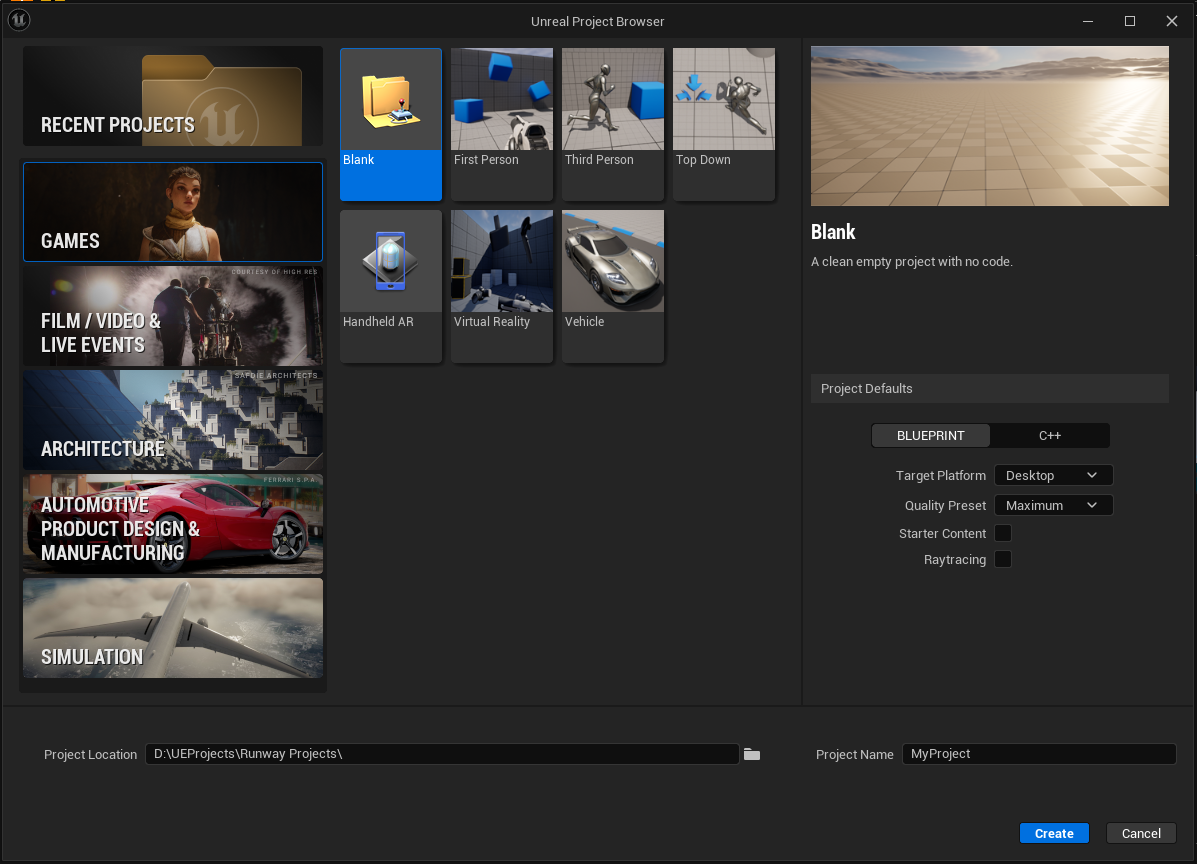
**How to make a platformer using Unreal Engine 5 and visual scripting**

What we need:

1. Unreal Engine 5.3.2

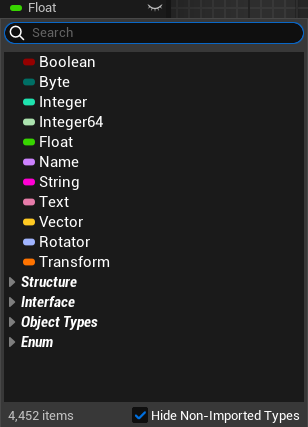


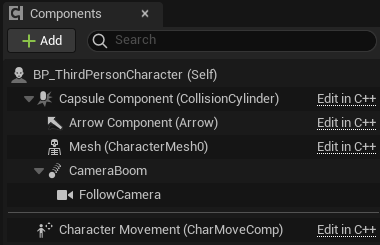
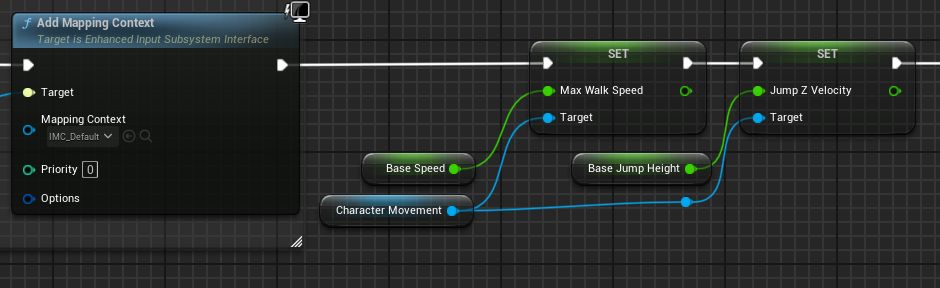
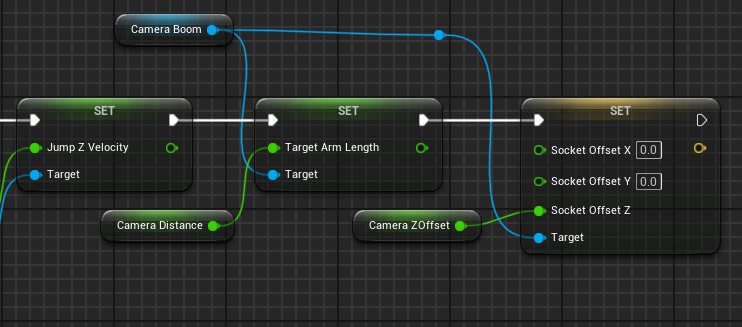
1. Create a new project using the “Third Person Template”(Games->Third Person, Make sure blueprint is selected, uncheck “Starter Content”)



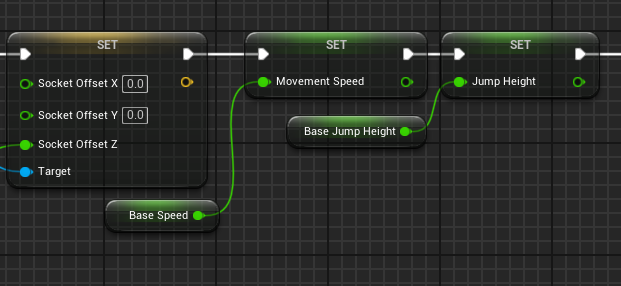
Step by Step:

Step1: Making the Playable Character

1. Open BP\_ThirdPersonCharacter. We are going to create 4 float variables named “CameraDistance”, “CameraZOffset”, “BaseSpeed”, and “BaseJumpHeight”. The default engine values are 400, 0, 500, and 700. Also make 2 additional float variables to be used later on. Name these “MovementSpeed” and “JumpHeight”.
   1. Click the “+” button on the variables tab.
   2. Name the variable.
   3. From the drop down menu select “Float”
2. In the “Event Graph” at the end of “Event BeginPlay” node you will see a “Add Mapping Context” node. We will be working off the end of this.
   1. Drag the “Character Movement” component from the “Components” section into the event graph and select “Get Character Movement”

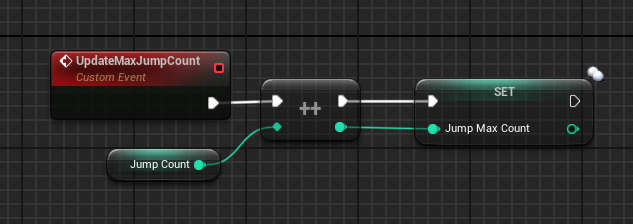
* 
  1. Now do the same with the BaseSpeed and BaseJumpHeight variables we made.
     1. Drag off of the Character Movement node and search for “Set Max Walk Speed” and create the node. Now connect the Base Speed node into the new node and also connect the Add Mapping Context node to this new node. Also do this for “Jump Z Velocity” from the Character movement.
* 
  1. Repeat that but for the Camera Boom, and search for “Target Arm Length” and “Socket Offset”. On the Socket Offset node right click the pin and in the menu select “Split struct pin”. We will also bring out CameraDistance and CameraZOffset from the variables tab. Connect CameraDistance to Target arm length and CameraZOffset to Socket offset Z.
* 

1. At the end of begin play we will set the values of the “MovementSpeed” and “JumpHeight” variables so we have fast access to those values when we make a temporary change to them.

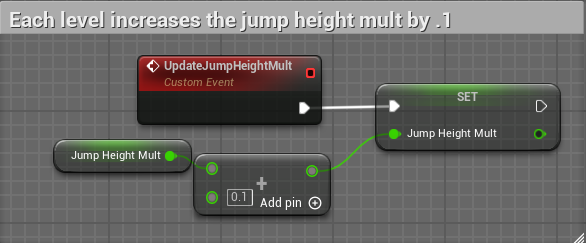
* 

Step 2: Power Ups and Upgrades

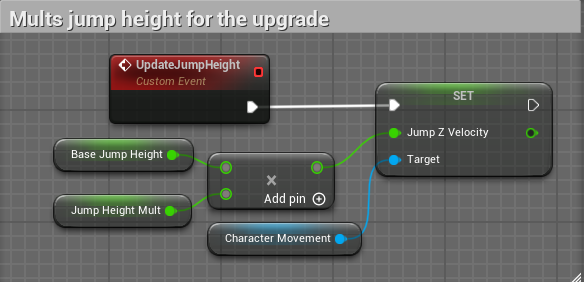
1. The first upgrade we will be looking at is an ability to jump multiple times. Start off by creating a variable and name it “JumpCount” and make it an Integer, default value should be 1.
2. Right click on the event graph and search for “Add custom event” rename the new event to “UpdateMaxJumpCount”. Add a get node for the JumpCount variable and create a new node branching from it and search for “Increment Int” also at this time plug in the event exec into the increment. Right click on the event graph and search for the “Set Jump Max Count” node, and set it equal to the new value of Jump count.

* 

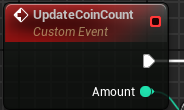
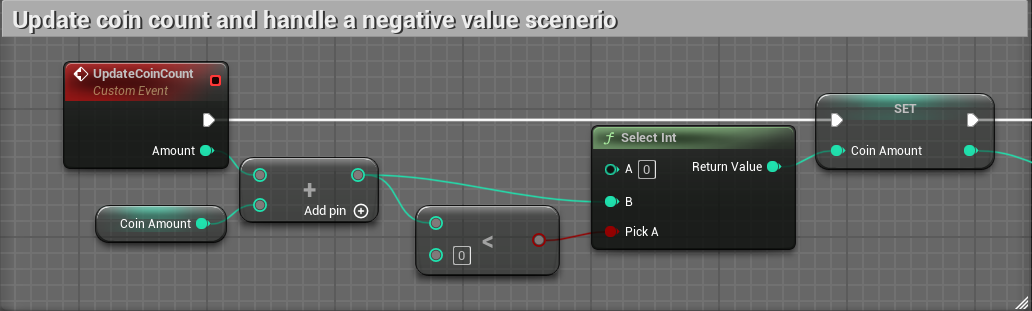
1. The next upgrade we will add will be increasing the player’s jump height. Create another custom event, name this one “UpdateJumpHeightMult”.
   1. Create a new float called “JumpHeightMult” and set the default value to 1.0.
   2. For this event add 0.1 to the current value of JumpHeightMult and set it to the new value.

* 

1. Make a new custom event called “UpdateJumpHeight” add a call to it at the end of “UpdateJumpHeightMult” so it gets called whenever we update the multiplier.
   1. Multiply the base jump height by the multiplier and set JumpZVelocity equal to this new value.

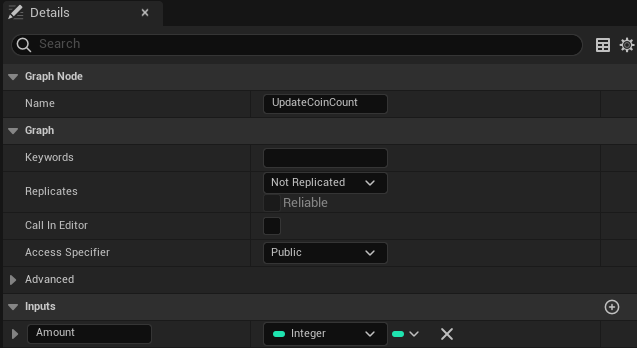
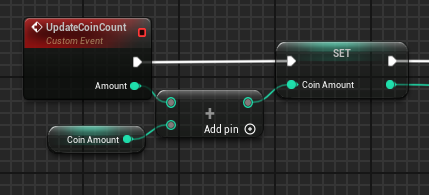
* 

1. Since we will be adding coin pickups and a shop system, let's get the basics for the coin counting setup.
   1. First create an int variable and name it “CointAmount”.
   2. Now right-click the event graph to create a custom event and name it “UpdateCoinCount” and add an integer input.

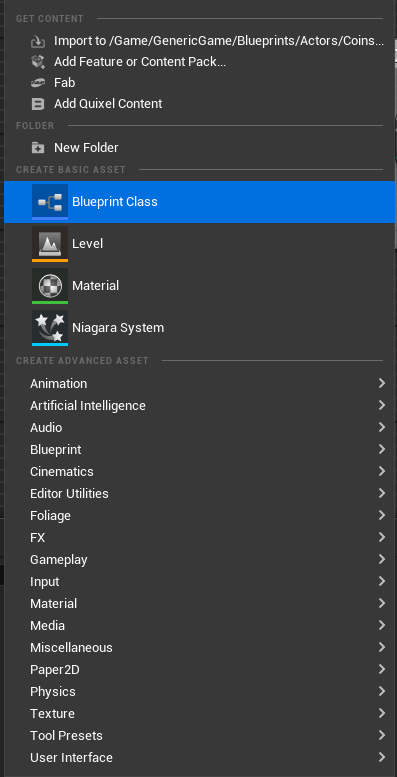
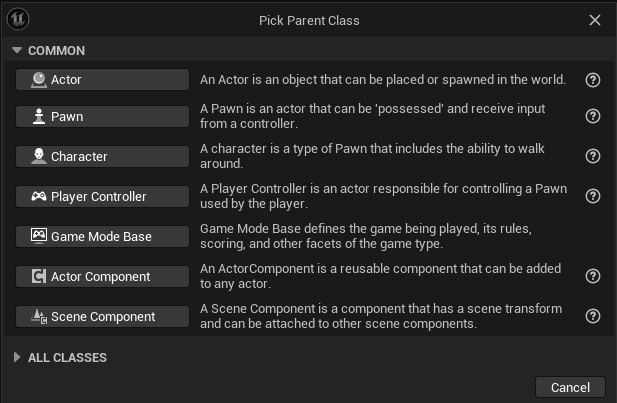
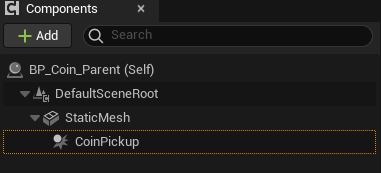
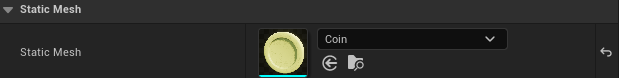
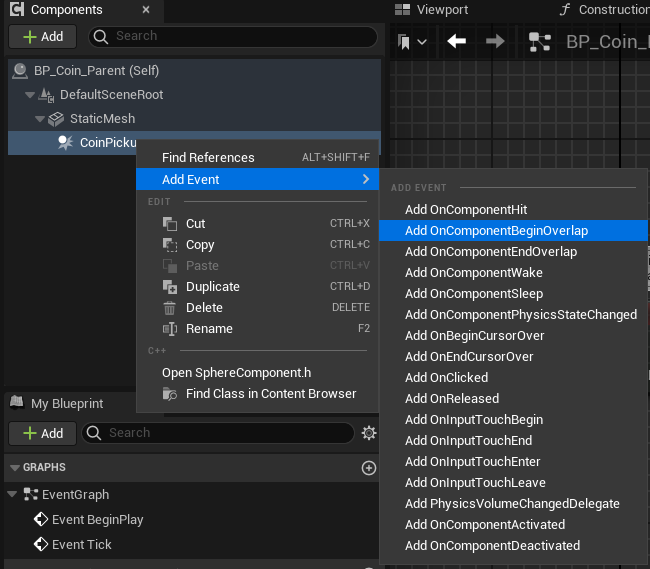
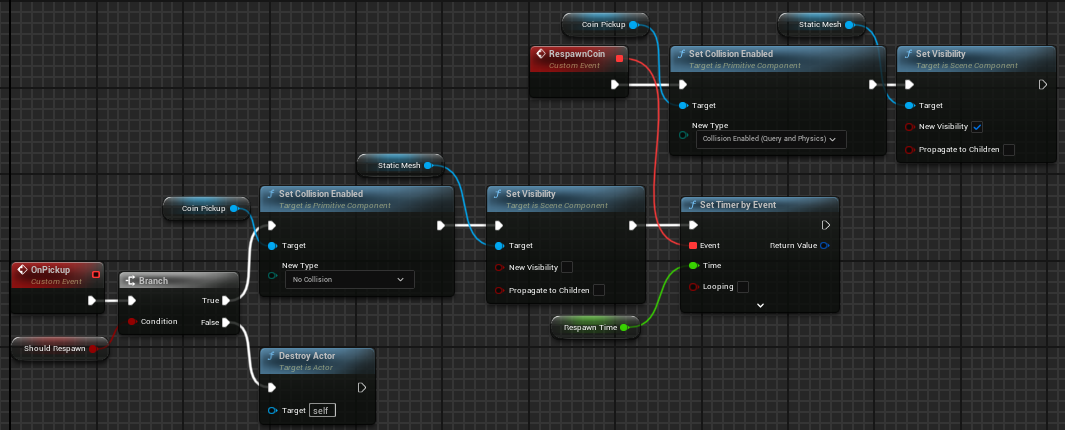
* 
  1. Next we will add our coin amount variable to this incoming amount and since we will be handling addition and subtraction here we need to add a check to make sure our coin value does not go negative since the player shouldn’t be allowed to go into debt.
     1. We will do this by adding a check to see if the result is less than 0, we will use a “Select Int on Bool” node here to handle this, this will take an incoming bool and if it is true will select option “A” and false will select option “B”
  2. The result of the select node will then be stored in our coin amount variable, once we make our HUD we will tie this into that, but for now this event is finished.
* 

Step 3: Coins and pickups

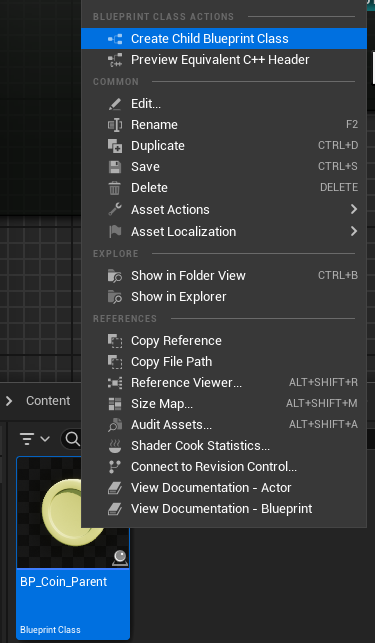
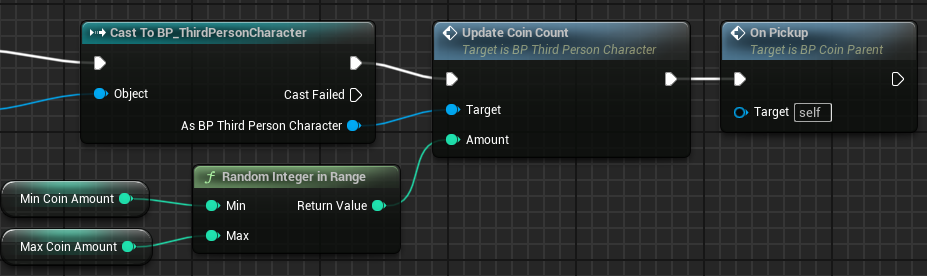
1. We will start by making a variable on the player to hold our coin count called “CoinAmount”. We will also make a custom event to update this number when we pick up a coin.
   1. Add an input to this custom event by clicking on the event node and in the details panel click the “+” icon on the input section.

* 
  1. In this event add the incoming amount to the current amount of coins and set the current amount to the resulting value.
* 

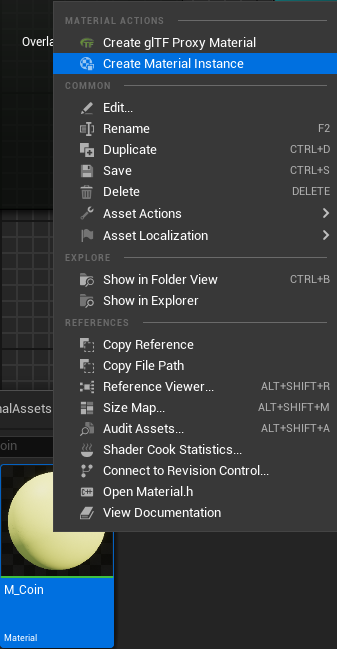
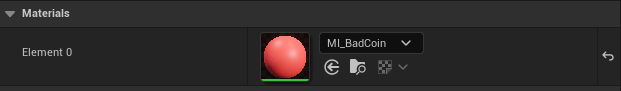
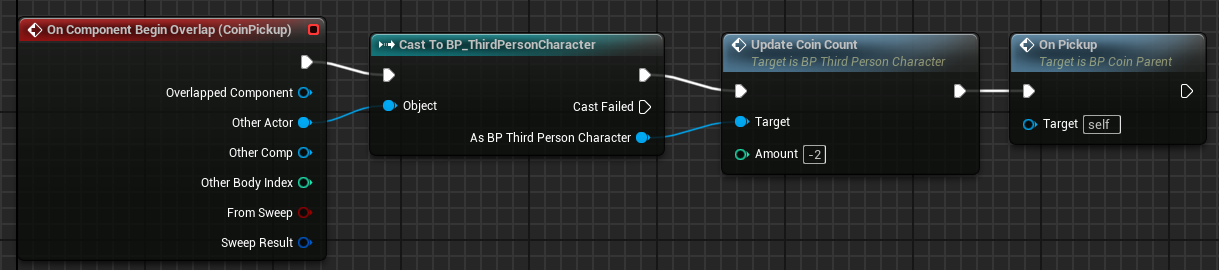
1. Next we will make a new actor blueprint.
   1. Right-click in the content browser, find and select “Blueprint Class”.

* 
  1. The parent class for the blueprint will be the “Actor” class and name it “BP\_Coin\_Parent”.
* 
  1. In the components tab add a “Static Mesh” component, this will hold our coin mesh. Next add a sphere collision component and drag it onto the static mesh which should make the sphere collider a child of the static mesh which in turn will allow up to apply any transformations we apply to the mesh to the collider as well. At this time you can also add a Rotator Component which will rotate the coin. Set the rotation along the Z-axis to 120.
* 
  1. In the details section of the static mesh, find “Static Mesh” it should say “None”. Click the drop down and search for the coin mesh and add it.
* 
* 
  1. Next, in the event graph right-click the sphere collider component. Find “Add Event” then find “Add OnComponentBeginOverlap” this will add the event to the event graph and will be called whenever a world object overlaps with our sphere collider.
* 
  1. With the begin overlap node drag off of the “Other Actor” pin and add the “Cast To BP\_ThirdPersonCharacter” node this will allow us to only execute our logic when the player overlaps rather than all overlaps and cause a crash.
  2. Create a new node off of the cast node and call the “UpdateCoinCount” event on the player character and set the amount to 1.
  3. Now we will make 2 new variables, both will be instance editable, one will be of type bool named “bShouldRespawn” and the other will be of type float named “RespawnTime” and set the default value to 15.
  4. We will make a new custom event named “OnPickup” the first node will be a branch node, and we will check the should respawn variable, if it is false we will call “DestroyActor”. If it is true we will set collision to “NoCollision” on the coin pickup, and set visibility to false on the mesh component and set a timer by event and use our respawn time variable to control how long until it respawns. We will create a new event to call once the timer completes, we will set the collision to enabled on the coin pickup and set the visibility to true on the mesh component. Now call “OnPickup” after the coin count is updated.
* 

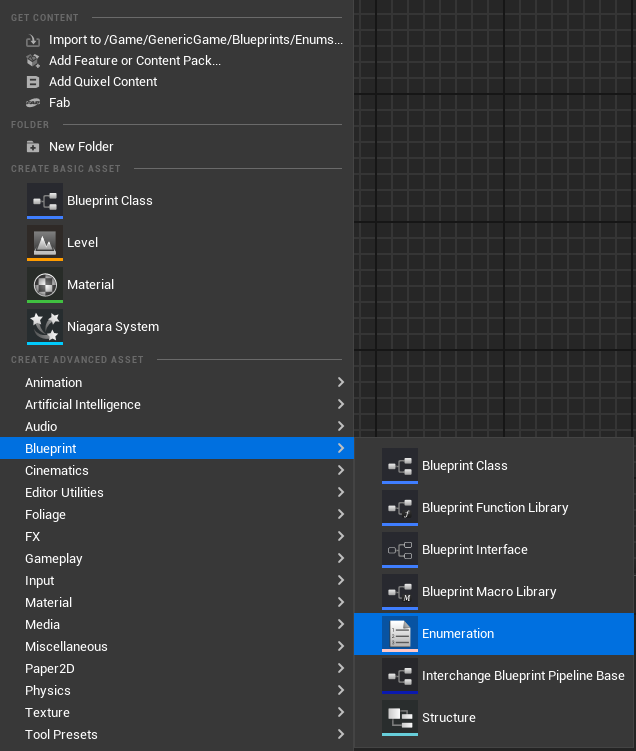
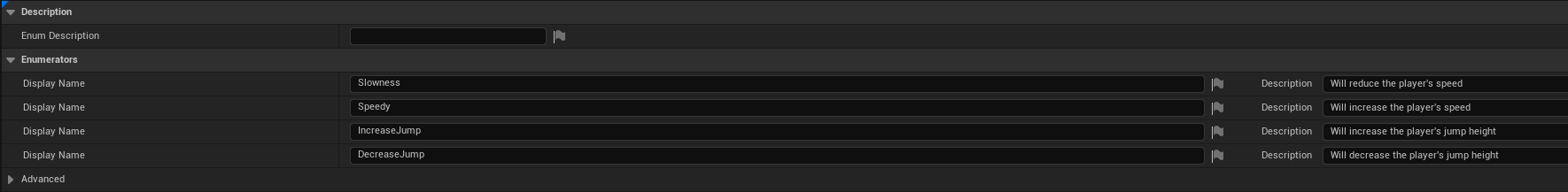
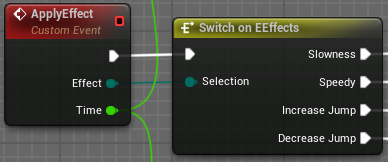
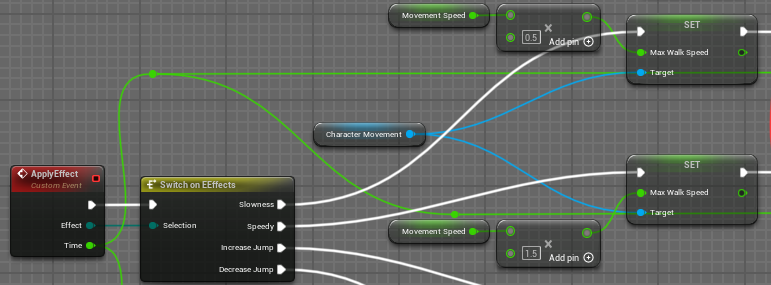
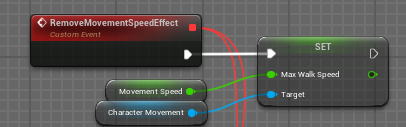
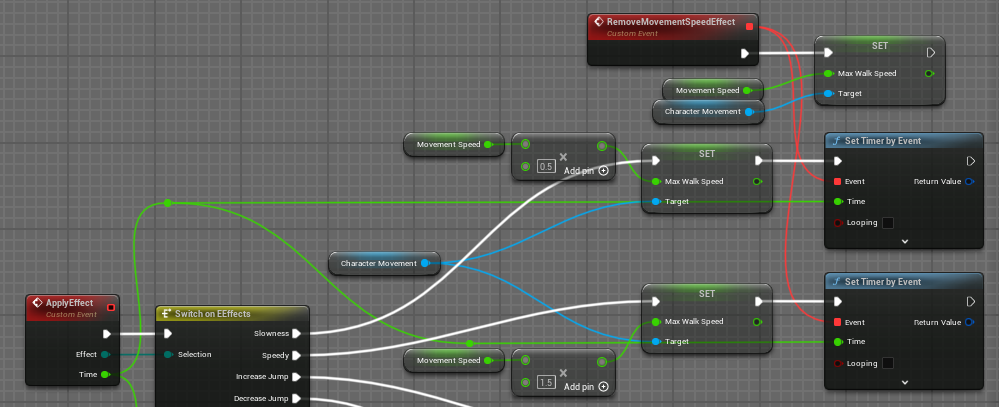
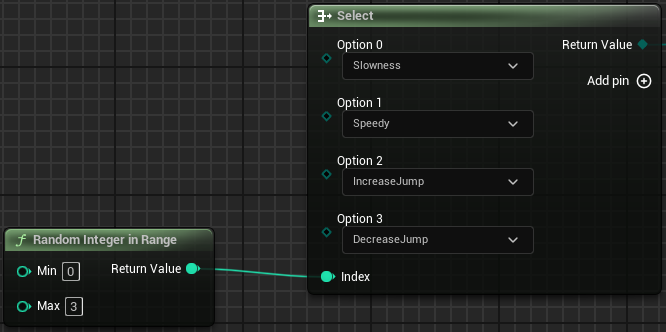
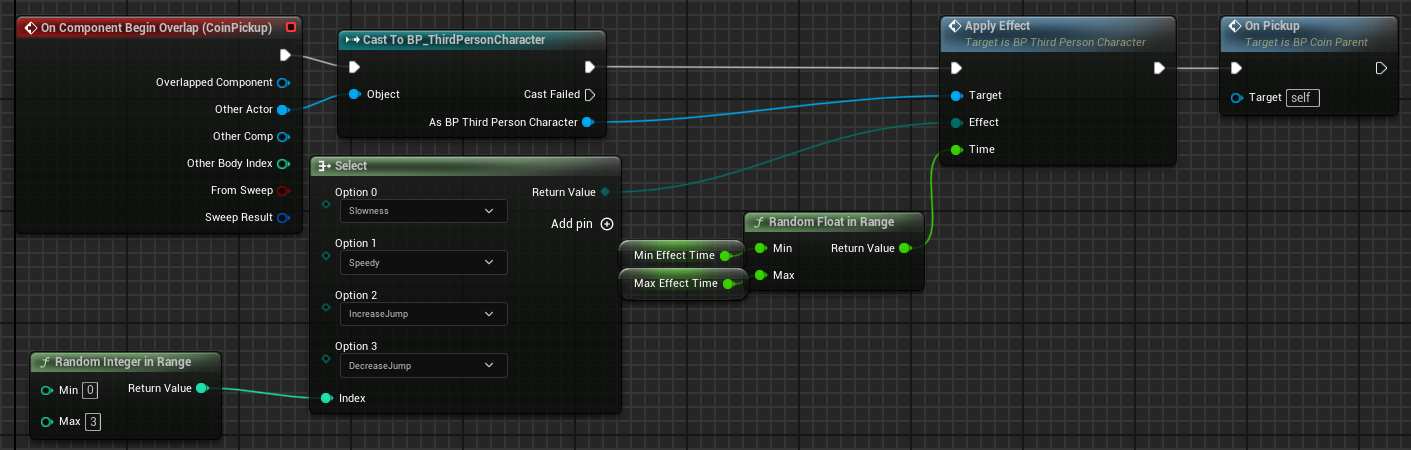
1. We will create a child blueprint of the Coin Parent to make a coin that will give a random amount of coins between 2-5 to add some randomness to the game.
   1. Start by right-clicking the “BP\_Coin\_Parent” blueprint and select “Create child blueprint class”. We will be naming this “BP\_RandomIncrease\_Coin”.

* 
  1. Once you have created the new blueprint class, open the event graph and we will copy and paste the same nodes from the parent blueprint to get the same basic functionality.
  2. Next we will add two integer variables, one named “MinCoinAmount” and the other “MaxCoinAmount” the values 2 and 5 are recommended for these.
  3. Now right-click the event graph and find the “Random Integer in Range” node, plug our two variables into their respective min and max pins on the node and then connect the return value pin into our amount pin on the event callback.
  4. Finally call our “OnPickup” event from the parent class.
* 

1. Now let’s make a coin that will remove coins from the player to create an obstacle for them to avoid. We will create a material instance for this and change it to a red color to indicate that this is a bad coin.
   1. To create a material instance right-click on the “M\_Coin” material and select the “Create Material Instance” near the top.

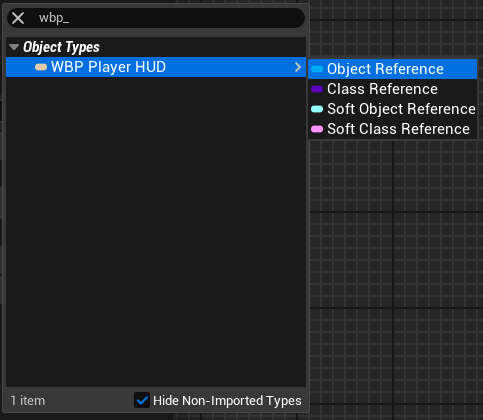
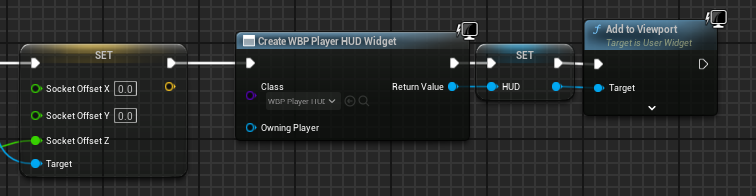
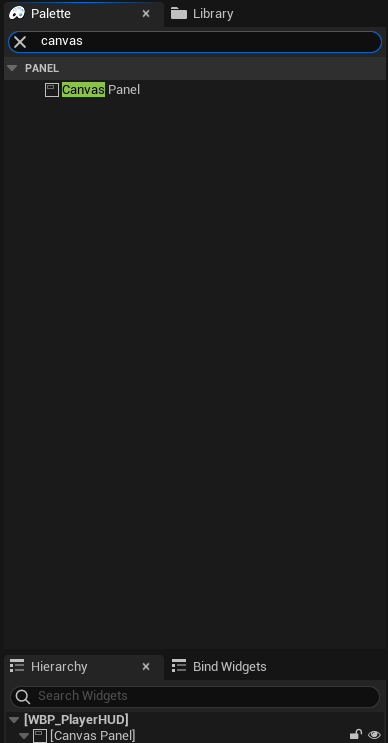
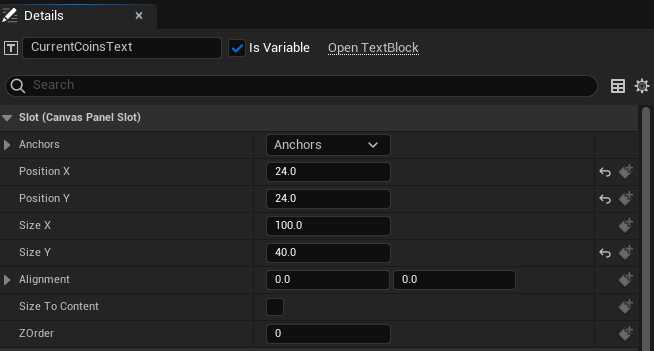
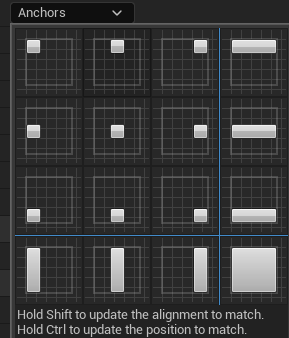
* 
  1. We will be naming this “MI\_BadCoin”. Open the new material instance, and in the details panel under “Global Vector Parameter Values” find “Param” and make sure the box is checked to be able to modify the values. Set Red to 0.75, Green and Blue to 0.05 and keep Alpha at 1.0. You can now save this asset and close the tab.
* 
  1. Next we will create a new child blueprint from the coin parent the same way as the previous step, name this “BP\_Decrease\_Coin”
  2. Once you have opened this new blueprint select the static mesh in the components tab, then in the details section underneath “Static Mesh” you will see “Materials” and we will change element 0 to the material instance we just created. Click on the drop-down and search for the material name.
* 
  1. Now our coin should appear red which will indicate to us that we want to avoid it in game. Now create the On Component Overlap Event for the “CoinPickup” collider. Now copy and paste the same logic from the parent to this child because it will be exactly the same but instead of our input amount being “1” we will make an integer variable that is instance editable so we can alter the value on each instance of the object.
* 

1. Next we are going to create a coin that when picking it up will apply a random effect to the player for a certain amount of time.
   1. First step we will make an Enumeration type to help us handle this easier. Right-click in the content browser, find “Blueprint” then select “Enumeration”. All enums are denoted with an E prefix, name the new asset “EEffects”

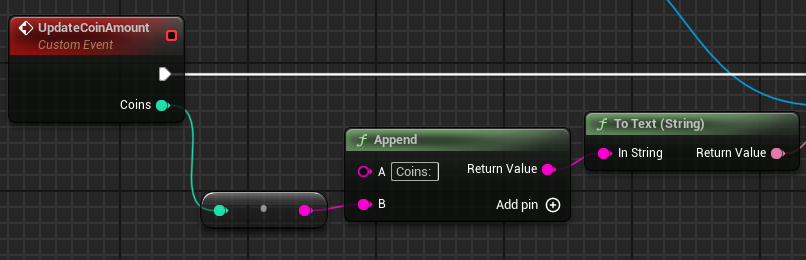
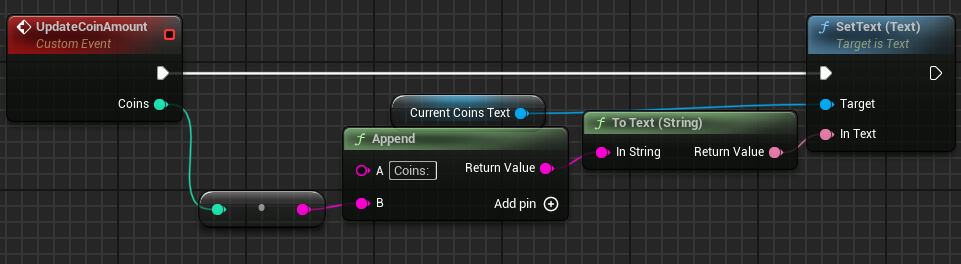
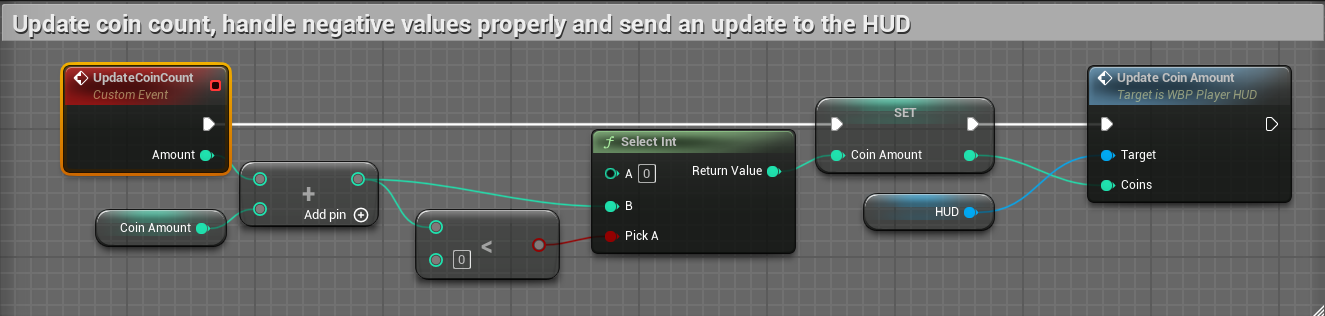
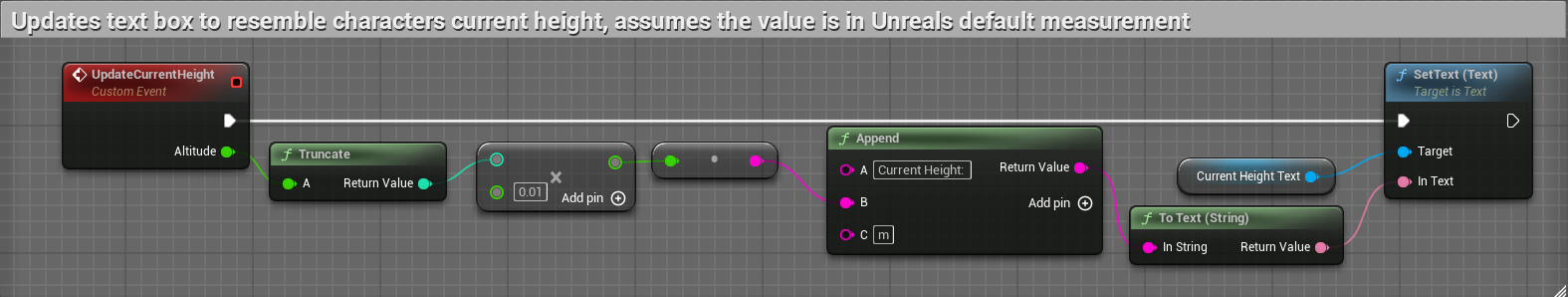
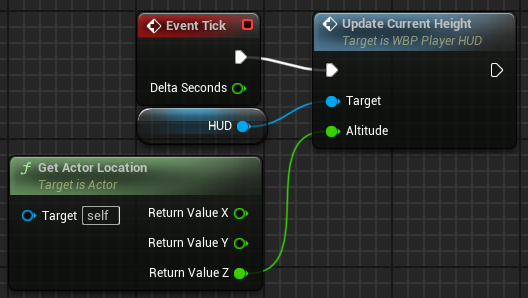
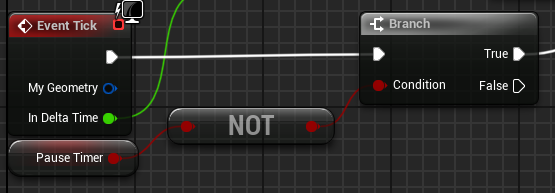
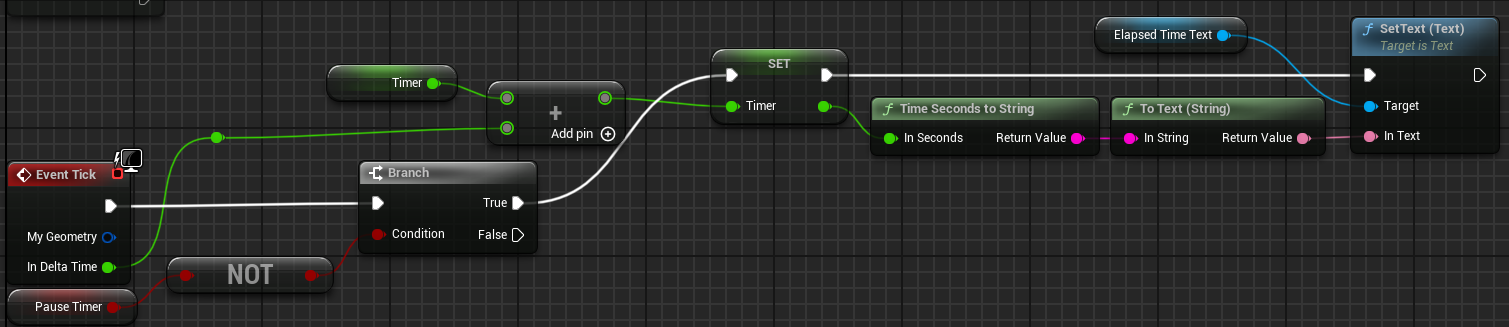
* 
  1. Open the new enum asset. For now we will create 4 enumerators, to do so click the “Add Enumerator” button near the save icon.
* 
  1. We will name these “Slowness”, “Speedy”, “IncreaseJump”, and “DecreaseJump”.
* 
  1. Next go into the player blueprint and we will create a custom event called “ApplyEffect” and add two inputs, one input being the enumeration we created just search for the name and the other being a float.
* 
  1. The first node to add to this event will be a switch node, we will then switch on the enum coming into the event and make a branch for each effect.
* 
  1. We will work on the “Slowness” and “Speedy” effects first and the increase and decrease jump ones will be almost the same, just modifying different values.
  2. For slowness we will take our movement speed variable and multiply it by 0.5 and then take our movement component and set the Max Walk Speed to this new value. The Speedy will be the same setup but instead of multiplying by 0.5 we will multiply by 1.5.
* 
  1. Next we will make a new event, we will call it “RemoveMovementSpeedEffect” and it will simply just set the Max Walk Speed to the value of the MovementSpeed variable.
* 
  1. Now to make these effects only be applied for a certain amount of time we will use a node called “Set Timer by Event” which will automatically call the supplied event once the timer hits 0. Connect the time parameter from ApplyEffect into the time pin on the timer node.
* 
  1. The jump height effects are almost the exact same setup, the only differences are we use the JumpHeight variable instead of the MovementSpeed variable, and we will set the Jump Z Velocity instead of Max Walk Speed. And the supplied event into the timer will be a separate “RemoveJumpEffect”.
  2. Now we will make a new child coin. Name it “BP\_Modifier\_Coin”. First thing to do is to add the begin component overlap event for the CoinPickup component. Also create 2 float variables named “MinEffectTime” and the other “MaxEffectTime”, set the min to 1.0 and the max to 2.5.
  3. Now we will use a select node to select which modifier to send to the player. We will select on an int value which will be created at random between 0 and 3 since we have 4 options.
* 
  1. Off of the begin overlap event cast to our player character and off of the character call the “ApplyEffect” event and connect the output of the select node into the effect pin. We will fill the time pin by using the “Random Float in Range” node which we will provide our min and max time variables to this. Then call our “OnPickup” event..
* 

Step 4: Player HUD

1. Right-click the content browser, select “User Interface” then select “Widget Blueprint” and in the popup window select “User Widget” and name the new blueprint “WBP\_PlayerHUD”.
   1. The first step after creating the HUD will be to make sure we add it to the player. Open the player blueprint and add a variable named “HUD” and when selecting the data type search for “WBO\_PlayerHUD” and select the object reference option.

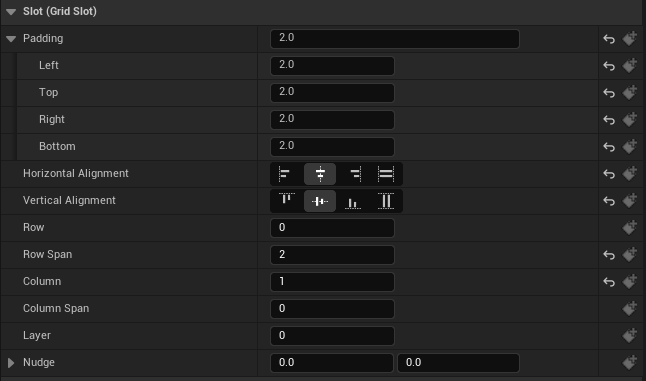
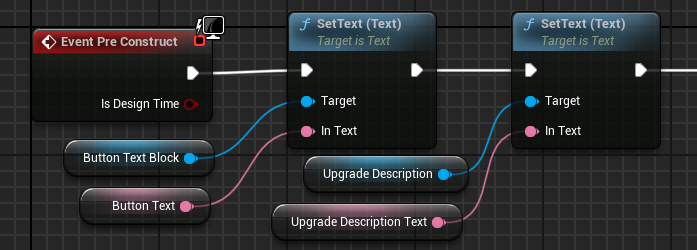
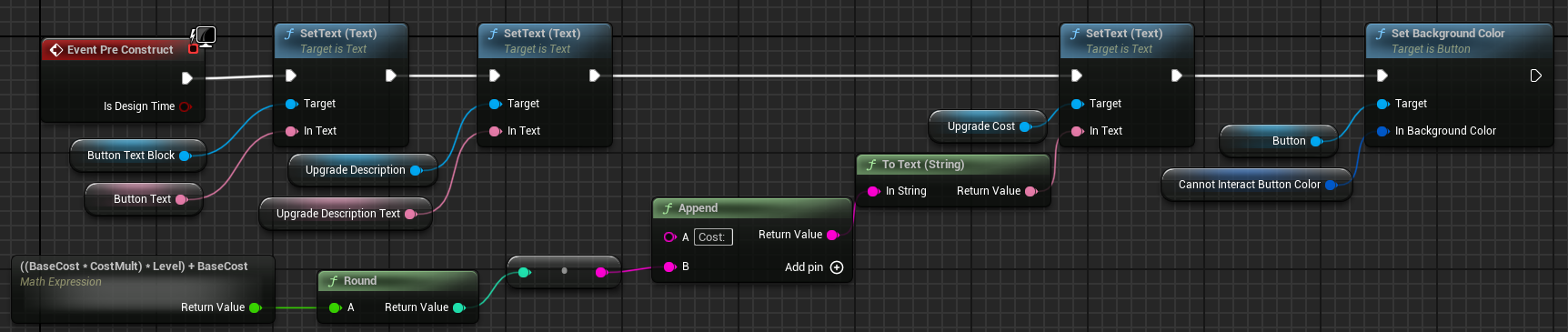
* 
  1. At the end of our execution line on “Event BeginPlay” we will add some more nodes. Add the “Create Widget” node and in the class reference find our new blueprint. Set the HUD variable to the return value of the create widget node. Then using the HUD variable find the “Add to Viewport” node.
* 
  1. We are going to keep the HUD very simple and just use text to display vital information to the player.
     1. In the pallet tab on the left side of the editor search for “Canvas Panel” this is what will allow us to place our text boxes on the screen. You can just drag and drop this onto the designer board and will populate there and in the hierarchy panel.
* 
  + 1. Now search for “Text” this is found under the common tab. We will be using 3 of these. The first one in the details panel we will name it “CurrentCoinsText” and make sure the “Is Variable” checkbox is checked. The X and Y positions will both be 24 and the default text will be “Coins: 0”.
* 
  + 1. The next text item we will add will be called “CurrentHeightText” and make sure the variable checkbox is checked. The X position will be 24 and the Y position will be 72 and the default text will be “Current Height: 0.00m”.
    2. The next one will be named “ElapsedTimeText”. Make sure the checkbox is checked and this one we will change the anchors since we want to keep it in the center no matter the screen size. Open the anchors tab in the details panel and Shift-click the top center one to have the origin at the top center of the canvas panel.
* 

1. Next we will set up the events to update all of these variables when they need to be. We will start with the coins.
   1. Create a new event named “UpdateCoinAmount” and make sure it has an int input named “Coins” drag off that pin and looks for the “To String” node, we will then append the result of that to the string “Coins: “, then we will covert that string into text since we need to send a text variable to the last piece of our puzzle.

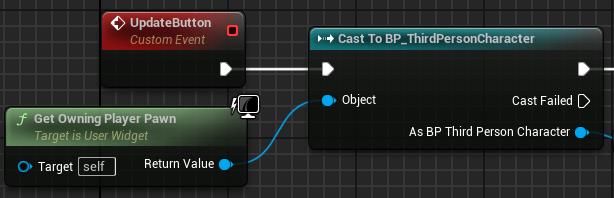
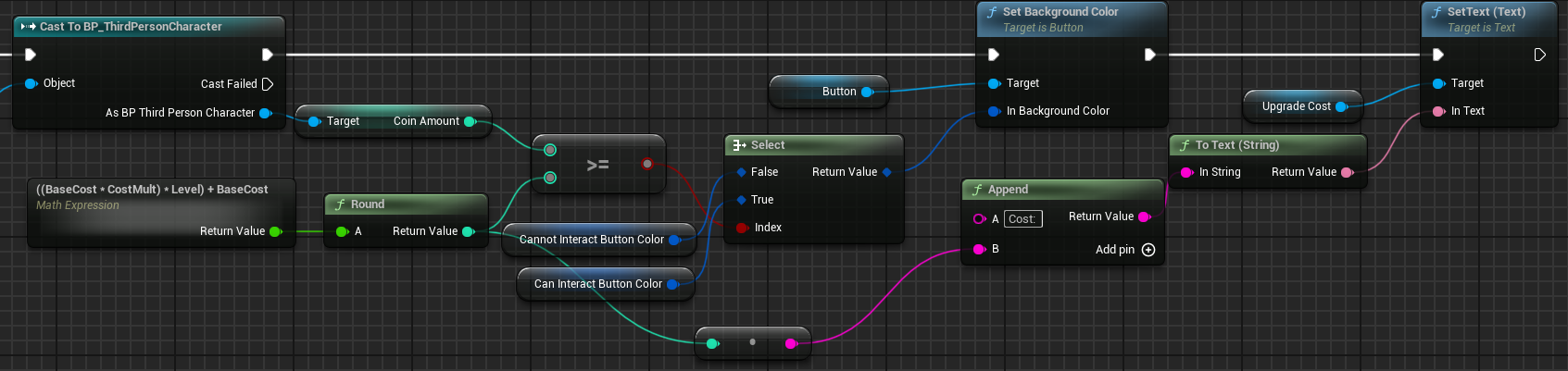
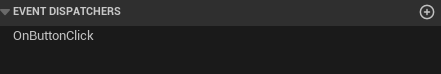
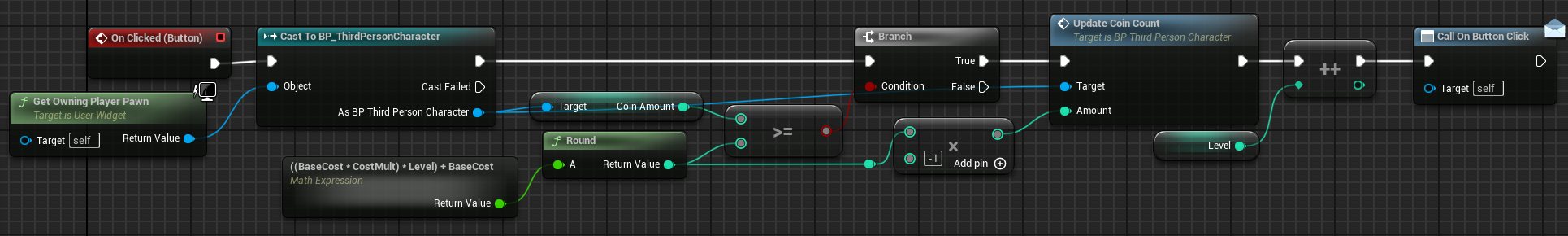
* 
  1. We will then drag out a reference to our current coins text variable and, dragging off of its pin, find the “SetText” node and put our text value into the text pin.
* 
  1. Now in the player blueprint find the UpdateCoinCount event and at the end we will add a call from the HUD to call the UpdateCountAmount event we just created to update the HUD whenever we change the coin amount.
* 
  1. Now we will add the event to handle the height change. Name it “UpdateCurrentHeight” and add a float input named “Altitude” this float value coming in is in cm and we’re displaying meters and only to 2 decimal places, so we will first truncate the float which will return an int, we then multiply this int by 0.01 to get our meters and only 2 decimal places.
  2. We will then convert that float into a string, then append a string to the front and back of it like so, “Current Height: “ + float + “m”, then convert that string to a text object and send that into the CurrentHeightText set text node.
* 
  1. Now in the player blueprint we will add the event tick event to the event graph, and we will call “UpdateCurrentHeight” off of the HUD variable and use the “GetActorLocation” node and split the struct pin and connect the Z value to the altitude node.
* 
  1. Now to update the elapsed time text variable, we will be using the event tick for this to steadily update our timer every frame, we will also add 2 variables one float named “Time” and a boolean named “bPauseTimer”.
     1. Hold “b” and click on the event graph to create a branch node and connect the event tick and branch exec pins. Drag out our pause timer variable, drag off of it and look for the “NOT boolean” node and check that in the branch.
* 
  + 1. Now we will take our timer variable and add delta time to it and off of the true branch we will set timer equal to that new value. Then we will find the “Time Seconds to String” node, convert the string into text and use that to update our elapsed time text variable.
* 

Step 5: The shop

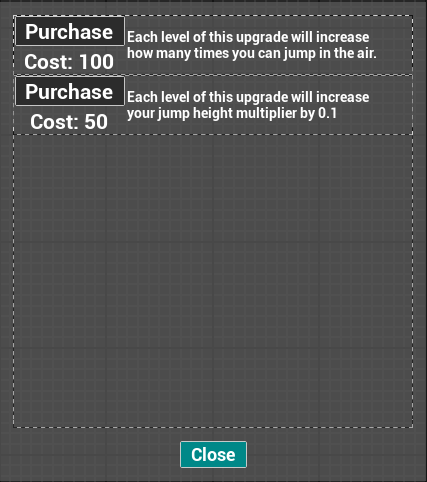
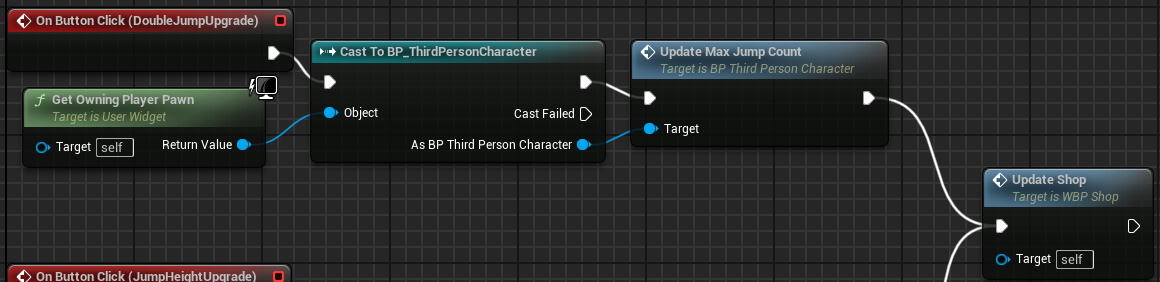
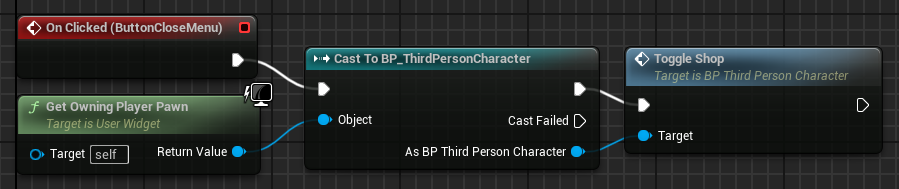
1. We’re going to start by making 2 new widget blueprints. One named “WBP\_Shop” and the other “WBP\_UpgradeTemplate” this upgrade template is going to give us a very modular shop system that will make it easier to add more upgrades in the future.
2. Starting on the template widget, open it in a new tab. In the designer portion of the blueprint search for a “Grid Panel” this will allow us to group items together and add them all together as a cohesive group to our parent blueprint. We are going to add 3 items to the grid panel, add a button and 2 textboxes, and add an additional textbox as a child to the button. Make sure all of these child items are variables. Rename the button textbox to “ButtonTextBlock” then the other 2 will be named “UpgradeDescription” and “UpgradeCost”.
   1. Formatting the grid. All elements will have a padding of 2.0 on all sides. Select the button, all other values will remain at their default value of 0. Select the UpgradeDescription text block, this we will set to be in row 0 and span 2 rows, and will have it be in column 1. On this we will set Auto Text Wrapping to true, and set Wrap Text At to 400. Select the UpgradeCost text block and we will change the row value to 1.

* 
  1. Next we will dynamically set all of our text values. Switch to the graph portion of the blueprint and we’re going to add some variables. Add 2 text variables name them “ButtonText” and “UpgradeDescriptionText” and make sure to click the eye icon which will allow every instance we make of this blueprint to have unique editable values for these variables, this is called **Instance Editable**. Set the default button text to “Purchase” and the default text on the description to be “Description here”.
* 
  1. Now off of the Pre Construct event we will set the text on the button to our button text variable and set the text for our description box to our description text.
* 
  1. Now we will update our upgrade cost text. We will need 3 new variables to do so and to keep it modular, 2 integers named “BaseCost” and “Level” make sure the base cost variable is instance editable, and we will need a float value that is instance editable named “CostMult”. BaseCost and Level will have a default value of 0, and CostMult will be 1.0. These will all be part of a math expression node.
     1. To make a math expression node right-click on the event graph and search for “Math Expression”. Then we type out the equation we want to represent, and if we use our variable names it will automatically fill in the input values. Our equation is ((BaseCost\*CostMult)\*Level)+BaseCost.
* 
  1. We will take the output and round it using the “Round” node, then we will cast it to a string and append “Cost: “ to the front of the string and set our upgrade cost text value.
  2. Now we will add 2 instance editable variables that are of the type “Linear Color” this will be used to visually indicate to the player whether or not they can interact with the button or not. The names will be “CanInteractButtonColor” and “CannotInteractButtonColor” set the default values for can interact as R: 0.0 G: 0.5 B: 0.5 A: 1.0, and R: 0.05 G: 0.05 B: 0.05 A: 1.0 for the cannot interact.
  3. Now we will take the button reference and call “Set Background Color” and set it to the cannot interact color on the Pre Construct event.
* 

1. Next we will handle the updating of the button, this will be done whenever the shop is opened or something is purchased in the shop.
   1. Create a custom event called “UpdateButton” then we will look for a node called “Get Owning Player Pawn” and type cast the return value to our player character.

* 
  1. We will reuse that math expression from earlier, copy and paste it by our new event, and off of our cast node we will get the players coin count and see if it is greater than or equal to the rounded result of our math expression. We will then use the produced boolean and select one of our colors based on the value and set the button background color accordingly. We will also append the rounded value to the end of the string “Cost: “. This will be all for our UpdateButton event.
* 
  1. We will now handle when the player clicks on the button. We will create an event dispatcher to handle this from the shop blueprint. Click the “+” button on the event dispatcher section and name it “OnButtonClick”.
* 
  1. The first little bit of the button clicking looks very similar to the UpdateButton event, we will get the owning player pawn and cast it to our player blueprint, get the player’s coin count and see if it is greater than or equal to the rounded output of our math expression. We are going to create a branch node and put our boolean value on that input pin, if it is true we will update the player’s coin count by the rounded output of the math expression multiplied by -1 that way it will take away from the value and not add to it. Then take our level variable and increment it by 1, then drag the OnButtonClick event dispatcher out and select “Call”.
* 
  1. TODO: add sound

1. Now that we have the button logic done we will make the actual shop. Open the WBP\_Shop blueprint, in the designer section add a canvas panel.
   1. Add an image, this will act as a semi-transparent background to the menu. Shift-click the top center anchor, set the Y position to 120, and the size to 640x720. Set the color to R: 0.1 G: 0.1 B: 0.1 A: 0.75.
   2. Add a scroll box widget and name it “UpgradeBox”, set the same anchor as the background image. The Y position is 140, size is 600x620.
   3. In the palette we can search for the widget we made, add 2 of them as children to the scrollbox
      1. Name these “DoubleJumpUpgrade” and “JumpHeightUpgrade”. Set the BaseCost and CostMult of these to 100, 1,5 and 50, 0.75 respectively.
      2. Add a button, name it “ButtonCloseMenu” and add a text block to it and set the text to “Close”. This button will use the same top-center anchor as the other elements. The position will be X: 0 Y: 780, size X: 100 Y: 40. Set the background color to R: 0 G: 0.5 B: 0.5 A: 1.

* 
  1. In the event graph the first event we will make will be “UpdateShop” which will call the update button event for each of the buttons. Get a reference to the UpgradeBox, call “Get All Children” and with the array it returns use a “For Each” loop to iterate over all children. Cast the array element to our WBP\_UpgradeTemplate and then call update button.
  2. We will handle all of the player interactions when pressing buttons on this menu. Select one of the upgrade buttons and in the details panel you will see an “Events” section click the plus button next to the OnButtonClick one to add it to the graph. This is a very simple event because of how we set it up on the template. It will only call this event if the player can afford it so we do not need to check that here. Call get owning player pawn, cast the return value to our player blueprint and call “UpdateMaxJumpCount” then call the update shop event.
* 
  1. The JumpHeight button is the same, only difference is we call the UpdateJumpHeightMult event.
  2. Add the “OnClicked” event for the close button. Here we will get the player pawn and cast it to our player character, and if it succeeds we will call our “ToggleShop” event.
* 

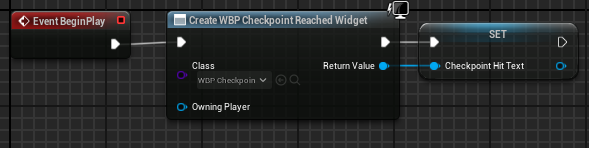
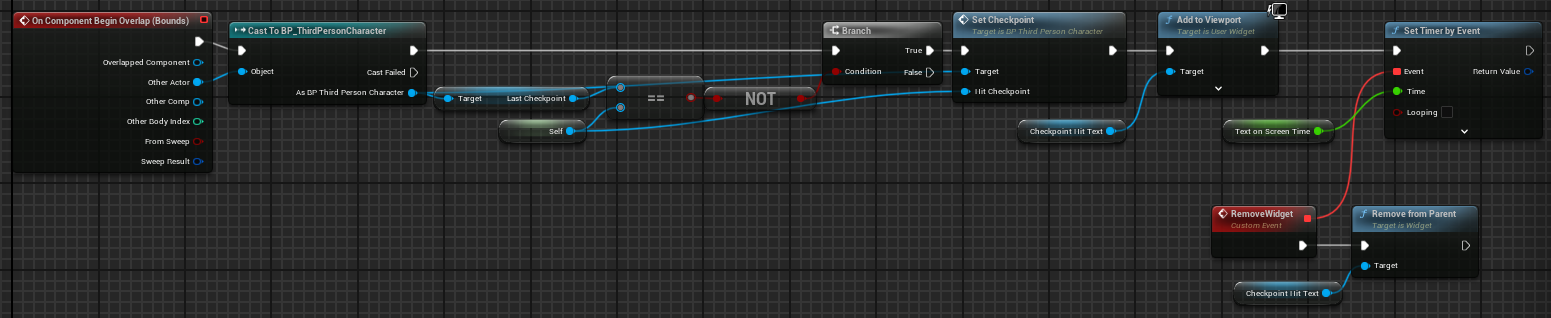
1. Going back into the player we are going to add an event to handle the opening and closing of the shop. Create an event called “ToggleShop”
   1. Add a new boolean called “bShopOpen” which will then be put into a branch node, on the false side we will handle the opening of the shop.
      1. Call “AddToViewport” on the shop menu variable, then call UpdateShop afterwards so all of the items are updated so the player will see the most updated values. We will then call “GetPlayerController” and call “SetInputModeGameAndUI” on the controller and set show mouse cursor to true then set our shop open variable to the NOT value of what it currently is.
      2. The true side will handle closing the menu and it is very similar, start by calling “RemoveFromParent” on the shop menu variable, then using the player controller call “SetInputModeGameOnly” and set show mouse cursor to false and set the shop open value to the opposite of what it currently is.

* 

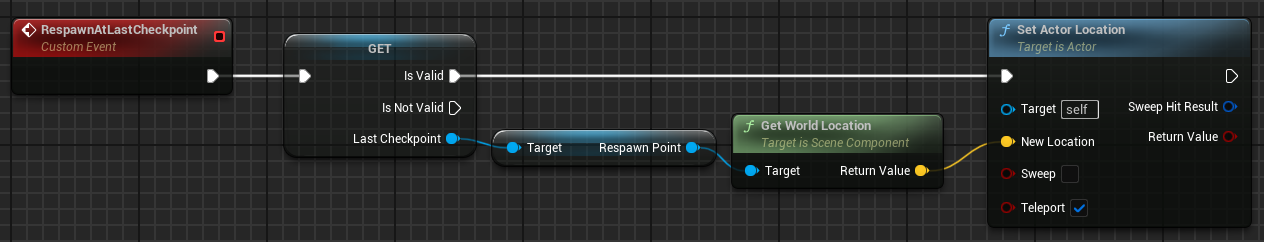
1. TODO: add actual input/button handling to open the shop

Step 6: Checkpoints

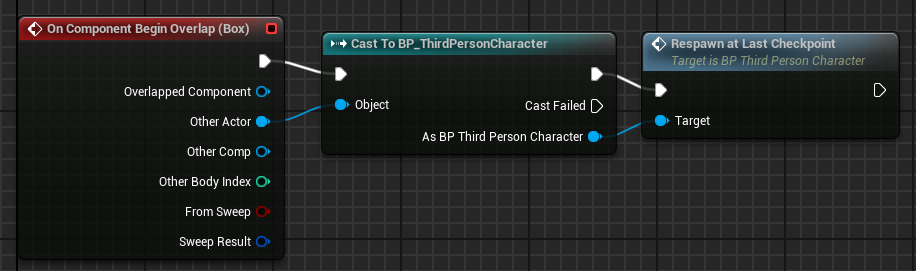
1. Make a new actor blueprint named “BP\_Checkpoint”. Then open the player blueprint and add a variable of type “BP\_Checkpoint” named “LastCheckpoint” and add a custom event named “SetCheckpoint” with an input of our checkpoint type, we will come back to the player and add logic there when we finish the logic on the checkpoint blueprint.
2. Now we will make a simple widget to display text to the player to indicate that they have reached a checkpoint.
   1. Create a user widget named “WBP\_CheckpointReached”. Add a canvas panel to the widget. Add a text box, shift click the bottom middle anchor point, set the X position to 0 and the Y position to -160. Set the color to R: 1 G: 0.1 B: 0.1 A: 1. Set the text to “You have reached a checkpoint!” and now we are finished with this blueprint.
3. Back into the checkpoint blueprint to add some components.
   1. The first one will be a box collider to mark the bounds of the checkpoint so we can visualize the size, set the default scale to X: 3 Y: 3 Z: 1, rename this to “Bounds”. Next we will add a capsule collider which will represent where the player will respawn, name this component “RespawnPoint”.
   2. Now add 2 variables, the first one will be named “CheckpointHitText” and will be type “WBP\_CheckpointReached” and the other will be named “TextOnScreenTime” and will be type float.
4. Time for events.
   1. Off of the BeginPlay event call create widget and make sure to set the class to our new checkpoint reached widget and set our variable equal to the returned value.

* 
  1. Now on the Bounds component right-click and find “Add Event” and add the “OnComponentBeginOverlap” event.
  2. Drag off the “Other Actor” pin and cast it to our player character. As the player get the last checkpoint variable and see if it is NOT equal to “reference of self” if it is not equal we will call our “SetCheckpoint” event and pass in a reference of this object. We will add our checkpoint hit text to the viewport. Now add an additional custom event named “RemoveWidget” which simply calls “RemoveFromParent” on the text widget. This will be called using the “Set timer by event” node and pass in this new event as the parameter and our on screen time variable for the time.
* 

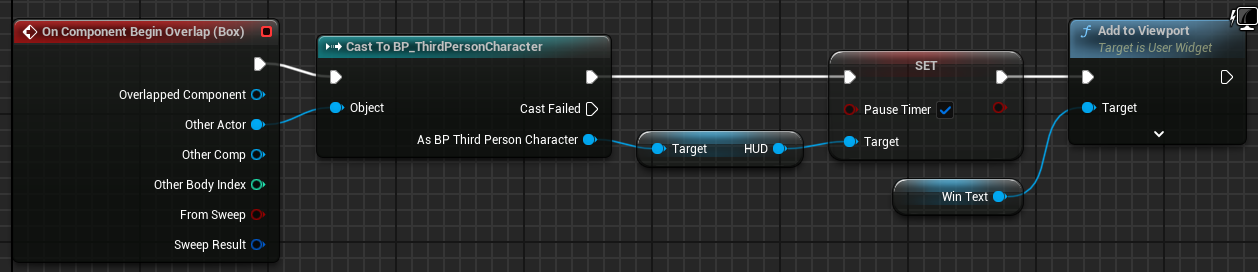
1. Let’s add some logic to the player now.
   1. For the set checkpoint event all we are doing is setting our LastCheckpoint variable to the incoming HitCheckpoint value.

* 
  1. Now we will add another custom event named “RespawnAtLastCheckpoint”. Drag a reference of our last checkpoint variable out, right-click on the node and select “Convert to validated get” since we do not want the player trying to respawn to nowhere when they haven’t hit a checkpoint. Off of the last checkpoint pin, get the respawn point, get its world location and use the “SetActorLocation” node to teleport the player to that location.
* 

1. Now we will setup a zone where if the player touches it they will be respawned at their latest checkpoint. Create a new actor blueprint named “BP\_AutoRespawn”.
   1. First add a box collider, and implement the begin overlap event.
   2. Off of the other actor pin cast to our player character, then call “RespawnAtLastCheckpoint” on the character.

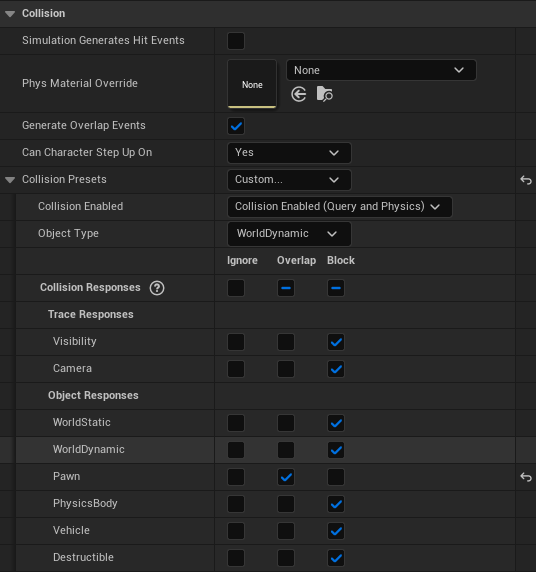
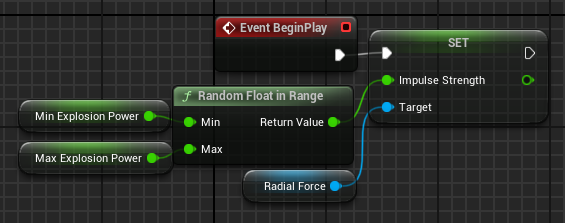
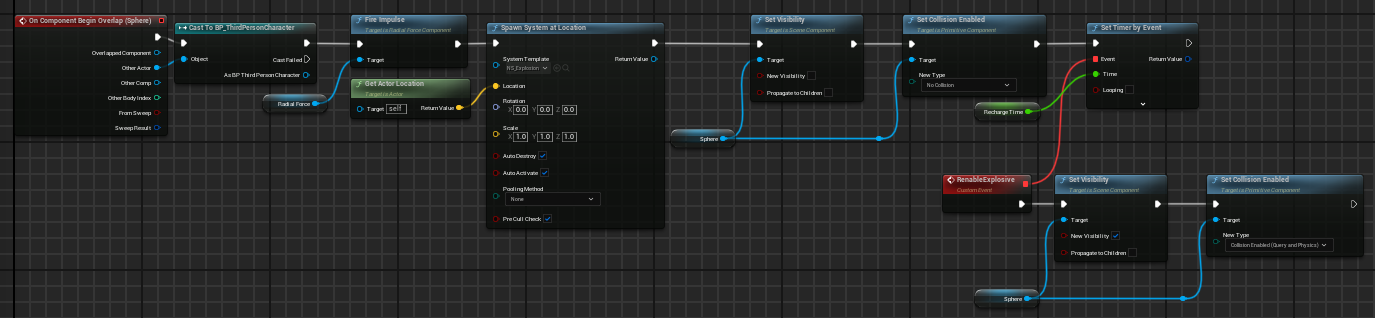
* 

1. Now we will create the win condition which will just be a trigger zone that will display a message for the player and stop the timer.
   1. First create a new widget blueprint and name it “WBP\_WinText”.
      1. Add a canvas panel, next add a text block and set the text to “YOU WIN!”. Set the color to green. Set the anchor to the middle of the screen and set the Y position to -250
   2. Now create a new actor blueprint and name it “BP\_WinZone”.
   3. Create a new variable of type “WBP\_WinText” and name it “WinText”. Add a box collider.
   4. Now implement the begin overlap event on the collider. Cast the other actor pin to our character.
   5. Then get the HUD variable from the player, and set the pause timer on the HUD to true. Now call “AddToViewport” on the win text variable.

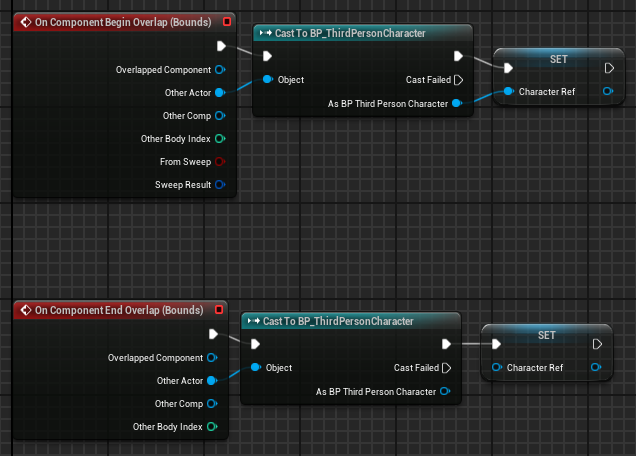
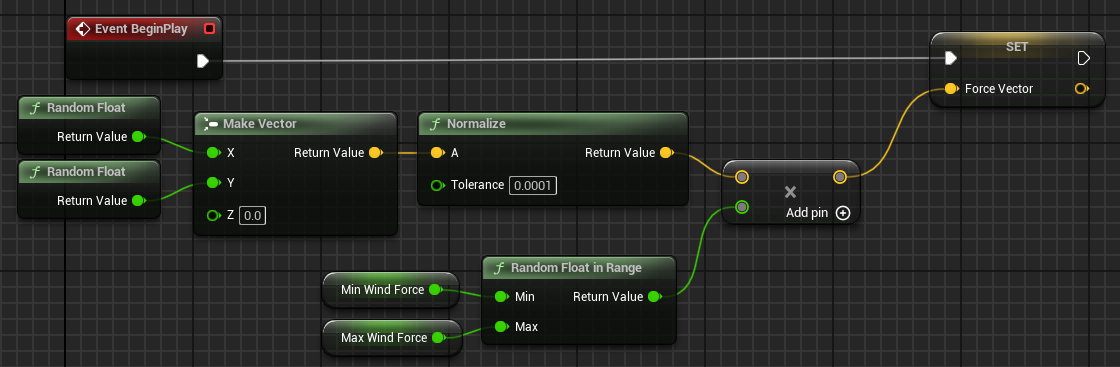
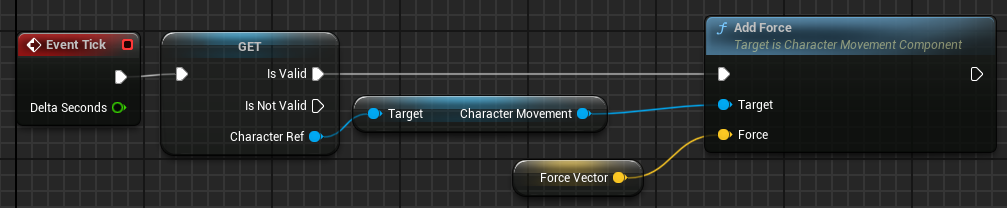
* 

Step 7: Hazards

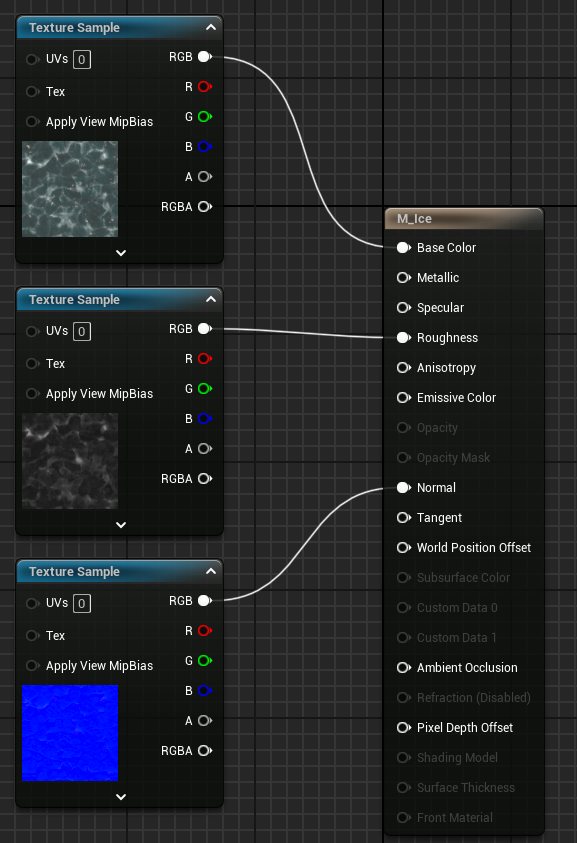
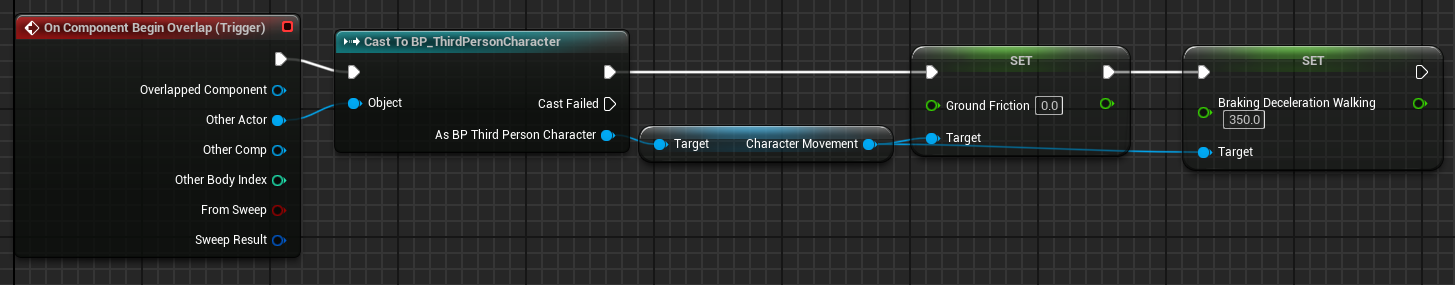
1. The first hazard we are going to make is an explosion hazard where if the player steps on it they get launched.
   1. First create a new actor blueprint named “BP\_ExplosionHazard”. The first component we are going to add is a sphere mesh, set the scale to X: 1 Y: 1 Z: 0.2. To indicate to the player that they should avoid this we are going to reuse our “MI\_BadCoin” material instance we created earlier. We are going to customize how the collision is handled for this mesh, find collision in the details panel and change the “Collision Preset” from “BlockAllDynamic” to “Custom” and we are changing the way Pawns interact and change it from “Blocking” to “Overlap” this will make it so the player will not collide with the sphere and will instead walk through which allow us to use the begin overlap event.

* 
  1. The second component to add is the “Radial Force” component, leave all the settings at the default.
  2. We will create 3 float variables the first one will be “MinExplosionPower” with a value of 150000, next variable is “MaxExplosionPower” with a value of 175000, and the last variable is “RechargeTime” with a default value of 1.
  3. On begin play, take the radial force component and set the “ImpulseStrength” value to a random float between the min and max explosion power.
* 
  1. Right-click the sphere component and add the component begin overlap event. First we will check if the other actor is the player character by casting to it. Next call “FireImpulse” from the radial force component. Now call “SpawnSystemAtLocation” and the system template will be the “NS\_Explosion” and call get actor location and set the location pin to the returned value. After the explosion we will set the visibility to false and set collision to “NoCollision” on the sphere component, and using a timer to call an event we will set visibility to true and re-enable collision.
* 

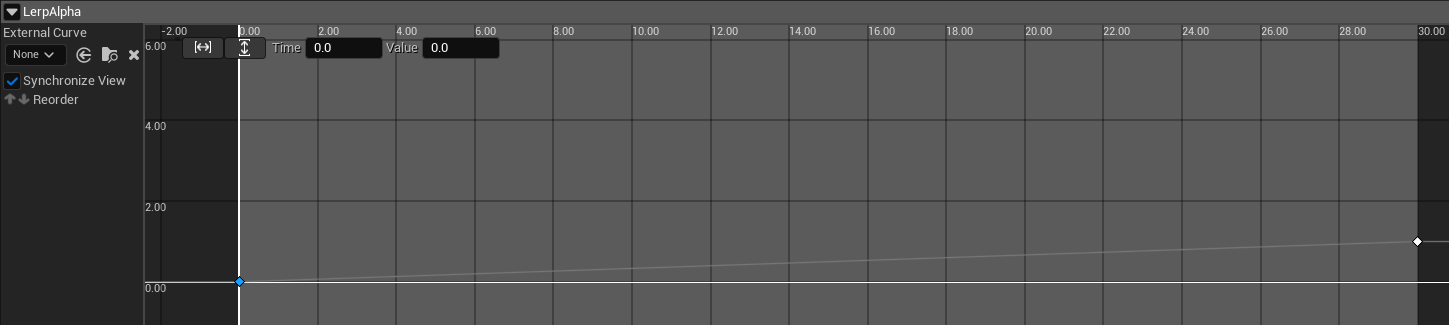
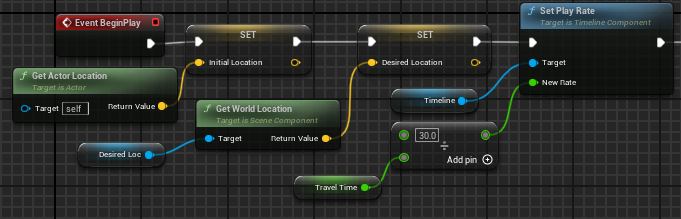
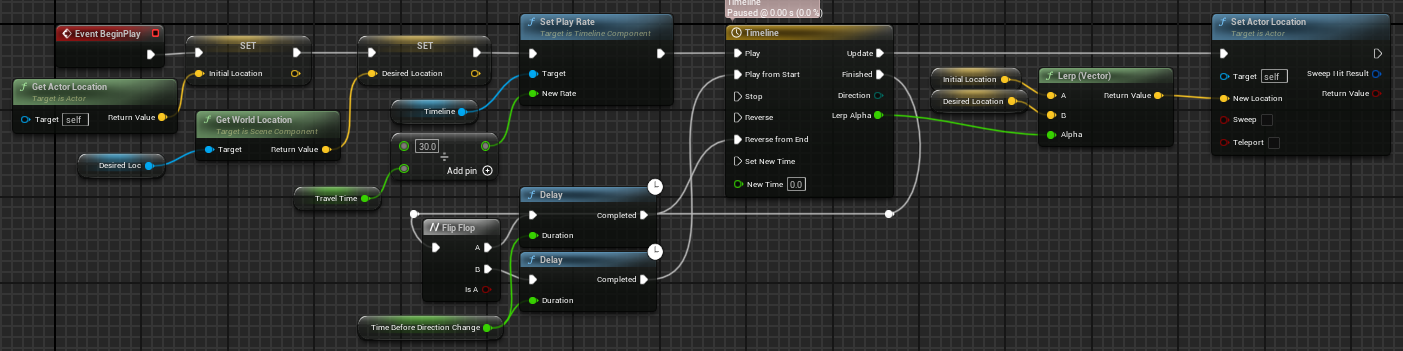
1. Next we will create another actor blueprint named “BP\_WindGustHazard”.
   1. Add a box collider component and name it “Bounds”. Add 2 float variables, one named “MinWindForce” with a value of 200000, and the other named “MaxWindForce” with a value of 250000. We will also add a vector variable named “ForceVector” this will hold our calculated vector to use when applying the force to the player. Last add another variable named “CharacterRef” of type “BP\_ThirdPersonCharacter”.
   2. Right-click the bounds component and create the events begin and end component overlap.
      1. On begin overlap we will check the other actor, cast it to the player character and on success set the character reference to the player character.
      2. On end overlap we will check the other actor, cast it to the player character and on success set the character reference to null.

* 
  1. On the begin play event we will get our force vector setup for later use. First make a vector and for the X and Y pin we will generate a random float and leave the Z component as 0, then normalize this vector. Multiply the normalized vector by a random float between our min and max wind force variables and set our force vector equal to the result.
* 
  1. On the tick event, use a validated get of the character reference, get the character movement component and call “Add Force” using the force vector.
* 

1. Now we will make a platform hazard that will cause the player to behave as if they are on ice. Name this BP\_IcyPlatform.
   1. Start by adding a cube mesh component to the blueprint, set the Z axis scale to 0.05. For the material we will make one using some provided textures.
      1. Right-click the content browser and create a material.
      2. In the material drag off of the base color and search for “TextureSample”, do this also for roughness and normal.
      3. Base color will use the texture “Ice\_Color”, the roughness will use the texture “Ice\_Roughness”, and the normal will use the texture “Ice\_Normal”. Finally add this material to the cube object in the icy platform blueprint.

* 
  1. Now add a box collider and rename it to “Trigger”. Set the scale values for the collider to be X: 1.56 Y: 1.56 Z: 1, and set the position to X: 0 Y: 0 Z: 75.
  2. Right-click the trigger component and add the begin and end overlap events.For the begin overlap, cast the other actor to our player character, get the character movement component. On the character movement component we will set the “GroundFriction” value to 0.0, and set the “BrakingDecelerationWalking” value to 350.0.
* 
  1. On end overlap we will copy and paste the nodes from the other overlap event and set the variables back to their respective default values of 8 for the ground friction and 2048 for the braking deceleration walking.

1. Now we are going to make a couple obstacles, first one will be simple, a rotating platform. We will name this “BP\_RotatingPlatform”.
   1. First to do is add a cube mesh and set the scale to X: 1 Y: 1 Z: 0.1.
   2. Next add a rotating movement component.
   3. Add 3 float variables, one for X, Y, and Z axis, then on begin play on the rotating movement component set the rotation rate values to their respective variable values.
2. Next up we will make a movement platform that will move between its starting point and a desired point in space. Name this “BP\_MovingPlatform”.
   1. Add a cube mesh and set the scale to the same as the rotating platform. Next add a capsule collider component and name it “DesiredLoc” and this is going to represent our point in space that we want the platform to move to.
   2. Let’s now make some variables, we will make 2 vectors, one named “InitialLocation” and the other “DesiredLocation", these will remain at their default values. Then add 2 floats, one named “TravelTime” set this to be 1 by default and this will control how long it takes to get from point A to point B, and the other float will be named “TimeBeforeDirectionChange” and this will determine how long the platform will sit still before moving again.
   3. In the event graph we will be using the begin play node, we will start by setting our InitialLocation to the actors current location, and we will set DesiredLocation to the world location of our capsule collider.
   4. Next we are going to make a timeline, right-click on the event graph and search for “Timeline”.
      1. Open the timeline and make a new track named “LerpAlpha” set the length to 30. We will make 2 keys, the first key will be at time 0 and value 0, the second key will be at time 30 and value 1. This will give us a nice blend of our alpha mask to be used by the lerp function for movement.

* 
  1. Now get the timeline from the components section and call “SetPlayRate” and the new rate will be 30(our length) divided by our travel time variable.
* 
  1. We will then play the timeline, and on update we will call “SetActorLocation” with a Lerp(Vector) where A equals InitialLocation and B equals DesiredLocation and the LerpAlpha pin off of the timeline node will be our alpha. Then our finished pin will go into a “FlipFlop” node and A and B will each go into their own “Delay” node which will take in our TimeBeforeDirectionChange variable.The A delay will be connected to the “Reverse from End” pin and the B delay will be connected to the “Play from Start” pin.
* 

Now using the components we created you can create a level for the game.