Attribute Menu

# Game Plan

Attributes show: Primaries, Secondaries, Vital on bars

+ Button for Primary, so that we can increase the primary when level up; the secondary will increase automatically with inherited values via modifiers.

A screenshot of a video game

Description automatically generated

Will require some design thought.

* Values are show with own frame so there’s a widget we can reuse
* FN to set value of numerical data represented
* Rows with text and framed value for each attribute
* Own widget
* Primaries are similar but also have button
* Row widget + Own widget for widget
* Reuse functionality from 2ndary?
* One big widget for menu itself
* How to construct?
* Not all 2nd attributes shown
* Scrollbar to move down to see additional values

A screenshot of a game menu

Description automatically generated

So:

* Framed Value widget for numbers
* Row widget contains text box and also framed value widget
* Primary is a row widget with and additional button widget
* Attribute menu widget contains many row widgets and scrollbar
* Health and mana own widget progress bars
* Close menu widget

A black background with yellow text

Description automatically generated

# Framed Value

New UI BP

Attribute menu an Overlay? New folder is tidier

WBP\_FramedValue based on AuraUserWidget

+SizeBox

FillScreen to Desired

Override Height and Width to arbirary values 80/45 to work with for now, change later if needed

Box Size (rename to Box Size Root) set to variable; in Graph set Height and Width of box as variable floats

A screenshot of a computer

Description automatically generated

Set category for W + H

Drag in Size Box Root and call set overrides from Event PreConstruct

A screen shot of a computer

Description automatically generated

Check defaults, currently at zero and collapse to Function



Size box now exists. To add to the menu:

* Background colour/image
* Border
* Text

Since things are going to be on top of one another will need to add an overlay that is a child of the size frame and set to fill it

A screenshot of a computer

Description automatically generated

Add an image of a child of the overlay, give it a value, either plain black RBG sliders or an asset:

A screenshot of a computer

Description automatically generated

This is an animated asset like the health/mana globes and has params we can set to preference:

A screenshot of a computer

Description automatically generated

Renamed the new elements and set as variable, then set in the graph:

A screenshot of a computer

Description automatically generated

Remember to set a default value for the brush

A screenshot of a computer

Description automatically generated

This way we can set variations, like colour, in child BPs if we want

Collapse to Fn

Add new image for the border

Fill H + V

For image pick desired border asset

A screen shot of a computer screen

Description automatically generated

Change Draw As from Image to Border and choose margin size – I went with 0.5 for all.

The size box can be changed in size and the border and background will stretch to match

A screen shot of a computer

Description automatically generated

For now I won’t parameterise the border, we can always change that later if we want!

Add Text block. Will set default as a 2 digit number for convenience

A screenshot of a computer

Description automatically generated

Set alignment (centre)

Set Justification (center)

Change font

A screenshot of a computer

Description automatically generated

Play with font size and outline settings for appearance

Set the text as a variable so we can set the value with Fn later

# Text Value Row

New Widget WBP\_TextValueRow

Add Sizebox, override W and H, as before with the frame

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

We’ll add elements left to right; first text, then the framed value widget

Add Horizontal Box and text to that box, horizonal align left, centre vertical

A screen shot of a computer

Description automatically generated

Set default text, font, fontsize outline and fontspacing to taste

A screenshot of a computer

Description automatically generated

Add FramedValue Widget, set to fill, right, center

A screenshot of a computer

Description automatically generated

Add a spacer after it to modify the size a little

A screenshot of a computer

Description automatically generated

Test with longest attribute names (intelligence, critical hit resistance)

A screen shot of a computer

Description automatically generated

Too long! Make the size box bigger!

A screen shot of a computer

Description automatically generated

Better!

The spacer works well, but remember we will also need to add a button in a child class

For this we can add a Named Slot

A screenshot of a computer

Description automatically generated

Slot will allow children to have more widgets on the named slot

Tinker a little for overall taste:

A screenshot of a computer

Description automatically generated

# Text Value Button Row

New widget based on the Text Value row

A screenshot of a computer

Description automatically generated

Includes all elements by default but only the named slot is accessible.

However, we will need to change the text and the numerical value for each instantiation, so we need to expose them!

For now, let’s add the button. Start with an overlay and an image for the button border/background

A screenshot of a computer

Description automatically generated

Center and draw as image, set size (I picked 45 \* 45 to match the value box size)

Add button

A screenshot of a computer

Description automatically generated

Now pick styles for Normal, Hovered, Pressed and Disabled

This project has existing assets to use out of the box

A screenshot of a computer

Description automatically generated

(set to draw as image)

We can add a single character of text, a +, so we know it’s for adding to the attribute!

Alight and justify as usual, set a font and appearance to taste

A black and grey plaid

Description automatically generated

We’ll need functions to set text and numerical value and make the button functional

# Attribute Menu Construction

We now have 3 widgets to use in the attribute menu

As before, sizebox/overlay - for now hard code widgth/height as this is a specific object

Now we start stacking.

We want a border, title, value widgets for rows with buttons etc

Draw as border, 0.5 borders etc

A screenshot of a computer

Description automatically generated

Add a wrap box to help space and align things properly

We’ll drag the elements onto the wrap box

Text block for manu name:

ATTRIBUTES, Fill space, set a size, centered Horizontally

Give wrap box a padding

A screenshot of a computer

Description automatically generated

Adding fields

Wrap box will wrap to a new line if over size, example:

A screenshot of a computer

Description automatically generated

Setting the new text box here, which starts right after the other box, to on “Fill span when less than “with a value greater than the set size of the sizebox, will result in the text box being pushed to a new line

A screenshot of a computer

Description automatically generated

To make some space from the title, add a spacer

A screenshot of a computer

Description automatically generated

The Y size can be set to make bigger spaces

Ctrl-C/Ctrl-V to copy

Next a Text Value Row (no button) for Attribute points we can spend then the box with button for the primary attributes

In the WBP\_TextValueRow, expose the Box Height/Width variables and compile, then add a new TextValueRow to the menu

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

New:

A graph paper with a grid

Description automatically generated

Rearrange a little as the points will not be an attribute, and add button rows

A screenshot of a video game

Description automatically generated

This can be modified to taste, maybe some more padding and smaller row length again?

Pad the wrap box to 40, set width 720

Primary attributes look good

The secondary attributes will go below the primary, and we’ll put them all in their own scroll box

Set a size and fill empty space, add a spacer above

Drag the scrollbox onto the existing size box

Drag a TextValue onto the scrollbox

A screenshot of a computer

Description automatically generated

Repeat for all secondary attributes

A screenshot of a computer

Description automatically generated

For secondary attributes things look a little cramped and the box slot is taking up a bit too much room, so we can edit properties

Since there are a few of them, maybe have a function to set the font and other properties like space dimensions and named slot later

Finally lets add an image background for the whole menu

A screen shot of a computer

Description automatically generated

Need some padding

And a background image

A screenshot of a video game

Description automatically generated

# Button Widget

Add new heading for Secondary attributes (copy the primary text box); adjust spacer size to make things all fit better

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

We now also need a X button to close the menu after we have looked at the values

Add a sizebox outside of the wrap box

A screenshot of a computer

Description automatically generated

I’ve put this at the top right like in Windows – I feel it’s more intuitive

Set size to 54/54 and padded 25 from right and top

Add an overlay as normal, then a button as before:

A screenshot of a video game

Description automatically generated

We’re making a lot of very similar buttons; we should probably just BP buttons as widgets themselves, copying the attributes we’ve use for other buttons, like sizebox, overlay, border, text etc

A screenshot of a computer program

Description automatically generated

# Wide Button Widget

New button from the main overlay to open the Attribute menu

Based on WBP\_Button: WBP\_WideButton

# Opening the Attribute Menu

To make Attributes button functional:

Make it a variable in the overlay.

Event Construct: assign to onclicked event for this button to a custom event

A screenshot of a computer program

Description automatically generated

Disable the button on clicked:

A screenshot of a graph

Description automatically generated

And create widget and add to viewport:

A screenshot of a computer screen

Description automatically generated

For the moment using 0 player controller but will add multiplayer later

This opens an attribute menu with clickable non-functional buttons but it fills the screen

To reduce the size, wrap the entire thing in an overlay:

A screenshot of a computer

Description automatically generated

And bam, done!

A video game screen with a game screen and a game screen

Description automatically generated with medium confidence

We can add padding and set the size like with Messages:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a video game

Description automatically generated

Hardcoded X, Y as 50 but other values, or variables, will work

# Closing the attribute menu

In WBP\_AttributeMenu

A screenshot of a computer screen

Description automatically generated

Closes menu but does not turn the Attributes button back on!

Currently the Overlay has a dependency on the Attribute Menu, so we should NOT give the Attribute Menu a dependency on the Overlay!

Send an event dispatcher that the overlay can subscribe to – it’s the BP version of a delegate!

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

When this is called any other BP subscribed to it will have the event triggered

A screenshot of a computer screen

Description automatically generated

So, there are 2 possibilities here, either the Attribute menu could be created and destroyed each time, or it could be toggled visible/invisible. Both are OK and there’s no significant performance issue from creating/destroying (a tiny one) but this way we avoid a widget living in the background and potentially responding to callbacks while not visible

# Plan for Displaying Attribute Data

Now we have an attribute menu we want to show the values, names and maybe also mouseover tooltips?

Need to get the data to the attribute menu.

Following UI architecture we need a Widget controller, which can subscribe to delegate for attribute changes and broadcast them to the widgets

Do we just sub to delegates for all delegates and make a new delegate for each attribute and broadcast, and continue this every time we add a new attribute?

This will make for a LOT of delegates and broadcasts! Very brute force approach!

A screenshot of a video game

Description automatically generated

Not very scalable; if we add new attributes we need to bind new callback/lambda in the widgetcontroller to the broadcast and declare a new delegate to broadcast to widgets. 1 new attribute means a lot of code changed in the widget controller and maybe also in BP as well…hard to maintain and expand!

What if Widget controller has code designed to receive broadcasts and then broadcast a generic delegate plus more info, for example each attribute is associated with a gameplay tag, so if str changes, the controller broadcasts one delegate – OnAttributeChanged - plus the str tag and any other information needed (name, value, tooltip description etc. Maybe package into a struct). The widget then checks the tag and does whatever needed. Each row receives the delegate, checks for tags and if a match takes specific data for updates

A screenshot of a video game menu

Description automatically generated

Flow of info: Widget controller binds to delegates broadcast from Ability System

When attribute changes, widget controller knows about it.

Controller takes the attribute, figures out what the tag associated with the attribute is, then performs lookup to find correct struct and send data to widgets.

If Data driven – data table or asset that can package the data

Data assets lookup by more than just name – can use anything we want, for example a gameplay tag

A screenshot of a computer

Description automatically generated

1. Need widget controller for menu set up to get broadcasts from ability system onattributechange
2. Figure out correct gameplay tag associated with attribute- map together
3. Reference tags in C++ - RequestGameplayTag(FName(”Attributes.Primary.Strength”))
   1. Asking Gameplay Tags Manager for a tag struct, giving it name to look up, then if GTM has tag return that tag for us
   2. Error prone, not good to have to type out the string name by hand. Some kind of class that stores global source of tag variables
4. Data Asset capable of receiving gameplay tags and returning aura attribute info struct
   1. As soon as we know the tag for a given attribute data asset can give all the data we need to send up to widgets. Widgets then check their own gameplay tags against the struct and update accordingly.

A screenshot of a computer

Description automatically generated

A screenshot of a black background with yellow text

Description automatically generated

Next steps:

1. Handle tags better, not call requestgameplaytag all the time, centralised source of tags on C++ side
2. Store structs, perform lookups based on gameplay tags to find attribute info needed to broadcast to widgets
3. Contains all info needed to send
4. Fill in Asset
5. Controller ties everything together; listens to ability system broadcasts, performs lookups from data asset, retrieves and broadcasts struct

# Gameplay Tags Singleton

So, we want to add tags for all our attributes. We know we can add them directly from the Project Settings, or from a Source config file and from Data tables using the gameplay tag table row.

However, when we need to use those tags in C++ we have to use RequestGameplayTag (like in Overlay Widget controller)

A computer screen shot of a program

Description automatically generated

This is a good use case for Requerstgameplaytag, but we don’t want to have to do this every time we need a gameplay tag, we want to use variables and have a central source, so we don’t have to type out strings and worry about typos etc

So create centralised location for the tags we use in C++ domain.

So we’ll create a C++ class to store them – a unique struct enforced by using Singleton pattern

New C++ class in the project root, parent class: None

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generatedA screen shot of a computer

Description automatically generated

Completely empty!

But not minimalist enough!

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

Stripping out the class declaration and constructor/destructor to make it a struct:

A screenshot of a computer program

Description automatically generated

Structs will default to public permissions, but I’m \*explicitly\* creating public, protected and private sections here

‘Native’ in the sense that the tags are created within C++ and available there (but will be available in editor and BPs also

Note the inclusion of GameplayTagContainer – we will be using the GameplayTag type

Because this is a singleton we’ll design this in a specific way

* Create a static Getter function
* No need of an instance of the class to exist to call one of it’s static functions

A screen shot of a computer

Description automatically generated

static const FAuraGameplayTags& Get() { return }

Static function

const because not making changes ofc

Uses the type of a reference to the one and only instance of FAuraGameplayTags in the project (this)

Function is Get() and it returns the only instance that exists

Instance will be a static variable on this class – the class itself will have a variable to store the only instance of itself

A screen shot of a computer

Description automatically generated

So, once this is constructed we can call the static function Get to get the only instance of this struct

So a static variable like GameplayTags we need to go into the CPP and explicitly declare the type here:



Next we will need a public function to initialise the tags – to create the native gameplay tags

A screenshot of a computer

Description automatically generated

How to start initialising:

Get the Gameplay Tags Manager – when we create a tag, it’s registered with the tag manager, a unique object within the project



Note: GameplayTagsManager.h – Tags with a S!

A screenshot of a video game

Description automatically generated

Look at this! A static function called Get() that returns the one, and only, GameplayTagsManager. That’s the same pattern we’re using here

So we get that and can use functions on it like:

A screenshot of a computer program

Description automatically generated

This requires the tag name and has a tag dev comment – specify name and an optional comment

We’ll define a native gameplay tag, starting with secondary attributes:



Attribute name, Dev comment.

Now we have a Singleton that we can always get with the Get function and we have created the static gameplaytags instance here, which is a global singleton

At what point should we call this function to initialise the tags?

Probably as soon as we can, to initialise them before anything tries to use them, using a custom asset manager.