Game UI Architecture

Most Game project involves attribute such as health and they are displayed on the screen in widgets

Widget BPs are derived from UUSerWidget and need to receive data to show the info correctly; healthbar needs to show health data, an ability icon needs to show that ability info etc

Attribute set include data for attributes; Playerstate might contain variables that need to be shown, and many other classes might also contain data that needs to be displayed!

A screenshot of a game

Description automatically generated

Possible approaches: Lots of pointers and references in the widget object to access data directly!

Is that the best way? How should this system be designed, how should it scale up? How is it done professionally?

In a well-structered project there should be a separation of concerns

UI: 3 different domains:

THE VIEW

* Display of visual effects
* Health and mana bars
* Ability icons
* All the widgets!

THE MODEL

* Models underlaying rules and results of the game
* Drives all the widgets that we will see in the view domain
* The data
* Player Health
* Player Level
* Mana
* Health
* Experience
* Unlocked abilities
* Their level
* What buttons are assigned to

THE WIDGET CONTROLLER

* A class
* Retrieves data from Model
* Calculations
* Algorithmic processes to process data
* Broadcasts to View
* NOT the same as a controller or player controller class!

Interactions:

* View focusses on how data is displayed only
* Receives Data from the controller ready to use
* May contain interactable widgets that result in changes to the model
* feed info back to the controller
* Facilitate changes made from the Widget interaction, by the player, to the model

A diagram of a game

Description automatically generated

Concerns:

Each domain is isolated so system is highly modular and prevents hard coding dependencies that would make the system ridged

Model should not care about which widgets are representing data: Controller depends on the classes in the model

Controller should not know which specific widgets are receiving data broadcast to them, the widgets depend on the controller

One-way dependencies allow model to have widget controller to be swapped out without changing any other code in the Model’s classes; also controller can have widgets switched out as well without code changes

Flexible codebase

A screenshot of a game

Description automatically generated

Controller UAuraWidgetController

Subclass of UAuraUserWidget

Next:

1. Custom Base Widget class – UauraUserWidget
2. Controller UAuraWidgetController
3. AAuraHUD

AuraUserWidget:

Public:

UPROPERTY(BlueprintReadOnly)

TObjectPtr<UObject> WidgetController;

This takes a generic object that can be set as the controller

protected:

UFUNCTION(BlueprintImplementableEvent)

void WidgetControllerSet();

This allows BPs to react to a widget controller being set for a given widget

These should be tied closely:

void UAuraUserWidget::SetWidgetController(UObject\* InWidgetController)

{

WidgetController = InWidgetController;

WidgetControllerSet();

}

Variables needed for this:

UPROPERTY(BlueprintReadOnly, Category = "Widget Controller")

TObjectPtr<APlayerController> PlayerController;

UPROPERTY(BlueprintReadOnly, Category = "Widget Controller")

TObjectPtr <APlayerState> PlayerState;

UPROPERTY(BlueprintReadOnly, Category = "Widget Controller")

TObjectPtr <UAbilitySystemComponent> AbilitySystemComponent;

UPROPERTY(BlueprintReadOnly, Category = "Widget Controller")

TObjectPtr<UAttributeSet> AttributeSet;

## Health Globe UI widget

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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 screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Result:

A screen shot of a blue circle

Description automatically generated

BPs of the globe can be created and added to a Canvas widget, and this overlay called in the HUD code:

//viarable to store the overlay widget

UPROPERTY()

TObjectPtr<UAuraUserWidget> OverlayWidget;

protected:

virtual void BeginPlay() override;

private:

//variable to store OL widget class

UPROPERTY(EditAnywhere)

TSubclassOf<UAuraUserWidget> OverlayWidgetClass;

};

The BPing the HUD in Unreal allows it to be added to the custom game mode Beginplay

## The widget controller

AuraWidgetController has 4 key variables: PlayerController, PlayerState, AbilitySystemComponent and AttributeSet

To create a widget controller we need to set these variables

Create a struct with these variables so easy to initalise!

So, variables are:

A screenshot of a computer program

Description automatically generated

Therefore the struct would be:

A screenshot of a computer program

Description automatically generated

First since it’s a struct to use inside unreal give it a ustruct macro (with a BP type in case we need to create and use one of these in BPs)

Generated body because USTRUCTS need that

Give it a default constructor (empty with empty body), then one to receive the variables

FWidgetControllerParams(APlayerController \* PC, APlayerState \* PS, UAbilitySystemComponent \* ASC, UAttributeSet \* AS)

Define the constructor inline with initialiser list

: PlayerController(PC), PlayerState(PS), AbilitySystemComponent(ASC) , AttributeSet(AS) {}

You need the variables to initalise, so Add local TObject pointers to the variables and set UPROPs

Initialise to nullptr to avoid compiler errors at start

Initialise the pointers with the input parameter so PlayerController with PC etc

Function to initialise:

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

Create new child of the Widget controller for an overlay controller

## Construct the overlay in the HUD class

FN to create widget controller to create if not yet created; if created return that controller

In AuraHUD:

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

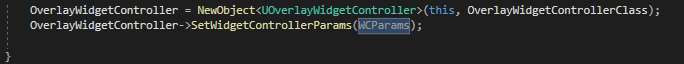
Check for controller, if none, create with NewObject. NewObject requires this (HUD) and also the class type for the widget being created)

Since HUD is a child of widgetcontroller, we need A black background with white text

Description automatically generated

This getter function (GetOverlayWidgetController(const FWidgetControllerParams& WCParams))

takes WCPARAMS so we have it now, so we take OverlayWidgetController, call the public SetWidgetControllerParams fn and pass in WCParams



And return OverlayWidgetController!

Now we have the getter working we need to contruct the widget controller

A computer screen shot of a code

Description automatically generated

BUT to construct it we need WCParams!

And to initialise WCParams we need access to the key variables which BeginPlay does not have access to!

This should be done somewhere that has access to the variables

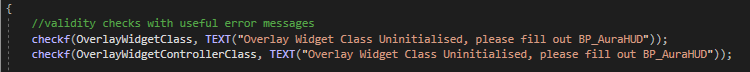
HUD needs a fn to call to pass in variables, so we can create and assign (not in begin play)



Move the create widget and add to viewport to the new function

A screen shot of a computer program

Description automatically generated





Create the widget and cast to AuraUserWidget so we set its controller; once this is done we can set it’s Overlay widget from the HUD.h (which is an aurauserwidget)



So, next we need a struct of type FWidgetControllerParams



Which takes params as before

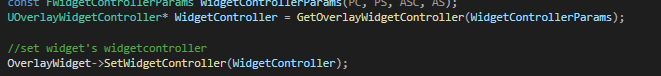
Initialise with PC, PS, ASC, AS



(make it const, no need to be mutable)

Now params are initalised we can construct the controller!

We can just call GetOverlayWidgetController and store it in a pointer



The function now in hud and as long as we pass in the controller, state, ability system component and attribute set, it will construct the overlay and controller and set the controller then add to viewport!

But When and Where do we call this?!

Objective:

Make sure all key variables are initialised

* Controller
* State
* ASC
* AS

State was set in InitAbilityActorInfo

* Server
* Client

Solution:

AAuraCharacter::InitAbilityActorInfo()

This function is called at OnPossess and at OnRepPlayerstate so it works fine for client and server.

Player controllers are always accessible to characters so GetController()

GetController returns an AController, so we need to cast to the game specific controller.

Works fine casting directly to custom class

Cast<AAuraPlayerController>(GetController()

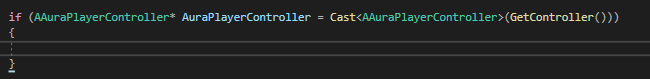
Store inside a pointer of correct type:

AAuraPlayerController\* AuraPlayerController = Cast<AAuraPlayerController>(GetController());

So do we check pointer in case it’s null, or will it never be?

The game supports multiplayer, and the server has access to all player controllers HOWEVER, clients do not have access to other client controllers. So this will work OK in single player but crash on clients if we try to use multiplayer!

So, check validity!



Are there other parts of code with similar problems?

YES!

A black screen with white text

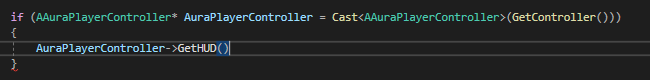
Description automatically generated

In this case we should be checking the Subsystem already before using it instead of using an assert!

A black background with white text

Description automatically generated

So:



This returns an AHUD, but we want our HUD, so

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Adding a breakpoint to the AddtoViewport shows us:

A screenshot of a computer program

Description automatically generated

## Broadcasting Initial Values

In the HUD, the Overlay Widget has a pointer to it’s controller, which we can access even from BP.



Widget controller knows only that the overlay widget has access to 4 key variables, nothing else.



This is how we will broadcast to the widget: with a delegate!

All widgetcontrollers will need to broadcast to widgets, so we will add a function to the base class

A screen shot of a computer

Description automatically generated

Leave blank in the base class and define the specifics in derived classes

Ctrl K O to open header

New dynamic multicast delegates in .h

Dynamic so can assign events in BPs

Multicast because multiple widget BPs may want to bind to update

For each we’re sending only one value so:

A black background with white text

Description automatically generated



This only declares the type, we need member variables of this type that we can assign in BP

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

This will broadcast a health of 100f hardcoded. BUT, what if the character starts with less, like a weaker enemy? We need to get that attribute!

Broadcast the actual health attribute initial values for health and max health

OverlayWidgetController is an AuraWidget controller, therefore it has AttributeSet member variable, which is a TObjectPointer.

We want to access the actual health value of the AuraAttributeSet Health Value

Cast to UAuraAttributeSet

A screen shot of a computer program

Description automatically generated

Now we can use the getters

A computer screen shot of a program

Description automatically generated

Complete function is

A screen shot of a computer program

Description automatically generated

When to call?

Should be after the AttributeSet is valid and after the widgets have had controller set

A screen shot of a computer program

Description automatically generated

Widget is created

Controller created

Controller set result of SetWidget controller



Controller params have been passed in so the set is now valid



Controller has been set, so now we can tell the widget controller to broadcast initial values

In Unreal:

WBP\_Overlay:

A screenshot of a computer

Description automatically generated

Set globes as variables

A screenshot of a computer screen

Description automatically generated

Back in code:



When SetWidgetController() is called, the BP will trigger the event WidgetControllerSet, which will set the same controller for the globes

This will then trigger the WidgetControllerSet event for those widgets AS WELL in their blueprints!

So we can now add events to the globes BPs

A screenshot of a computer screen

Description automatically generated

If we can cast to OverlayWidgetController we can bind to the delegates

However, we can’t yet:

A screenshot of a computer

Description automatically generated

Because the the OverlayWidgetController needs to be a Blueprint type

A black screen with white text

Description automatically generated

Back in BP:

A black and white text on a black grid

Description automatically generated

Now can be used in BPs because BP type, but it’s also BPable, so we can make a new BP to use in the Hud BP!

A black and grey background

Description automatically generated

Now cast health globe to the BP widget controller:

A screenshot of a computer

Description automatically generated

And access OnHealthchanged delegate

On Event Assign OnHealthChanged:



So:

A screenshot of a computer

Description automatically generated

Promote to variables:

A screenshot of a computer

Description automatically generated

Globe Progress Bar base class will take a new function with a new float variable Percent

A screenshot of a graph

Description automatically generated

Set Progress Bar Percent is now available in the HealthGlobe and ManaGlobe child classes

A screenshot of a computer

Description automatically generated

SafeDivide function means if MaxHealth is 0, we do not divide by 0, just returns 0

A screenshot of a computer

Description automatically generated

Health is now at 100 because accessing the real value and max value

## Responding to attribute changes

Widget controller is now capable of broadcasting initial value

We can cast attribute set to an AuraAttribute set and access the values with the Accessors. However, we need to respond when those attributes change!

ASC has a function for doing that.

Take the ASC pointer and call GetGameplayAttributeValueChangeDelegate()

This function needs to know which attribute we are talking about, this can be gotten from AuraAttributeSet if Health – pass in AuraAttributeset and call GetHealthAttribute

A screen shot of a computer

Description automatically generated

AuraAttributeSet->GetHealthAttribute() returns an FGameplayAttribute

which is what GetGameplayAttributeValueChangeDelegate() requires

– and therefore GetGameplayAttributeValueChangeDelegate(AuraAttributeSet->GetHealthAttribute()) returns the delegate, and we can now bind to that delegate.

It returns an FOnGameplayAttributeValueChange delegate

A screenshot of a computer

Description automatically generated

To bind to it, we use the dot operator and UFUNCTION AddUObject()

If this were a dynamic multicast delegate (which it is not) we could use AddDynamic(), but FOnGameplayAttributeValueChange is

Declaration for FOnGameplayAttributeValueChange shows it is of type FOnGameplayAttributeValueChange&



Declaration for FOnGameplayAttributeValueChange type shows that this is a Multicast OneParam delegate, not a Dynamic multicast, so we cannot AddDynamic()



We have to instead use AddUObject()

To bind an object to it we pass in followed this by the callback function

A computer screen shot of a computer screen

Description automatically generated

The callback function has to have a specific signature to be able to be bound to this delegate

A screenshot of a computer

Description automatically generated

So we need to create a function to bind to it

Binding callbacks to the dependencies of this widget controller is something that ALL widget controllers need to do, so we need to make a function to bind out callbacks to all those dependencies in the parent class!

A screenshot of a computer

Description automatically generated

Declared the Fn and Generated a definition, but left blank because will override it in the child.

A screen shot of a computer

Description automatically generated

So, we need a callback that we can bind to that delegate whenever our attributes change, such as Health and MaxHealth, so we really want a callback function for each of these

To work, the function will require a signature that has a const reference to FOnAttributeChangeData

A blue and white screen

Description automatically generated

These functions have a signature that qualifies them to bind to the delegate that broadcasts when an attribute changes

reminder:

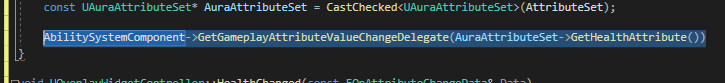


So we can do that in BindCallbacksToDependencies()

We can take the line

const UAuraAttributeSet\* AuraAttributeSet = CastChecked<UAuraAttributeSet>(AttributeSet);

from BroadcastInitialValues() because we need that, then take the ASC and call GetGameplayAttributeValueChangeDelegate() and pass in the attribute we want to bind



The highlighted expression, as we established, returns the delegate and we can then dot AddObject(this, and the address of the callback function)



HealthChanged has now been bound to the change delegate

The highlighted section is the attribute we specify:



This is the callback



Therefore for the delegate binding for MaxHealthChanged we would use



So, what do we actually DO when health changes?

We want to broadcast the OnHealthChangeDelegate so the widgets respond to it, and we do that inside the HealthChanged function.

We do not need to access it from the AuraAttributeSet however because we already have it, so we can just do this:

A screen shot of a computer

Description automatically generated

This can be made const

A screen shot of a computer

Description automatically generated

So now whenever Health and MaxHealth update, the delegates are broadcast and the widgets are already bound to the functions

Time to test!

We know that the Health Potion actors adds to the health stat so we can test this

In AuraAttributeSet init health at 75:

A screenshot of a computer program

Description automatically generated

A screenshot of a video game

Description automatically generated

A red liquid in a circle

Description automatically generated

Health goes to 100, UI remains the same

This is because we never called BindCallbacksToDependencies!

A black and white text

Description automatically generated

The best place for this is inside the HUD right after setting the params and just before returning the widget controller:

A screen shot of a computer program

Description automatically generated

That way we know the widget controller callbacks will be bound so when attributes change we’ll get it broadcast

A screenshot of a game

Description automatically generated

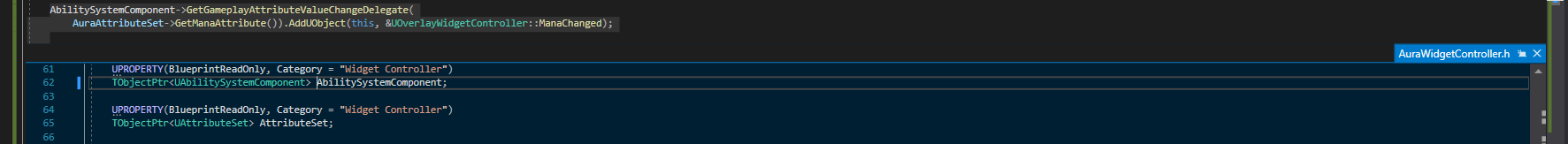
This seems like a LOT of work to set up, but we now have a class designed to handle retrieving and controlling widget data. The dependencies are all one-way, ASC knows nothing about the OverlayWidgetController, and the OverlayWidgetController knows nothing about the widgets, so controllers and widgets are now modular

A black and white screen with white text

Description automatically generated

A screen shot of a computer program

Description automatically generated



A computer screen with text

Description automatically generated

A screen shot of a computer code

Description automatically generated A screenshot of a computer screen

Description automatically generated