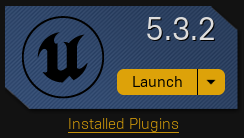
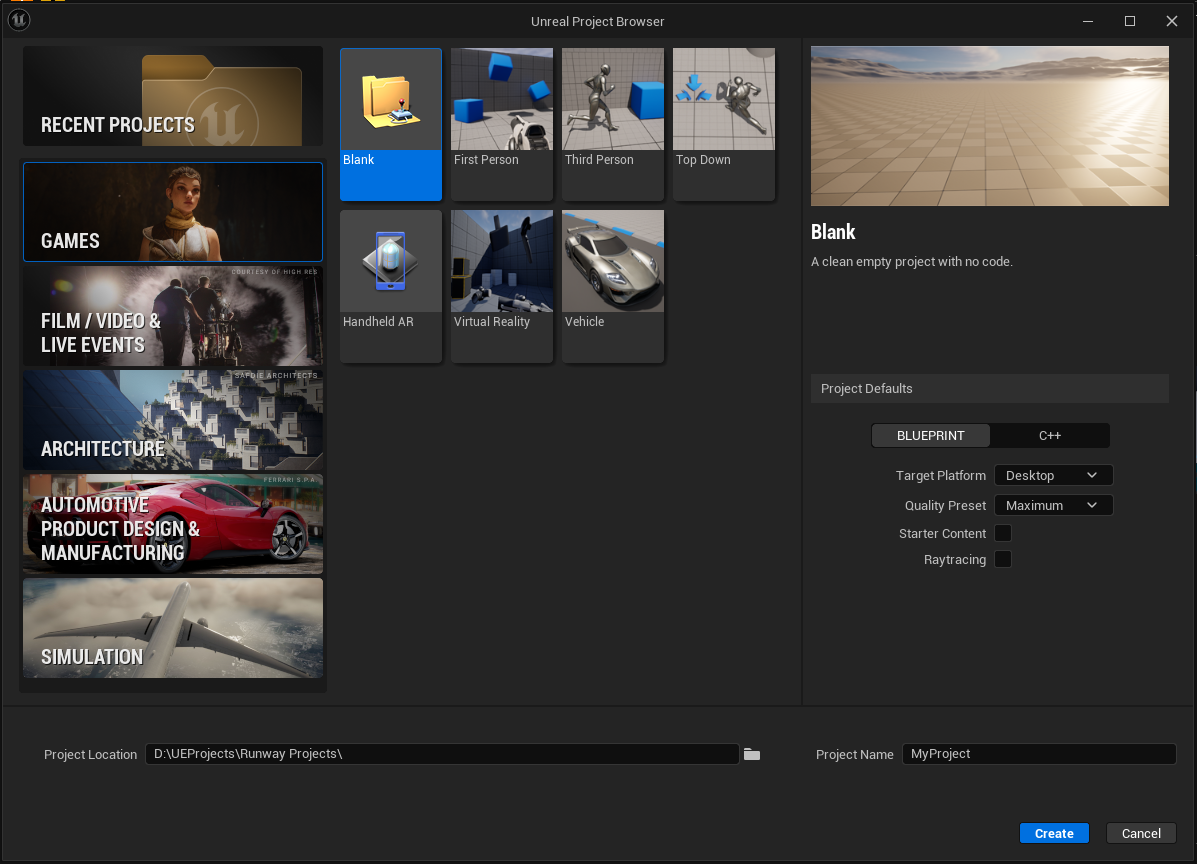
**How to make a bullet hell game using Unreal Engine 5 and visual scripting**

What we need:

1. Unreal Engine 5.3.2

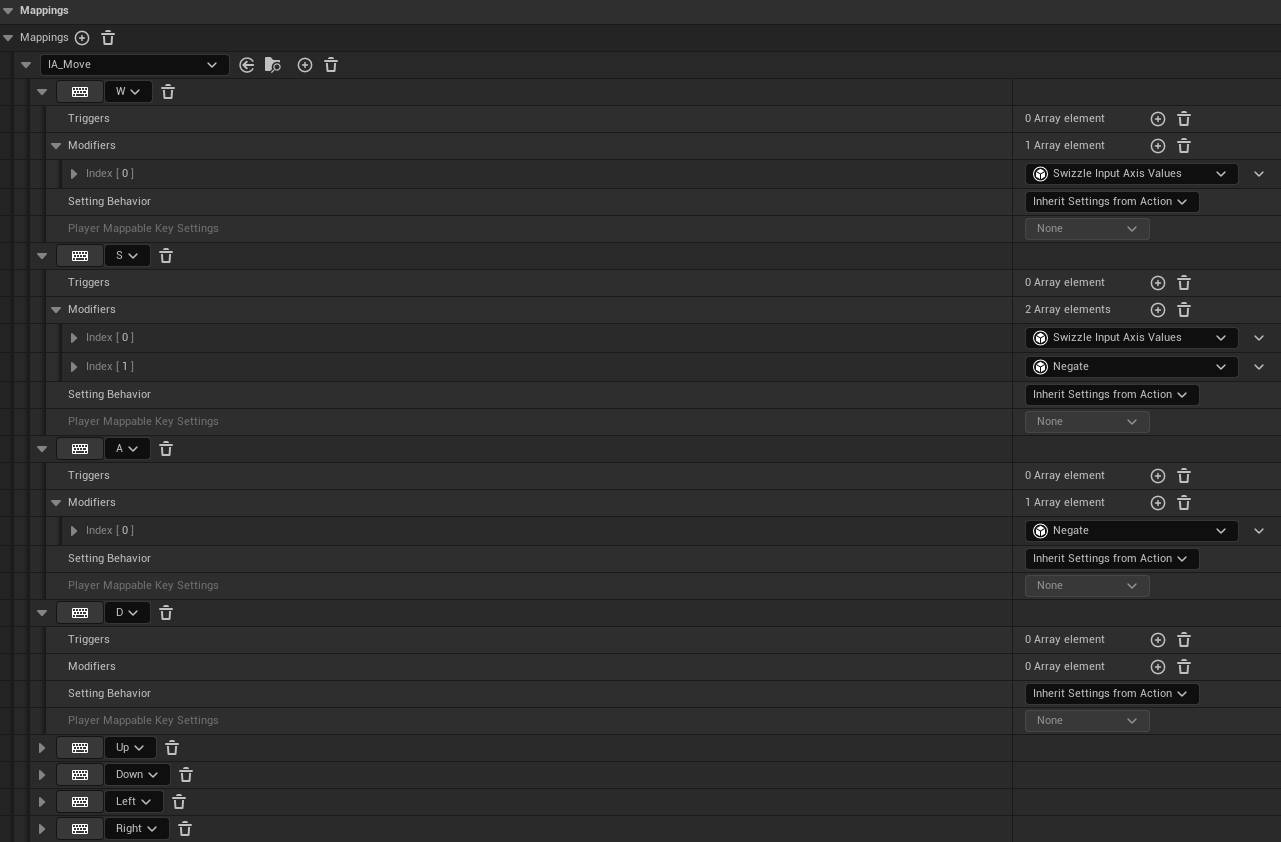
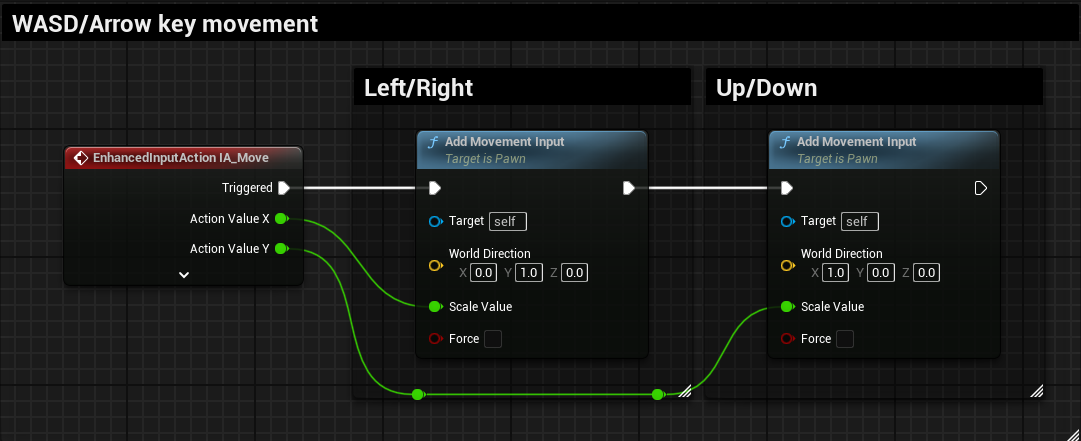


1. Create a new project using the “Top-down Template”(Games->Third Person, Make sure blueprint is selected, uncheck “Starter Content”)



Step By Step Instructions:  
Step 1: Fixing the player

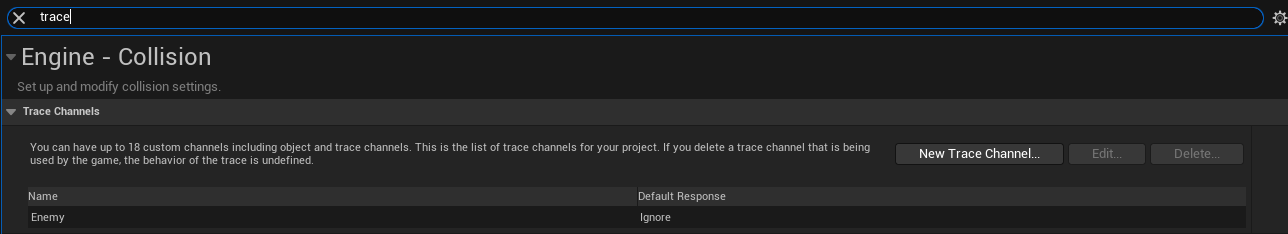
1. The top-down template gives us what we want where the mouse cursor is visible and useful to the player during playtime, however we don’t want to move around by clicking and instead want to use WASD.
   1. So our first course of action will be to add some input actions and alter our input mapping context.
   2. The first input action we will add is our movement one, name this “IA\_Move”, the only value here that we will change is the value type, change this to “Axis2D(Vector2D)”.
   3. While we’re here we will also setup our input for performing our special attack. Make a new input action and name it “IA\_SpecialAttack”, this value type will be of type “Digital(Bool)”. One this one we will also add a “Pressed” trigger to the triggers array.
   4. In our mapping context we can remove all of the default items. First we will add our movement action.
      1. Our first key mapping will be ‘W’, in our modifiers array add the “Swizzle Input Action Values”
      2. Our next mapping will be ‘S’, this will also have the swizzle modifier and it will also have the “Negate” value.
      3. The next mapping will be ‘A’, this will only have the negate modifier.
      4. The next mapping will be ‘D’, this will have no modifiers. You can also add all the equivalent arrow key movements at this time

* 
  + 1. At this time also add our special attack action, set the key mapping to the left mouse button, this will have no modifiers.
  1. We will handle the special attack in a later section, but right now we will handle our new movement.
     1. Remove the old input action events and add the “IA\_Move” event to the event graph.
     2. For the left/right movement we will call “AddMovementInput”, for the world direction we will use the vector X: 0 Y: 1 Z: 0, and the scale will be the action X value. For up/down movement we will call the same function, the vector will be X: 1 Y: 0 Z: 0, and the scale value will be action Y value.
* 

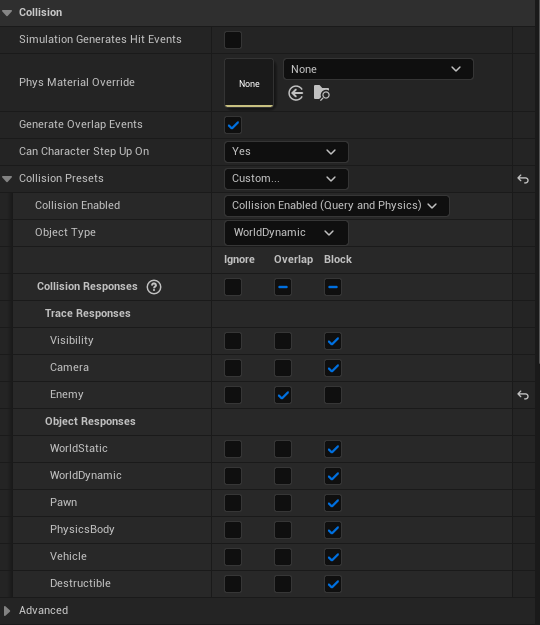
1. To make this more how we would like, change the Y rotation of the spring arm to -90 degrees.
2. Add a new float variable to the player named “MovementSpeed” and on begin play set the “MaxWalkSpeed” variable on the movement component to the value of movement speed.

Step 2: Player basic attack

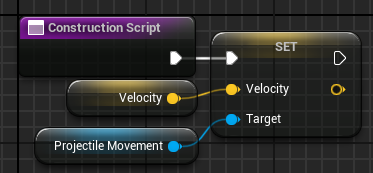
1. The basic attack will be an automatic attack that fires out anytime there is an enemy on the screen and will shoot towards the mouse cursor.
2. First thing we will do to make this more performant will be to go to “Edit->Project Settings” then in the search bar search “Trace” and under the “Trace Channels” section click “Add new trace channel” call this new channel “Enemy” and set the default response of the channel to “Ignore”

* 

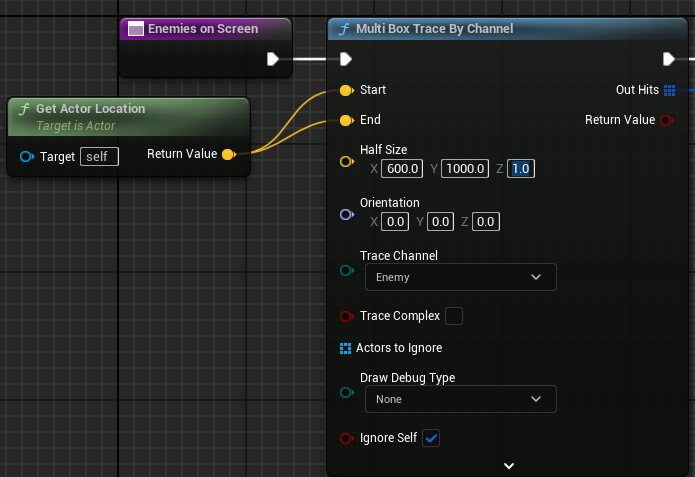
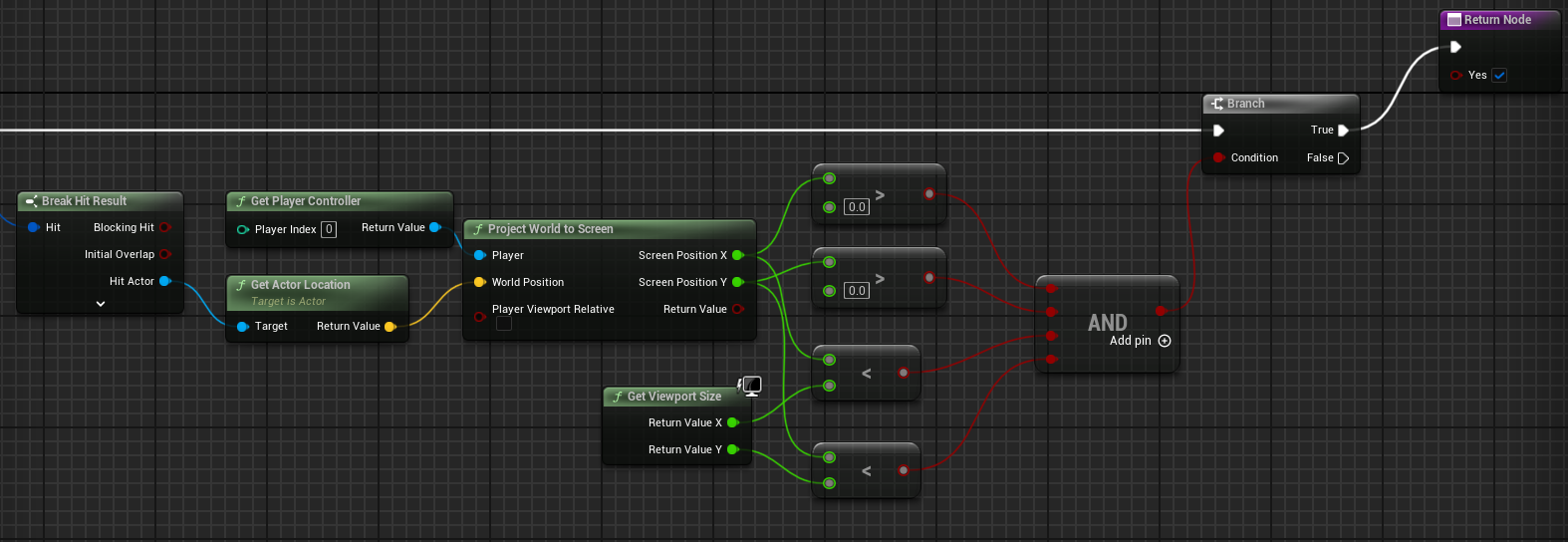
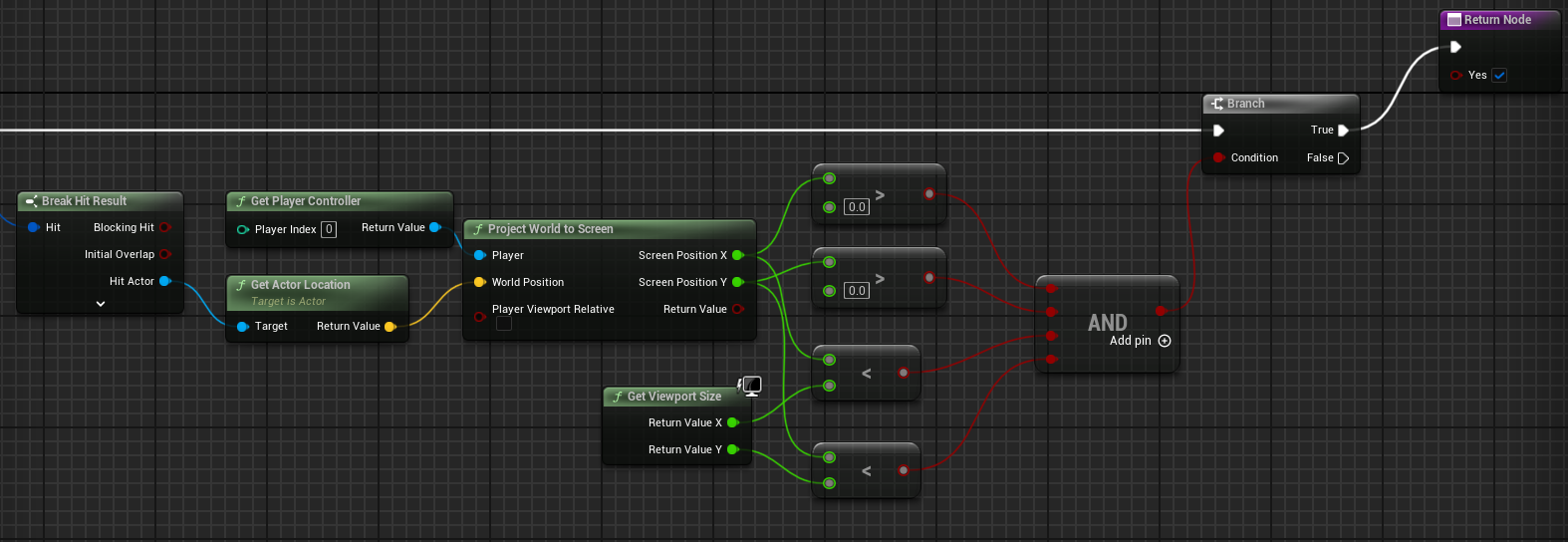
1. Now just to test our functionality we will add a new actor blueprint and name it “BP\_TestActor” add a cube mesh and in the details panel look for the collision section, set the preset to custom and leave everything as it was and change our new custom channel from ignore to overlap

* 

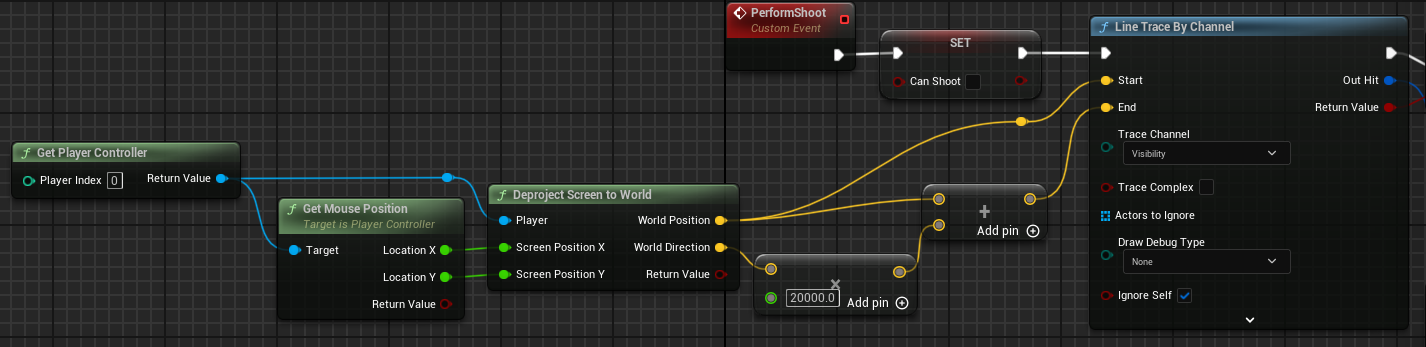
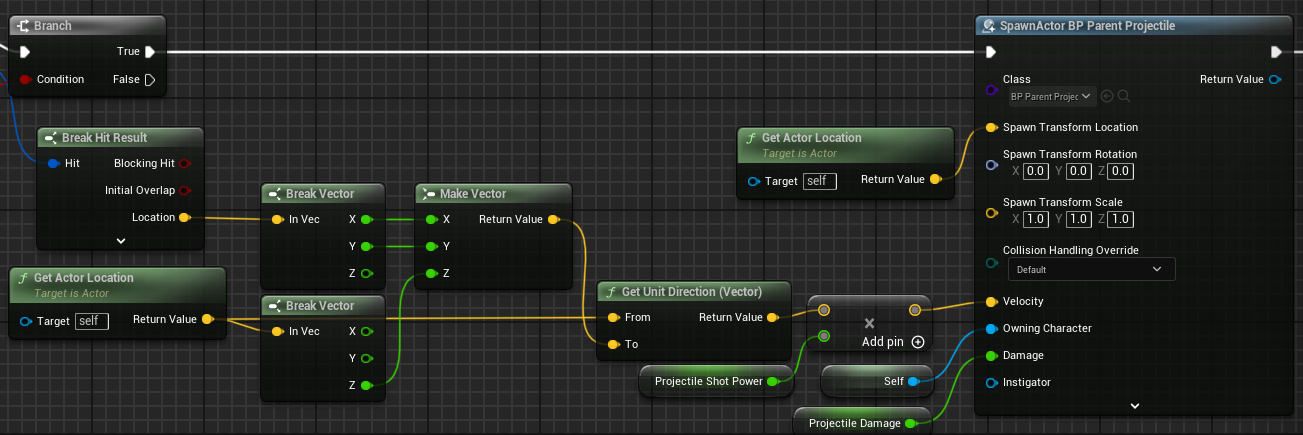
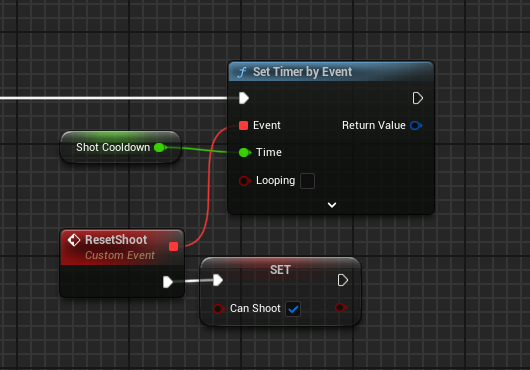
1. We will create another new actor blueprint, this one we will keep and come back to later on, name this “BP\_ProjectileParent”.
   1. Add a sphere mesh and set the scale on all axes to 0.25.
   2. Also add a projectile movement component, everything except 2 values will remain default, change “RotationFollowsVelocity” to true and set the gravity scale to 0.
   3. Now add a vector variable named “Velocity”, make sure to set this variable to instance editable and “Expose on Spawn” this way when we spawn this either from enemies or the player we can easily modify its movement.
   4. In the construction script panel set the velocity of our projectile movement component to the value of our velocity variable.

* 

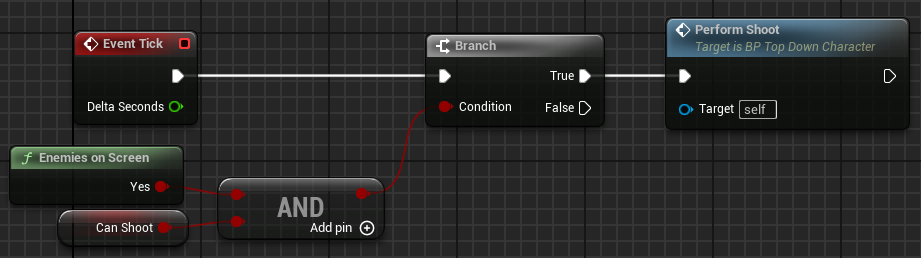
1. As mentioned we will return to the projectiles later on. Open the player blueprint we are going to add some variables that will be useful for our shooting. Add 3 float values named “ShotCooldown”, “ProjectileShotPower”, and “ProjectileDamage”, the default values of these will be 1, 1000, and 1. Also add a boolean variable named “bCanShoot” and by default this will be true.
2. Now for a big section of the attack, detecting if enemies are on the screen. If we get all actors of a class that will take a lot of resources and computing power to do this and if we have to do this every frame it will considerably slow down the game, so instead will we will do trace testing on our enemy channel that way we’re only checking a small area around the player and not grabbing every single actor that exists.
   1. First make a new function named “EnemiesOnScreen” and have a boolean return value.
   2. The first node we will call will do our heavy lifting, call “MultiBoxTraceByChannel”. The start and end location will be the actor location, the half size will reach just outside of our view frustum, put on the value X: 600 Y: 1000 Z: 1, leave the orientation to all 0 since this box does not have to rotate at all. Set the trace channel to our “Enemy” channel, and leave the rest of the values default.

* 
  1. Next we will use a for each loop on our hits array. We will break the hit struct and get the location of the hit actor. We will get the player controller and call “ProjectWorldToScreen” and by doing some basic checks we can see if the given actor is within our viewport.
     1. For the X and Y position check if they are greater than 0.
     2. Now use the “GetViewportSize” node and check our X and Y values against these values to see if our values are less than.
     3. Now use an AND boolean node to check if all 4 of these conditions are simultaneously met. If they are all true return true, it is important we return here and not just set a variable to true because this allows us to save computing time if an enemy is on screen we are going to shoot no matter what so we do not need to check the others.
* 
  1. Now to handle if none of them are in the viewport we will then return false on the completed branch of the for each loop.
* 

1. Now we will make a new custom event named “PerformShoot”, first thing we will do is make sure to set “CanShoot” to false so that way we won’t shoot more than once before the cooldown is up.
   1. Next we need to figure out where the mouse cursor is so we can reliably shoot at the mouse cursor. To do this we will utilize line traces. Call “LineTraceByChannel”.
      1. For the Start location first we will get the player controller and call 2 functions “GetMousePosition” and “DeprojectScreenToWorld” both will take in the player controller reference, the output of the mouse position will feed into the input of deprojection.
      2. The output world position will be the start pin, the end pin will be the sum of the world position and the product of the direction and 20000.
      3. The trace channel will be Visibility and the rest of the parameters can be left as default.

* 
  1. Now using a branch node we will check to see if the return value is true or false.
  2. If true we will spawn our projectile class at the player’s location. Now we want to figure out the velocity vector of our projectile, we want it to stay at the same altitude as the player and not go down so we will need to do some vector manipulation.
     1. First we will get the hit location vector and break it, feed the X and Y components into a “MakeVector” node, take the player's location and break that vector and feed the Z component into our make vector node.
     2. Now with the 2 vectors we want to get the unit direction vector using the appropriate node, the from pin will be the player's location and the to pin is our newly constructed vector.
     3. Now to get the velocity multiply the returned vector by our projectile shot power variable.
* 
  1. Now we need a way of handling our shot resetting so we can shoot more than once.
     1. Off of the spawn actor node call “SetTimerByEvent”, for the time pin use our shot cooldown node, and for the event drag off of the pin and create a new custom event and name it “ResetShoot” and all this event will do is set our can shoot variable to true.
* 

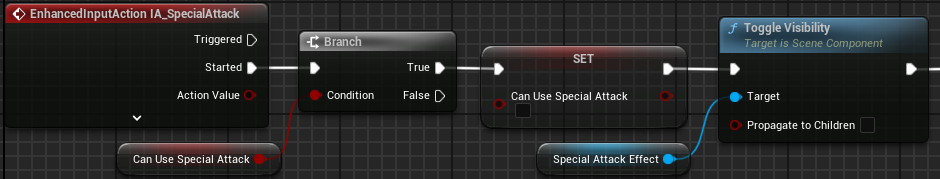
1. On the tick event we will see if there any enemies on the screen, and the output of that node will be put into an AND node with our can shoot variable. If both conditions are true we will call “PerformShoot”.

* 

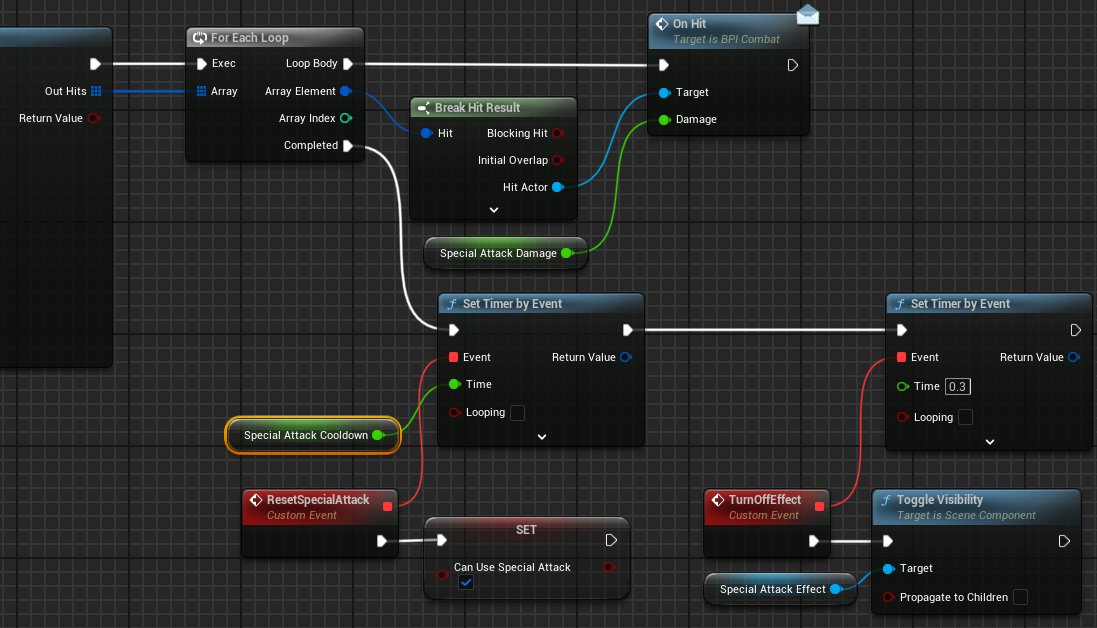
1. Now our player will automatically shoot at the cursor whenever something with the trace channel enters the screen. (Note: this method for some reason does not work when playing in editor, you have to play test in a separate window)

Step 3: Player special attack

1. Firstly we will add the “IA\_SpecialAttack” event for the player and add a bool named “CanPerformSpecialAttack” which will have a default value of true.
2. We will check to see if this variable is true, if it is true the first thing we will do is set that variable to false.
3. We also are going to add a sphere component named “SpecialAttackEffect” with a location of X: 0 Y: 0 Z: -90 with a scale of X: 3.25 Y: 3.25 Z: 0.25. And we will use our custom material instance “M\_SpecialAttackEffect”. Make sure the default visibility is set to false.
4. Now after setting our can use special attack variable to false we will call “ToggleVisibility”.

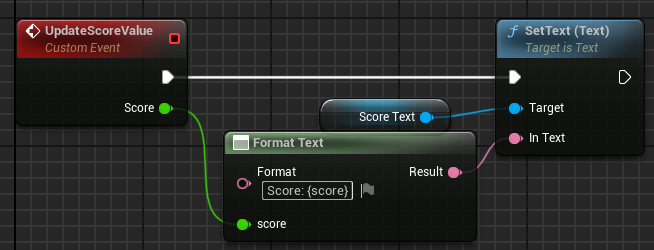
* 

1. Now we will create a new variable for use in the next steps named “SpecialAttackRange”, “SpecialAttackDamage”, and “SpecialAttackCooldown” both will be floats and their default values will be 150, 5, and 1 respectively.
   1. We will call “MultiSphereTraceByChannel” with the start and end locations set to the player location and the radius will be the variable we just created. Our trace channel will be our custom “Enemy” channel.
   2. Now on the out hits variable we will do a foreach loop, for each iteration of the loop we will break the hit result and call our “OnHit” message.
   3. Then off of the completed exec pin we will set 2 timers by events, the first event will reset our can use special attack variable to true on a timer using our variable, and the other will again call “ToggleVisibility” on the effect component with a timer of 0.3 seconds.

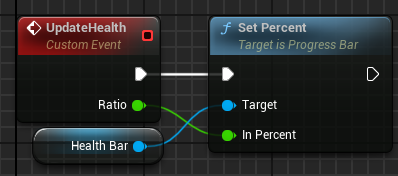
* 

Step 4: Player HUD

1. We will now start on the HUD for the player. Make a new widget blueprint and name it “WBP\_PlayerHUD”.
2. First we need to add a canvas panel to our widget. The first element we will add is a progress bar, rename this to “HealthBar”.
   1. Set the fill color to R: 0 G: 1 B: 0 A: 1.
   2. Set the anchor to the top middle of the widget.
   3. Set the position to X: 0 Y: 48 and the size to X: 500 Y: 40
3. Next add a text element, name this “ScoreText”.
   1. Set the anchor to the top middle.
   2. Set the position to X: 0 and Y: 110. Set the “Size to Content” box to true.
   3. Set the default text to “Score: 0:
4. Now in the event graph add a new custom event named “UpdateScoreValue”, this will have a float input.
   1. Now with our “ScoreText” reference we will call set text, the text input will be formatted text and the format will be “Score: {score}”.
   2. Our score input value will be put into this text format.

* 

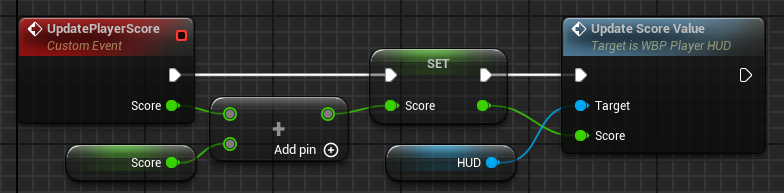
1. Now we will make a new custom event named “UpdateHealth”, this will have a float input that will represent a ratio.
   1. We call “SetPercent” on our health bar reference using our ratio as the percent.

* 

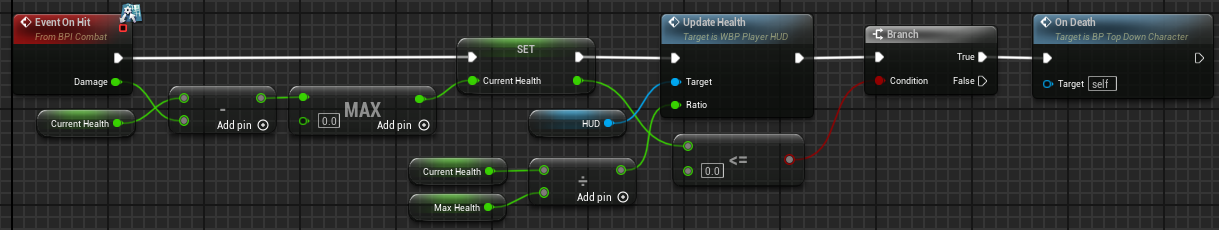
1. This is everything done on the HUD, now let’s add the needed variables and functionality to the player.
   1. Add a new variable of type “WBP\_PlayerHUD”. Off of begin play call create “CreateWidget” and select our blueprint class and set our variable reference to the output object and then call “AddToViewport”.

* 

1. Add a custom event to the player and name it “UpdatePlayerScore” and have a float input to indicate the amount of score to add.
   1. Add a float variable to the player and name it “Score”.
   2. Off of the event node we created we will get our current score and add our input value to it and set our score equal to the sum, then call our update score method on our HUD reference.

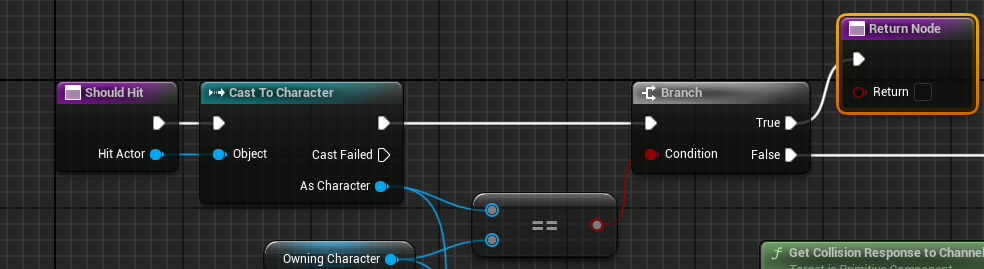
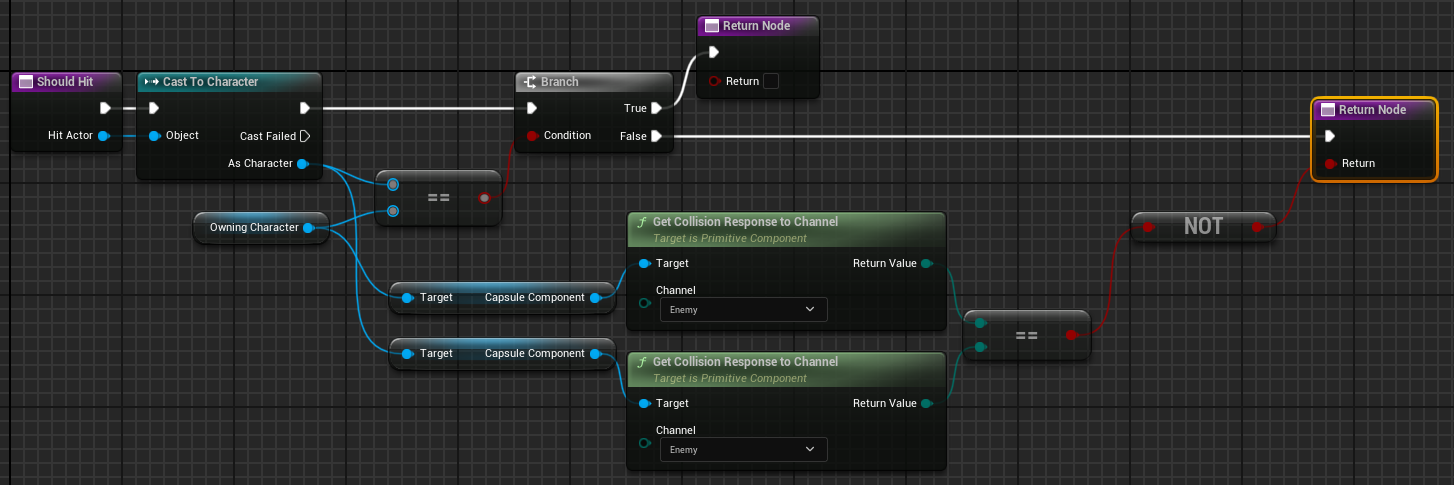
* 

1. Now we will work on the health portion, we will start by adding 2 float variables, one will be “MaxHealth” and the other will be “CurrentHealth”, set the default for max health to 3.
   1. On begin play we will set our current health value to our max health value.

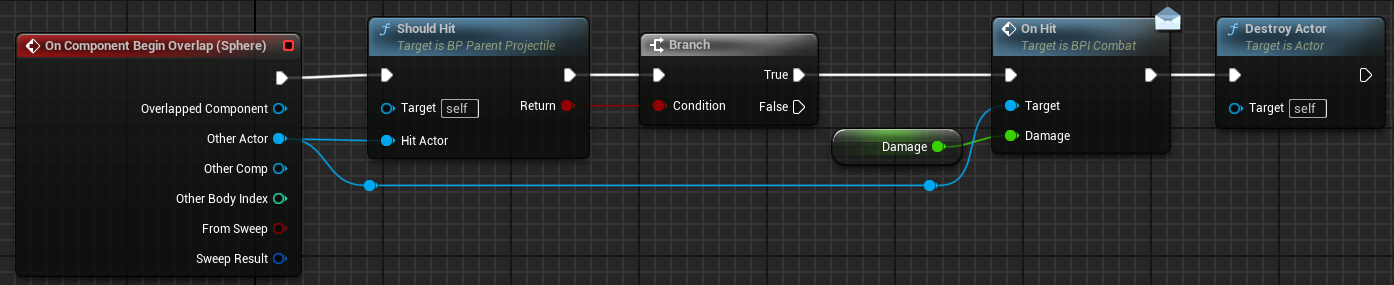
* 
  1. Now we will add a blueprint interface named “BPI\_Combat” this will be useful for both our player and our enemies. It will have 1 function for now named “OnHit” with a float input to indicate the damage amount.
  2. Change the class settings on the player blueprint to implement this new blueprint interface.
  3. Under the interface section we can now implement the event. First we will subtract incoming value from the current health of the player, then take the max value between that value and 0, this will limit it so the player’s health cannot drop below 0. Then set our current health to the output value of the max node. Then update our health value on our HUD. Check if the current health is less than or equal to 0, and if it is we will perform death.
* 

Step 5: Projectiles:

1. Now we will reopen our projectile parent blueprint. We add 2 new variables to this blueprint, one of type “Character” named “Owning Character” and the other of type float named “Damage”, these will both be instance editable and exposed on spawn.
2. We will create a new function named “ShouldHit”. It will have an input of type Actor and a return type of bool.
   1. First we will cast the incoming actor to type “Character” if it succeeds we will check to see if it is equal to the owning character, if they are equal we will return false.

* 
  1. Then using our custom we can determine if the owner is an enemy or not and check if the hit actor is an enemy. We will call “GetCollisionResponseToChannel” on the capsule component on both the owning character and the hit character, and check if the responses are equal, then return the NOT value of that check.
* 

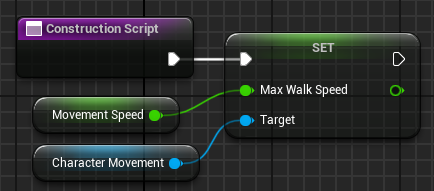
1. Now we will add the component begin overlap event. Off of the exec pin call our new “ShouldHit” function feeding in the “OtherActor” into our hit actor pin. Then use a branch node using our output, on true call our “OnHit” message on the hit actor, and put our damage variable in, then we will destroy the actor.

* 

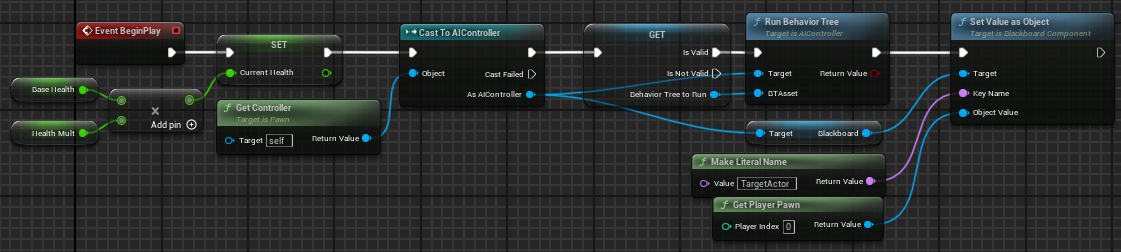


Step 6: Enemies:

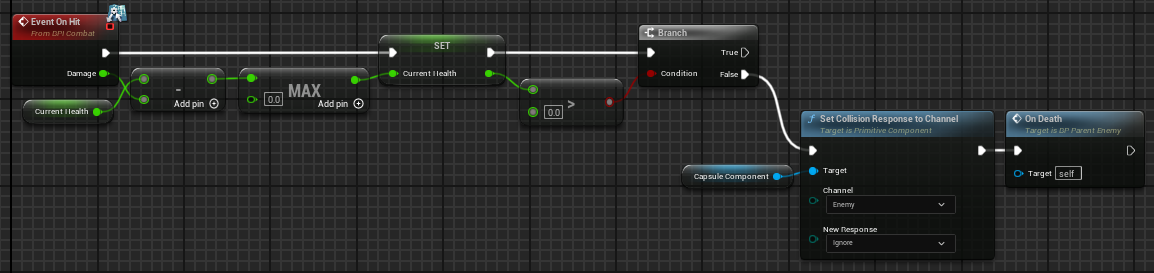
1. We will start by making the base enemy character, for this duplicate our player character and rename it “BP\_ParentEnemy”.
   1. Now we need to set this up for our AI, delete all of the player variables other than “MovementSpeed” and “CurrentHealth”.
   2. Now let’s add 4 more float variables, name them “ScoreValue”, “BaseHealth”, “BaseHealthMult”, and “HealthMult” make sure health mult is instance editable and exposed on spawn. Movement speed will have a default value of 400 and base health mult will have a default value of 1.
   3. We will also create a variable of type “BehaviorTree” and name this “BehaviorTreeToRun”.
   4. We can remove the camera and spring arm components from the character and delete everything in the event graph.
   5. On the mesh component set the collision preset to “Ragdoll”. On the capsule component switch it to “Custom” and leave everything default except for “Enemy” in the trace channels.
2. In the construction script we will set our max walk speed variable to our movement speed variable

* 

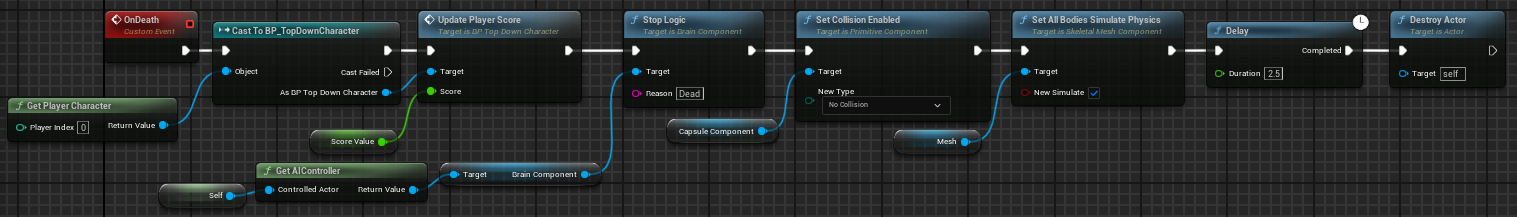
1. Now add the begin play event and we will firstly set our current health to the base health times the health mult. Then we will call “GetController” and cast it to the “AIController” type, and get our behavior tree to run, make sure this is a validated get, and we will call “RunBehaviorTree” on our casted controller, then get our blackboard and call “SetValueAsObject”, the name will be “TargetActor” and the object will be a simple call to “GetPlayerPawn.

* 

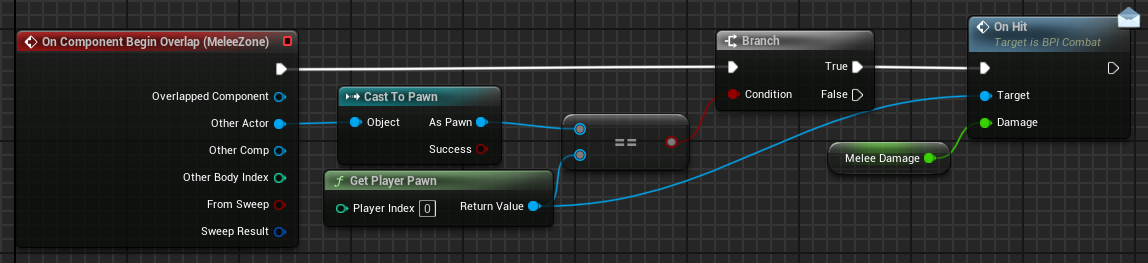
1. Now we will implement our “OnHit” event.
   1. First we will subtract the incoming damage from the current health, then put that return value into a max node and the other value will be zero, then set our current health to the return value. Check if the return value is greater than 0, if false get the capsule component and call “SetCollisionResponseToChannel” and the channel will be our enemy channel and set the new response to “Ignore”.
   2. We will also create a new custom event named “OnDeath”, call this at the end of our on hit event.

* 

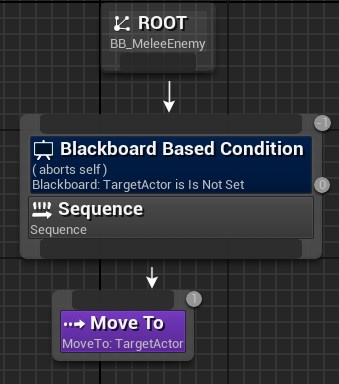
1. Now lets work on the “OnDeath” functionality.
   1. First thing we will do is get our player character and update our score on the player. Then we will get our AI controller, get the brain component and call “StopLogic” to stop running the behavior tree. Then we will disable collision on our capsule component, set the mesh to simulate physics, add a 2.5 second delay and then destroy the actor.

* 

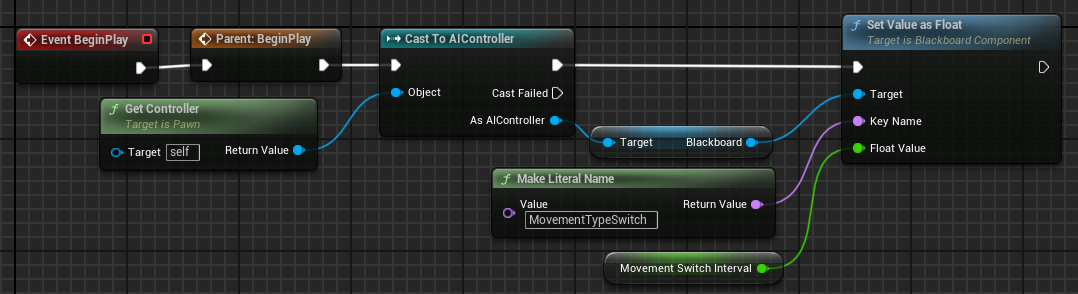
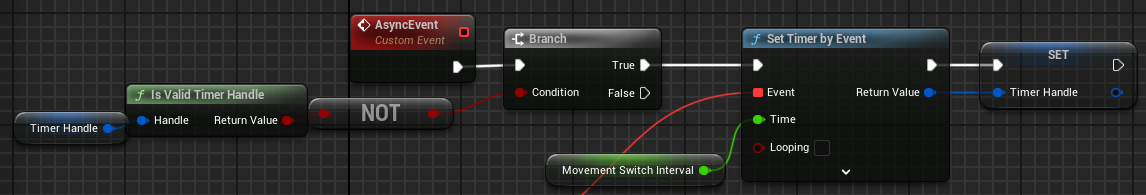
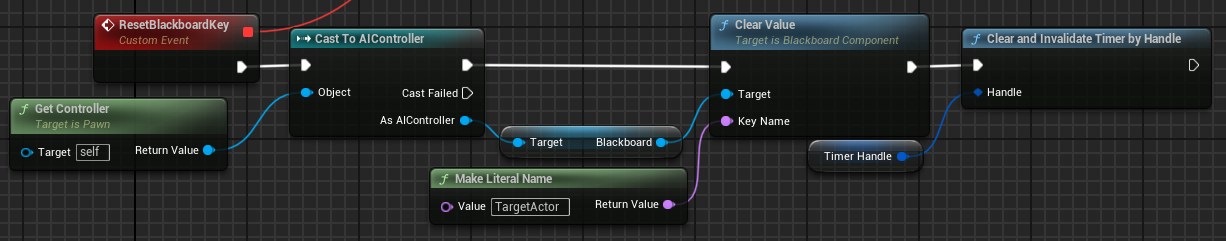
1. Now we will create our first child enemy, on the parent enemy blueprint right-click it and select “Create Child Blueprint Class”, name this class “BP\_MeleeEnemy”.
   1. This class is very simple, add a float variable named “MeleeDamage” this will be set to 1.
   2. Now we will set the inherited values, our movement speed will be 400, the score value will be 25, the base health will be 5.
   3. We will now add a sphere collision component, name it “MeleeZone”. Set the radius to 110. For the collision we are going to do custom preset and turn everything to ignore and change pawn to “Overlap”.
   4. Now we can implement our on overlap begin event on the sphere component.
      1. Take the hit actor and cast it to a pawn and check if it is equal to our player pawn, and if it is our player we will call our “OnHit” message and pass in our melee damage variable.

* 
  1. And now we will add a little extra to our on death event. Destroy our melee zone component and then call the parent version of the event.
* 

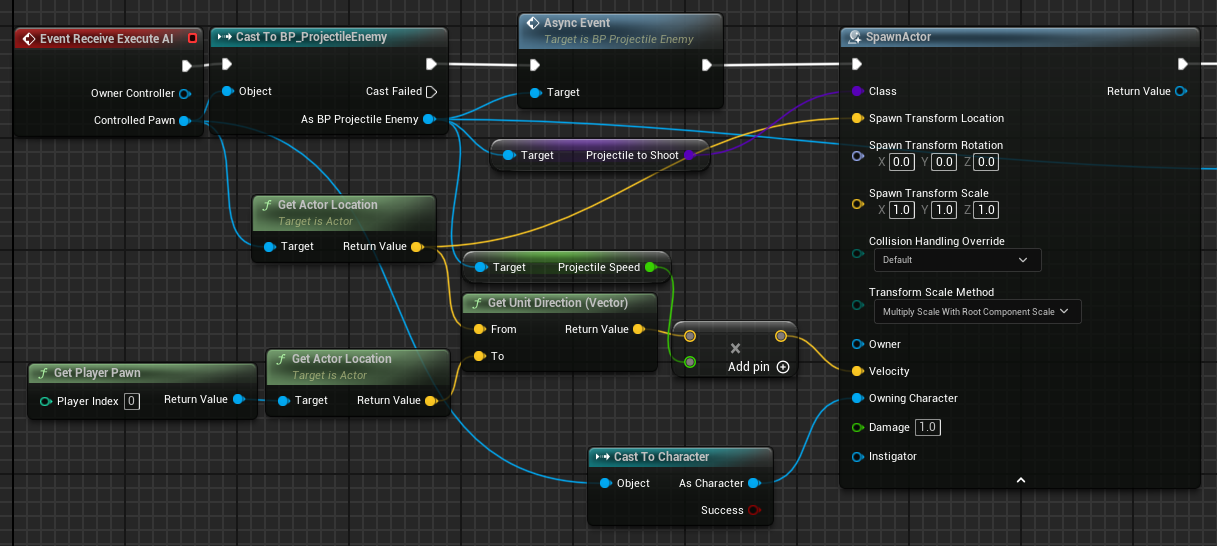
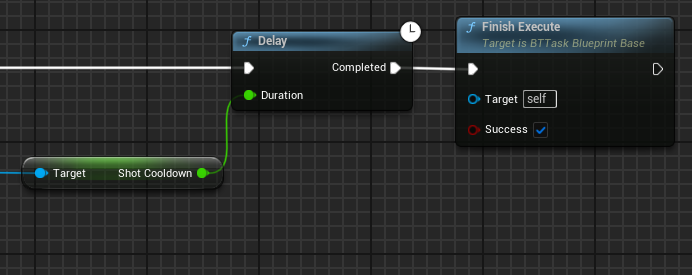
1. We will now make the behavior tree and blackboard for the melee enemy.
   1. Name the behavior tree “BT\_MeleeEnemy” and the blackboard “BB\_MeleeEnemy”.
   2. In the blackboard the only additional key we will be using is an object type with the base class set to “Actor” and name this “TargetActor”.
   3. The behavior tree is also going to be very simple, off of the root node we will add either a sequence or selector node, it does not matter since we only have 1 behavior node to run, and this will be the “MoveTo” node and set the move to location to be the target actor.

* 
  1. Now this blueprint can be added to our melee character as the “BehaviorTreeToRun” variable.

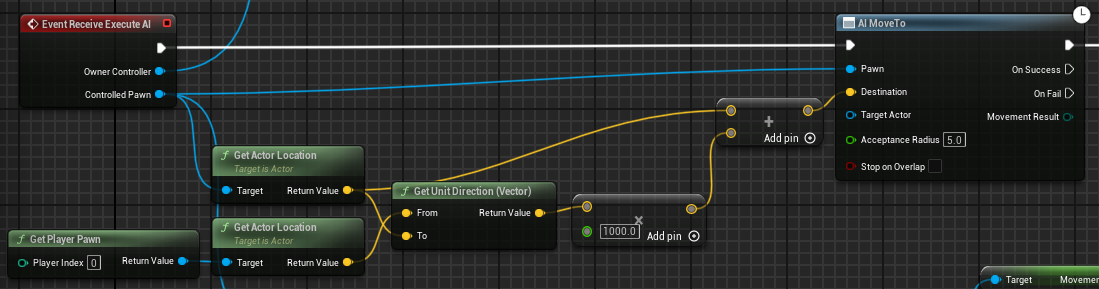
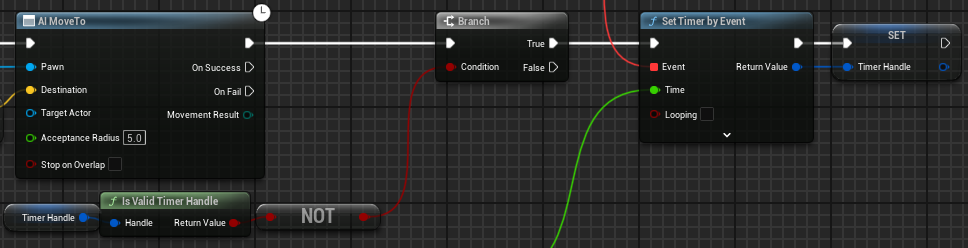
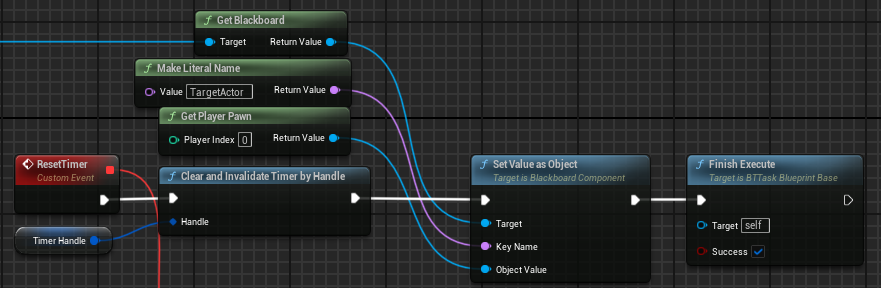
1. Now we will create the enemy that will switch between chasing the player and shooting to running away from the player, we can achieve this by duplicating our melee enemy and changing a few things. Change the name to “BP\_ProjectileEnemy”.
   1. First get rid of the sphere collision component since we won’t be needing it. The damage variable can also be removed at this time.
   2. Now we will add some variables.
      1. Add a variable of type “BP\_ParentProjectile” class reference, name this “ProjectileToShoot”, the default value will be the parent class.
      2. Now add 3 float variables “ProjectileSpeed”, “MovementSwitchInterval”, and “ShotCooldown”, the default values for these will be 750, 5, and 1 respectively.
      3. And finally we will add a timer handle variable to handle some async tasks for the behaviour tree.
   3. On begin play first we will call the parent event, then we will get the AIController and get the blackboard and call “SetValueAsFloat” the name to be passed in will be “MovementTypeSwitch” and the value to be passed in will be our “MovementSwitchInterval” variable.

* 
  1. Now we will make 2 new custom events one named “AsyncEvent” and “ResetBlackboardKey”
     1. For the async event our first node will be a branch node and we’re checking to see if the timer handle is not valid, on true we will set timer by event where the event is our reset blackboard event and the time will be our movement switch interval and then set our timer handle equal to the return value.
* 
  + 1. Now for our reset blackboard key event, first off we will get the AIController and then get the blackboard variable and clear the “TargetActor” key, then clear and invalidate our timer handle variable.
* 

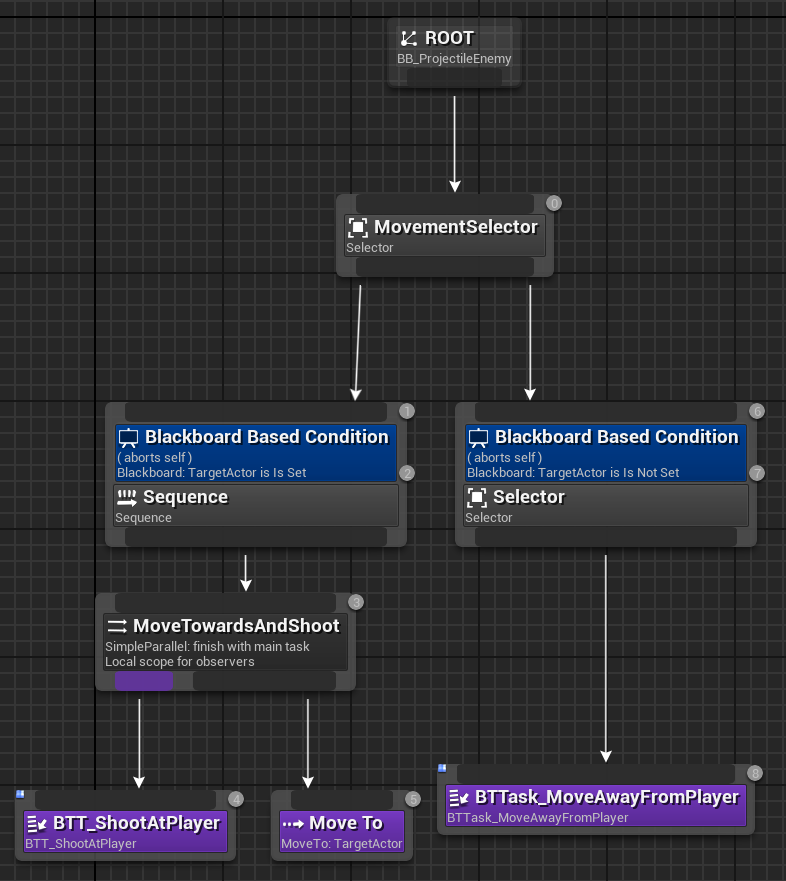
1. Now we will make some behavior tree tasks. Create a new blueprint with the parent type “BTTask\_BlueprintBase” and the first one will be named “BTTask\_ShootAtPlayer”
   1. Firstly override the “ReceiveExectueAI” function, cast the controlled pawn to our projectile enemy type call our async event and then we will call “SpawnActorFromClass”.
      1. For the class pin get the projectile to shoot variable from the projectile enemy.
      2. The spawn location will be the controlled pawn location.
      3. Now for our velocity pin, we will get a unit direction vector from the controlled pawn location to the player pawn location. Multiply the unit vector by the projectile speed off of the projectile enemy.
      4. Then set the owning character to the controlling pawn casted to a player.

* 
  + 1. Now we will use a delay node with the time of “ShotCooldown” off of our enemy, and then finally a finish execute node with it returning true.
* 

1. Now to make another task, this one will be named “BTTask\_MoveAwayFromPlayer”.
   1. We will override receive execute AI. First node will be “AIMoveTo”. Now we need to fill out all of the pins for this node.
      1. The pawn will be our controlled pawn.
      2. Our destination will be a calculation, we will get a unit direction vector from the players location to our controlled pawns location, and multiply the result by 1000 and then add this vector to our controlled pawns location.

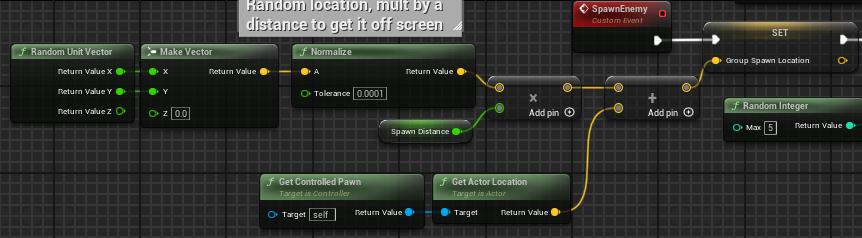
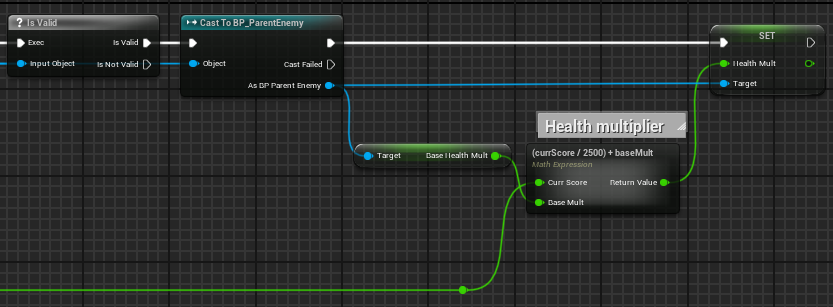
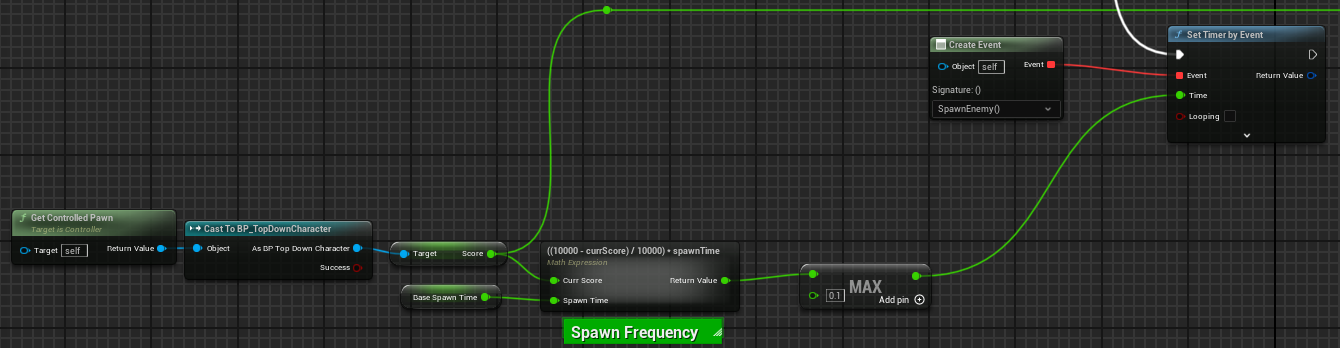
* 
  1. Then we will make a branch node which will be filled out shortly. Off of the true node we will create a timer by event the time will be our movement switch interval divided by 2, and our event will be a new custom event named “ResetTimer” then promote the return value to a variable that we will use.
  2. Now for our branch we will see if our new timer handle is NOT valid.
* 
  1. Now for our reset timer event, firstly we will clear and invalidate the timer handle, then we will set a value as object on our blackboard from our owner controller node, the name will be “TargetActor” then the object value will be our player pawn. Then we will call “FinishExecute” with true.
* 

1. Now we will create our behavior tree and blackboard for our projectile enemy, one will be named “BT\_ProjectileEnemy” and the other “BB\_ProjectileEnemy”.
   1. On our blackboard we will add 2 keys, one of type object named “TargetActor” with the base class of Actor, and the other of type float named “MovementTypeSwitch”.
   2. Now onto our behavior tree, we will use a selector, the left side of the tree will be a selector node with a blackboard based condition which will abort itself and will abort if the TargetActor is set, then use a simple parallel node so we can move and shoot at same time, on the left side we will call our shoot at player task, and on the right side call move to with our target actor as the location. On the right side of the initial selector node we will have another selector with a self abort condition when target actor is not set and there we will use our move away from player task.

* 
  1. Now on the projectile enemy class we will set our behavior tree to run variable to this new behavior tree.

Step 7: Enemy spawning algorithm:

1. First we will start off with a new custom event named “SpawnEnemy”.
   1. We will make some new variables, 1 of type vector named “GroupSpawnLocation”, and 2 float variables named “SpawnDistance” and “BaseSpawnTime” the 2 default values for the floats will be 900 and 5 respectively.
   2. Now to start on the logic, the first node we will use is to set our group spawn location, to do this we will get a random unit vector, break the struct and we will make a new vector using only the X and Y components with the Z component being 0, then normalize the resulting vector. Multiply the normalized vector by our spawn distance, then add this result to the location of the controlled pawn.

* 
  1. Now we will make a for loop, and the first index will be 0, and the last index was a random int with a max of 5. In the loop body we will spawn actor from class.
     1. The class pin will be a true/false selection of our 2 enemy types.
* 
  + 1. The spawn location will be our group spawn location.
    2. Set the Collision handling override to “Try to Adjust Location, Don’t spawn if still colliding”
  1. Check if the spawned actor is valid then cast it to our parent enemy class and we will get the base health mult, and use a math expression to set our current health mult.
     1. The math expression will be “(currScore/2500) + baseMult”
* 
  1. Now on the completed pin on our for loop node we will call set timer by event, for the event drag off of the pin and use the “CreateEvent” node and in the dropdown use our “SpawnEnemy” event.
  2. Now for the time this will decrease based off of the players score.
     1. We will get our score from the player and use another math expression which will be “((10000-currScore)/10000)\*spawnTime” and our spawn time will be the base spawn time variable.
* 

1. Now on begin play we will call this event to kick off the gameplay loop.