UNREAL WIDGET DEVELOPMENT

TASK

- Develop a Widget which allow to insert a variable number of options with their associated details, according to the image 1 and the included Challenge document;

- The component must be easy to reuse;

- The component must support styling for artistic fine-tuning.

SCOPE

We want a reusable Widget component which allows Unreal developers to quickly insert into the game a generic list of items at left with the selected item details centered at right. The data is dynamic and the component must provide a way to insert, update and remove items at runtime.

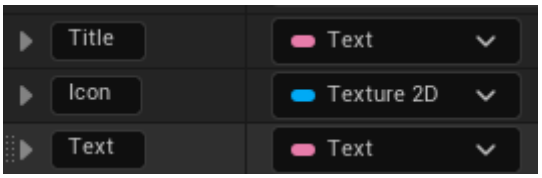
The component must support multiple input types (mouse/keyboard and gamepad) on different platforms. We should also be careful to layout the components in such a way which looks good in different screen resolutions and aspects.

ARCHITECTURE

We will make use of the built-in Unreal CommonUI plugin to allow multiple input types and visual styling.

We will make use of the built-in Unreal Viewmodel plugin to allow better separation of concerns.

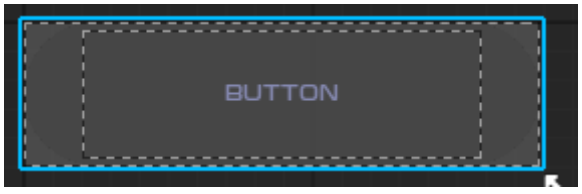
Data fields are provided to the component as a Struct:



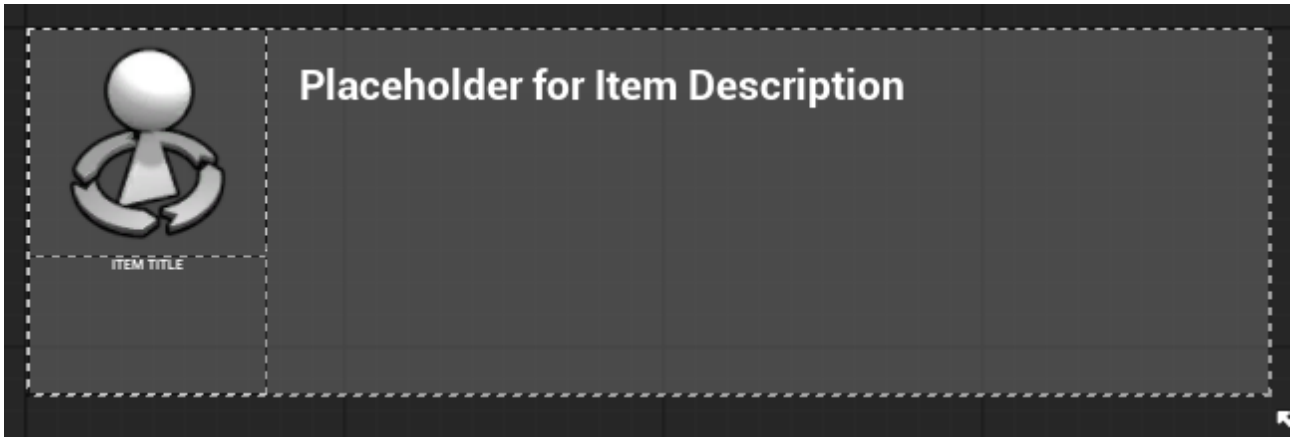
- **Title:** the item title (Text) which is displayed at the left list;
- **Icon:** the item icon (Texture2D) which is displayed at the right (details) list;
- **Description:** the item long description (Text) which is displayed at right (details) list.

There will be three main classes:

MainListItem: a sub-component based on CommonUserWidget, which will be the list at left; visually, this sub-component will only include a list of clickable buttons. The button text will be dynamically set with the item title.



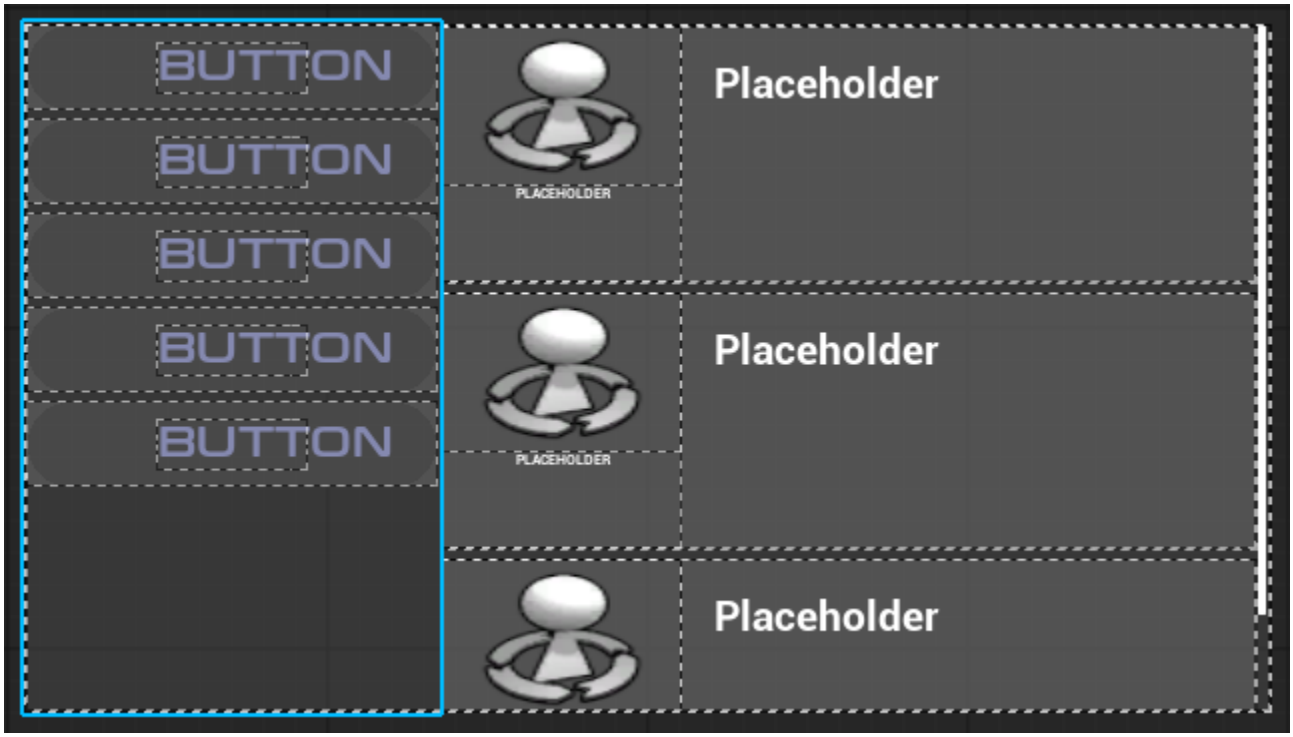
SubListItem: a sub-component based on CommonUserWidget, which will be the list at right; visually, it will show the item icon and the item description.



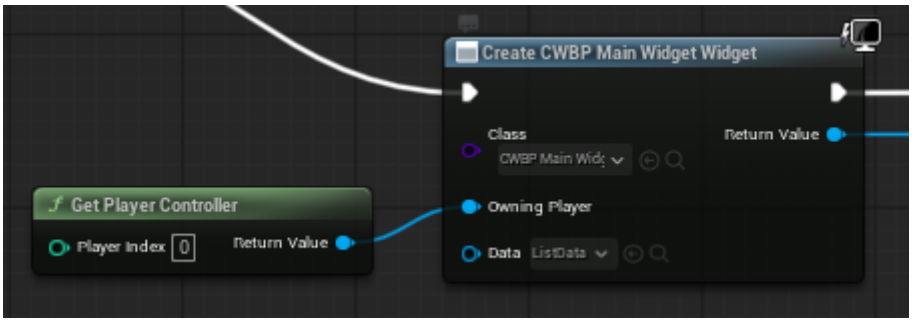
MainWidget: based on CommonActivatableWidget, this will be the main component and also the logic orchestrator. The MainWidget will contain the two sub-components, MainListItem and SubListItem.

MainWidget will accept an optional DataTable as a construction parameter, to provide immediate population. It will automatically insert the items into the two sub-components.

MainWidget will also provide two public functions: InsertItem and RemoveItem. These functions allow for insertion and removal of items at will, during gameplay.



HOW TO USE

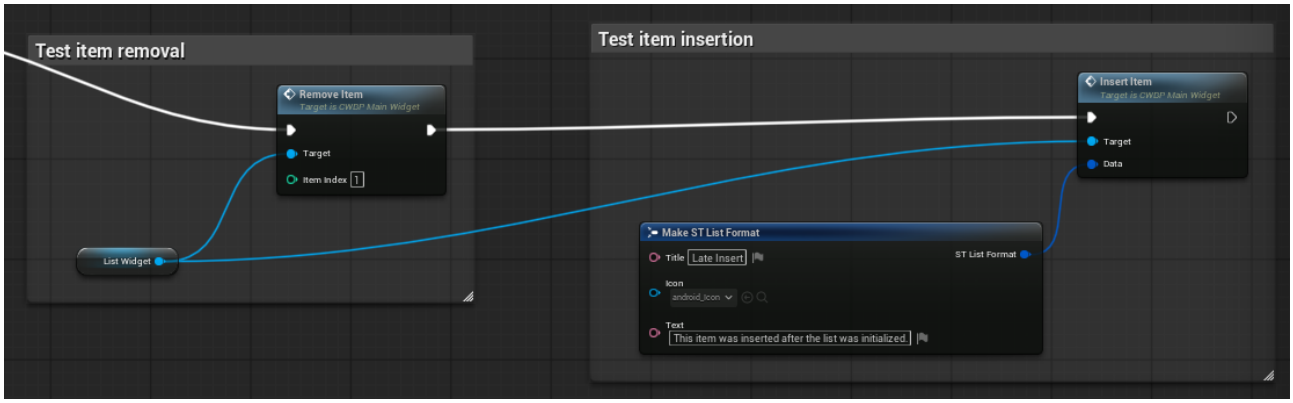


Just create the MainWidget and provide an optional ListData Data Table (a table using the specified structure described above).

The components styles can be fine-tuned by editing their corresponding files in the Styles directory:

BackgroundBorderStyle: *widget background style*
BaseButtonStyle: *left buttons background style*
BaseButtonTextStyle: *left buttons text style*
DescriptionBorderStyle: *right descriptions style*
DescriptionTextStyle: *right descriptions text style*
DescriptionTitleStyle: *right descriptions title style*
SelectedButtonStyle: *selected left buttons background style*
SelectedDescriptionBorderStyle: *selected descriptions background style*

If later on in the game you need to insert or remove items, use the two provided functions:



Localization should be done on the Input Data itself, which happens before the Widget.

POTENTIAL IMPROVEMENTS

- Move into a plugin

We could move all the widget resources into a plugin, to improve isolation and reusability.

- Multiple styles (themes) support

We could add a second Data Table with theme configurations (each field would be a filename to a style for each component -- button, text, background etc)

Like:

Name: *Theme name*
BackgroundBorderStyle: *widget background filename*
BaseButtonStyle: *left buttons background filename*
BaseButtonTextStyle: *left buttons text filename*
DescriptionBorderStyle: *right descriptions filename*
DescriptionTextStyle: *right descriptions text filename*
DescriptionTitleStyle: *right descriptions title filename*
SelectedButtonStyle: *selected left buttons background filename*
SelectedDescriptionBorderStyle: *selected descriptions background filename*

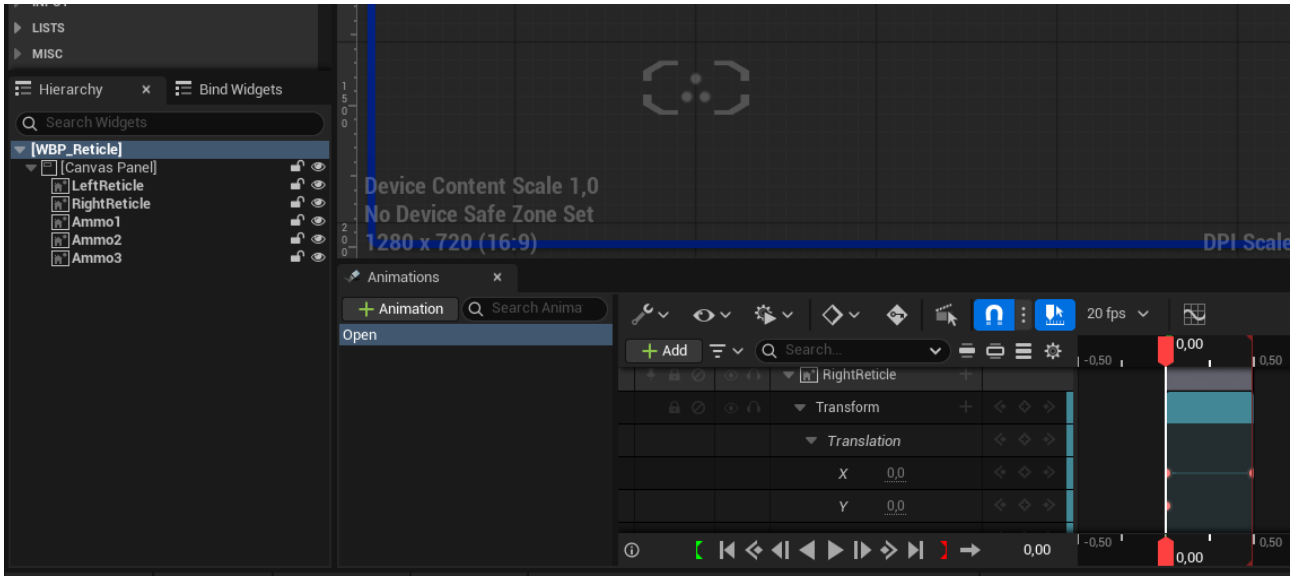
Then we could, on construction, read those styles from the selected row in the Data Table, effectively providing multiple themes for the widget.

RETICLE DEVELOPMENT

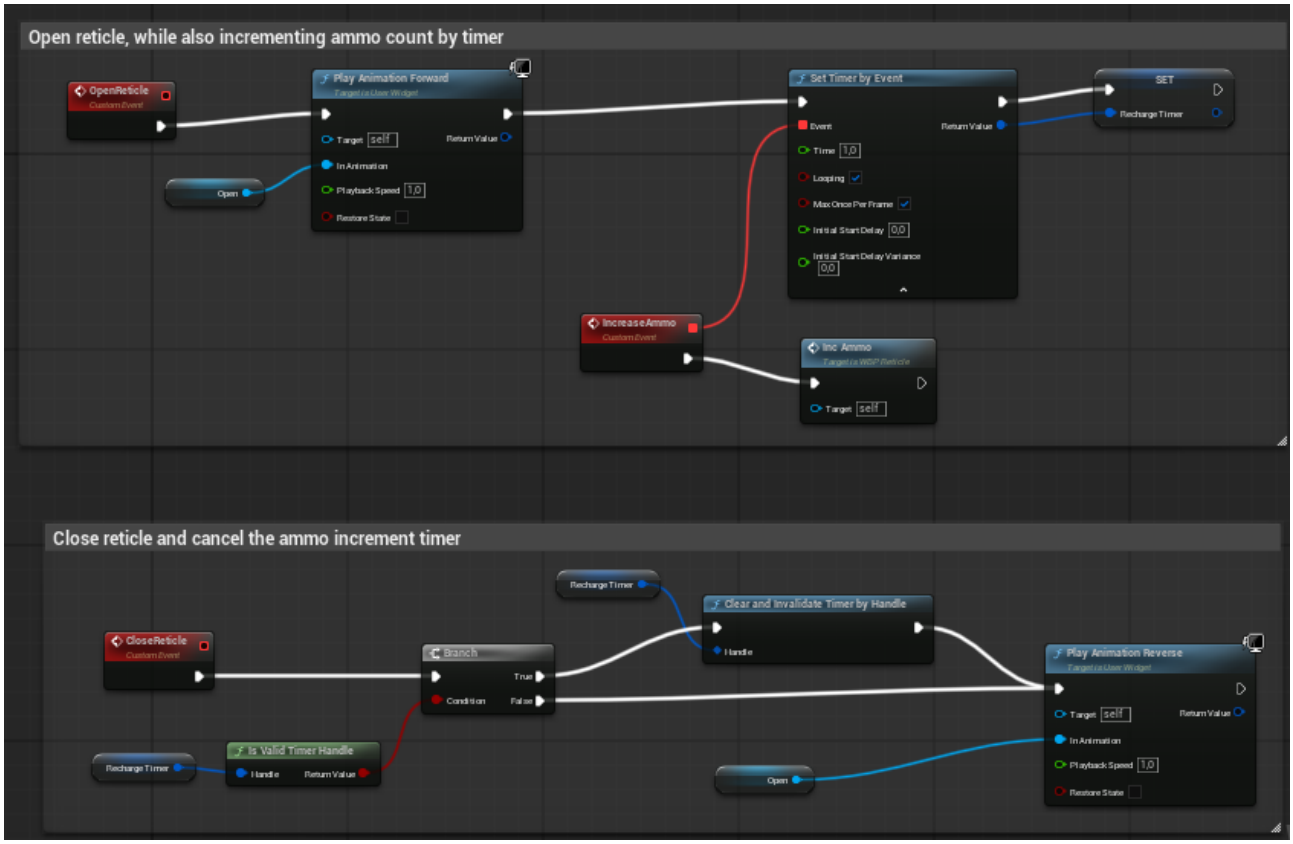
We can use a UserWidget for the aim/recharge reticle. Widget animations can be created to reproduce the UX mockup.

Three textures should be created: left bracket, ammo and right bracket:

We then can use Widget Animation to open and close the recharge mode.



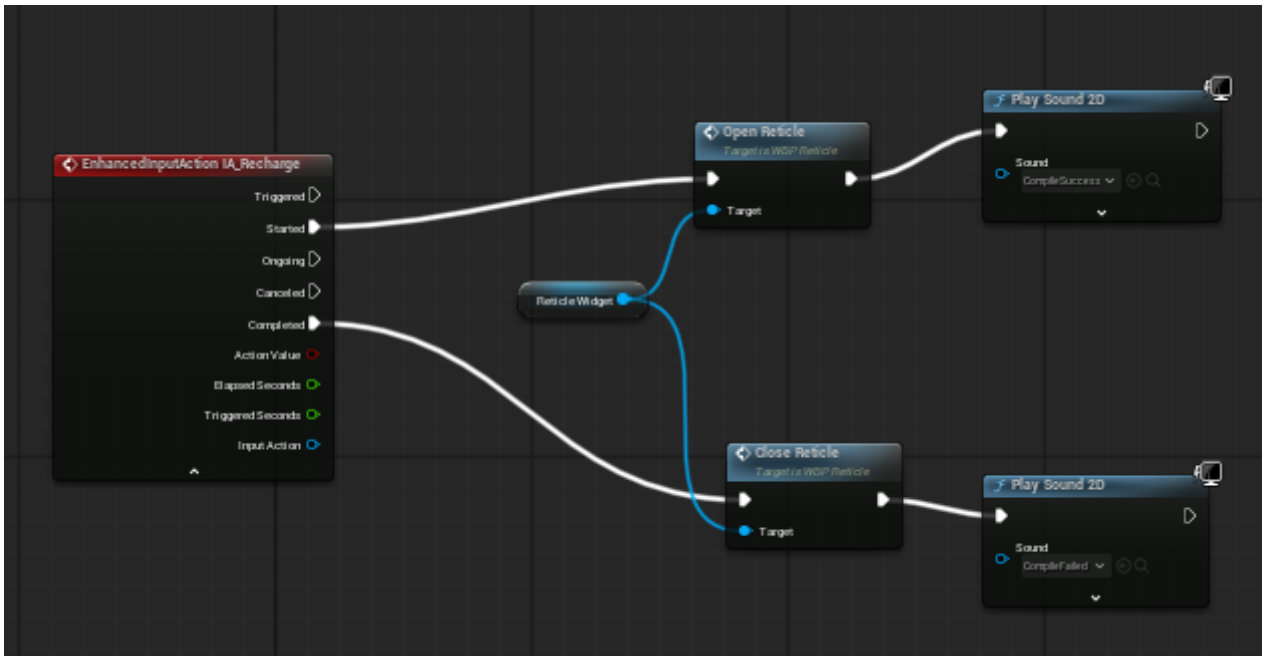
The Widget should provide two public functions OpenReticle and CloseReticle:



The OpenReticle also spawns a timer which will increment the ammo count, also updating its visual representation (available ammos become brighter on the reticle).

The CloseReticle will cancel the timer stopping the recharging.

Those functions are to be called from the Player Pawn, when the player press the Input Action (right Mouse click or Gamepad right shoulder, for ex).



In game result (no real shooting is implemented as of this version, but ammo discharging happens):

BLUEPRINT IMPLEMENTATION

Although this developer loves C++, on this specific exercise only Blueprints were used. The decision was because most of the code logic was small, and Blueprints are really quicker to implement in such cases. For production/bigger logic, C++ would most probably be chosen though.