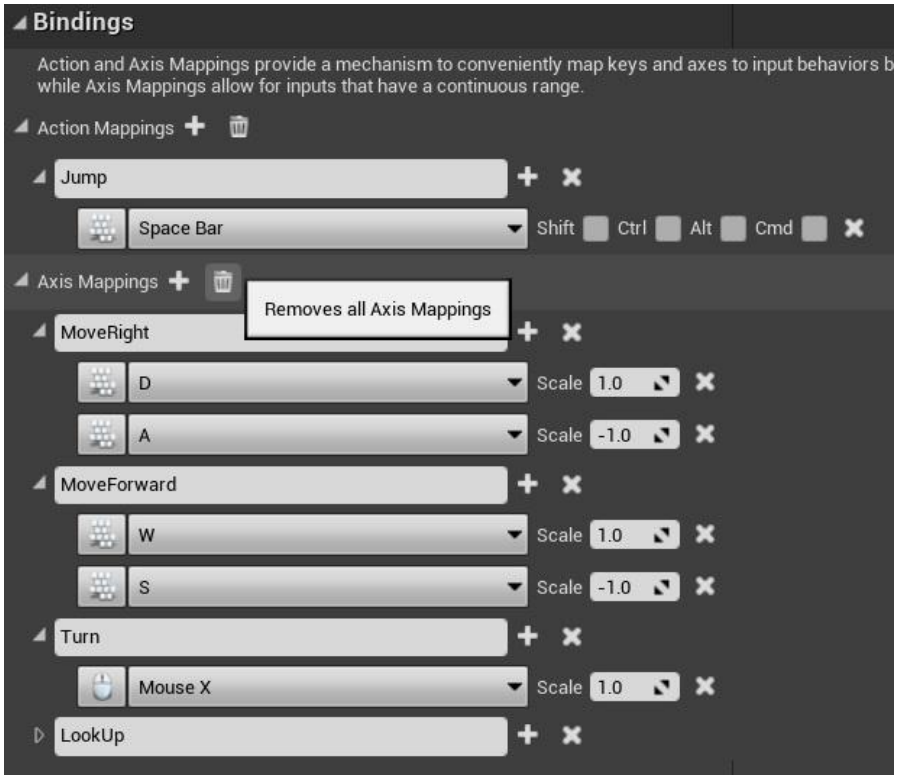
- *Detailed analysis, overview and justification of your design. Why did you develop your code the way you did? Does it do what you want it to do?*

Detailed Analysis:

We planned on making a First-Person-Shooter when we first started planning the project. We took the game "duck hunt" as a reference, but then we deviated a little differently to have a coin collecting mechanism as well.

Key and Mouse Bindings

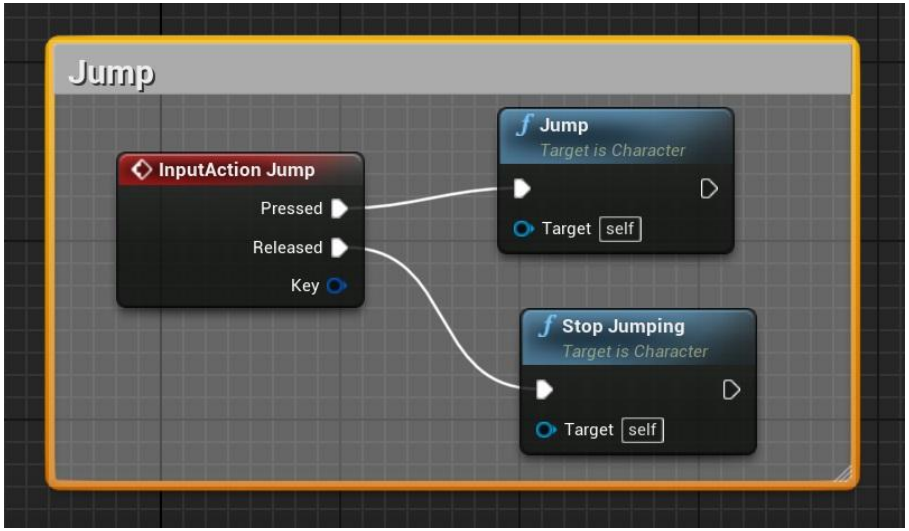


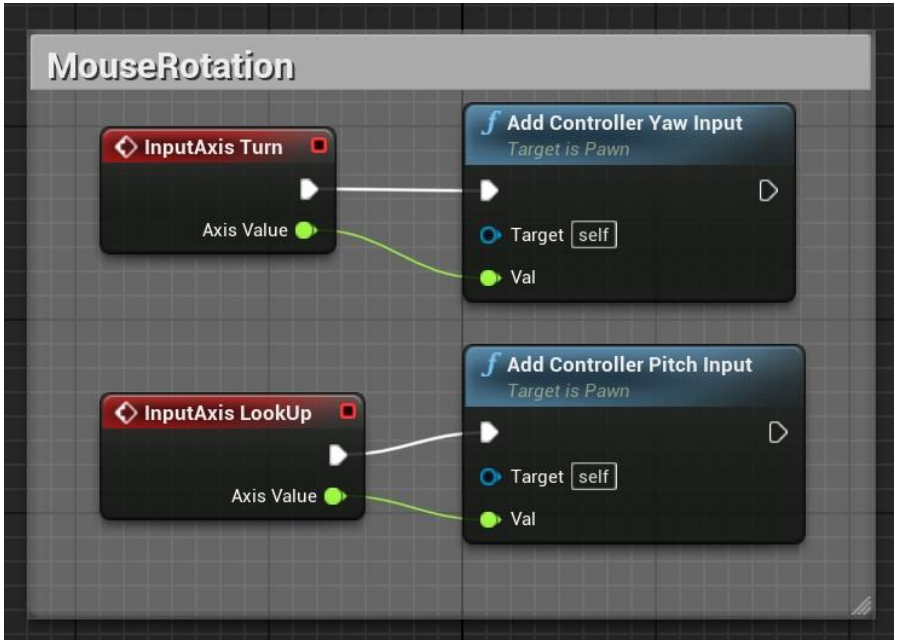
We used some key bindings for the movement of the "actor" throughout the environment. Here we just initialized the functionality with the key bind. The documentation for the functionality is posted below.

⇒ We initialized the "Jump" key bind with the Space Bar in the keyboard.
⇒ We also used the keys "D," and "A" to move right and left.
⇒ We also used the keys "W" and "S" to move up and down.
⇒ And Finally, we used the movement of the mouse in the x-direction as "Turn" and in y-direction as "LookUp/LookDown"

Jump

Since we've linked the Space Bar with the Jump function, whenever we press it, it triggers this function.
⇒ The jumping motion is pre-built into Unreal engine, so we just need to call the function.
⇒ Thus, when the SpaceBar is clicked the "actor" jumps.

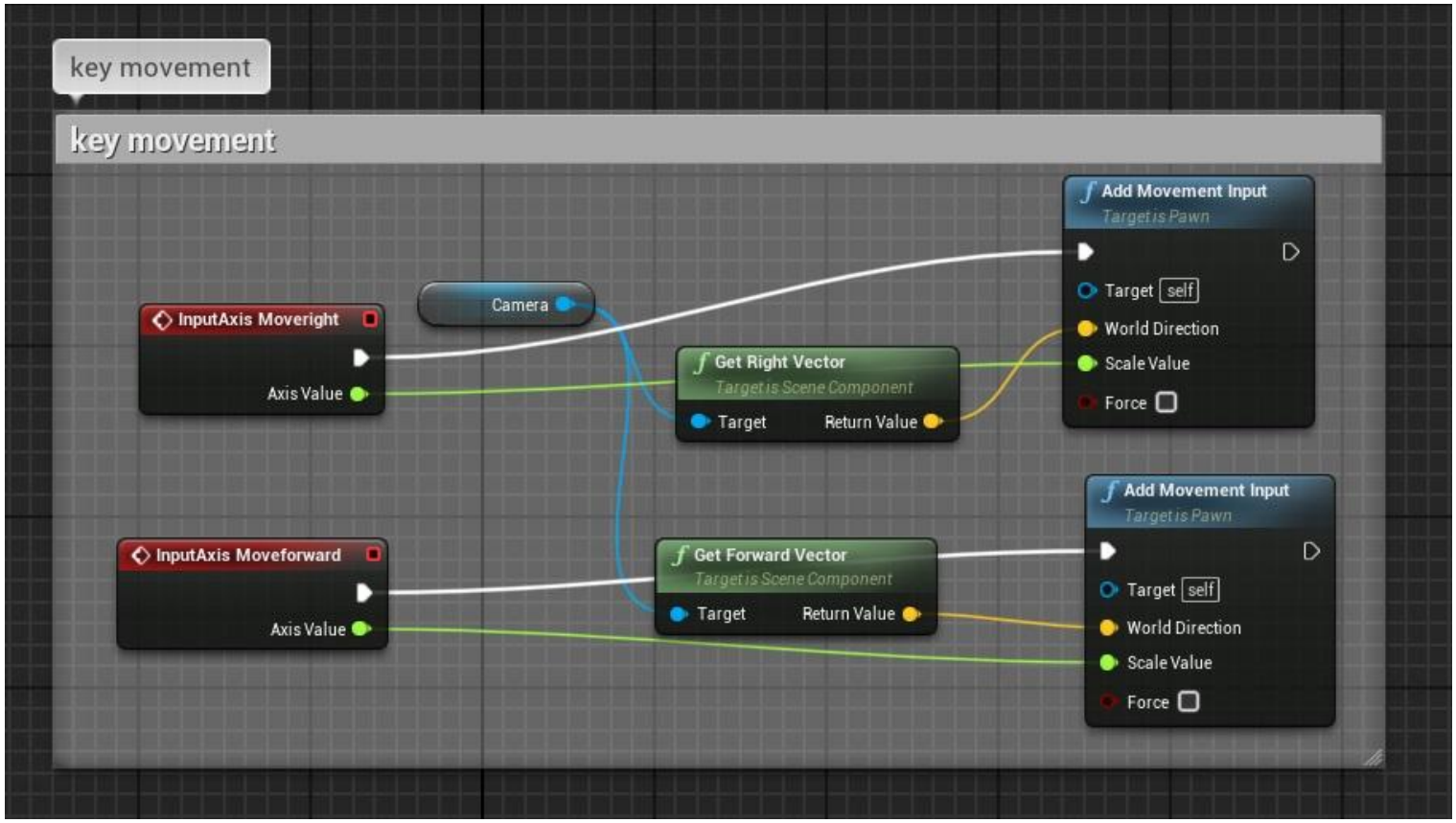




Mouse Rotation

For looking left/right/up/down, we use the in-built functions for Yaw and Pitch in Unreal Engine.
⇒ Whenever we turn in the y-direction it moves by that angle using the Controller Pitch Input. Pitch defines the movement from up to down.
⇒ Whenever we turn in the x-direction it moves by the angles using the Controller Yaw Input. Yaw defines the movement from left to right.

WASD Keybind



⇒ For Left/Right movement, we press "A/D." Then, it gets the right vector of the camera and translates it to the world direction.

⇒ For Right, the scalability is multiplied by 1

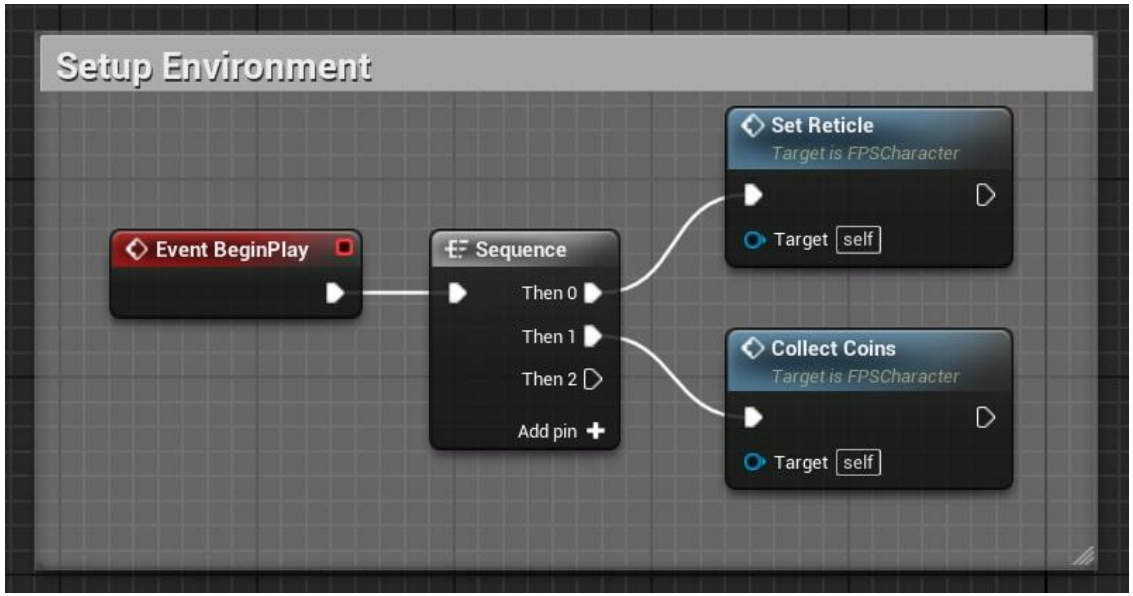
⇒ For Left, the scalability is multiplied by -1 so that it moves left (opposite direction)

⇒ For Front/Back movement, we press "W/A." Then, it gets the forward vector of the camera and translates it to the world direction.

⇒ For Front, the scalability is multiplied by 1

⇒ For Back, the scalability is multiplied by -1 so that it moves left (opposite direction)

Setup Environment



⇒ When we start the event, we run two things in a sequence (at the same time).

⇒ We initialize the set Reticle function, we set the cross hair of the gun, the timers, scores (coins dropped, collected, accuracy) and make it ready.

⇒ We also initialize the Collect Coins function that starts collecting coins if we walk over it.

Starting

⇒ When we call the set Reticle function in the setup environment, we run a sequence of two things.

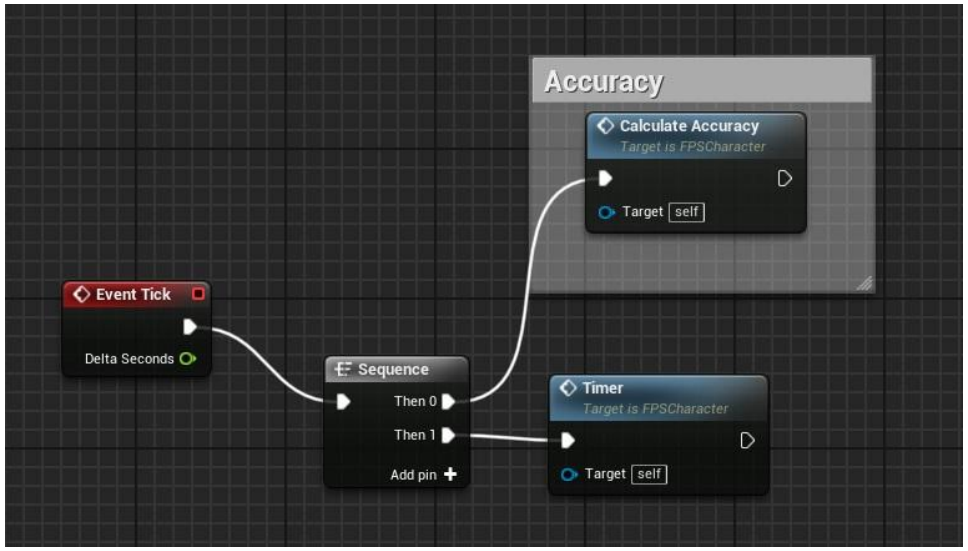
⇒ First, we Create Reticle Widget (through which we display the info) and Add it to the Viewport (the screen)

⇒ The Owning player in the widget is set to the actor or the First-Person player playing the game.

⇒ Second, we stop the rifle audio from playing because it shoots whenever we set the reticle.

⇒ You can see how we used Sniper Gunshot as the target for this one.

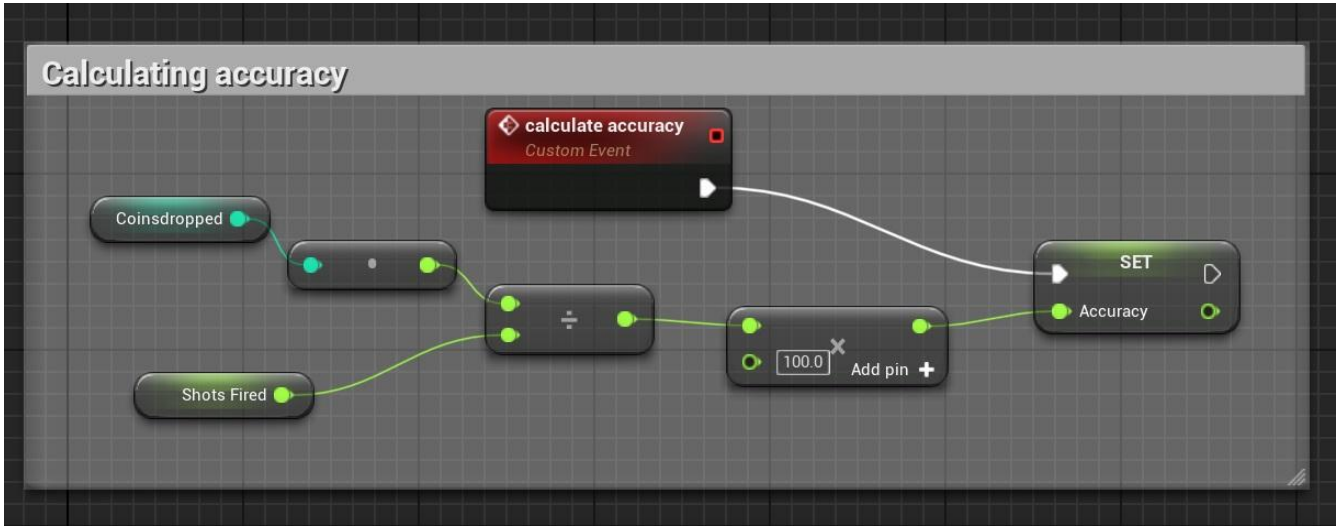
Accuracy



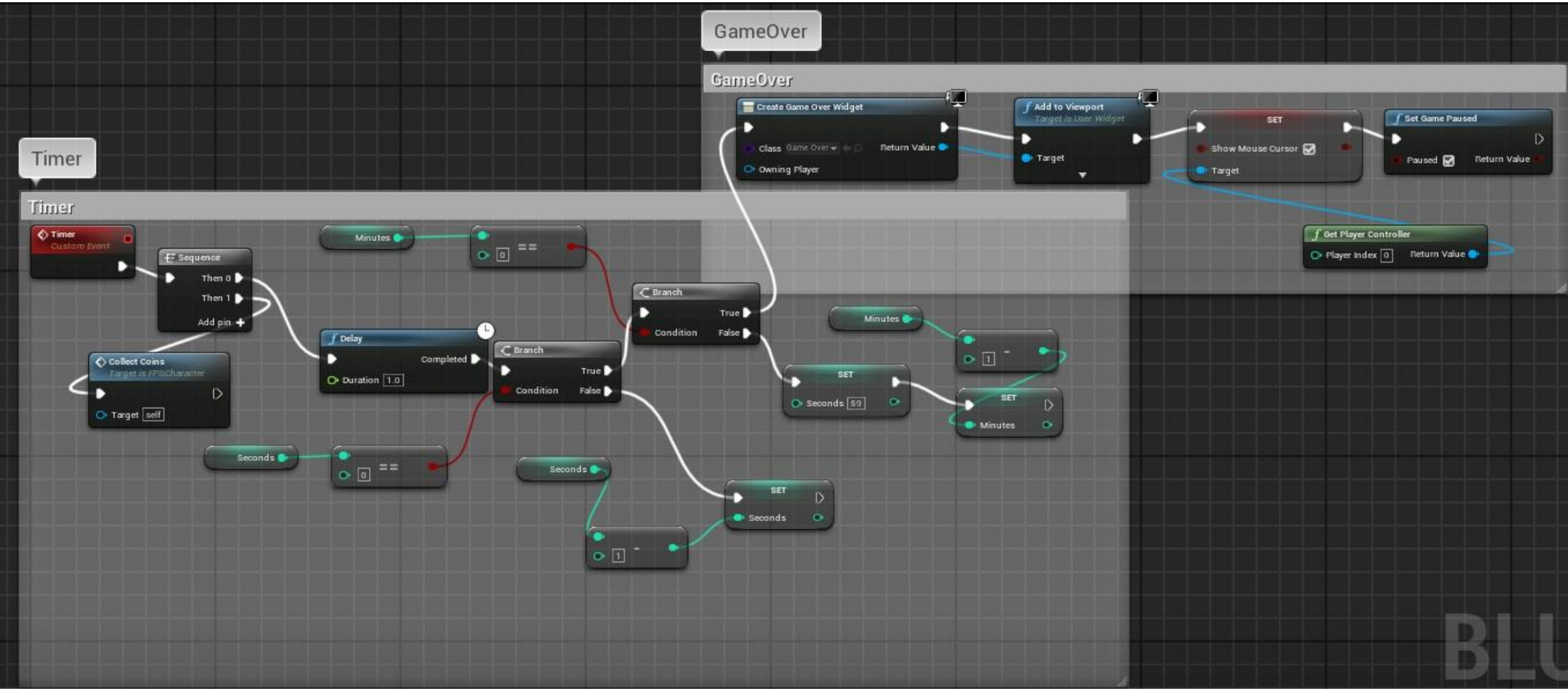
⇒ We use the Event Tick function that basically ticks or calls this function after every second. (Basically like a ticking clock).
⇒ After each tick, we call the Calculate Accuracy and Timer function in sequence.
⇒ The Calculate Accuracy function Calculates the accuracy of our shots.
⇒ The Timer function starts counting down the timer.

Calculate Accuracy

⇒ To calculate the accuracy, we use the number of coins dropped and shots fired and multiply it by 100% to get the accuracy.



Timer / Game Over



⇒ As the game starts, the timer function is called, which starts ticking down the time.

⇒ It's default value can be adjusted for both minutes and seconds.

⇒ It runs in a sequence - collecting coins and delaying the timer by 1 sec.

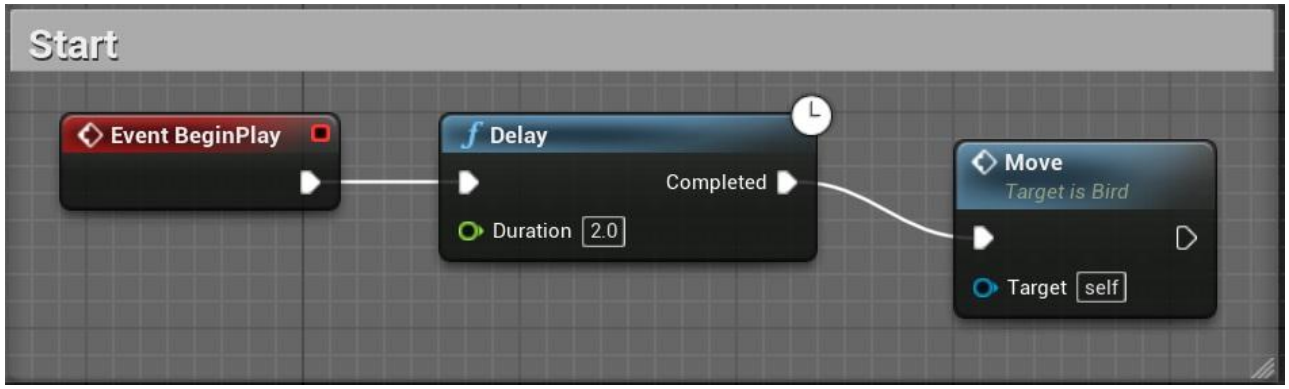
⇒ We have a few branches in the system. The first one checks if the seconds are equal to 0. If not, it'll decrease by 1.

⇒ If yes, it'll check if the minutes are 0 or greater. If greater, it'll decrease the minutes to 0. If minutes is 0, then it'll Create the Game Over Widget.

⇒ We add the Game Over HUD Widget to the Viewport .

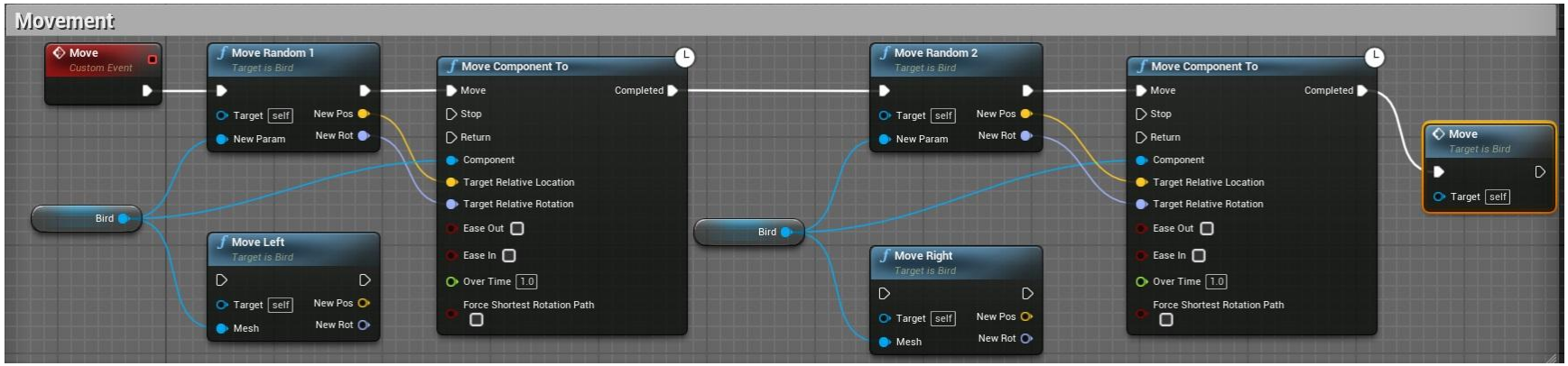
⇒ Then, we set the mouse cursor to the screen so they can choose to restart the game if they wish to do so.

Start Bird Movement



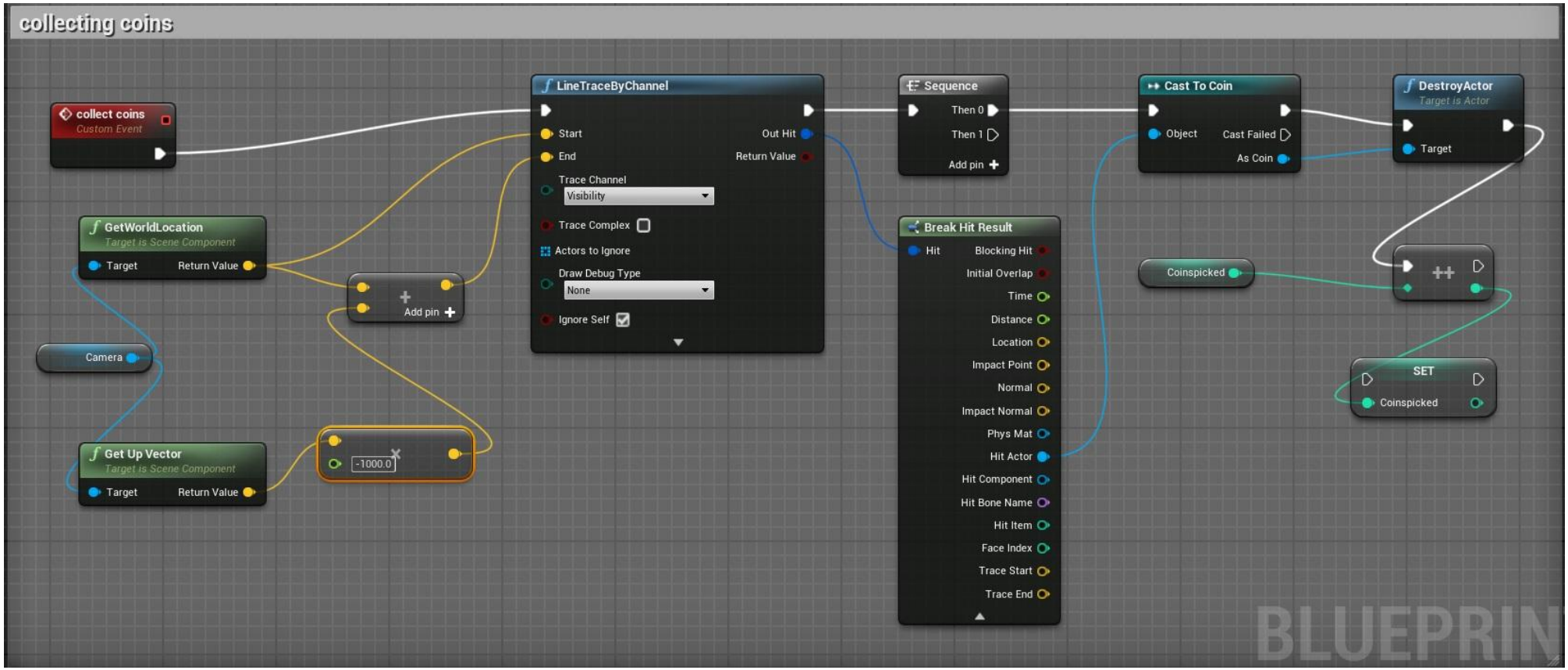
⇒ When we start the game, we delay the movement of the birds for about 2 seconds before they start moving.
⇒ We call the Move function to start moving the birds randomly.

Bird Movement



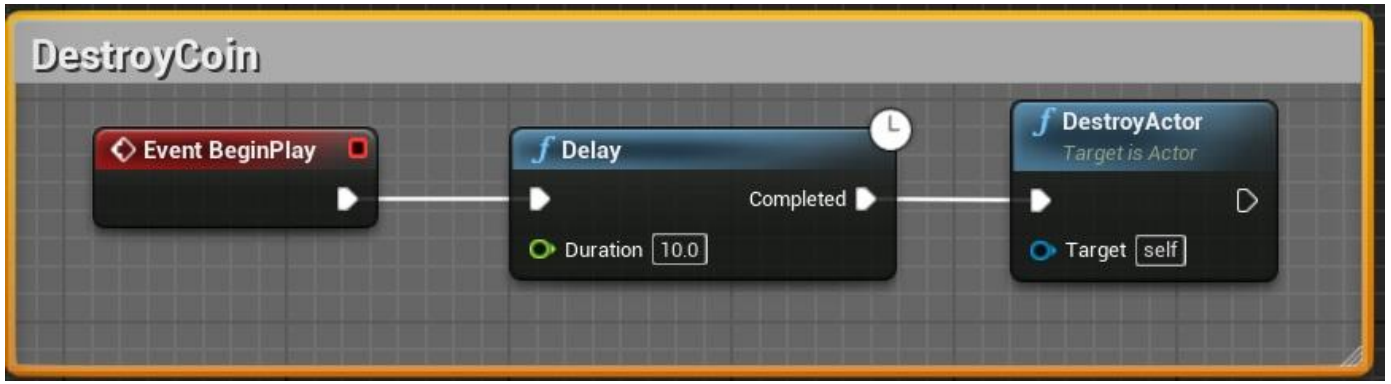
⇒ When the move function for the bird is called, we call the Move Random 1 function that generates a vector in a random x-y plane direction.
⇒ Using this random vector, we translate the vector left based on the randomized vector.
⇒ The bird moves in this random direction for 1 second.
⇒ Then, we call another Move Random 2 function that generates another vector in a random x-y plane direction.
⇒ Using this random vector, we translate the vector right based on the randomized vector.
⇒ The bird moves in this random direction for 1 second.
⇒ After the bird moves in these random directions, we call the Move function again (recursively) to repeat the same process.

Collecting Coins:



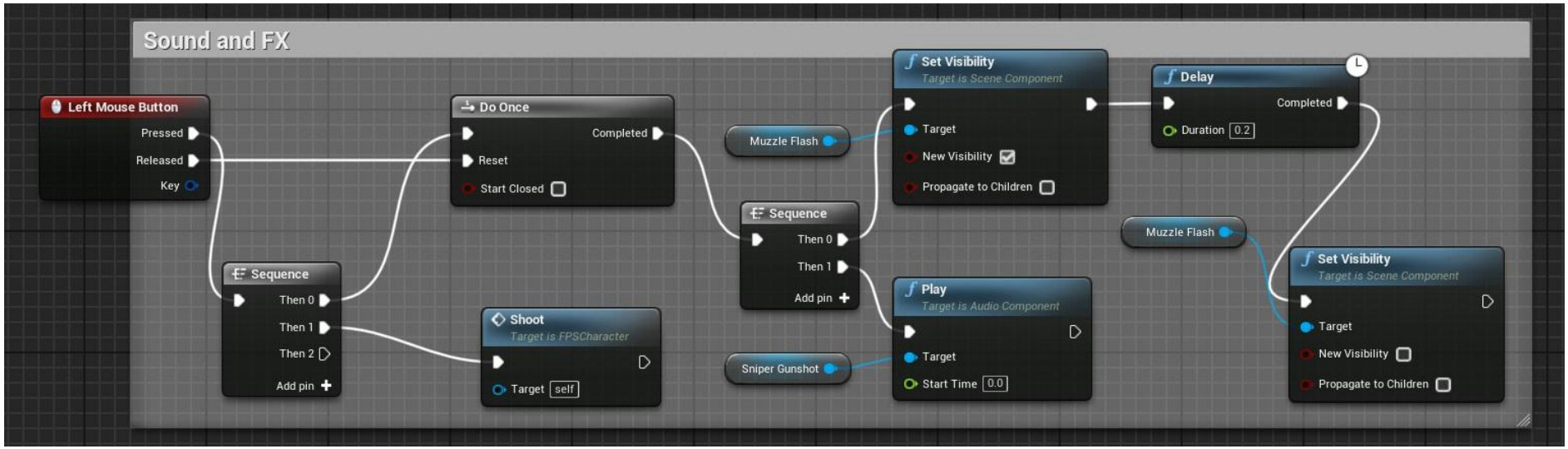
⇒ The movement of the coins (dropping) is due to the built in physics inside Unreal Engine.
⇒ After the collect coins function is called, we use the world location of the "actor" and the camera's up vector (multiplying it by -1000 so that it points down) to calculate if we're collecting the coins or not.
⇒ The LineTracebyChannel sets a starting point of the vector (Our world position) and an endpoint (the Downward Vector).
⇒ If another object (in this case the coin) hits the line between the start and end point, it calls the Cast to Coin function that further calls the Destroy Actor function that removes the coin from the environment and increments the coins picked variable.

Destroy Coin



⇒ When the game starts and the bird starts dropping the coin, there is a 10 second buffer between the actor picking up the coin and it being destroyed (disappeared).

Sound and FX

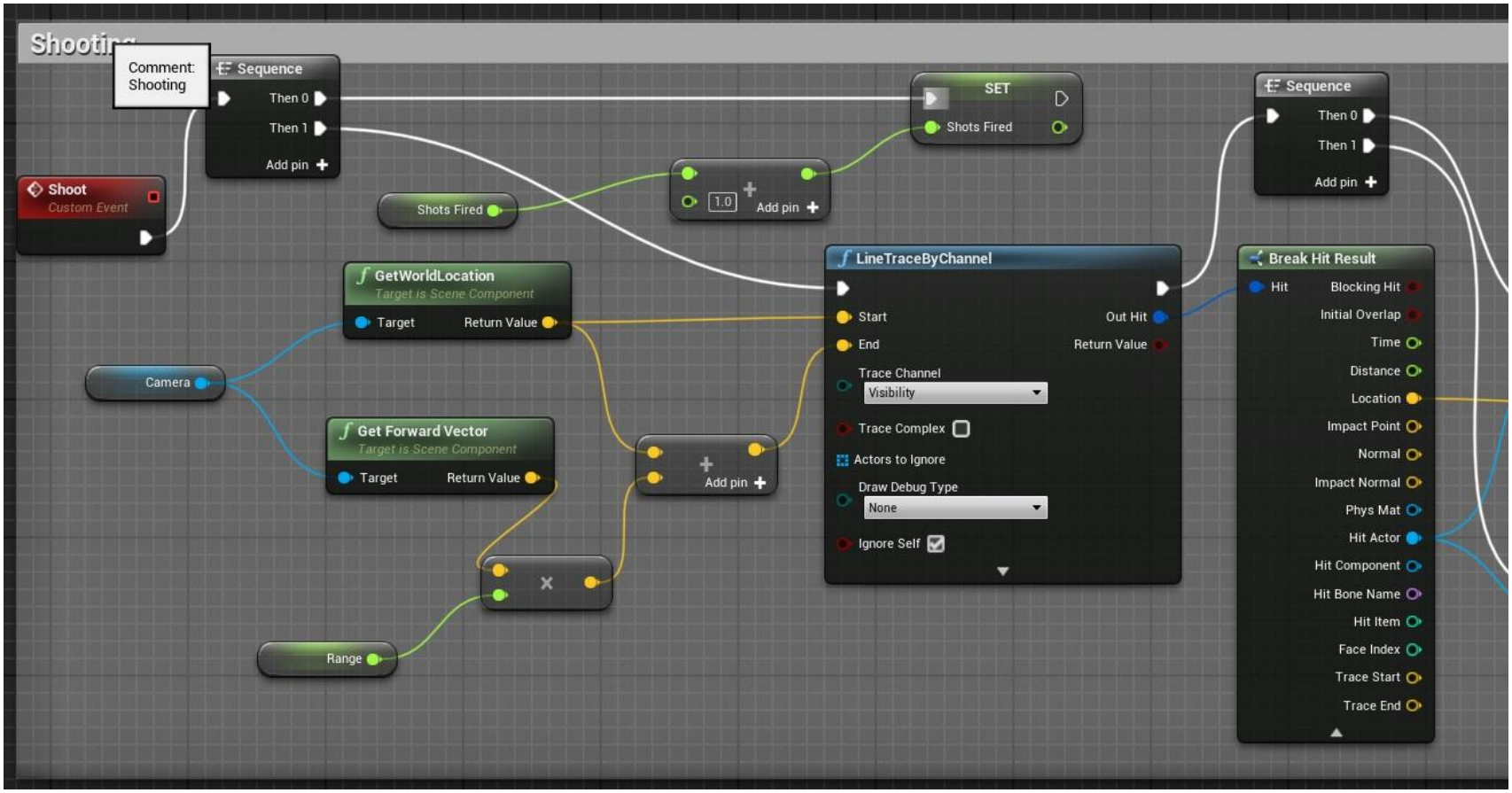


⇒ When the left mouse button is pressed, firstly the shoot function described above will be called.

⇒ The action is only done once so that the muzzle does not keep turning on if the button is kept pressed. The Do once function is reset after the button is released.

⇒ The flash on the tip of the gun is set visible for 0.2 seconds and turned off again. The gunshot's audio as a .wav file is also played once.

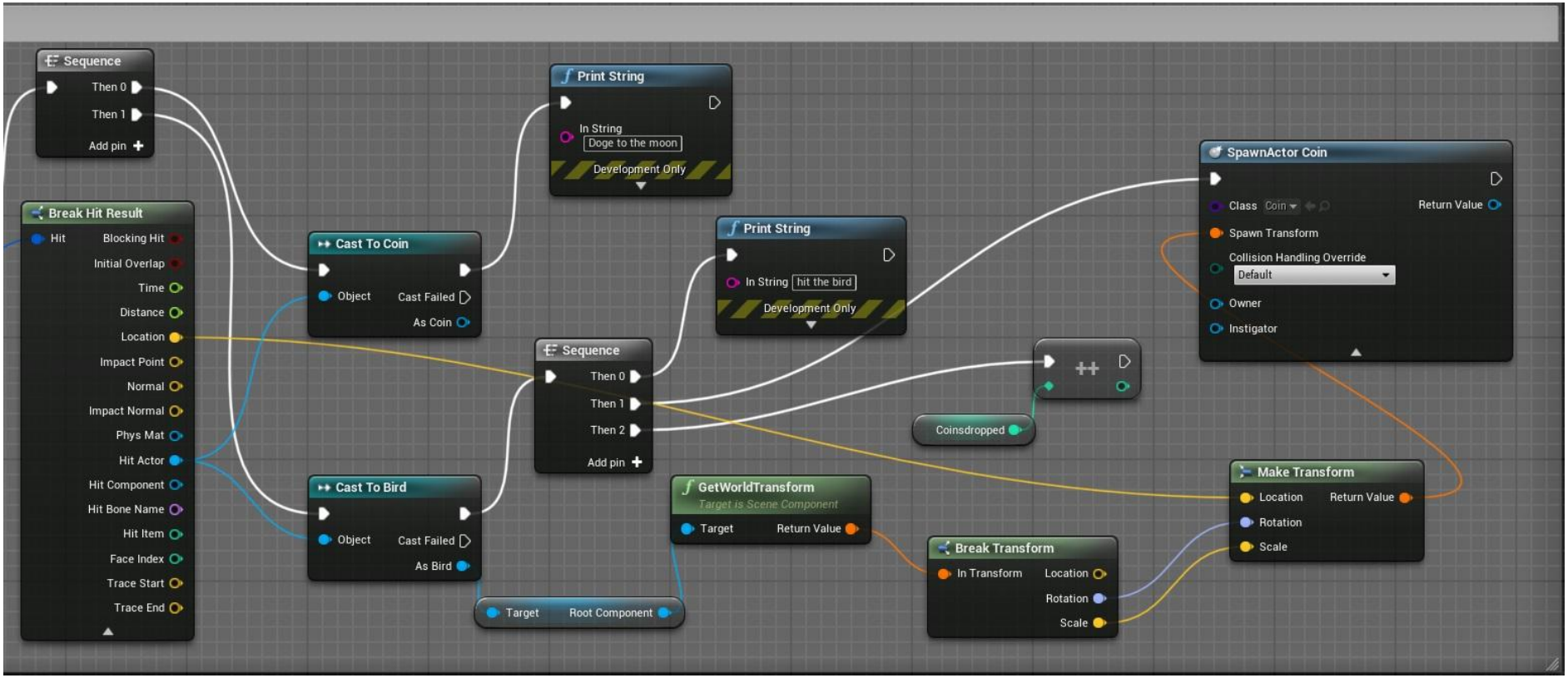
Shooting



⇒ When a shoot function is called, first the shots fired variable is incremented to display on HUD and calculate accuracy.

⇒ We then get a unit vector in the forward direction of the camera and multiply it with the range of the gun. We then add the vector to the world location of the camera to get a ray in forward direction or the shooting range of the gun.

⇒ The start of the ray will be the world location of the camera and will end at the range. It then checks if the existing line or ray collides with any object in the scene that has its collision on.



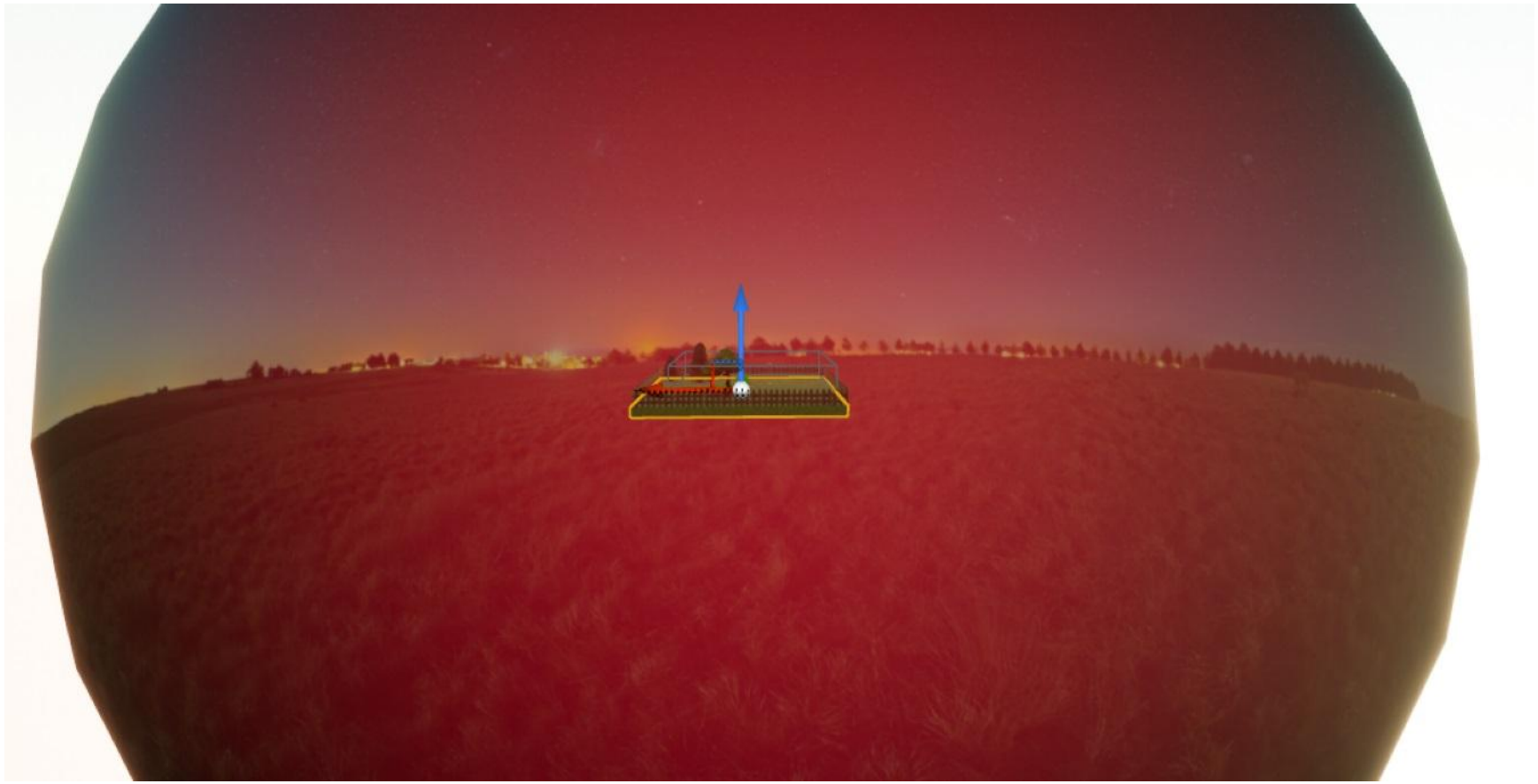
⇒ Any object with collision will block or stop the ray. If the colliding object is a coin or a bird, only then the specific object and its location is called for further processing.

⇒ If the bird collides with the ray, the number of coins dropped is incremented and accuracy is updated.

⇒ All the transformation vectors done to the bird is passed to the coin so that its location would be where the bird was hit.

⇒ A new instance of coin class is called and it is transformed and located in rendered in the world space where the bird was hit. It then simulates physics to drop to the ground.

Skysphere

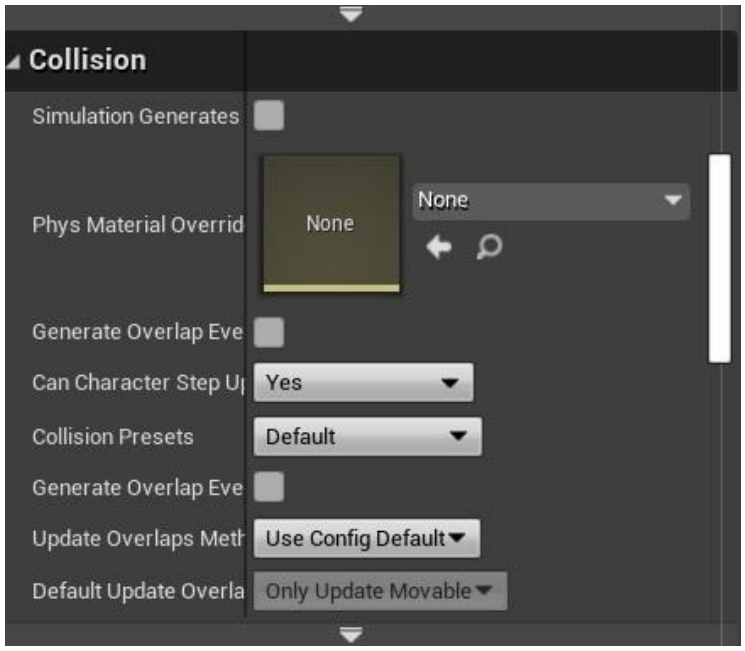


⇒ Outside view of the Skysphere in Unreal Engine .

Lighting

⇒ There are many lighting options in Unreal Engine. Among which, we didn't have enough time to explore each well.
⇒ Plus, we ran into a bug with Sky Light where it just turned dark if we moved into a certain area of the map. We tried looking for a solution online, but couldn't find it.
⇒ Thus, we ended up using the Ambient and Diffuse lightings within SkyLight and left the shadows for now.

Collision Detection



⇒Unreal Engine has built-in collision detection which we used for collisions in the game.
⇒ The one objection that we had to make was for the sky sphere since if it's collision is turned on, the actor couldn't move around at all in the map.