Damage

# Meta Attributes

So far the gameplay effects that modify Health are reaching into the attribute set and changing the value of Health directly. Typically in RPG games with GAS this is not how it is done, because applying damage to an enemy there are a number of calculations that need to be made, and those calculations have to do with attributes on the attacker and victim. For this reason, to keep the mathematics simple, we often use a placeholder attribute that behaves as an intermediary that allows us to perform all the necessary maths before setting the value of the health, in the case of damage

These intermediary values are called **Meta Attributes**

Normal attributes:

* Often Replicated
* Changed on server and replicated to clients

Meta attributes:

* Temporary
* Not replicated
* Only used on server
* Allows for calculations

Damage:

* Meta Attribute would be *Incoming Damage*

How Incoming Damage works:

A GE causes 11 damage to a target. Instead of subtracting 11 health, the GE instead adds 11 to Incoming Damage

There are ways t respond to attribute changes in the attribute set eg PostGameplayEffectExecute is used to check if the attribute that has changed is the Incoming Damage Meta Attribute. If so, we then calculate if there was a Block, a Crit, is there an Intelligence/Strength Bonus. Whatever, we perform those calculations to determine the final real amount of damage actually inflicted and then subtract that number from Health

A screenshot of a computer

Description automatically generated

The Gameplay Effect is not the arbiter of Health subtracted from the attribute, it’s the rules set by those equations that determines the final value

The Incoming Damage is set by the GE, and the Attribute Set performs the calculations it needs to and then zero out the Incoming Damage Meta, Set the Value of Health and then respond in any other ways needed eg floating text (which may be a different colour depending on bock or crit), apply debuff effects, damage over time, stuns, hit react anim

Meta attributes are therefore often the way that GAS projects handle damage

# Damage Meta Attribute

In GE\_Damage:

A black and white striped background

Description automatically generated

This is an Instant Gameplay Effect that adds a negative value directly to Health. This is really a placeholder Gameplay Effect – it hardcodes a Negative 10, doesn’t use a curve table

Instead we’ll modify a meta attribute, then allow the Ability Set to process the data. We’ll also use the damage meta in the custom calculations as a custom calculation class as Damage will be one of the more complicated types of modifiers

We’ll add the Meta to the Attribute Set

A screen shot of a computer program

Description automatically generated

Easy enough. Note that the UPROPERTY includes a category and BP Readonly, like the other attributes, but there is NO ReplicatedUsing, because this is not a replicated attribute

So how do we actually **use** the meta attribute?

In the CPP:

A computer screen with text

Description automatically generated

Here we can see it the attribute data has the attribute IncomingDamage, in exactly the same way we check for GetHealthAttribute and GetManaAttribute

A screen shot of a computer program

Description automatically generated

So, inside the if check we know we are getting a change to the IncomingDamage Meta Attribute

As IncomingDamage is a Meta, it should be used for it’s value and then reset to zero – we will consume that data!

To do this we will make a local float variable that has the value IncomingDamage

A black screen with white text

Description automatically generated

So we’re caching that value locally and then we can set that meta attribute to zero

A computer screen with white text

Description automatically generated

IncomingDamage is now ready to take a new value, because we have received the data into this function

Now we have the Local incoming damage we can decide what to do with it, and we should only do something with it if its value is greater than zero

So far we have nothing to do with the Meta Attribute other than set the health value, so we can do that for now and handle any additional clamping

A screen shot of a computer program

Description automatically generated

At this point, we can tell certain things about the damage that has been done; eg if NewHealth is 0 then we know that the damage done is fatal. So we can do something like this:

A screen shot of a computer

Description automatically generated

If NewHealth is equal to or less than 0 we know that bFatal will be true - enough damage was done to kill the owner of this attribute set

So this is basically how it should work – we use the meta instead of changing health and the gameplay effects that currently affect health will be changed to affect the Meta on the target, and they will be adding to it, not subtracting, because the subtraction is done here in NewHealth – LocalIncomingDamage

In GE\_Damage:

A screenshot of a computer

Description automatically generated

The Attribute is now set to Incoming Damage and we’ll increase the damage a bit for testing

A video game with a firework

Description automatically generated with medium confidence

The bar is going down; we’re doing damage with a meta attribute, which allows us to do all kinds of things; we can check the damage done, perform custom calculations

There’s no hit react or death at zero, but we’ll worry about that later

GE looks better but we’re still hard-coding 25 for the damage – would be better if this were linked to the Gameplay Ability somehow, so if the Gameplay Ability has it’s own damage variable, how do we make sure that the Gameplay Effect is using the Damage from the Gameplay Ability?

That would be the 4th Magnitude type:

A screen shot of a message

Description automatically generated

# Set By Caller Magnitude

A Gameplay Ability may have a base damage value that we’d like to set on the Gameplay Effect that the ability is causing

Currently the Effect Spec when launching a projectile is handled like this:



We are creating a Gameplay Effect Spec with MakeOutgoingSpec and then taking the SpecHandle and setting that on the projectile – the projectile has a Damage Effect Spec Handle variable that we are setting

So we’re setting the Damage Gameplay Effect for the projectile spell, but as soon as we set the Spec Handle for the Projectile – which will apply it later – that Spec Handle should carry the information for how much damage to cause. We should be able to set that here from our ability, if the ability wants to have some value – the ability itself should determine how much damage to do, not the effect

To do this we can use the Set By Caller option on the Gameplay Effect

Set By Caller Magnitudes are Key/Value pairs; where the key is a gameplay tag, so we need a gameplay tag to identify the Set By Caller magnitude

First we’ll make a Gameplay Tag for Damage in the AuraGameplayTags singleton

A screen shot of a computer

Description automatically generatedA black screen with blue and green text

Description automatically generated

Now we have a native Gameplay Tag for Damage and we can use it in code

In AuraProjectileSpell.cpp:



We’ll Get the gameplay tags right before applying the Spec Handle:

A screen shot of a computer

Description automatically generated

And now we can use a Set By Caller magnitude

This is done using the AbilitySystemBlueprintLibrary

A screenshot of a computer

Description automatically generated

Assign Tag set By Caller Magnitude requires a Gameplay Effect Spec Handle, a Tag and a Magnitude

We’ll use SpecHandle for this.

Next it requires a Gameplay Tag called DataTag – this is how we identify the magnitude value, because you can assign multiple SetByCaller magnitudes to a gameplay effect spec.

We’ll set a Key/Value pair, where the Key is the Damage tag

And finally the magnitude is whatever we set here; the projectile spell could have it’s own Damage variable.

For the moment, as a proof of concept, we’ll hardcode 50



So now the spec handle is carrying along a key/value pair – the Damage gameplay tag and the value 50

Now we just need to access that key/value pair from within the Gameplay Effect when applying a modifier

In GE\_Damage:

A black and white background

Description automatically generated with medium confidence

Once Set By Caller is applied there’s a new dropdown. Because we can have multiple Set By Caller magnitudes on a gameplay effect spec we need to choose which one, and they are identified by the gameplay tag key

A black and white screen

Description automatically generated

As soon as we pick this tag, this modifier will do 50 damage, because that’s the value we associated with this key when we called AssignTagSetByCallerMagnitude

A screenshot of a video game

Description automatically generated

So the damage is now controlled by the Gameplay Ability, not the Gameplay Effect (even if we’re still hardcoding for the moment). We can set a base damage property based on the Gameplay Ability’s level. So we can add a member variable to the AuraGameplayAbility class to handle damage

# Ability Damage

We’re currently hardcoding 50.f for the set by caller magnitude in the effect spec handle set on the projectile

Aura projectile spell is going to control the damage and rather than hardcode this we can have a variable. We know that abilities have the concept of level, so if the ability level increases maybe we want the damage to increase as well, and we’ve seen a good way to handle increasing values based on levels – curve tables

We’ve also seen that when we select a curve table in a gameplay effect, for example

A screenshot of a computer

Description automatically generated

Here we’re using a scalable float magnitude where we have selected a curve table (CT\_Potion), choosing a curve from that table (HealingCurve) we can enter a value and the scalable float magnitude will be scaled by the value in that curve based on a level

We can do something like this in our gameplay ability – the scalable float mag is an FScalableFloat, and when you have a member variable of this type you get these properties – Ability to have magnitude, a curve table dropdown and curve and a preview slider

It would be good to have one of these useful properties in the Gameplay Ability class

It’s true that we could have abilities that do not do damage, and we could subclass AuraGameplayAbility to have an AuraDamageAbility from which all damaging abilities inherit, but for simplicity we’ll just give this to AuraGameplayAbility and then any abilities which do not do damage we simply won’t use that property – the memory it takes up is negligible

A computer screen shot of a program

Description automatically generated

We’re adding the property here for damage for the ability, which will be a scalable float and we can create curve tables and set them per ability

A screen shot of a computer

Description automatically generated

In editor GA\_FireBolt now has an FScalableFloat  
  
A screenshot of a computer

Description automatically generated

Entering a value here will scale by the curve table selected. We could enter a base value of 10 and then scale it, or set it to 1 and then set the curve for values for levels

Now we can set a damage curve table to scale Firebolt, and create different curve tables for different abilities so they scale differently when level increases!

A black and white text

Description automatically generated

A screenshot of a computer

Description automatically generated A graph with a line going up

Description automatically generated

A screenshot of a computer

Description automatically generated

In AuraProjectileSpell.cpp:



We will replace the 50.f with a query for the value at level so:

A screenshot of a game

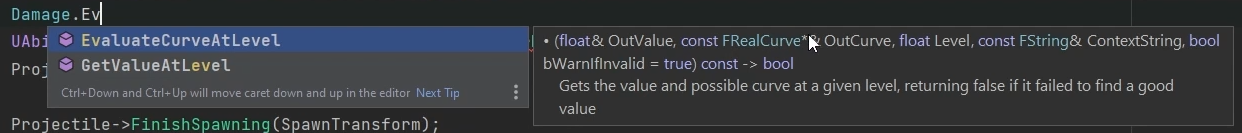
Description automatically generated

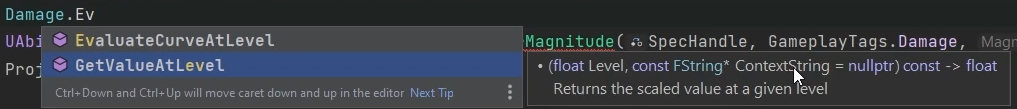
This returns an Int32

For an Integer we could use:



Or possibly:



This gets the value and possible curve; We would need to pass in an FRealCurve pointer by reference; or better for this:  




This \*does\* return an int32 instead of a float, but it performs an implicit conversion so we can just pass in this expression



However it’s better if we make a const float and pass that in



For visibility will add a screen message at this point:

A blue sky with red text

Description automatically generated

Nb We see the damage printed immediately, before it hits, because we’re printing it in the DamageEffectSpec

If we change the value 

We get more damage

A blue sky with red text

Description automatically generated

# Enemy Hit React

So far applying damage only affects the Health Bars, but an enemy reaction would be nice

Gameplay Abilities can be used for this too, activating an ability when non-fatal damage is received and play a hit react montage specific to the enemy, and also prevent other things from happening while the enemy is hit-reacting, like using attacks or movement.

Essentially we want the enemy to be immobilised, and while that is the case we can give the enemy a gameplay tag

In: A screenshot of a video game

Description automatically generated

There’s a hitreactspear anim for the spear goblin, and while this animation is playing, we don’t want the enemy to move or attack or do anything else, because it would look weird

While that lack of action is happening, we can give the enemy a gameplay tag – gameplay effects are a good way to apply tags to enemies, so let’s make a gameplay effect that will give this enemy a tag!

And if we decide that we want that effect to modify attributes or do anything else we can add more functionality in the future

For now, basic Gameplay effect to applya tag, and we add native tags to the singleton to use them in C++

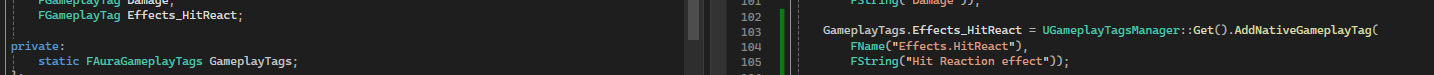
We want to use in C++ so we can respond with callbacks to delegate broadcast by the ability system when an ASC is granted a tag

To do:

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated  A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

The Gameplay Effect will now grant the tag

Now we can add to the ASC a response to getting the tag granted to it, we can do this in C++

In AuraEnemy class:

What we’ll do here is listen for a gameplay tag change on the ASC

In BeginPlay we have access to the ASC, and we will bind a callback/lambda to a delegate broadcast from ASC when a tag is added

A screenshot of a computer program

Description automatically generated



This function RegisterGameplayTagEvent() allows events to be registered for specific gameplay tag events, like adding or removing new tags. So we can specify a tag and specify what happens in response - we can have a callback for it

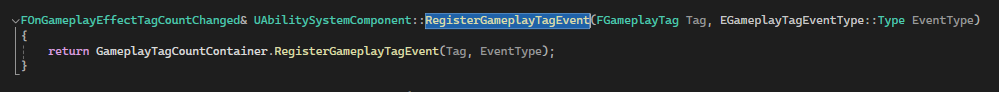
This function returns a delegate of type OnGameplayEffectCountChanged 

So we can see what the signature needs to be for a callback that can be bound to this

Let’s look at the definition

A screenshot of a computer

Description automatically generated



We can see the type (FOnGameplayEffectTagCountChanged) and go to *it’s* declaration



And here we have the required signature. It’s a 2 param multicast delegate, that takes a const gameplay tag and an int32 – this is actually for the new tag count – we can have tag counts and multiple tags of the same kind

In Public:



A screen shot of a computer

Description automatically generated

The Callback has been created, so we need to bind it in Beginplay, and for that we take the tag:

The GameplayTags are currently not included in Enemy.cpp so… 

With the include, we can use  to get the HitReact tag, and therefore pass it directly into the function



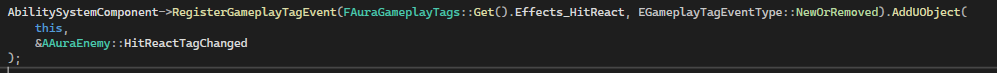
The next argument is an Enum for Event Type:

A screenshot of a computer

Description automatically generated

We can respond either to any count change, or to complete add/remove; we’ll use the second

This expression will return the delegate that we can bind the callback to, using AddUObject:



So now the callback is bound and we can add functionality to it

First, let’s check the count. If the count > 0 we should hit react; if it is not > - we should not be hitreacting



Knowing if we are hitreacting is important; we may need to know about it in Blueprint, enemies will get behavior trees and there might be BP circumstances we need to know eg Anim BPs, so we’ll need a member variable, rather than a local like here, and expose it with a UPropertyA screen shot of a computer error

Description automatically generated A black screen with blue and green text

Description automatically generated

We can do pretty much whatever we like now, like set the MaxWalkSpeed of the Movement Component to zero, for example

A black screen with text

Description automatically generated

In this function the bool will now set the speed to 0 if hitreacting is true ie if NewCount is greater than 0, and if false set it to the new variable MaxWalkSpeed, which is a float that will default to 250.f and which will be set in BeginPlay

A black background with white text

Description automatically generated

Loading up the Editor we will now create a new Gameplay Ability, apply the hitreact effect and then play a montage and remove the effect when the montage is complete

A black screen with white text

Description automatically generated

Creating a new Gameplay Ability, we can apply the gameplay effect to owner easily, and promote the output, a GE Effect Handle, to a reference variable:

A screenshot of a computer

Description automatically generated

Next we will Play Montage and Wait, but the montage will be a variable, so we can use the same hitreact BP for multiple Enemies that will have different animations

A screenshot of a computer

Description automatically generated

We’ll get the info on the actor from the Actor Info, and then cast to the CombatInterface!

In the CombatInterface code we will add a new function that returns a pointer:

A black background with white text

Description automatically generated

(needs to be forward declared)

A black background with white text

Description automatically generated A screenshot of a computer

Description automatically generated

Note the UFUNCTION is Blueprint Native Event, we want it to be callable in BP, but if we make it BP Callable we cannot override in C++ This function does not have to be marked Virtual, it will automatically generate a virtual native version that exists in C++ that we can override, and that will be the version with \_Implementation at the end

In AuraCharacterBase:

A screen shot of a computer

Description automatically generated

This returns an Anim Montage, so HitReact will now have a Montage variable, and in private we’ll add:

A screen shot of a computer program

Description automatically generated

For the Implementation:

A black screen with white text

Description automatically generated

We can now call Get Hit React Montage:

A screenshot of a computer screen

Description automatically generatedSo, in BP all we are really doing is applying a GE Effect, storing its handle, then taking the avatar actor and casting to a combat interface, and calling the montage. If it’s a valid montage, we should see it played! So we need to set it in the Enemy bp

A screenshot of a computer

Description automatically generated

Currently the Ability doesn’t react, but that’s the next task!

# Activating the Enemy Hit React Ability

* Currently the Gameplay Ability will:
* Apply the Gameplay Effect
* Store the Handle for that Effect
* Get the Avatar Actor from ActorInfo and cast to the Combat Interface
* Use the GetHitReactMontage() BP Native Event to return a montage
* Play Montage and Wait

We have set the montage for Spear and Slingshot but the ability does not yet activate – we want to do this when enemies are damaged, but not dead

Let’s look at the Attributes set!

In the attribute set we can see Incoming Damage

A computer screen shot of a program code

Description automatically generated

We can even see if it’s fatal damage

A screen shot of a computer

Description automatically generated

If the bool bFatal is not true (hits are 0 or lower) then we should have the enemy activate it’s hitreact ability

That means the enemy should HAVE that ability as well!

We need to:

* Grant the hitreact ability to the enemy
* Activate the ability from within the attribute set

Making sure the enemy gets the ability:

We have a data asset for ‘things all enemies should share’ – the C++ class CharacterClassInfo

All enemies may want to share gameplay abilities so we can have a TArray of Gameplay Ability

classes that we would like to apply

So we’ll add that to the CharacterClassInfo, so that we can then add that Gameplay Ability to the

Enemy class

The TArray will be of type TSubclass of type UGameplay Ability

A black background with white text

Description automatically generated

It’ll be common to all classes so we’ll put it in that category and now we have created this, we can set this in the data asset and make sure all enemies have these abilities given to them!

How do we grant abilities to the protagonist, Aura?

In AuraCharacterBase there’s an AddCharacterAbilities function

A computer screen with text

Description automatically generated

In this function we get the ASC for Aura and call AddCharacter Abilities, and the add abilities takes in a TArray of starting abilities

Here’s the definition for that:

A screen shot of a computer program

Description automatically generated

It loops through the TArray, creates an ability spec and makes it level 1, then starts the startup input tag, and adds that tag to the DynamicAbilityTags before calling GiveAbiilty()

This is all specific to the player controlled character, enemies don’t need to care about input tag, all

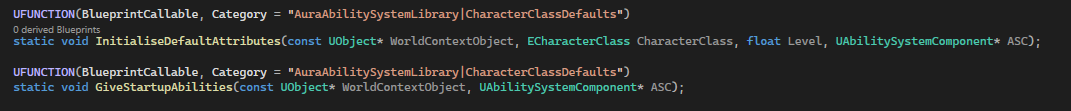
we care about is the AbilitySystemComponent calling GiveAbility

But the abilities are going to be on the data asset, which exists on the game mode, and the startup

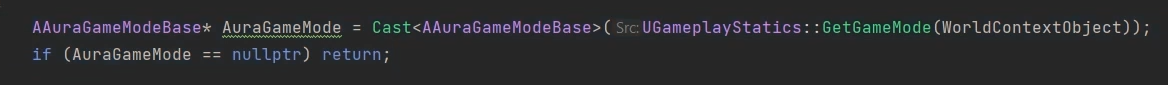
attribute effects are given using the Ability system Blueprint Library AuraAbilitySystemLibrary using the static function InitialiseDefaultAttributes

We could do this for the enemy CommonAbilities as well; all it would need is the world context object and the ability system component, no level or class needed, and it can get the character class info from the game mode

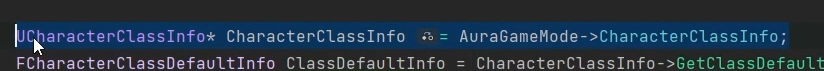
So we’ll make a new static function in the library to initialise enemy gameplay abilities



The functionality will be fairly simple- we get the Game Mode like in InitialiseDefaultAttributes, passing In the context object and return if game mode mode is null;



Once we have that, we can get the character class info from it



and from that the array of abilities,

A screen shot of a computer program

Description automatically generated

loop through it with a for loop:

A screenshot of a computer program

Description automatically generated

Nb this is a TSubclassOf GameplayAbiilty so we can make things clearer by changing the auto)

A screenshot of a computer program

Description automatically generated

and use GiveAbility()

A screenshot of a computer

Description automatically generated

This requires an abilityspec, but we can get that by going back to AuraAbilitySystemComponent and seeing what we did there

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generatedWe can just copy that line!

End result:

A screen shot of a computer code

Description automatically generated

So, now we have a function we can call to give abilities to the enemy, and we can call it in enemy.cpp, in Beginplay, as soon as we know the ASC has been initialised!

A screen shot of a computer

Description automatically generated

So we scope it to the library, call it, pass in this as the context object and pass in the ASC, and it should get those abilities, provided that those abilities are in the data asset

So, we need to remember to add the hitreact ability to the common abilities array in the data asset, but once that’s done, this function should give those abilities

Now to actually activate it:

In AuraAttributeSet::PostGameplayEffectExecute, as soon as we know the damage is not fatal:

A black screen with white text

Description automatically generated

We can use an if !bFatal to activate, but we want to avoid making the attribute set dependant on the enemy class, so we want it to be nice and polymorphic

We can activate abilities by ability tag

A screenshot of a computer program

Description automatically generated

We’ll use the props and target asc and call TryActivate by tag. This takes in a Tag container with an optional bool

This remote activation bool means we can call it from a client and activate it predictively. We’re doing it on the server so it doesn’t matter, but we need a [tag container](5%20Gameplay%20Tags.docx)

We can just create one here and populate it from the singleton:  
A black background with white text

Description automatically generated

“Activate an ability if you have one that has this tag”

We know the enemy will have an ability that has this tag so time to compile and run!

Apart from failing to forward declare the UGameplayAbility class in CharacterClassInfo.h, no problem!

To do:

Data Asset needs the HitReact ability added to it, so in the DA\_CharacterClassInfo there’s a new field called CommonAbilities, and we can add GA\_HitReact

Once this is in place, we can add damage in the attribute set, and that damage if it is not fatal, is going to to activate TryActivateAbilitiesByTag, which should activate the HitReact Ability, if it has that tag!

Does it have that tag?

No.

A screenshot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

Nb: fixed an error in AuraProjectileSpell where a line was commented out breaking health bars; it now reads UAbilitySystemBlueprintLibrary::AssignTagSetByCallerMagnitude(SpecHandle, GameplayTags.Damage, ScaledDamage);

GA\_HitREact set to instanced per actor so each enemy is treated separatelyA screenshot of a computer

Description automatically generated

Also:

A screenshot of a computer

Description automatically generated

Using the cached tag Active GE HitReact, we remove and end the ability. The same is applied if the animation ends, is cancelled or is interrupted

A screenshot of a computer

Description automatically generated

# Enemy Death

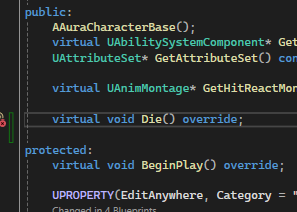
We now have a hit react Gameplay ability. Time to implement Death, defined as when the enemy has zero hit points. In other words, in AuraAttributeSet, when bFatal is true, we call a function for the character to die

Death is something both Aura and enemies should share, so we should add a function to the Combat interface

We’ll make it a virtual void pure



so that it has to be implemented and we’ll first implement in AuraCharacterBase



Goals to do in Die():

* Drop the weapon
* Ragdoll the enemy
* (we’re going to ragdoll to dodge having to play a montage)

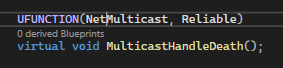
We \*could\* create a death gameplay ability and play a montage like with HitReact, but let’s do this differently as a new learning opportunity

To ragdoll the enemy we need to do stuff on the server and the client, so it needs to be replicatable, so we’ll need a multicast RPC

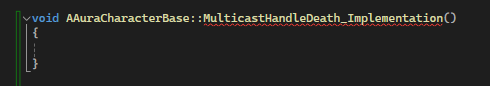


So we can override in child classes if we want to do specifically, but this is what we’ll use to handle things on all clients when a character dies

It’s a multicast so we’ll add that to the UFUNCTION and specify reliable, because death should be reliable



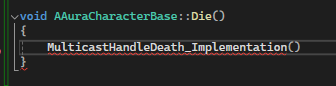
Since it’s an RPC Implementation is required



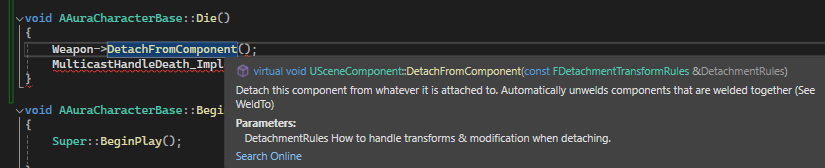
We’ll handle what happens on all machines here

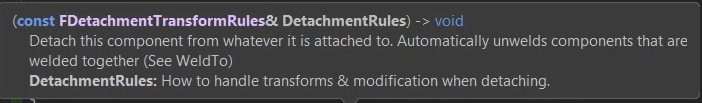
The Die function can handle what happens specifically on the server and we’ll only call it there

The Die function will call MulticastHandleDeath, so we’ll put it in the override implementation

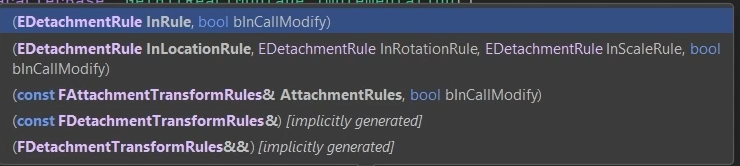


And before we call it we’ll drop the weapon





This required FDetachment Transform rules, which is a struct with a contructor that requires:

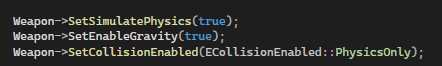


* EDetachmentRule
* KeepWorld is good for this
* A bool that we’ll set to true

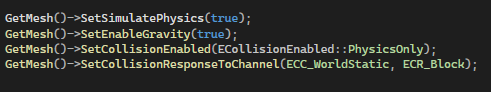


This will result in detaching the weapon, which will automatically be a replicated action

We’ll add some effects for the weapon drop:

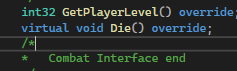


And ragdoll the character mesh and disable the collision capsule

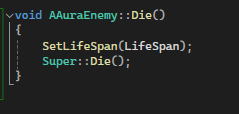




Now for the Enemy class:



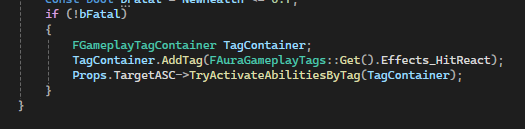
We’ll include the Super::Die because there’s good functionality there, and add a lifespan parameter (defaulted to 5 sec, same as WalkSpeed



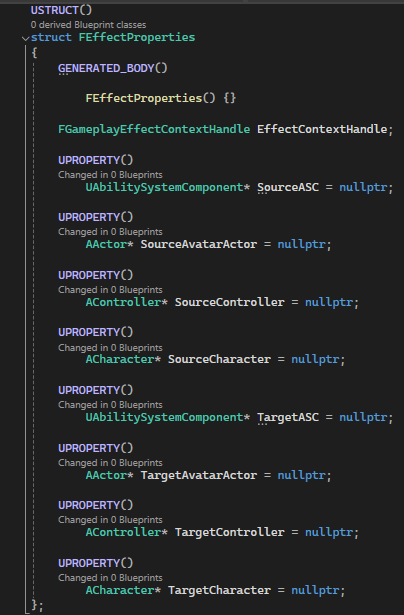


We now have a usable interface function, which we will use in the attribute set

Here’s the fatal damage check

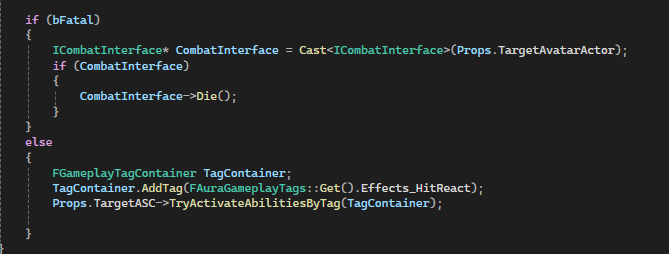


And here’s the props struct from the Attribute set header



We didn’t add a pointer to the combat interface, but that’s OK, we can call the function if the target actor implements the interface

We’ll call it with if bFatal, and then add an else case and put all the things currently in !bFatal there



In the event of fatal damage, the code will get for and cast to the target’s combat interface and call Die; if not fatal it’ll play the hitreact montage as before!