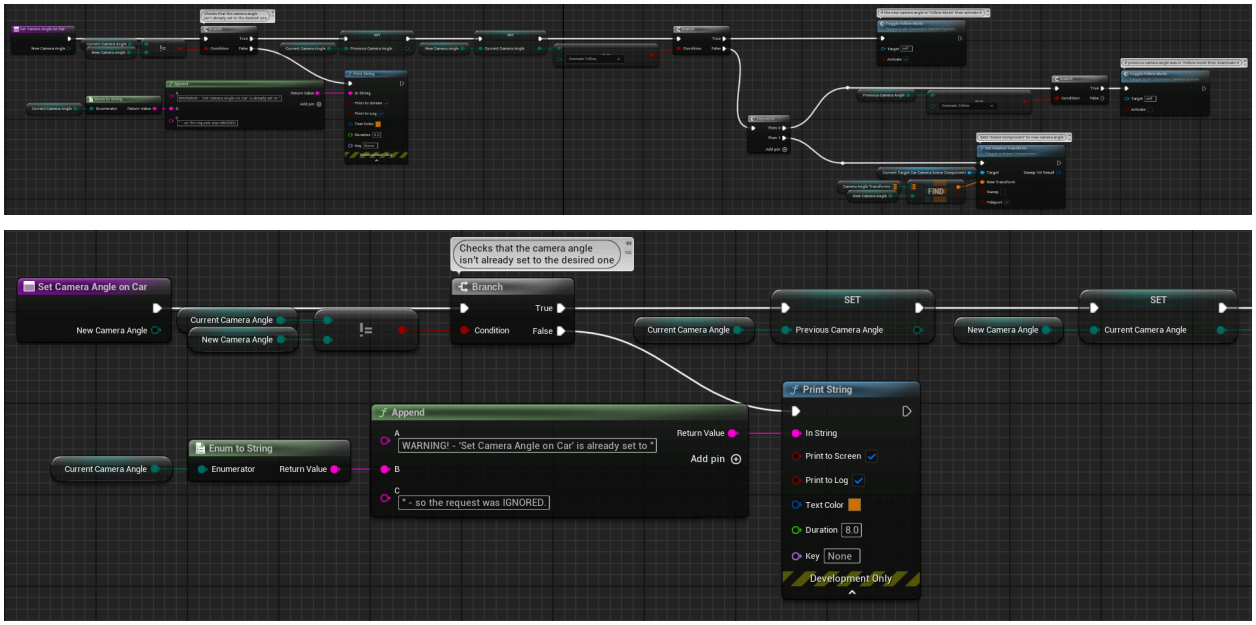
The reason it gets “subtracted” by 1 is because the numbers start at 0 as far as the ‘For Loop’ is concerned, but the result of ‘Get Number of Spline Points’ doesn’t account for this when retrieving the amount (thankfully). What this means is that if there are 15 spline points then that first spline point would be seen as ‘spline point 0’. Also, the reason “subtraction” is done by adding the value to a negative one, is atone for my fear at the impossible possibility that you could encounter a situation with a subtraction node that you are subtracting against a negative number, which of course would result in a positive number. This kind of situation could cause untold issues that would likely never be discovered. So yeah it’s staying like that.

As the ‘For Loop’ runs, it will take the current index found at the ‘For Loop’ and find the corresponding index spline point’s world location along the ‘cinematicFollowSpline’. This vector value will then have it’s distance checked against the ‘anticipatedTargetLocation’

and check to see if it’s return float value is less than the one set for the ‘shortestDistance’ float (Default value of 999999.0 to ensure the first point checked is the base line). If it is, then that value will become the new ‘shortestDistance’ and the index point will be saved as ‘currentNearestIndexPoint’.

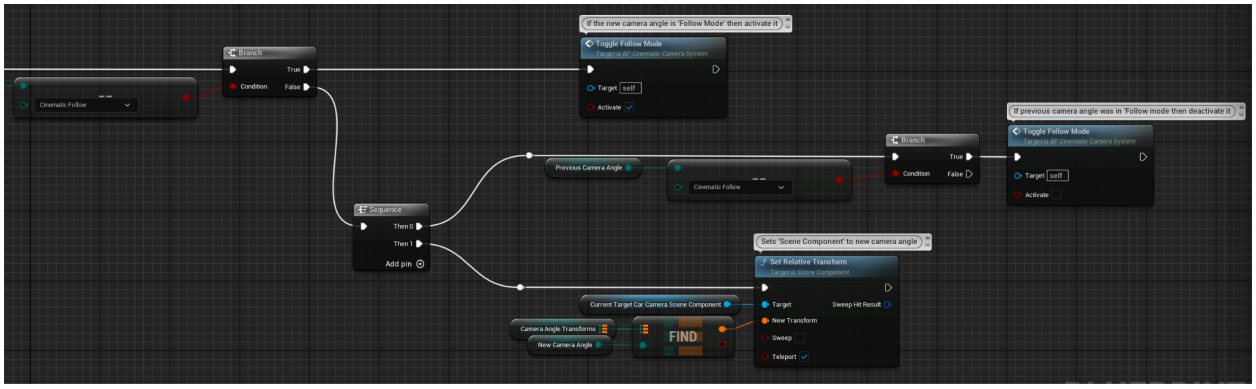
Finally once the ‘For Loop’ has completed, it will take the final value stored within the ‘currentNearestIndexPoint’ and set it the return node's output value of ‘splinePointCameraShouldBeAt’.

Set Camera Angle on Car (Function)



On ‘Set Camera Angle on Car’ the supplied value of ‘newCameraAngle’ will be checked against the value of ‘currentCameraAngle’ if it’s found to be the same value, a warning will be displayed and function will end. This is to prevent unnecessary setting of the camera to the location it is already at.

So long as they are the same value, the ‘currentCameraAngle’ will have its value stored to a local variable of ‘previousCameraAngle’ and the ‘newCameraAngle’ will set the value of ‘currentCameraAngle’.

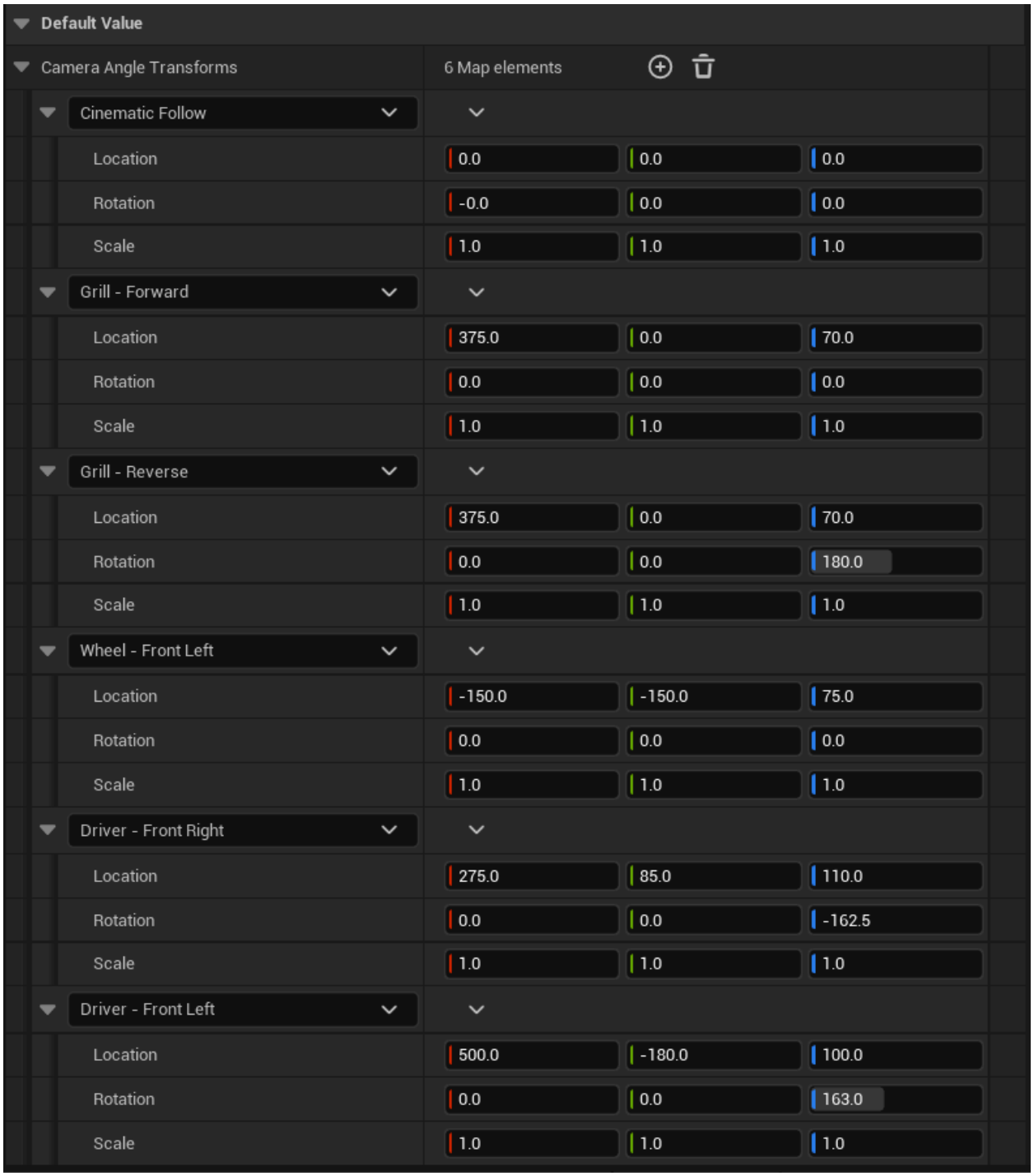


Following this the new value of ‘currentCameraAngle’ will be checked to see if it is equal to ‘Cinematic Follow’ if TRUE then it will run the ‘Toggle Follow Mode’ event with ‘Activate’ ticked.

If FALSE, then it will first check if the ‘previousCameraAngle’ was equal to ‘Cinematic Follow’, and if it was, then it will run the ‘Toggle Follow Mode’ event with ‘Activate’ unticked.

Next it will set the ‘currentTargetCarCameraSceneComponent’ relative transforms to those which correspond to the ones found within the ‘cameraAngleTransforms’ map.

Camera Angles Transforms (Map Variable)



This variable is solely responsible for having the ‘E_CameraAngles’ ENUM correspond to a specified transform. In short, ***this is how you set the location and rotation the camera will be relative to the car.***

To edit the transform values, you need to click on the ‘cameraAngleTransform’ variable found on the left side of your screen under ‘My Blueprint’ and ‘Variables’. Once selected, on the right side of your screen, under the ‘Details’ panel, at the bottom you will find the ‘Default Value’ tab, this is where you edit the values.

If you add a new camera angle listing to ‘E_CameraAngles’, it will not show up in the map, you will first need to hit the plus icon located just under the ‘Default Value’ tab.

To retrieve a camera angles location you simply need to get a reference to the ‘cameraAngleTransforms’, drag off it, and search for ‘Find’, this will allow you to select whichever camera angle found and get the transforms associated with it.

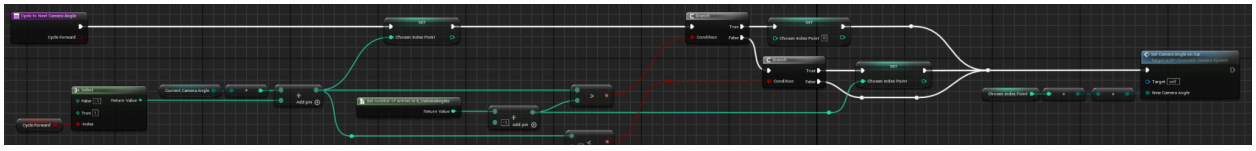
However, you will likely quickly learn just how much a pain in the ass it is to add a new map variable. You can only have one instance of a particular angle type present within the map at any given time, if you try and add two of the same, you will get an error and nothing will happen. The main issue with this is whenever you add a new map variable, the default value becomes the top listing of ‘E_CameraAngles’, and if that value is already within you map, you will get the error and won’t be able to add anything. The way to deal with this is to play a really fucking annoying sorting game of unsetting certain listings, making temporary ones within the ‘E_CameraAngles’, and fumbling through it until you can add what you want. The only strategy to make this slightly easier, is to always have the top value of ‘E_CameraAngles’ at the bottom of the map list, because once something is in the map list *you can not sort it*, things are sorted sequentially.

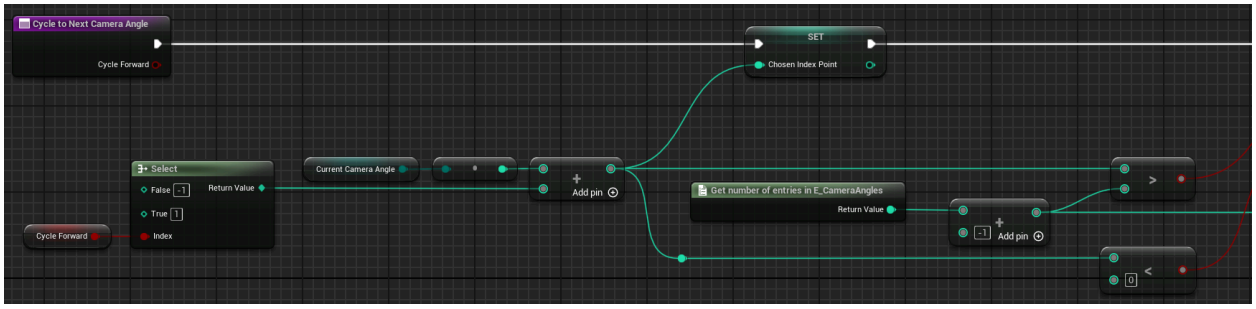
Another important note, within ‘**E_CameraAngles**’...

DO NOT EVER CHANGE THE ORDER OF THE ENUMERATOR LISTINGS!!!!!!!!

If you do, it will more than likely completely fuck up your settings within the ‘cameraAngleTransforms’ map. And when I say “fuck up” I mean it can swap the transform values of the camera angles or even remove listings entirely. This is a ***MASSIVE*** pain in the ass, so I am begging you **DON’T MESS WITH IT!**

Cycle to Next Camera Angle (Function)

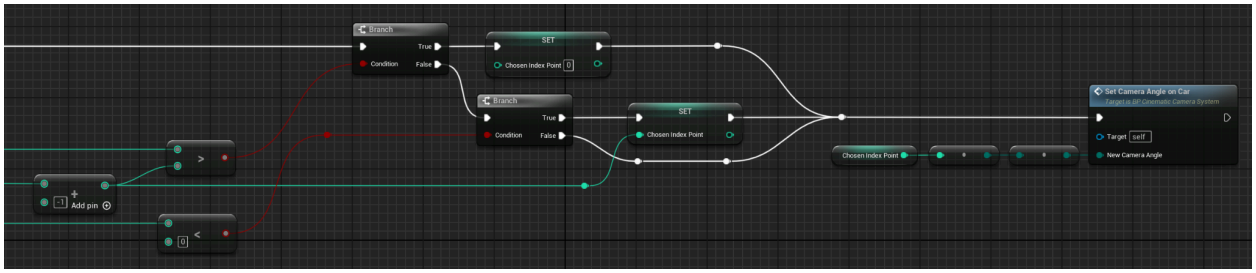




This function is only to be called via the 'BP_Racer_Controller' blueprint.

When the 'Cycle to Next Camera Angle' function is run, it will first retrieve the supplied value of its 'cycleForward' bool and feed it into a select node. The purpose of this bool is to determine if the camera angle listings should cycle *forward* or *backward*, forward being a positive 1, while backward being a negative -1.

The value of 'currentCameraAngle' will be retrieved and converted to an integer value, (this integer value is its listing position within the 'E_CameraAngles') the return value of the previously determined 'Select' node will be added to together, and it's result will become the new set value of 'chosenIndexPoint'.



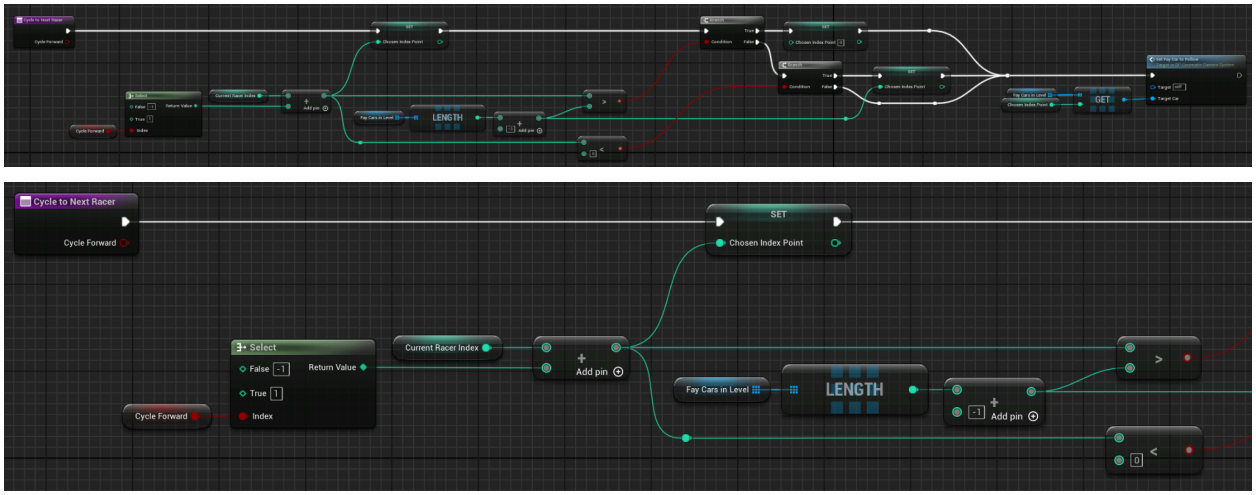
Next, that same value of the addition node will be checked to see if it's greater than the number of listings present in 'E_CameraAngles' (- 1 to prevent an EMAX error). If this value is determined to indeed be greater, this will result in the 'chosenIndexPoint' being set to 0. This is how the angles loop when you reach the last one in the listings.

If it wasn't determined to be greater than, then it will check if that same value is less than 0, if it is, then the 'chosenIndexPoint' will be set to the last listing in 'E_CameraAngles'. This is how the angles loop when you try to go backwards from 0.

If the value wasn't determined to be less than 0 either, then that value will continue unchanged.

The value of ‘chosenIndexPoint’ will then be converted to a byte, that byte will then be converted to the associated ‘E_CameraAngles’ entry found at that listing, and the ‘Set Camera Angle on Car’ function will be ran with that value input to its ‘newCameraAngle’ variable.

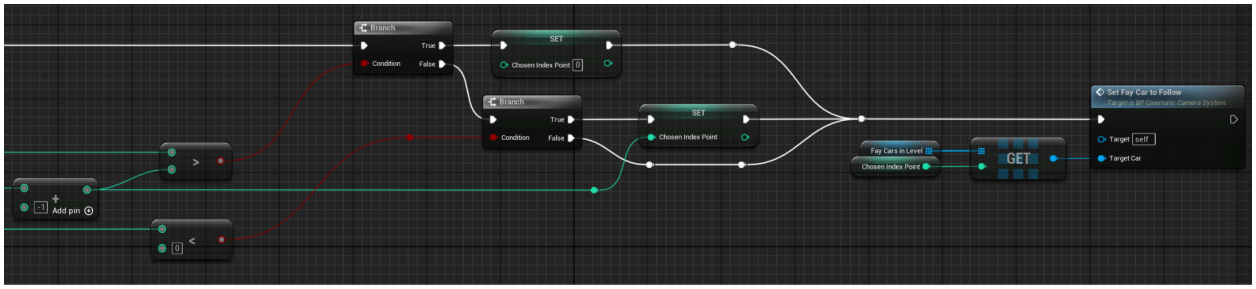
Cycle to Next Racer (Function)



This function is only to be called via the ‘BP_Racer_Controller’ blueprint.

When the ‘Cycle to Next Racer’ function is run, it will first retrieve the supplied value of its ‘cycleForward’ bool and feed it into a select node. The purpose of this bool is to determine if the racer index should cycle *forward* or *backward*, forward being a positive 1, while backward being a negative -1.

The value of integer value of ‘currentRacerIndex’ and the value of the previously determined ‘Select’ node will be added together, and it’s result will become the new set value of ‘chosenIndexPoint’.



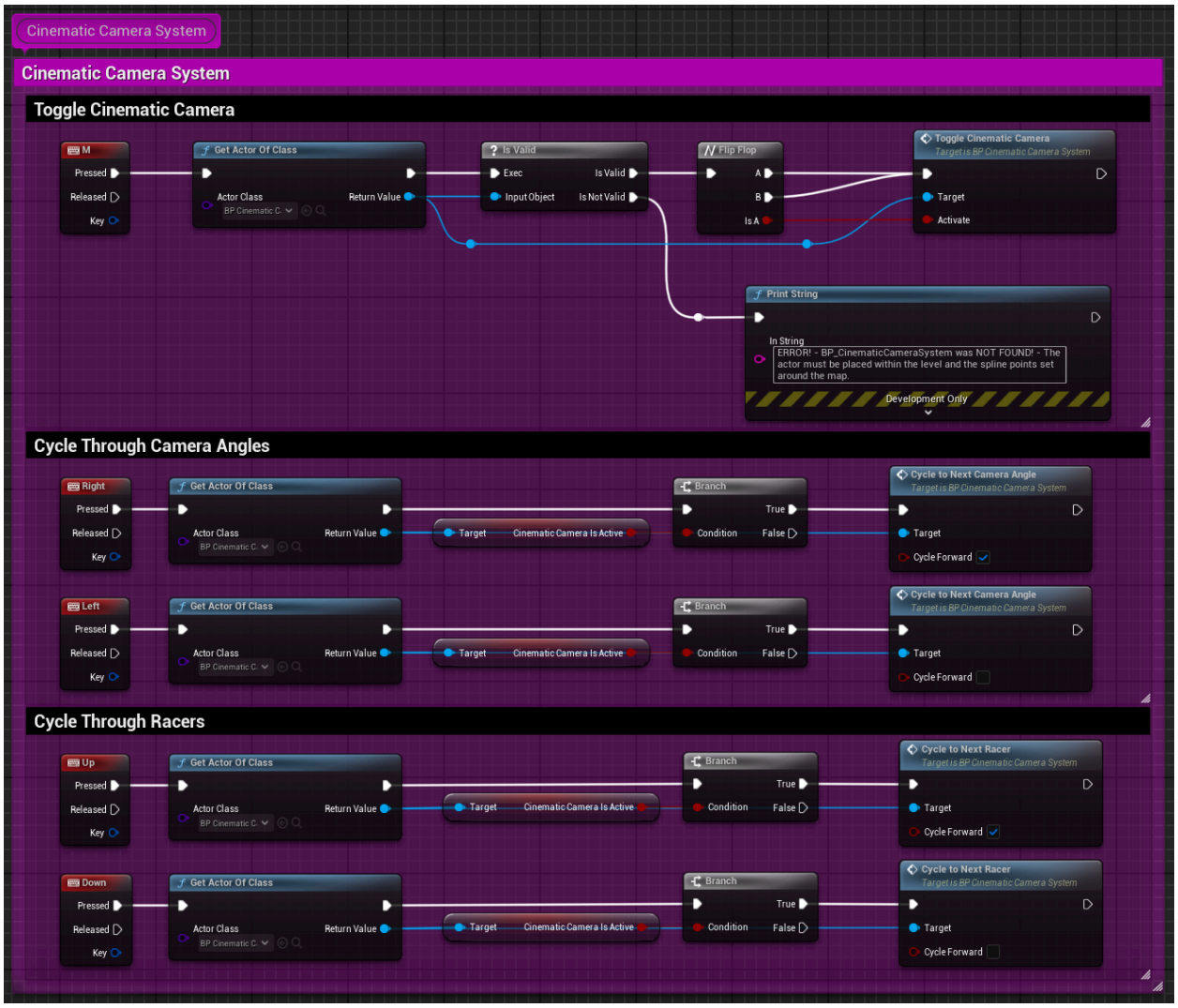
Next, that same value of the addition node will be checked to see if it's greater than the array length for 'fayCarsInLevel' (- 1 since index starts at 0). If this value is determined to indeed be greater, this will result in the 'chosenIndexPoint' being set to 0. This is how the racer index loops when you reach the last one in the array.

If it wasn't determined to be greater than, then it will check if that same value is less than 0, if it is, then the 'chosenIndexPoint' will be set to the last listing in the 'fayCarsInLevel' array. This is how the racer index loops when you try to go backwards from 0.

If the value wasn't determined to be less than 0 either, then that value will continue unchanged.

The value of 'chosenIndexPoint' will then be used to retrieve the item at that corresponding index location of the 'fayCarsInLevel' array. Following this the 'Set Fay Car To Follow' function will be run, and the result of that 'Array Get' will be fed into the 'targetCar' input.

BP_Racer_Controller



Within the ‘BP_Racer_Controller’ is where the inputs for the ‘BP_CinematicCameraSystem’ are found. At the moment, these inputs are event keys and not controller bindings.

Toggle Cinematic Camera

On ‘Keyboard Event M’, a ‘Get Actor of Class’ node will search for the ‘BP_CinematicCameraSystem’ actor within the level (there should only ever be ***ONE***), it will then check if the return of that search was valid. If it was then it will feed into a ‘Flip Flop’ node that will run ‘Toggle Cinematic Camera’ with the ‘Activate’ bool value bound

to the ‘Is A’ of the ‘Flip Flop’. But if it wasn’t valid, then it will display a warning and notify the user of the problem.

Next Cycle Through Camera Angles / Racers

These functionally work the exact same way so I’m stitching them together here.

On ‘Keyboard Event [Arrow Key]’ a ‘Get Actor of Class' node will search for the ‘BP_CinematicCameraSystem’ actor within the level. It will then read the bool value of ‘cinematicCameraIsActive’ to make sure the player is actually in the ‘BP_CinematicCameraSystem’ before they’re allowed to cycle through angles or racers. Then the corresponding functions are run with the bool ‘cycleForward’ being ticked or unticked depending on whether you want that particular input to cycle forward or backward within the function.