Gameplay Tags

* Essential in GAS
* Exist Outside of GAS and can be used in non-GAS projects

# What are Gameplay Tags?

* Basically, names
* Type FGameplayTag
* Registered with the Gameplay Tag Manager
* Hierarchical
* Parent.Child.Grandchild
* Dot-separated level of hierarchy
* Can have more or less than 3 levels
* Versatile
* FNames at core, include an FName at core
* Can be compared for exact or partial equality
* Maybe root tags match?
* GAS designed to use tags in pretty much every single class
* Tags are given to actors
* More specifically to their ASC
* ASC now has that tag
* ASC implements interface: IGameplayTagAssetInterface
* Interface has functions to look at owned tags
* See if particular tag is owned
* Compare tags
* See if any tags match a specific tag
* All interface functions so all classes can implement
* Interface handles these in it’s own way
* Classes can have multiple tags so need to store inside a container
* Not TArrays though!
* “Gameplay Tag Container”
* Has some Gameplay Tag specific functionality and added efficiency
* “Tag Map Count”
* More than one instance of a single tag in container
* Map count tells us how many instances of a particular tag exist in container
* Could be zero depending on if added and removed
* Debug mode might see given tag in container even if map count zero

A screenshot of a game

Description automatically generated

Gameplay Effects might have tags that they grant to the ASC they are being applied to

Eg:

* Actor applies a GE to a given ASC
* For this example it’s duration based effect
* GE has a tag it grants to the TargetASC on application
* Now ASC has that tag
* GE Duration expires – it removes the tag as well

A screen shot of a video game

Description automatically generated

Many operations depend on Tags

Eg:

* ASC has some Gameplay Ability
* Gameplay Ability may have a ‘Blocked’ Tag
* If ASC has that tag and tries to active it cannot do so because blocked
* Or maybe ability has a list of \*required\* tags
* ASC must have these tags to activate and cannot if does not meet requirements
* Identify Inputs
* Abilities
* Attributes
* Damage types
* Buffs
* Debuffs
* Messages
* Data
* Anything else

A screenshot of a video game

Description automatically generated

## Creating Tags in the editor

Tags are Project wide

Options:

Project Settings -Gameplay Tags

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Hmm, so tags are associated with config files somehow?

A screenshot of a computer

Description automatically generated

Here’s how to add them in the editor though

A screenshot of a black screen

Description automatically generated

Example tag hierarchy

A screen shot of a computer

Description automatically generatedTag added!

A black rectangular object with a grey line

Description automatically generated

So, easy to do in the editor but the tags exist somewhere!

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Editing the ini will reflect in the editor after a reload

A black text on a white background

Description automatically generated A screenshot of a computer

Description automatically generated

That’s how to add from within the editor and it’s fine to do it but it’s not the ONLY way to do it

## Adding GameplayTags from Data Tables

New – Miscellaneous – Data Table

A screenshot of a computer program

Description automatically generated

Adding a new row:

A screenshot of a computer

Description automatically generated

Tag and comment:

A screenshot of a computer

Description automatically generated

Populated Data Table:

A screenshot of a computer

Description automatically generated

Back in Project Settings:

A screenshot of a black screen

Description automatically generated

It’s an Array of data tables

A screenshot of a computer

Description automatically generated

Elements need to have the GameplayTagTableRow structure

Before:

A screenshot of a computer

Description automatically generated

After:

A screenshot of a computer

Description automatically generated

There are other options beyond this method, but this is convenient because it’s easy to find them all in one place and make changes

A screenshot of a computer

Description automatically generated

# Apply Gameplay Tags with Effects

In GE CrystalHeal:

A screenshot of a computer

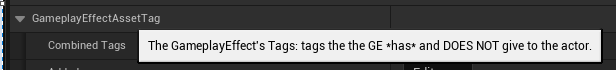
Description automatically generated

Lots of tags, they all do different things. Makes it versatile but confusing!

## Types of tags:

Gameplay Effect Asset Tag

List of tags the Gameplay Effect has and doesn’t give to the actor



A black and grey background

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated

We can put some tags here and when the effect is applied I won’t pass them on to the ASC

Combined Tags: “Tags that I inherited and tags that I added, minus tags that I removed”

This is what we’ll see in all tag containers; makes more sense in non-GE classes because GE classes generally do not inherit!

Added: “Tags that I have, in addition to my parent’s tags”

Removed: “Tags that should be removed if my parent had them”

* Gameplay Effect Asset Tags
* “We can put some tags here and when the effect is applied I won’t pass them on to the ASC”
* Granted Tags
* “These Tags are applied to the actor I am applied to”
* Granted Blocked Ability Tags
* Ongoing Tag Requirements
* Application Tag Requirements
* Removal Tag Requirements
* Remove Gameplay Effect Query
* Remove Gameplay Effects with Tags

## Adding a tag:

This example is a Granted Tag

A screenshot of a computer

Description automatically generated

Upon Compile:

aA screenshot of a computer

Description automatically generated

In-game:

A screenshot of a computer

Description automatically generated

This vanishes upon expiration of the effect

Note the parentheses: tags can stack

Difference: GE\_CrystalHeal has an aggregate stack limit of one tracked by target, so we can only stack one. But if we change the stack limit to 4, the tags still only stack once even if you pick up multiple crystals

The effect stacks, not the tag

However, if we remove stacking from the GE:

A screenshot of a computer

Description automatically generated

Now we have 3 separate tags applied because the non-stacking crystals are separate gameplay effects

Any stacking on effects means tags will not stack – a tag is added when a duration effect is added so we can respond to them!

## What about instants?

In GE\_PotionMana:

A black and grey squares

Description automatically generated

Nothing is added!

Granted tags only mean something when the effect has a duration!

But what if we want to get a tag ANYWAY?! For example, if we want to display a message about how we just picked up a health potion?

That’s where the Gameplay Effect Asset Tag container comes in

Lets look at the ASC.h



Called on Server whenever a GE is applied to self. Based on replication policy we should know whether or not GEs will be replicated

Reminder:

A screen shot of a black background

Description automatically generated

Mixed: Gameplay effects are replicated to the owning client only; cues and tags replicated to all clients

Minimal: GE not replicated at all; cues and tags only

/\*\* Called on server whenever a GE is applied to self. This includes instant and duration based GEs. \*/

FOnGameplayEffectAppliedDelegate OnGameplayEffectAppliedDelegateToSelf;

Let’s see what we can do with this one!

We need to know what the function signature should be for a callback

/\*\* Delegate for when an effect is applied \*/

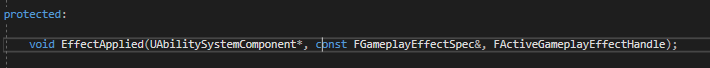
DECLARE\_MULTICAST\_DELEGATE\_ThreeParams(FOnGameplayEffectAppliedDelegate, UAbilitySystemComponent\*, const FGameplayEffectSpec&, FActiveGameplayEffectHandle);

Declaration for the FOnGameplayEffectAppliedDelegate shows it takes in a GAS pointer, a gameplay spec and an effect handle. So when it broadcasts this delegate it’s going to pass in this info and we can receive it

Often in Declare Multicast Delegate the macros sometimes have param names, but you can just have types without names

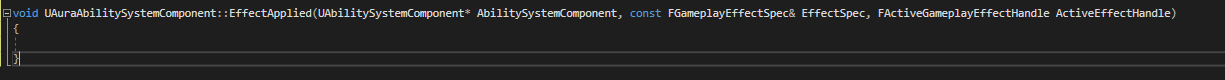
Callback needs these 3 input params

So far AuraAbilitySystemComponent.h has no content, so we’ll add a function to it



Parameter names here are optional, but it’s helpful to have them be descriptive





Now we bind to delegate:



Where to bind? Well, we typically do this when the actual game starts, but not in the constructors as these are fire off early, so we want a place to bind to this delegate early on – an easy way to do this would be to make a function we can call when something happens, like when we initialise abilityactorinfo

A screen shot of a computer

Description automatically generated

We could make this a virtual on AuraCharacterBase and implement in both character and enemy, and then when they call it they could also call some function on the ability system to let I know it’s time to bind to any delegates it wants to bind to

A screenshot of a computer

Description automatically generated

Character base is now an overrideable virtual, so we override and define it in the AuraCharacter

A black screen with white text

Description automatically generated

And also in enemy.h

A black and blue background

Description automatically generated

Enemy already calls InitAbilityActorInfo in it’s beginplay:

A screen shot of a computer program

Description automatically generated

So let’s just refactor a little instead:

A screen shot of a computer program

Description automatically generated

And we can do other stuff at the same time in the InitiAbilityActorInfo function

First let’s make a function here:

A screen shot of a computer

Description automatically generated

So that we know when the AbilityActorInfo has been set

We’ll do that in AuraCharacter in InitAbilityActorInfo, as soon as we call it:



We’re getting the ASC from AuraPlayerState but still need to cast

A screen shot of a computer program

Description automatically generated

We can do this with the enemy too

A screen shot of a computer program

Description automatically generated

Nb This now means that the character is dependant on the AbilitySystemComponent, so we will make sure that the AbilitySystemComponent does not have to know about the character, to keep the dependency one-way!

So now the character and the enemy, as soon as they set their ability info, will call AbilityActorInfoSet(), so the ASC can now bind to delegates

A screen shot of a computer program

Description automatically generated

According the the definition it’s NOT a dynamic delegate, so we cannot use AddDynamic as before, we need to use AddUObject

A screenshot of a computer program

Description automatically generated

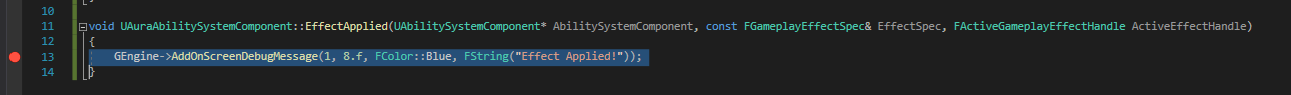
A screen shot of a computer code

Description automatically generated

2nd argument is the address of the Effect Applied function and needs to be fully qualified with the class name

Since we’re binding this here, we now have the function EffectApplied as a callback, and it is called in response to any effect being applied to this GameplayAbilityComponent object

To examine the data applying a breakpoint and printing a screen message



Callback has the ASC, owned by BP\_PlayerState:



And the effect spec:

A screenshot of a computer

Description automatically generated

We can see that the effect spec is GE\_CrystalHeal, we see the Tags, duration, periods, stack count etc

Here’s the active effect handle:



So we can get information about the effect being applied at runtime

# Get All Asset Tags

Currently there’s a debug message printing, but we would prefer that to be instead something in the HUD.

To show in HUD we need to consider dependencies – Aura AbilitySystemComponent knows nothing about the Widget controller, which broadcasts data to widgets so if we need to broadcast in screen that’s the View domain it needs to receive something from the middleman – the widget controller, which needs to get something from the ASC

A diagram of a game

Description automatically generated

So ASC should have a delegate.

What will it broadcast, what info do we want?

A good way to identify things is with gameplay tags – if we have a tag that can be broadcast from whatever effect was applied. If that effect carries that tag, we can broadcast it to the controller that can broadcast to the HUD.

So we need to find out what tags the effect has, and broadcast them in a delegate.

## How to get tags?

We have the effect spec, and EffectSpec. Will bring up a list of useful things

A screenshot of a computer program

Description automatically generated

There are a number of getters, and as a reminder Asset tags are carried along with an effect, but not granted to the target:A screenshot of a computer

Description automatically generated

Since granted tags only apply to duration effects, the asset tags can be used to get a list of all tags we might want to know about once an effect is applied, including instants

A screenshot of a computer

Description automatically generated

GetAllAssetTags takes as it’s argument a GameplayTagContainer – tags are not stored in a TArray, they’re stored in an optimised container. It’s a non-const reference.

We need an FGameplayTagContainer, so:

A screen shot of a computer

Description automatically generated

Now TagContainer has all those tags

We’d like to add a broadcast for every tag in the contrainer and we can figure out a more elegant solution later but for now let’s just loop through them:

A screenshot of a computer

Description automatically generated

Replace this with the type, and make it a reference so as not to copy the tag, and make it a const



A screen shot of a computer

Description automatically generated

So, when any effect is added we can get any all asset tags associated with that effect that we have added in the GameplayEffect BP and do what we want with them!

For the moment lets print the tags to screen:

A screen shot of a computer program

Description automatically generated

(manually add Asset tags in the GE\_ blueprints)

A screenshot of a video game

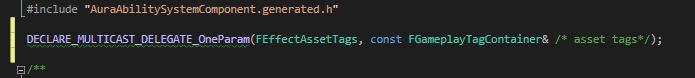
Description automatically generated

We can apply more than one tag inside a GE, so we can output multiple tags to the screen…this also means we can perform multiple Widget responses to multiple tags at once from a single GE as well

# Broadcasting Effect Asset Tags

Now we know when an effect is applied and have the ability to get the asset tags, it’s time to start broadcasting them

We’ll start by declaring the delegate inside the ASC.h



We need to broadcast this delegate, and the controller needs to bind to it, so we need to make it public.

A screenshot of a computer program

Description automatically generated

And then broadcast it at the point of collection:

A screen shot of a computer

Description automatically generated

## Lambda

In void UOverlayWidgetController::BindCallbacksToDependencies() :

Cast<UAuraAbilitySystemComponent>(AbilitySystemComponent)->EffectAssetTags.AddLambda(

[](){}

);

Lamba is an anonymous function with no name and no declaration – it’s just function and you define in the parentheses whatever you want called in response to the delegate broadcast

Since it’s a lamba we use the [](){} signature, but if we bind anything to a delegate it needs that signature as well, in this case the signature of a function that can take an FGameplayTagContainer

This lambda has parentheses, which is where the input parameter goes, so

A black screen with white text

Description automatically generated

So now we have this lambda anon function and when the broadcast happens for this delegate this lambda will be called and will receive the asset tags from the delegate

The functionality goes inside the curly brackets

Since we are now broadcasting the container, we no longer need the on-screen messages from the ASC

A screen shot of a computer program

Description automatically generated

We can refactor all that over to the overlay controller

A computer screen shot of a program

Description automatically generated

Note the variable name change to AssetTags in the refactored code

A screenshot of a computer

Description automatically generated

## UI Widget Data Table

Now we can tell when we pick up a GE with a tag and we know which tags, including if there’s more than one.

Specifically we’re accessing the Asset tags, which are not given to the actor, they’re carried along with the effect, and we’re sending them to the widget controller

Next we want to take those tags, do something with them and broadcast some information that the widgets can receive

Rather than sending a tag, we want an asset to look up info that can be broadcast. Eg a Data table that looks up by tag and retrieves a struct we can pass to the widgets

This struct will have its row structure defined in the C++

In OverlayWidgetController.h:

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Once compiled we can make a new data table of type UIWidgetRow and the fields are available

A screenshot of a computer

Description automatically generated

We do still need message tags for messages, so created some of those

A screenshot of a computer

Description automatically generated

Now we have tags, we need to let the overlay know when the controller has received a message tag

In the WBP\_Overlay BP the Event Widget Controller Set means we have the widget controller, so we can bind to delegates on the controller. Going back into the code, we have the lamba previously mentioned. Once we have received the tag container, we can see if any of the tags within have the Message parent tag, and if they do, we can look up \*something\* from a data table. Since we have a data table that has those assets in UI/Data, it would be good to have a variable of this type on the controller, that way we can always look up from that table to get any assets we want to broadcast to the widgets

A screen shot of a computer

Description automatically generated

Now in the WBP\_OverlayWidgetController we have the variable as an option

A screenshot of a computer

Description automatically generated

# Retrieving Rows from Data Tables

Referring now to the Data table DT\_MessageWidgetData in UE, there’s currently a single unset row. For the sake of demonstrating setting this to HealthCrystal

A screenshot of a computer

Description automatically generated

When looking up rows to find a specific row we can look by name. So row name should be the same as the message tag for ease of searching (beware typos!)



Data assets might be a better choice for some things, but it’s best to know tables as well

There are message tags, but there are also messages, so adding a message text:

A black and grey background

Description automatically generated

No message widget is set yet

There’s a default image, and we can set this now:

A screenshot of a computer

Description automatically generated

This is found in Assets/UI/Pickups

A screenshot of a phone

Description automatically generated

Only thing missing now is the widget, which will make later but the other rows can be created

A screenshot of a computer

Description automatically generated

We’ll be looking up the row names in the lambda, but first we should make sure the gameplay effects have the message tags

Updated the tags to remove the attribute tags and replaced with message tags:

A black rectangular object with small squares

Description automatically generated with medium confidence

A screenshot of a video game

Description automatically generated

When tested, this currently displays the name of the tag, not the tag message, because we’re not yet reading the table, only the name of the tag

A computer screen shot of a program code

Description automatically generated

Now we’re going to make a lookup in the data table.

* We need to find the data table row that corresponds to the gameplay tag
* We want to:
* Take a data table
* And a gameplay tag
* And return the row
* Function should be versatile
* Should return any sort of data table row
* Takes in a generic data table and tag

A screen shot of a computer

Description automatically generated

Nb since it’s a template function it’s defined in-line inside the header

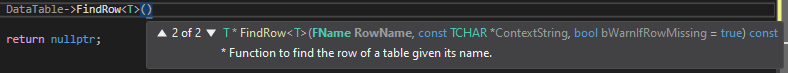
DataTable, the UDataTable type, has a function to help find rows, so we can use that

A screen shot of a computer

Description automatically generated

FindRow is, itself, a template function that requires a type. The type we want is T.

The function takes:



An FName for the Row and a context string (no worries) and an optional bool defaulting to true.

The DataTable Row is going to be the same as the tag, since that’s how we named them. We can therefore:



We don’t really care about the context string so a blank string is fine here



FindRow returns a T\*, because it’s a template function



We can store this in a local T\* Variable

Then we can say:

A black screen with white text

Description automatically generated

So, if there’s a Row, return it, else return nullptr. Nullptr is actually the default here and is baked into the definition, so we can condense this further:

A screen shot of a computer

Description automatically generated

In fact, since we have a local variable and we’re not doing anything else with it:

A screenshot of a computer program

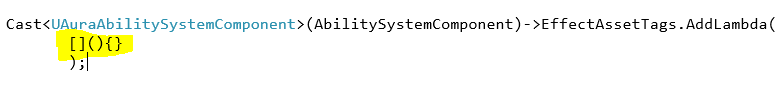
Description automatically generated

Technically we could even replace the entire function with just a line of code in the right place, but it’s an ugly line so using the function as it is here

A screenshot of a computer

Description automatically generatednb: Lambdas, as anonymous functions, are able to access global functions like the GEngine->AddOnScreenDebugMessage(-1, 8.f, FColor::Blue, Msg);, but because it’s anonymouse doesn’t know about the Overlay Widget controller because it’s outside of scope – it only knows global level stuff and things that are passed in.

That’s where the Square brackets come into play



The square brackets are for captures. If we want to access a member variable from something that class with the member variable must be captured within the lambda. To access member variables for the class OverlayWidgetController, we need to capture this object. Since we’re already inside the class, we can just use the **this** keyword

A computer screen shot of a program code

Description automatically generated

The object is now included withing the scope of the lambda!

GetDataTableRowByTag returns a pointer, so we can put this into a local variable



We can now do what we want with the row, and what we want to do is broadcast the row to the widget

We need a delegate that can send through an FUIWidgetRow

# Broadcasting Data Table Rows

We are now able to find the data table row associated with a given tag. Now to broadcast up to widgets that can bind to a delegate, receive the row and show in HUD data using assets from the data table row.



Since FUIWidgetRow is defined lower down we need to move the entire struct upwards to maintain scope

A screen shot of a computer program

Description automatically generated

HOWEVER….now the struct is up here, we’re using some types in the struct that are not defined yet! That’s OK, we can forward declare:

A screen shot of a computer

Description automatically generated

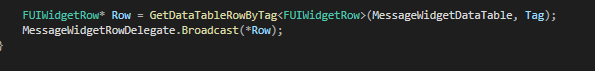
So, now that the delegate is declared with a signature we can make a member variable for it in public



If there’s one delegate for all messages, why can there not be one delegate for all attribute changes?

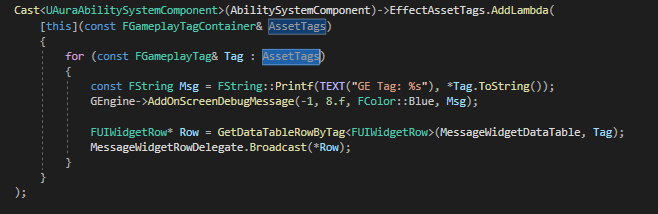
We can totally do that with some refactoring later.

Broadcasting the row:



(remember to deference the row as it’s a pointer)

Point of concern: what is AssetTags has a Gameplay Tag that is not a message tag?



What if it’s a tag for something else for the GE effect? We don’t want to restrict our Tag container to JUST message tags

The FGameplayTag type contains useful functions to help with filtering like this:

A screenshot of a computer program

Description automatically generated



This looks useful

A screenshot of a computer program

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

This function is called on a gameplay tag, passing in another gameplay tag

In Context: We want to match the message parent tag.



Store it:

A screenshot of a computer program

Description automatically generated

Nb RequestGameplayTag as a bool argument called ErrorIfNotFound that defaults to true, so we’ll get an error if we call this is a tag that does not already exist in the project. But we can use Message, because Message is still considered a tag

So



To clarify with a hypothetical:



We know that Tag in AssetTags could be one of the Message tags

A screen shot of a computer program

Description automatically generated

So, MatchesTags returns a bool of ‘True’ if the tag found is Message.<anything>

This means we can use an if and not bother collecting the row at all if the tag is not a message tag

A screen shot of a computer program

Description automatically generated

The Row pointer can be a const because we don’t want to change it. Now we have a broadcast and we just need to bind to the delegate in the widget

Binding the Widget:

In WBP\_Overlay:

A screenshot of a computer screen

Description automatically generated

From the widget controller reference cast to the BP and convert the output pin to a variable

Add the sequence so this is first, globes second and then a third pin

A screenshot of a computer screen

Description automatically generated

Make a ref to the BPOverlay, drag from it and find Message Widget, and bind event. The new event has a Row output and you can even break it for access to data

A screenshot of a computer program

Description automatically generated

# Message Widget

Output text is a great step, but we’re not going to be using debug text for production!

New Widget BP, parented by AuraUserWidget, WBP\_EffectMessage

A screenshot of a computer

Description automatically generated

A black grid 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

In data table:

A black and grey rectangular object

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

A screen shot of a graph

Description automatically generated

A screenshot of a video game

Description automatically generated

# Animating the Effect Message

The message is now appearing reliably, but it does not disappear, and we only really want it for a moment or two.

Best place to animate and remove is in the BP

A screenshot of a computer

Description automatically generated

A screenshot of a video editing

Description automatically generated

A screen shot of a graph

Description automatically generated

# Replacing Callbacks with Lambdas

Here are the callbacks:

A screenshot of a computer screen

Description automatically generated

All these functions are doing is broadcasting some value with the delegates

Here are the delegates:

A computer screen shot of a program

Description automatically generated

The OnMaxHealthChanged delegate uses a separate signature from the OnHealthChanged delegate, but in testing they functioned correctly with the same signature, because they have separate variables and are both just delegates that can broadcast a float; there’s no reason they cannot all use the same delegate for all the attributes and create a new variable of that same type for each one

The goal however is to replace the callbacks with lambdas, and the most important thing is that the lambda has the correct signature or it cannot bind to a delegate that requires this signature

The signature has a value of  - const reference to on attribute changed data

Cast<UAuraAbilitySystemComponent>(AbilitySystemComponent)->EffectAssetTags.AddLambda(

[](){}

);

The original code was:

Callback fn:

void UOverlayWidgetController::HealthChanged(const FOnAttributeChangeData& Data) const

{

OnHealthChanged.Broadcast(Data.NewValue);

}

Delegate bind:

AbilitySystemComponent->GetGameplayAttributeValueChangeDelegate(

AuraAttributeSet->GetHealthAttribute()).AddUObject(this, &UOverlayWidgetController::HealthChanged);

Lambda for asset tags:

Cast<UAuraAbilitySystemComponent>(AbilitySystemComponent)->EffectAssetTags.AddLambda(

[this](const FGameplayTagContainer& AssetTags)

{

for (const FGameplayTag& Tag : AssetTags)

{

// "A.1".MatchesTag("A") will return True, "A".MatchesTag("A.1") will return False

//

// For example, Tag = Message.HealthPotion

// "Message.HealthPotion".MatchesTag("Message") will return True, "Message".MatchesTag("Message.HealthPotion") will return False

FGameplayTag MessageTag = FGameplayTag::RequestGameplayTag(FName("Message"));

if (Tag.MatchesTag(MessageTag))

{

const FUIWidgetRow\* Row = GetDataTableRowByTag<FUIWidgetRow>(MessageWidgetDataTable, Tag);

MessageWidgetRowDelegate.Broadcast(\*Row);

}

}

}

);

To do:

Receive broadcast from ASC

Switch out the callback to a lambda

Pass in the signature

Do the thing

Remember to capture

Solution:

See annotations:

AbilitySystemComponent->GetGameplayAttributeValueChangeDelegate(

AuraAttributeSet->GetHealthAttribute()).AddLambda(

[this](const FOnAttributeChangeData& Data)

{

OnHealthChanged.Broadcast(Data.NewValue);

}

);

Rationalising the delegates:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

# Ghost Globe

Implement Ghost Globe effect

Make sure works for all globes

Ghost globe is a progress globe behind the regular globe, trails behind in a smooth manner and interpolates to it

A red liquid in a circle

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generatedA screen shot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Clamping

Ref the Aura Attribute Set’s PreAttributeChange function

A computer screen with white text

Description automatically generated

We see that we’re passing in the NewValue and clamping it, but where does it come from?

It’s from the calculation performed by the modifier in the gameplay effect that is performed when an attribute changes

This function is called even when attributes are set directly but if this is from a GE Effect the modifier returns the result of a calc and this checks it and clamps it.

In playtesting we discover that even though we can pick up health crystals until we reach 100 and the healing stops there, and that new crystals do not increase beyond this point, when we then subsequently step into the fire to take damage, the health attribute does not change! What’s going on?

If we look at PostGameplayEffectExecute:

A screen shot of a computer

Description automatically generated

We can evaluate the health attribute and look at it in-game:

A computer screen with text and images

Description automatically generated

Looks OK, but let’s check the preattribute change values:

A screenshot of a computer program

Description automatically generated

That’s too high!

Prettributechange gets the newvalue from the modifier on the GE that’s being applied…and then clamping it before the attribute changes. But the clamp is not going to change the modifier, just the value returned from querying the modifier! I we have some other GE that has it’s own modifier queried, it’s going to recalculate the value from currentvalue. So it’s like the clamp isn’t even applied!

If we check the attribute while standing in the fire we see it’s going down, so it’s working correctly mostly

We can correctly clamp by clamping a second time in postgameplayeffectexecute, because this happens after the GE effect has already been applied

This happens after the GE has been applied so we can clamp again without worrying it will replicate multiple times

A screen shot of a computer program

Description automatically generated

Why does this work? Because we’re actually SETTING the health; the preattributechange just changed the value returned from querying the modifier!

A computer screen shot of a program code

Description automatically generated