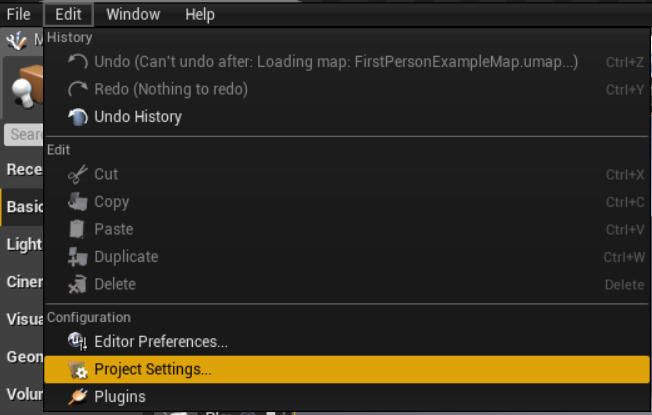
Week 4 – Basic Abilities

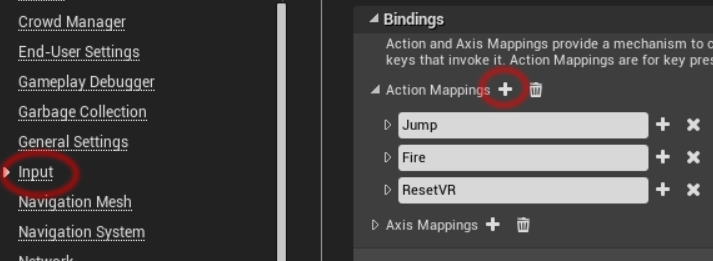
**Section 1 – Setting up a new input**

First we are going to set up a new input mapping for when the player presses Q. We will then bind this input to a function in our character.

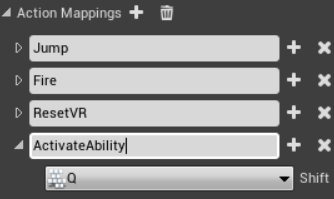
In editor navigate to Edit->Project Settings…



1. Navigate to Engine->Input on the left panel.
2. Press the “+” next to “Action Mappings”



Name it “ActivateAbility” and assign it to the “Q” key.



Open up visual studio and open your character header.

Add a new protected function:

//Function bound to activity activated key

void OnAbilityActivated();

In your .cpp file find the function called SetupPlayerInputCommand()

Add the following binding, replace AGEPCourseworkCharacter with your character class name

PlayerInputComponent->BindAction("ActivateAbility", IE\_Pressed, this, &AGEPCourseworkCharacter::OnAbilityActivated);

Back in the header add this event declaration below OnAbilityActivated()

//Blueprint event fired when ability is activated.

UFUNCTION(BlueprintImplementableEvent, Category = "Abilities")

void AbilityActivated();

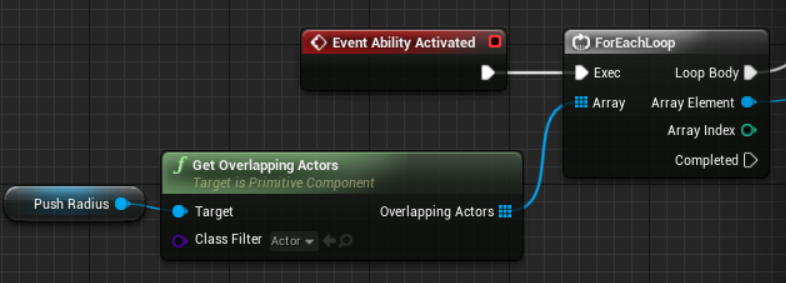
In the .cpp create a definition of OnAbilityActivated() it should only call AbilityActivated(). This is the same principle as what we did with OnFire().

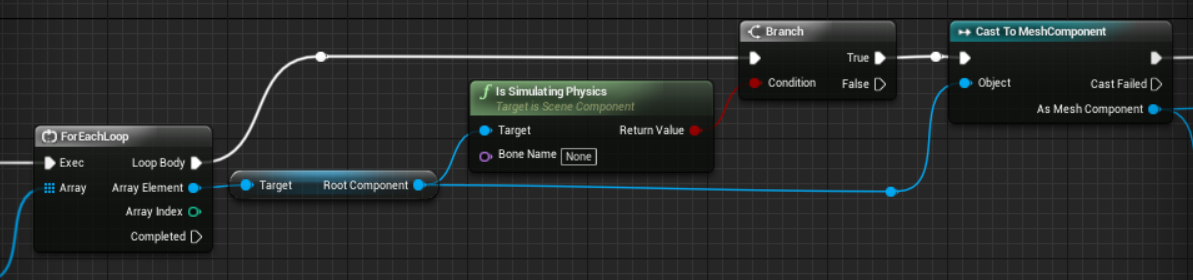
**Section 2 – Scripting a new hero**

In this section we are going to create a new blueprint deriving from our character class, it will have scripting specifically for this character, in this case, its unique ability.

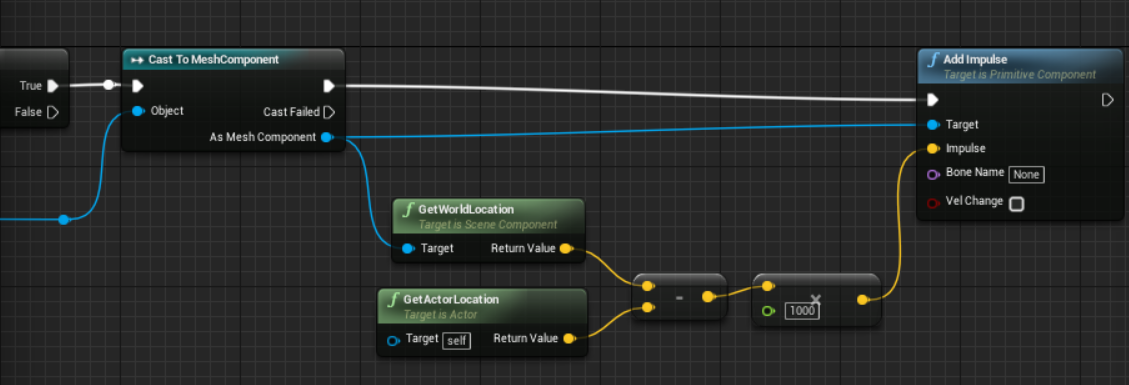
*It’s worth noting that this is the most basic way to do this that works for this scenario, i.e. if you needed customizable ability loadouts or multiple characters with the same ability you would need an ability class. The character would then have a reference to an ability derived from a base ability class. The base ability class should deal with things like cooldown, activation time/ticks etc.*

In editor create a new blueprint class deriving from FirstPersonCharacter. This example is going to have a force push ability so I am going to call it “ThePusher”.

1. Open up the blueprint editor
2. Add a Sphere Collision component and call it “PushRadius”
3. In its details panel, Set its radius to be 1000
4. In the event graph create an AbilityActivated node
5. Drag your PushRadius component from the components panel into the graph
6. Drag off it to create a GetOverlappingActors node
7. Set the class filter to be “Actor”
8. Drag out of the Overlapping Actors array and create a ForEachLoop node *(for loop)* 
9. Drag out of the Array Element pin of the ForEachLoop and get the Root Component
10. Check if that root component is simulating physics
11. If it is then cast it to a MeshComponent *(only primitive components such as mesh can apply force)*



1. Drag out of the “As Mesh Component” pin and create and Add Impulse node
2. Also drag out and get its world location
3. Get the actor location (for this character)
4. Minus the characters location from the actors and multiply it by 1000
5. Plug this result into the “Impulse” pin on the Add Impulse node.



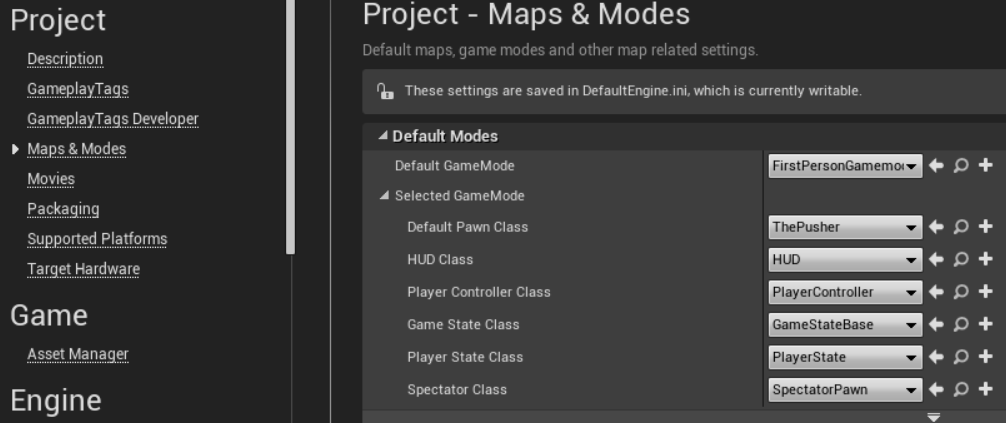
**Section 3 – Changing the default character**

Delete the FirstPersonCharacter that is already in the scene.

Go to the Project Settings ( Edit->Project Settings..)

Select Project->Maps & Modes then change the Default Pawn Class to ThePusher

*Remember you need to do this if you ever override the Gamemode, Player Controller etc.*



*There also seems to be functionality to do this in C++ in the GameMode.cpp but as far as I can tell project settings overrides it.*

**Problems:**

**The projectile seems to spawn above the player**

In the PushRadius component set its collision matrix to custom then make sure projectile is set to overlap

**The Cubes in the scene are not moving**

Select all the cubes in the scene, navigate to their collision settings and tick “Generate Overlap Events”

Still not working, try creating a print string node at the end of the graph we just created to see if it is executing.

**Challenges:**

1. Create another FirstPersonCharacter derived blueprint with the ability to blink forward.
2. Prototype another ability of your choice
3. In C++ create a base ability class, in the character (C++) give it a property similar to how “ProjectileClass” is set up. Create a blueprint derived from ability then assign it to the character in the inspector.
4. Refactor the code to work with your new ability class, the scripting side should still be done in blueprint.
5. In base ability add functionality such as cooldown time, i.e the event should only fire if the ability is ready.