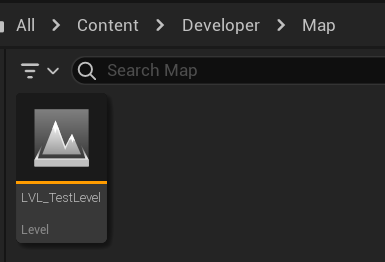
# **The End Times - Demo Documentation**

Name: Chong

UE Version: 5.3.2

Test Map:



## **1. Coding Rules**

I was following the rules from **The End Times - Programming Guideline and Naming Convention**

## **1.1 Naming Convention:**

Variable and Function:

Use PascalCase for class names.

Use camelCase for variables and functions.

The only difference with the guideline is I will add 'm\_' prefix for the private variables. (exp: m\_PlayerRef)

## **1.2 Organisation & formatting:**

For organisation and formatting, it will follow the sequence below:

AClassConstructor();

public:

UFUNCTION()

void func();

void func2();

public:

UPROPERTY()

Int a;

int b;

protected:

SAME FORMAT AS PUBLIC

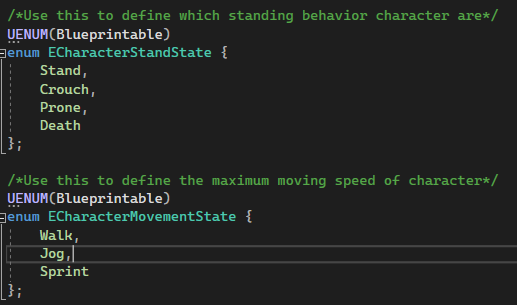
private:

SAME FORMAT AS PUBLIC

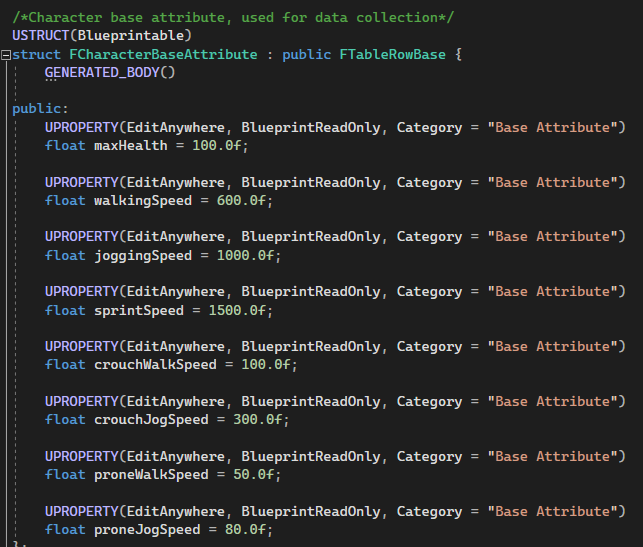
## **2. Character**

Both Zombie and Player shared the same base class called CharacterBase.h. Therefore, all characters should contain the Enum and Attribute like below:

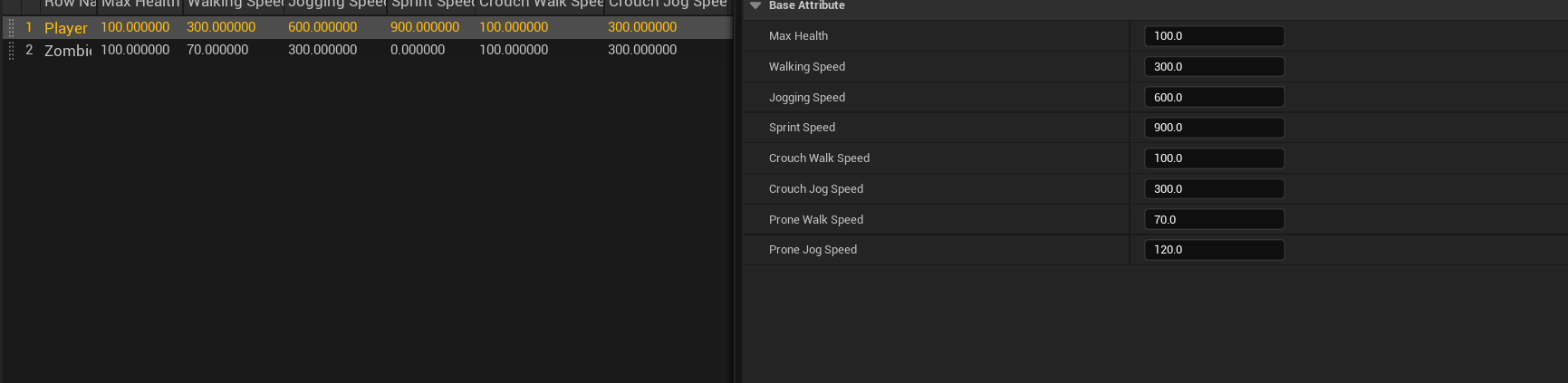
### **2.0.1 Character Enum**



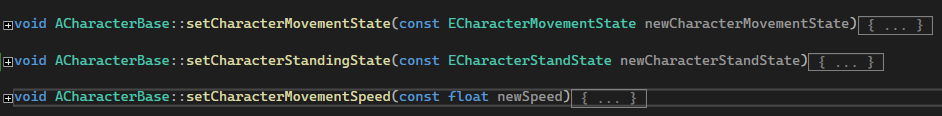
### **2.0.2 Character Attribute**



This struct will be used for Data Asset.

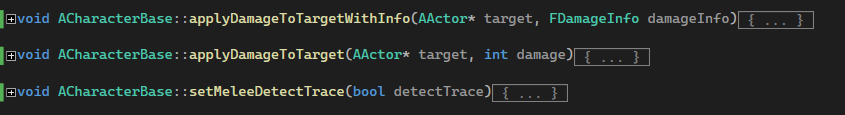


### **2.0.3 Character State and Speed**



Character state will be used to determine the animation and maximum speed for the character. This applies to both zombies and players. The enum state should allow the animation or functionality to change accordingly and be flexible.

### **2.0.4 Attack and Damage System**



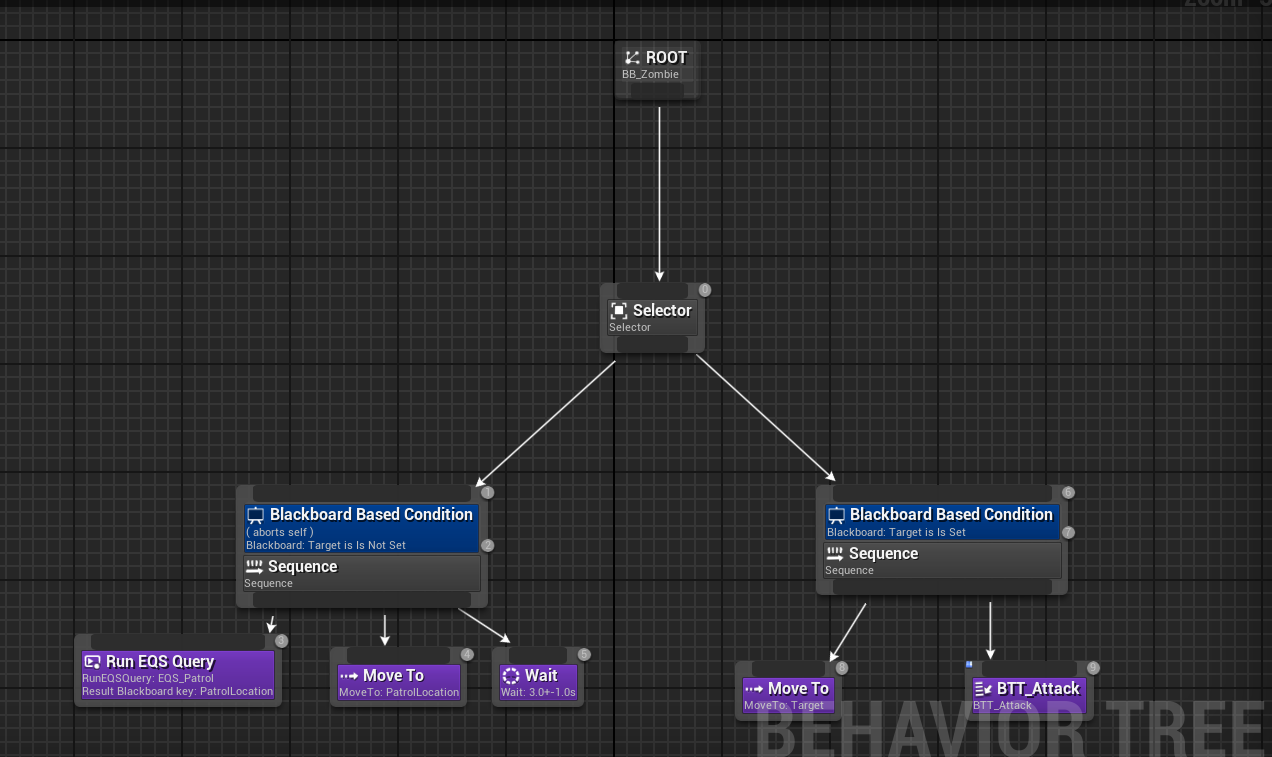
setMeleeDetectTrace was a temporary solution for all characters to trigger the damage collision trace. This function will be used in the AnimNotifyState to allow collision trace detection to trigger within a specific duration in Animation Montage.

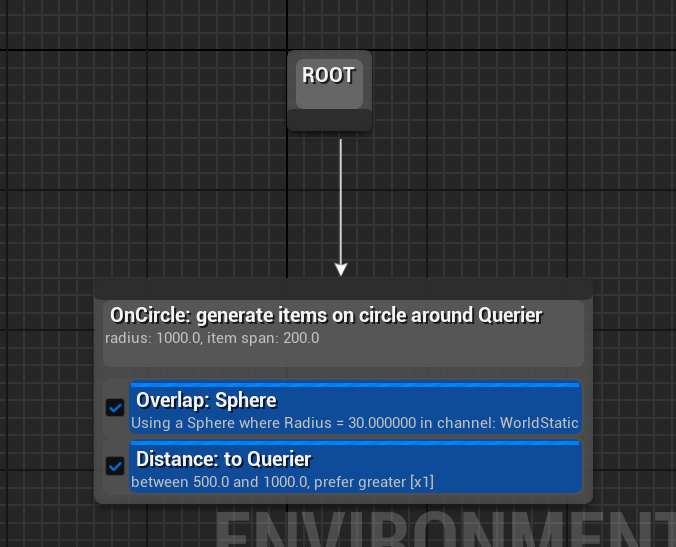
Both applyDamage functionality was part of the damage system but should be a temporary solution as well. Ultimately, these two functions were used to record which enemy target had already applied damage so that it could prevent the same target from being applied damage multiple times.

The best solution from what I imagine with doing this kind of attack behavior could use the Gameplay Ability System in Unreal Engine. GAS can make the system very modular and flexible.

## **2.1 Zombie AI**

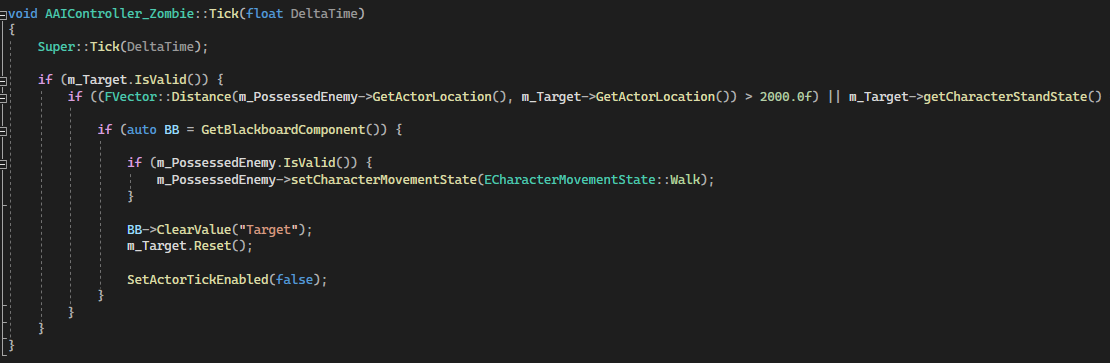
### **2.1.1 Behavior Tree and EQS**



The AI behaviour tree was quite simple. if the target is set, then AI will chase and attack it, otherwise, AI will just roam around.

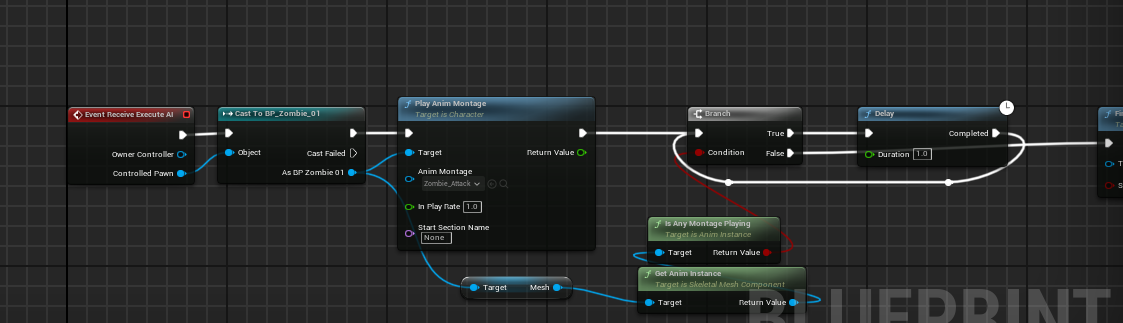
### **2.1.2 Perception and Enemy Target**



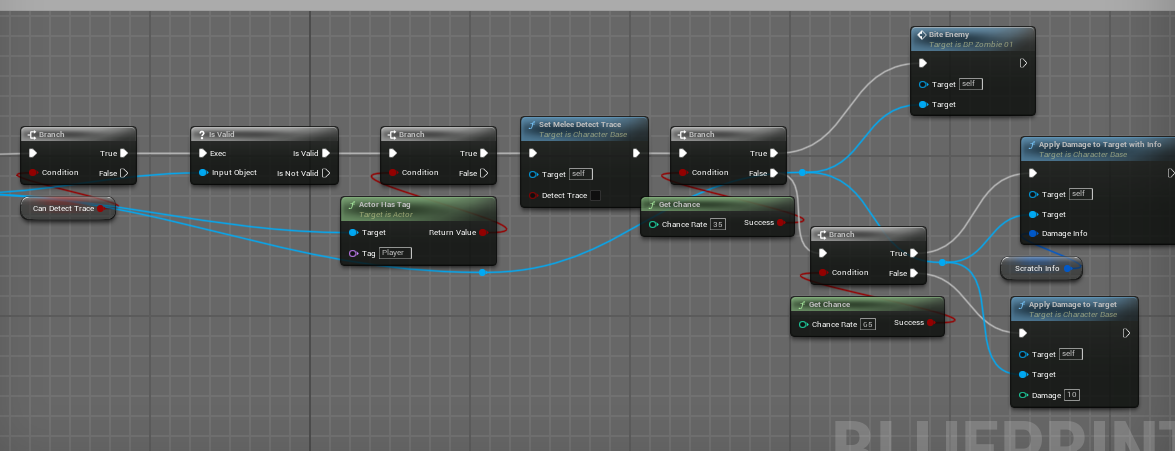


The perception will always be updated if no target is set. Whenever he senses a hostile attitude target, the tick will also enabled in order to clear the target when it is too far away or the target is dead.

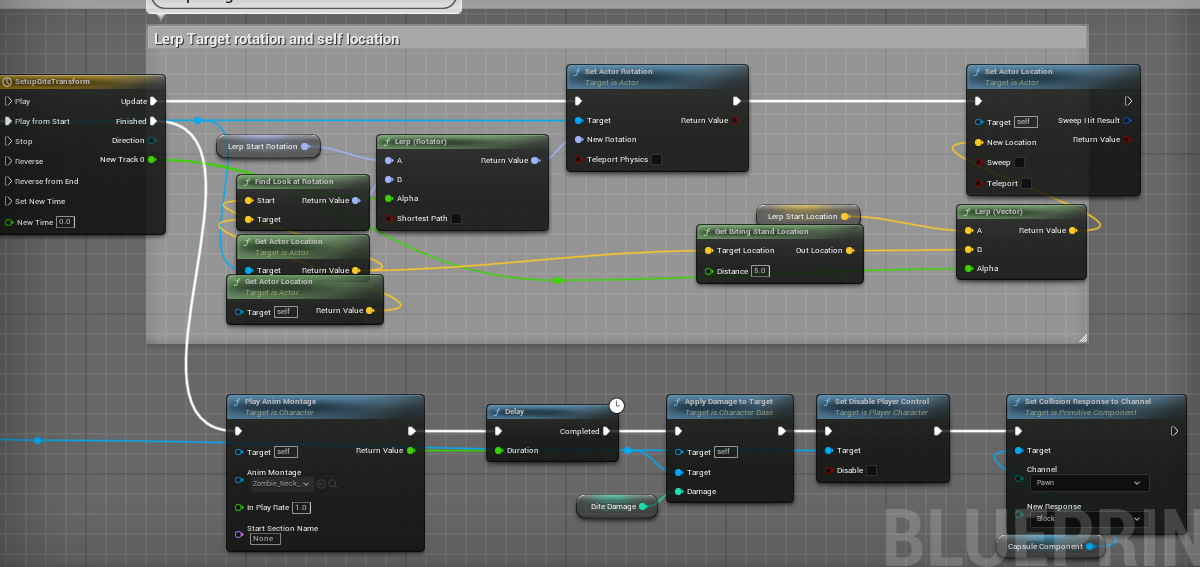
### **2.1.3 Scratch and Bite Target**



Both scratch and bite behaviours will use blueprints for quick prototypes. Although this may not be a good approach in terms of optimisation by doing it on Blueprint. I will discuss a better solution below at the end of this section.



The scratch simply involved playing animation in the AI Task. For the bite behaviour, it will trigger when the damage collision detects the enemy target and passes the 35% chance checking. Same thing with scratching debuff, it needs to pass the 65% chance checking condition in order to scratch the enemy. Otherwise, it will just apply damage to the target only.



For bite behaviour, using the timeline to setup the correct rotation of the enemy target and the desired location of enemy AI will be the current solution. After the timeline is finished, bite montage will trigger and apply damage to the target.

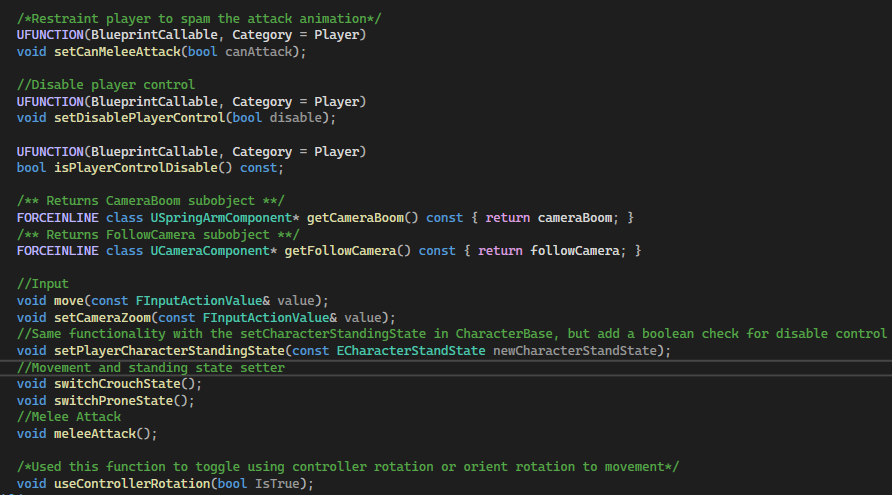
What will be a better solution in the actual project?

GAS will be involved again in this case. Zombie will have their own gameplay ability to define the scratch and bite behaviour. So we don't have to put everything in the zombie blueprint. We can even define the different attack behaviors that can shared among all enemy classes using Enum or GameplayTag. All the gameplay abilities and gameplay tags can be bound together through either Data Table or Data Asset.

## 

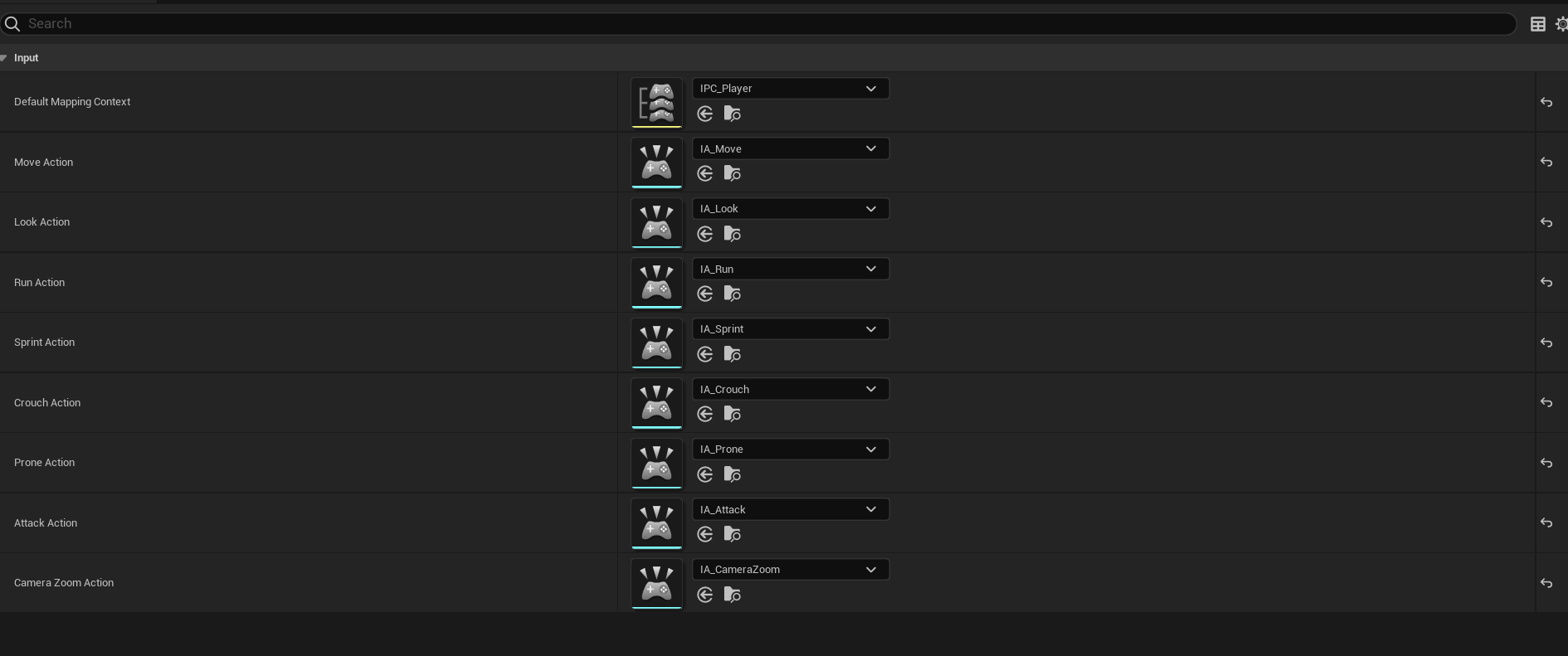
## **2.2 Player**

### **2.2.3 Player Behavior and Input Binding**

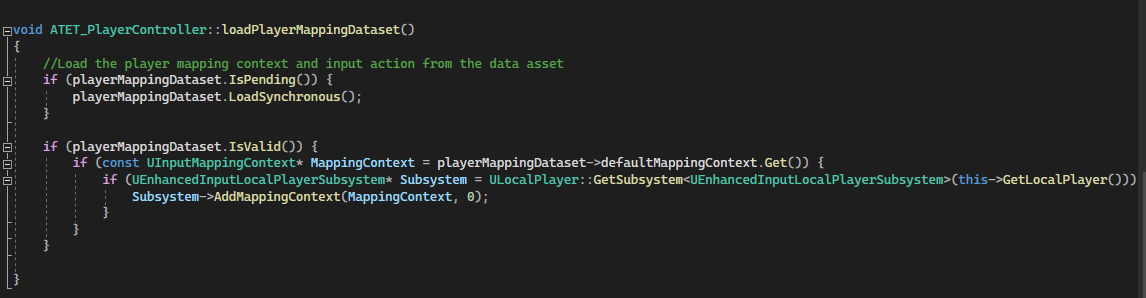




Most of the character functionality will be done in the PlayerCharacter class. The input binding will happen in the PlayerController class instead.



Input mapping data asset

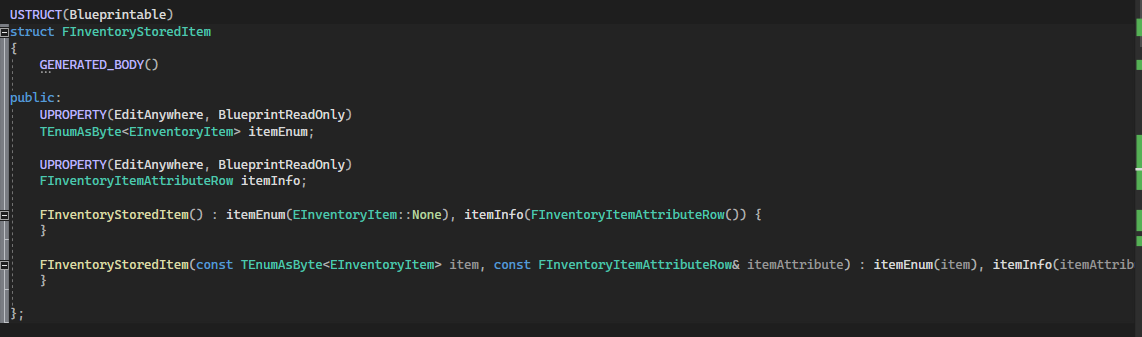


The player mapping data will be loaded using TSoftObjectPtr. After all binding events finish, it will be reset to null status to save memory.

## **3. Inventory System**

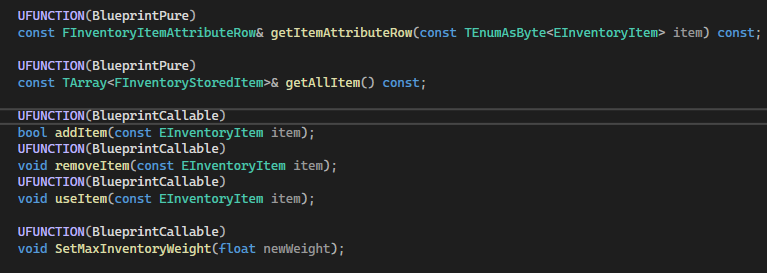
### **3.1 Inventory Enum and Struct**

The image above will be the enum and struct that used



However, the actual data will be stored in this struct instead. FInventoryItemAttributeRow is just the struct for the Data Table only.

### **3.2 Inventory Component**

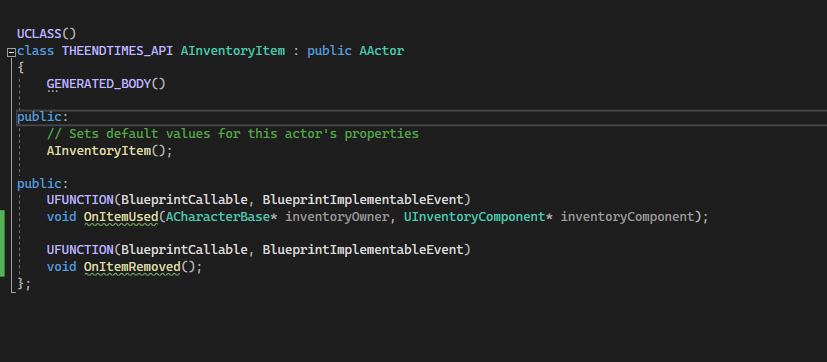


The common function, variable or delegate that can be used from other classes or blueprints.



As for item data storage, it will stored as a value instead of a pointer because the item scale is very small for now. But we can replace it with a smart pointer later when it involved with a large scale of items. Therefore, for the getItemAttributeRow function, it will return a const reference instead of a copy of the value from the array.

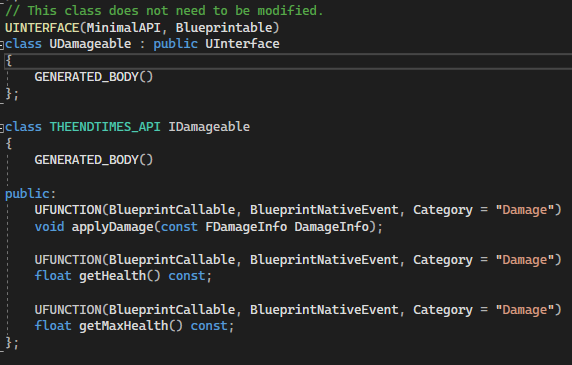
### **3.2 Inventory Item**



For the inventory item, it was using AActor for now in case it needed any mesh attached to the player character and play any used item animation. However, if no mesh is needed to attach for the player, this class can be replaced with UObject Class instead.

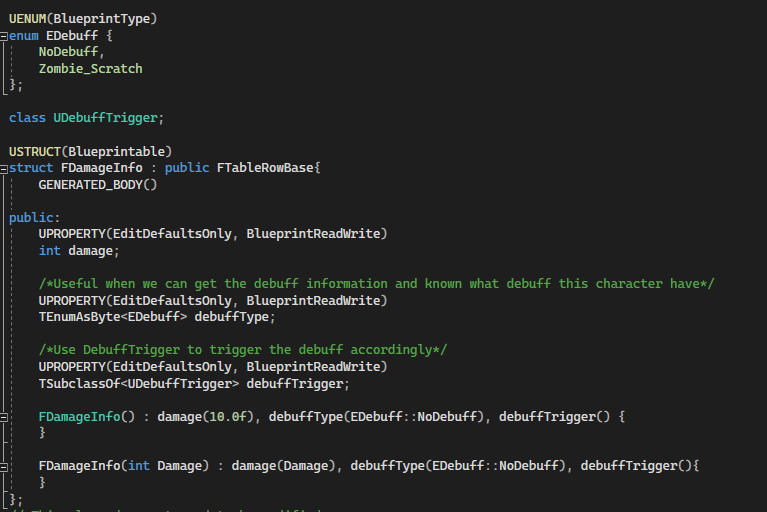
## **4. Damage and Health System**

### **4.1 Damageable Interface**



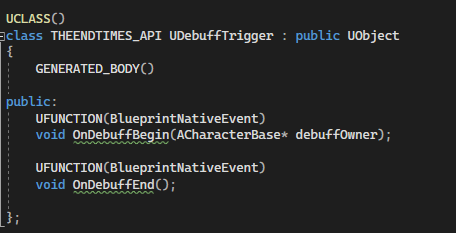
For the damage system, It was using the Interface with related functionality. In the CharacterBase class, a float variable health will be presented. The IDamageble will be implemented in the CharacterBase class as an Interface

### **4.2 Enum and Struct**



The enum and struct will be part of the damage system that applies different types of debuff to the player.

### **4.2 DebuffTrigger**



DebuffTrigger is a UObject class that can be used to trigger the debuff. For now, damage system didn’t have any functionality for this class since this is just a demo. However, in the actual project, the character may have a debuff component or container that stores all DebuffTrigger instances. The DebuffTrigger can be used to execute the debuff that is attached to the player