Dynamic Widget Switching

User Guide

Introduction

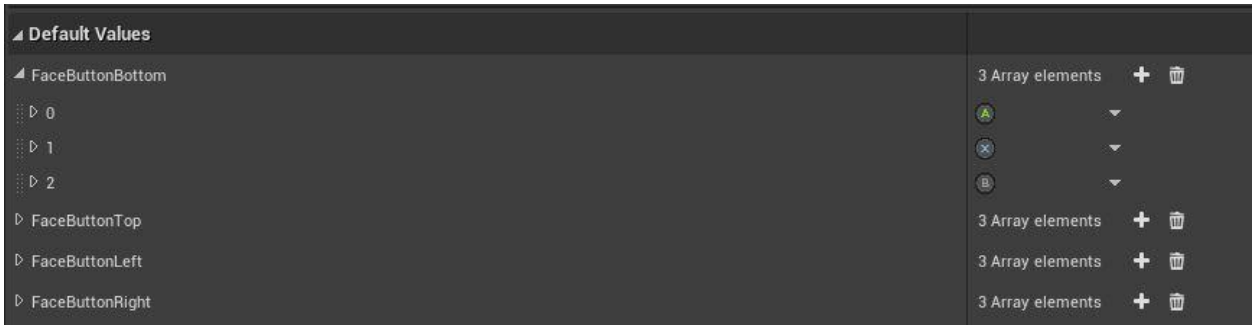
This asset is a library and player controller that allows you to automatically detect the input method being used by the player, and display the appropriate icons in your HUD and UI widgets. No ticks or timers required. You can also set up multiple Gamepad Styles, so the player can select the style of icon to display without the need to hardcode the logic into individual Widgets.

See the asset in action here: https://www.youtube.com/watch?v=lDrssV_4FDw

Updates

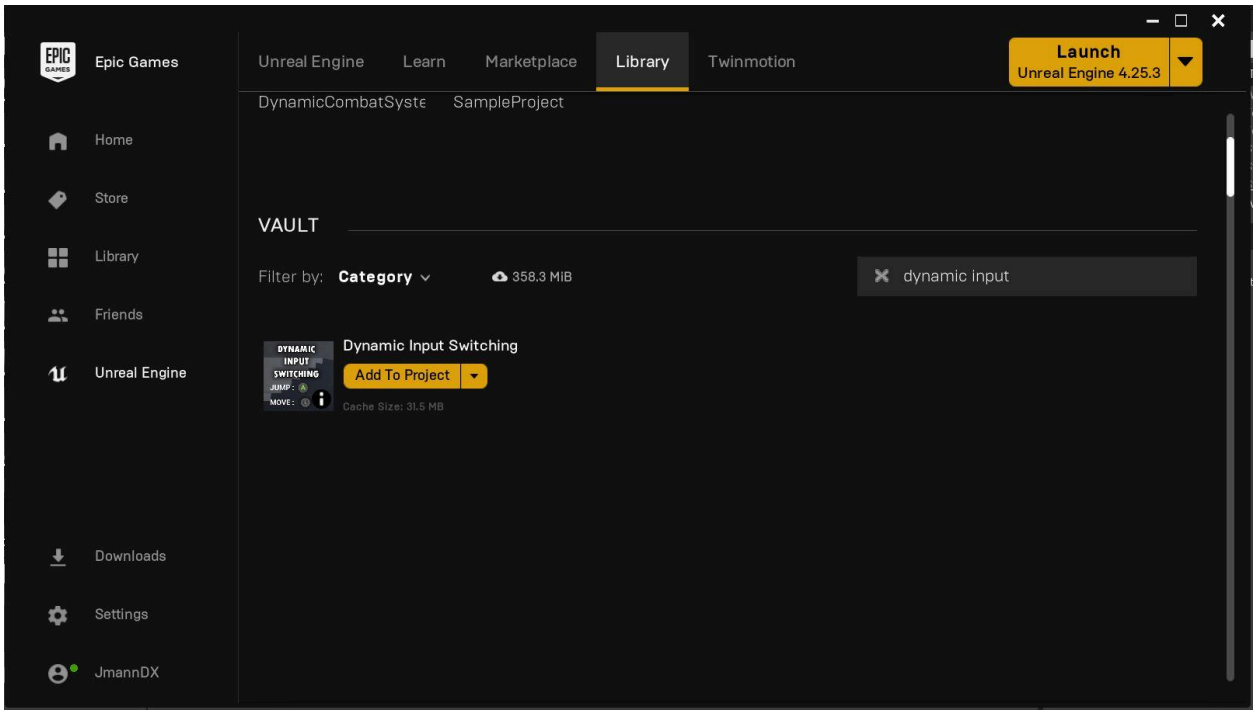
Update 1

Update 1 is a complete overhaul of how the library assigns the Slate Brushes. Now, each image is controlled inside S_GamepadIcons. Adding your own icons and Gamepad Styles is much more straightforward.



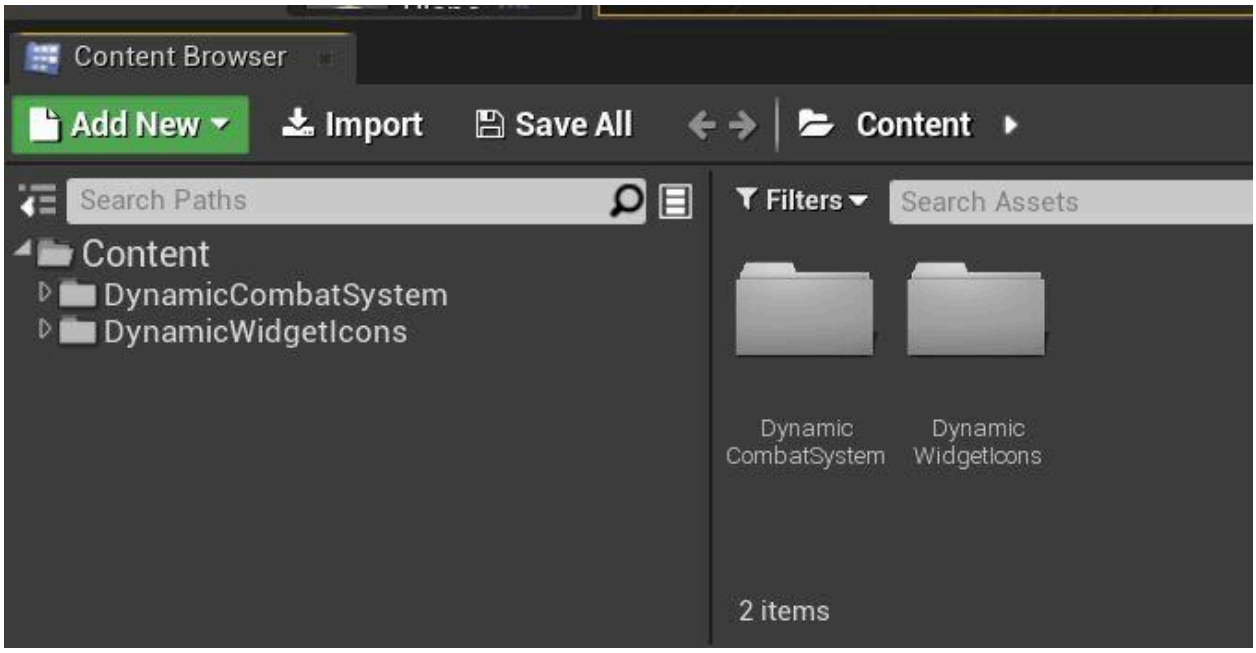
Integration

Integration of the asset is incredibly straightforward. First, just select “Add To Project” under Dynamic Widget Switching, and select your project.



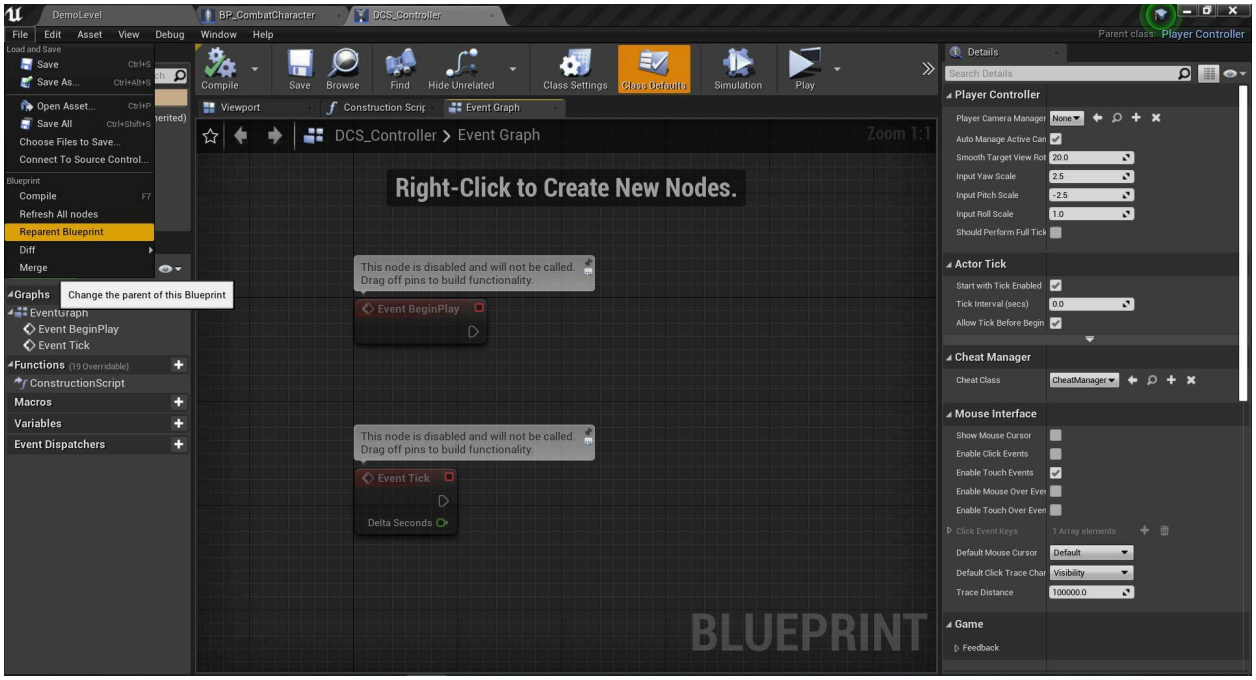
Now, when you open up your project, you’ll see a new folder called “DynamicWidgetSwitching”.

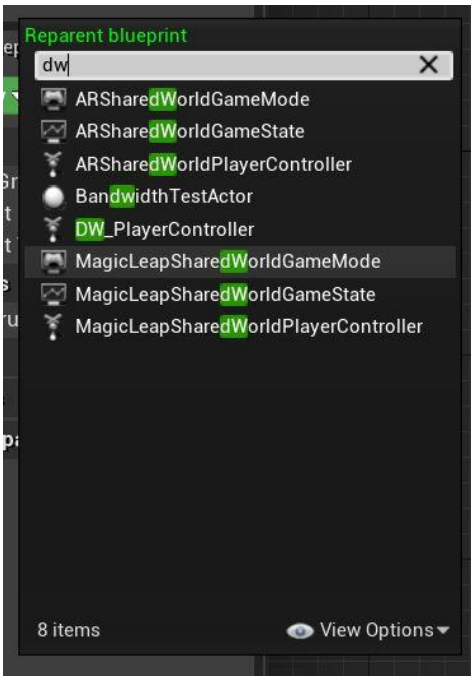
There are several subfolders here, and you are free to poke around to get a feel of the Library Functions and the enums.



Next, we want to make sure the Player Controller knows what's up. If you aren't using a custom Player Controller, you only need to open up your Game Mode and change Player Controller to "DW_PlayerController". Then you can start using all the Library Functions immediately.

If you are using a custom Player Controller, don't panic. Just open your player controller and select "File" > "Reparent Blueprint" and select "DW_PlayerController". You may need to search for it.



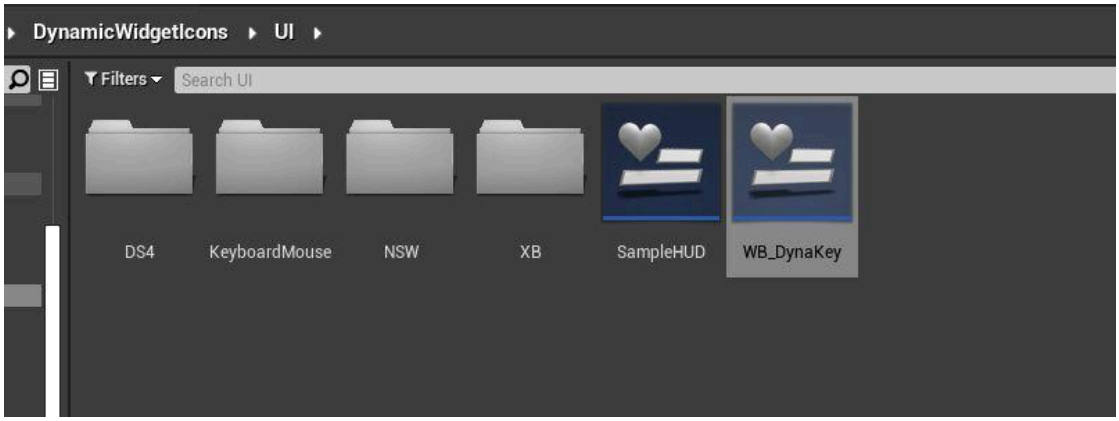


You'll know you've successfully reparented when you can see the default Gamepad Prompts in your Class Defaults.

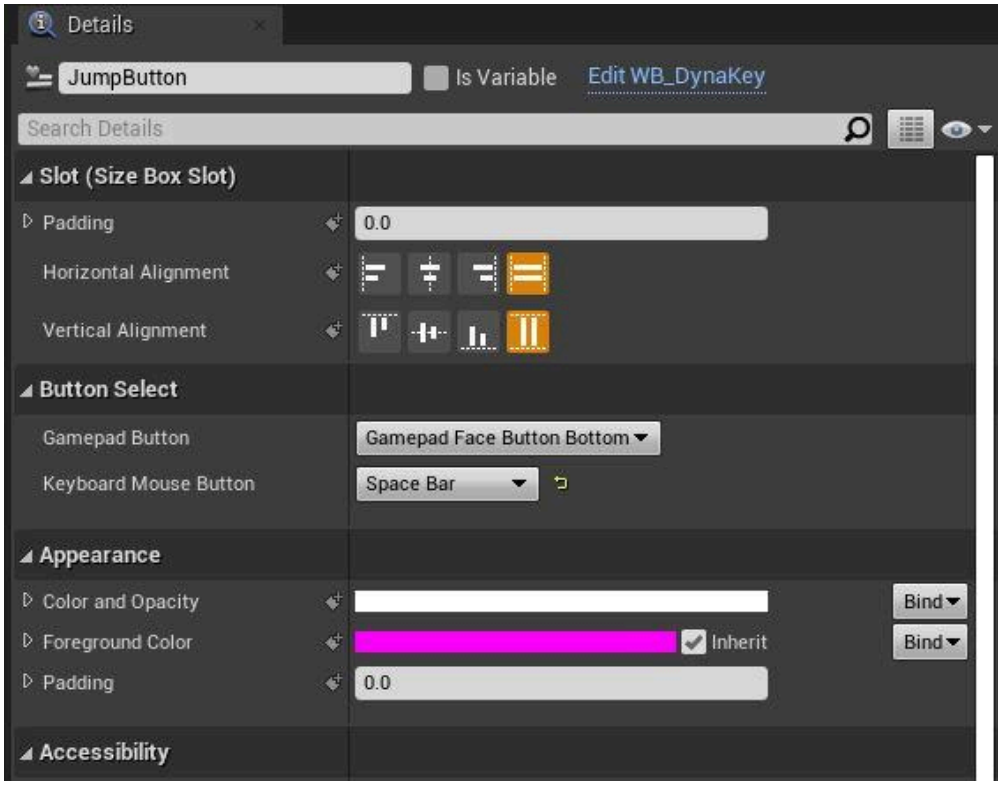
However, these prompts are no longer utilized in the latest version, and are only there for Legacy integrations.

DynaKey

In the latest update, I added a new Widget Blueprint called “WB_DynaKey”. It actually handles all the functions automatically, so if you utilize this Widget as the basis of your Input Icons, you don’t need to worry about casting (as long as the Dynamic Widget Switching Player Controller is either the current Controller or a parent of the current Controller).



When you add WB_DynaKey to an existing widget, you’ll be able to select both a Gamepad Icon and Keyboard/Mouse Icon to display. Then, in the background, it will grab the corresponding image and set it to a Slate Brush automatically based on the included Icon structures.



Utilizing the Library

There are two main Functions you can utilize inside your widgets.

GetGamepadControls

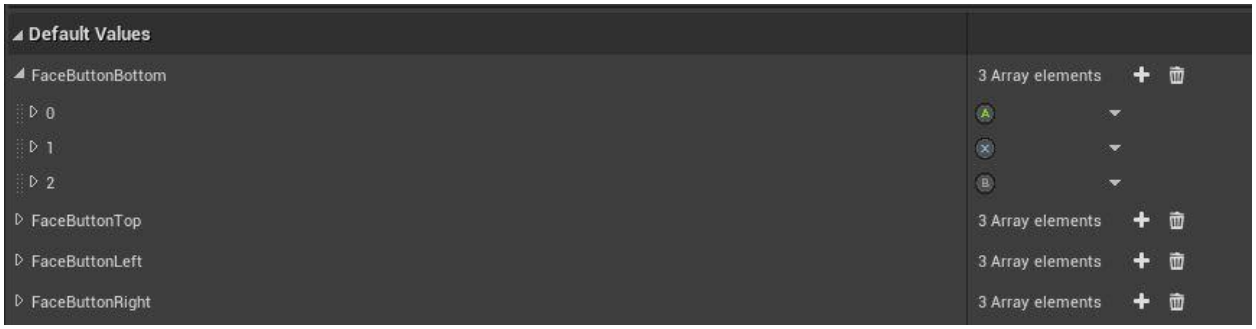
This function calls the player controller and returns the value of the “isGamepad” boolean. DW_PlayerController contains several variables related to the Gamepad and Gamepad Buttons. It also contains several events that allow for the dynamic input detection. That’s why reparenting your current Player Controller is necessary.

SetGamepadIcon [DEPRECATED]

This is the fun function. The FUNction. Anyway, just call this at any point to grab a Slate Brush of the selected Gamepad Button. All the Button Names are held in the “GamepadButtons” enum and are based on Unreal Engine’s naming convention for easily aligning Input Actions with the UI.
But the real value of this function is that it will return the Slate Brush of the currently active Gamepad Type. So using this function allows you to swap between Xbox, Playstation, and Switch prompts dynamically in-game.

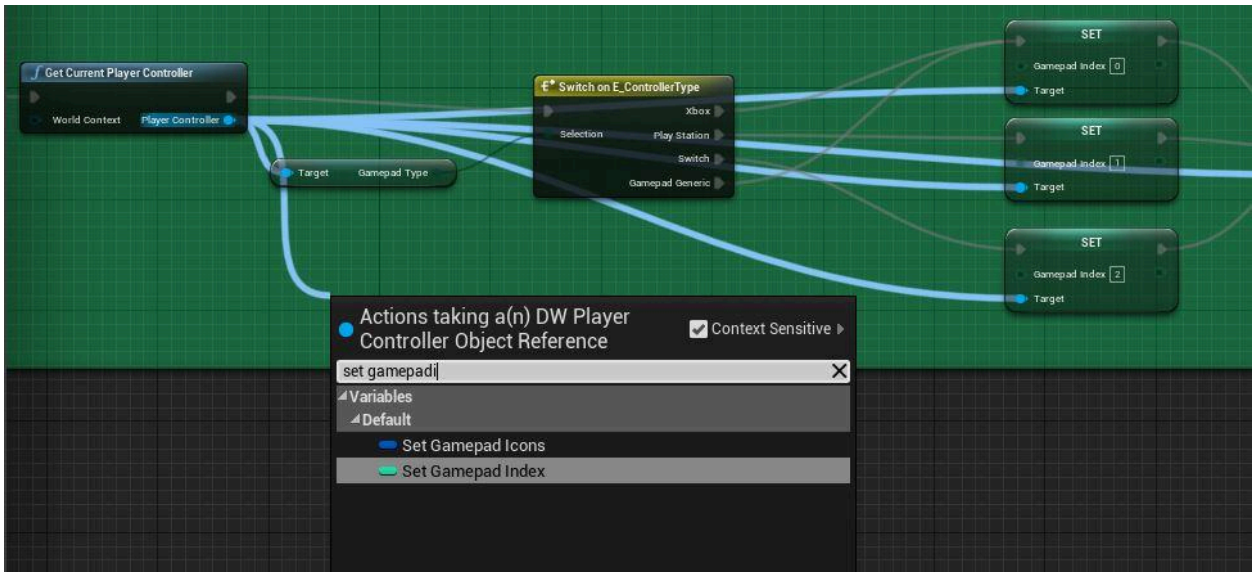
Adding Gamepad Styles

Adding your own Gamepad Styles is much more streamlined in the latest update.
First, add the name of the Gamepad Style to E_ControllerType
Then go into S_GamepadIcons and add your new icons to each Array. Keep an eye on the index of your new Icons. By default, Xbox is 0, PlayStation is 1, and Switch is 2. Your new icons will likely be at Index 3 of each array.



Next, you'll need to hop into BFL_Casting.
Enter the "SetControllerStyleIcons" function. If needed, right click on "Switch on E_ControllerType" and select "Refresh Node". Your new Gamepad Style should be on the list (if it wasn't before).

Next, drag off of the Player Controller reference and find "Set Gamepad Index".



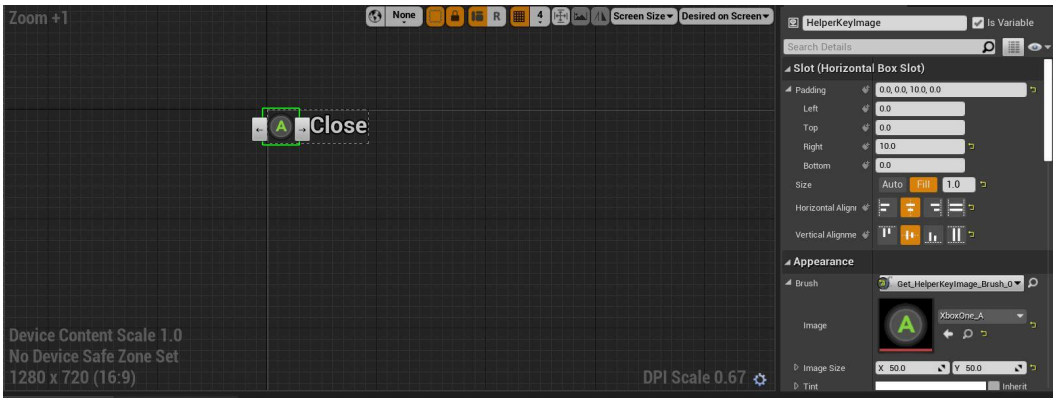
Type in the index of your icons, then connect that node to the proper Enum on the Switch as well as the "SetGamepadPrompts" function. Now you're all set to call your new Controller Type in-game!

Sample Integration

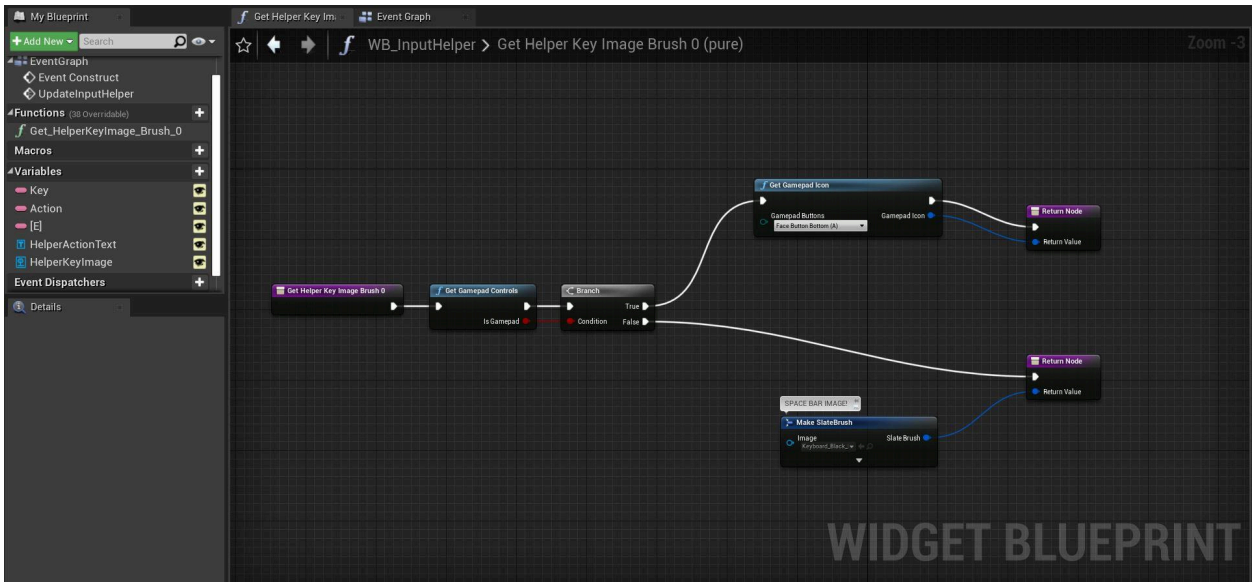
While I included a Sample HUD and ThirdPersonCharacterBP to show how to utilize the Library Functions, I thought it would be fun to do a sample integration.
Here we're using the "Dynamic Combat System" as the base project.

Following the Integration tutorial, we added Dynamic Widget Switching to the project, then reparented the player controller to DW_PlayerController. Technically, DCS doesn't use a custom controller so I just made one because I wanted to reparent instead of just assigning my Controller.

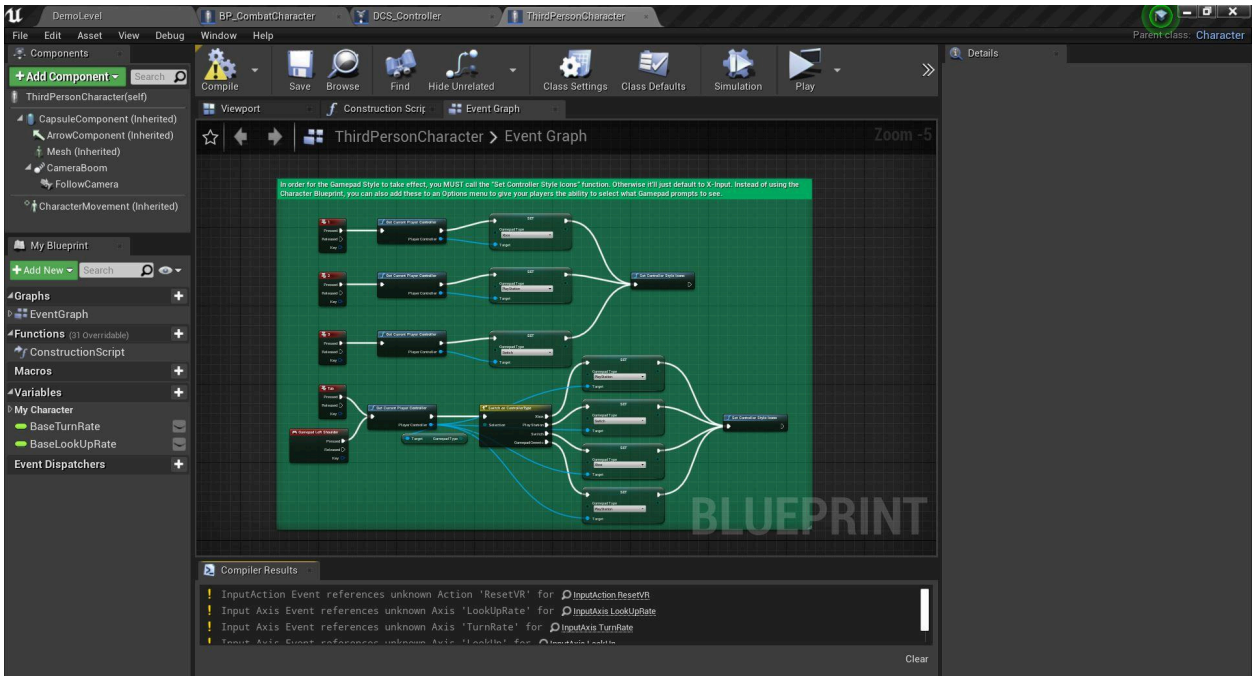
The fun began when I opened the "WB_InputHelper" widget. The first thing I did was replace the "HelperKeyText" with an image that I called "HelperKeyImage". Amazing.



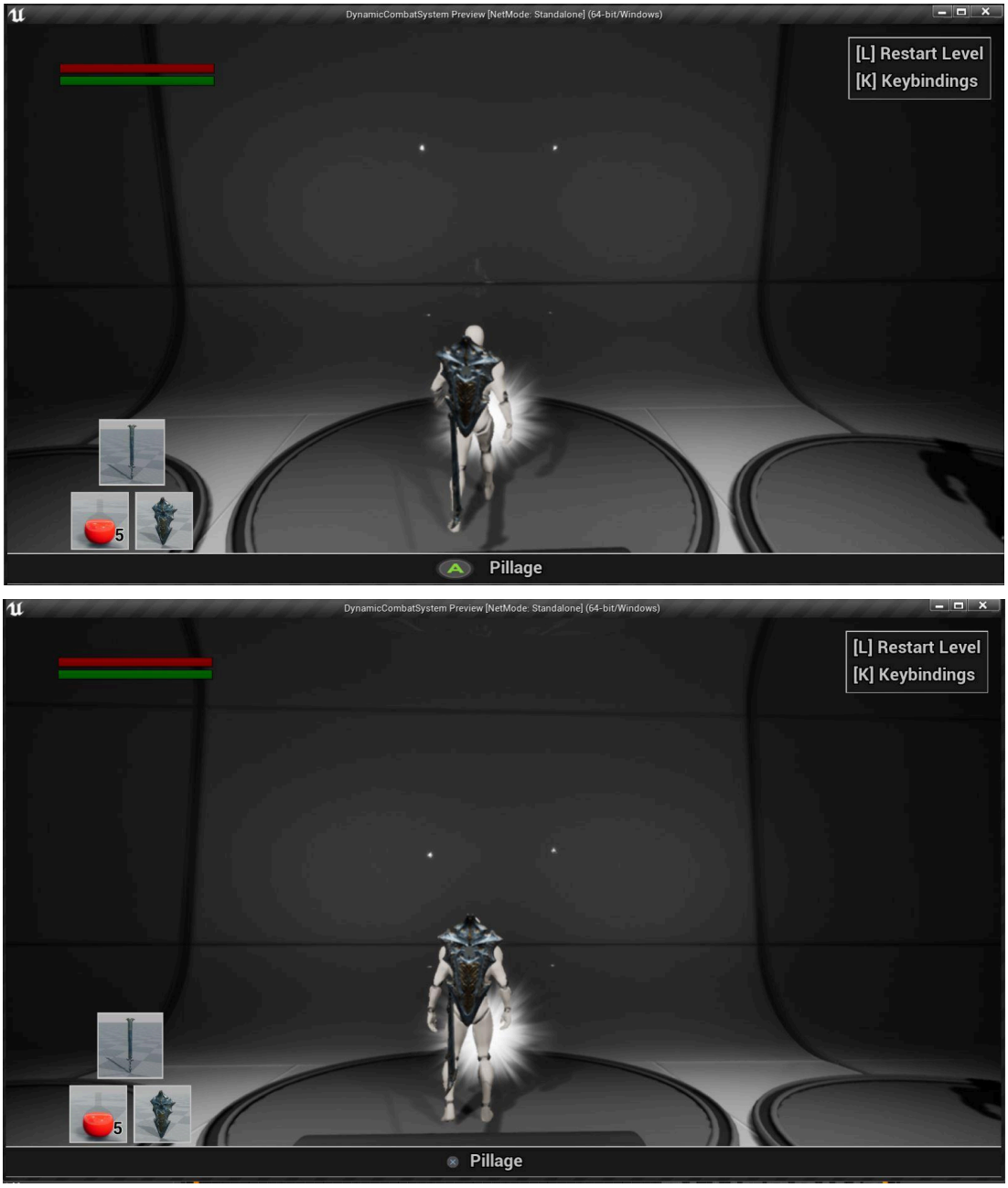
Then I removed the references to "HelperKeyText", then went back to the Design panel, selected HelperKeyImage, and added a Binding for the Brush.
From there I basically just copy/pasted the function from the Sample HUD. Except instead of "Space Bar" I used the "E" key image.



I also copy/pasted the code from the ThirdPersonCharacter to the CombatCharacterBP. I removed “Tab” as an Input Event since it’s already being used by the project. Everything else stayed the same.



Now, when you hit the Play button, you can walk over to the glowing orb of loot and instead of a horrifying mass of text greeting you, you get a nice button image. And pressing the left shoulder button swaps between Xbox, Playstation, and Switch prompts just like we want.



Some notes about this:
The Input Helpers in DCS are incredibly convoluted, and all the Input events are pulled directly from the Project settings and displayed as text. As such, since I didn't replace any of that functionality, it will always display the icon for "Face Button Bottom" regardless of the actual Input key needed.

So don't take this as the tutorial on how to ACTUALLY integrate the asset into DCS fully. This is just to show how easy it is to get the library working in an existing project. If there's any interest, I can write up an actual Integration Guide for DCS. Or really any other Asset.

Support Information

As always, if you have any questions, suggestions, rude comments, or issues, please don't hesitate to email me at jmannmarketplace@gmail.com.

Document Versioning

1.0	9/28/2020
2.0	10/01/2020
2.1	8/03/2021