Whiskey Business/ HUD

Architecture/Design Document

Table of Contents

Contents

Change History 2

1. Introduction 3

2. Design Goals 3

3 System Behaviour 3

4 Logical View 4

5 Process View of the HUD 7

6 Use Case View 9

Change History

Version: 0.3

Modifier: Daniel Azevedo

Date: 04 / 02 / 2025

Description of Change: Setup hearts and have them appearing on screen

Version: 0.4

Modifier: Daniel Azevedo

Date: 17 / 02 / 2025

Description of Change: HUD elements can be moved on screen with toggleHUD()

1. Introduction

This document illustrates how the QuetzalHUD class and the custom Slate classes work together. This project uses Slate instead of UMG for the HUD elements to have more customizability within the code. The three custom Slate classes created were SHealthWidget, SScoreWidget, and STextWidget.

The SHealthWidget class creates Heart icons on the HUD that will be pushed back into a vector of hearts for each player. These will be modified to have hearts removed or have hearts cut in half correlating to the health of the player.

The SScoreWidget class creates Score icons on the HUD that will increase based on the score of each player. If the player's score increases past 5, the score icons will instead be replaced by a number counting the player's score instead.

The STextWidget class creates Text on the HUD that currently increases based on the score of the player and will be set up to count down the time for a game timer.

2. Design Goals

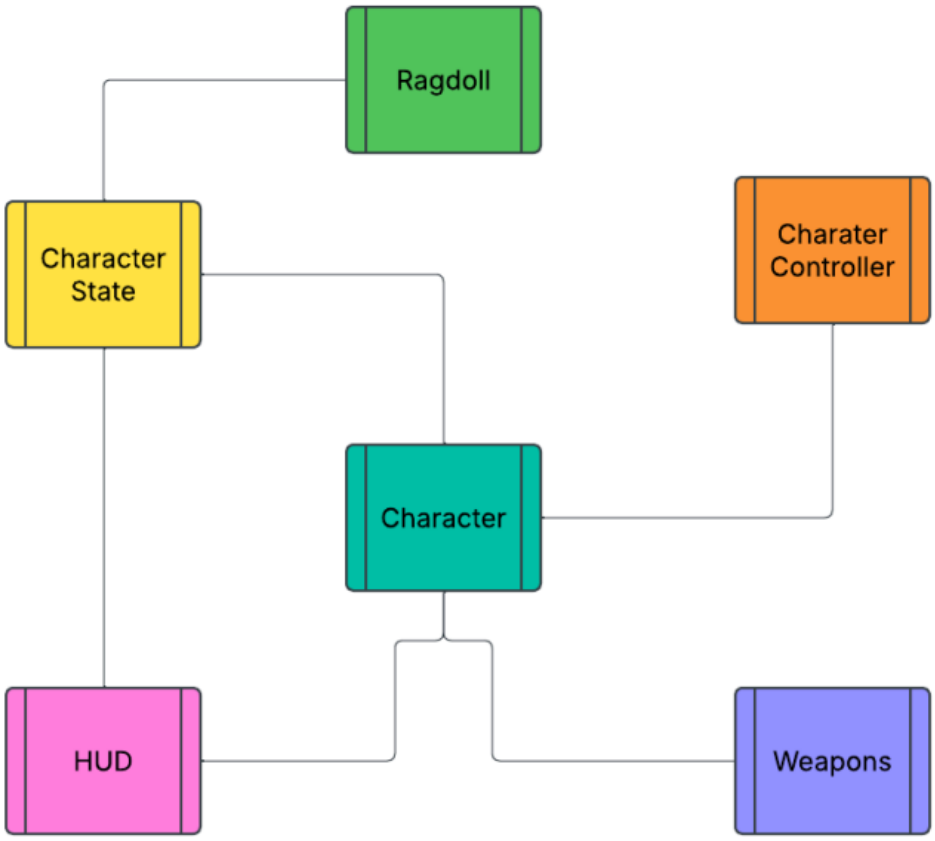
- **Ability to be toggled:** The HUD will have the ability to be toggled to allow the players to see the entire screen without any obstructions.
- **Not to clutter the screen:** When the HUD is on screen, it should be non-invasive and not block any gameplay elements.
- **Clear Designs:** Players should easily be able to determine what each HUD element represents upon seeing them.

3 System Behaviour

The HUD will be created by the GameMode and will have a pointer to the first player character to get their health when health is updated.

4 Logical View

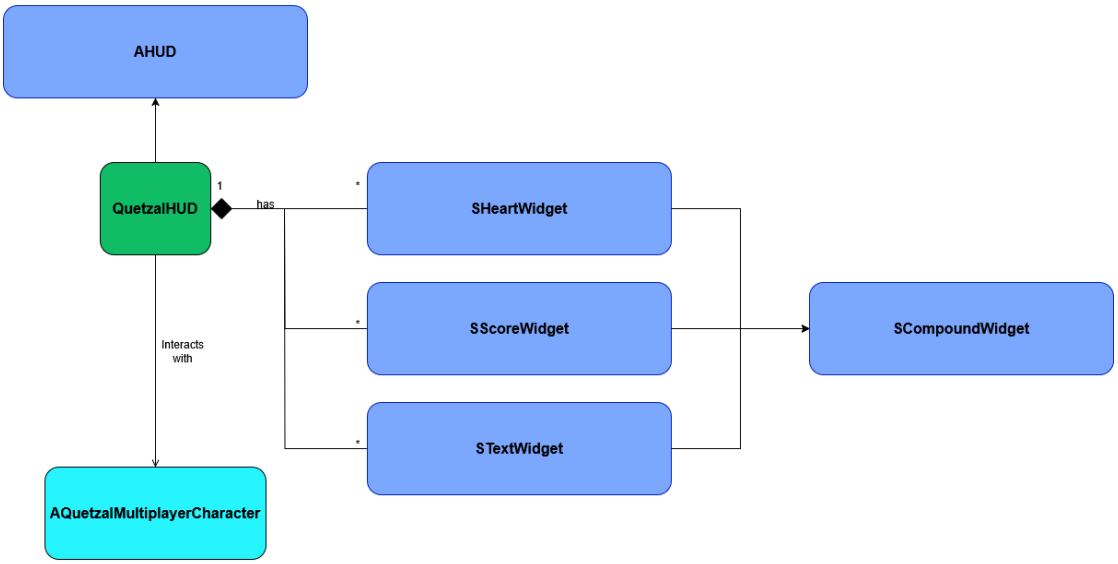
4.1 High-Level Design



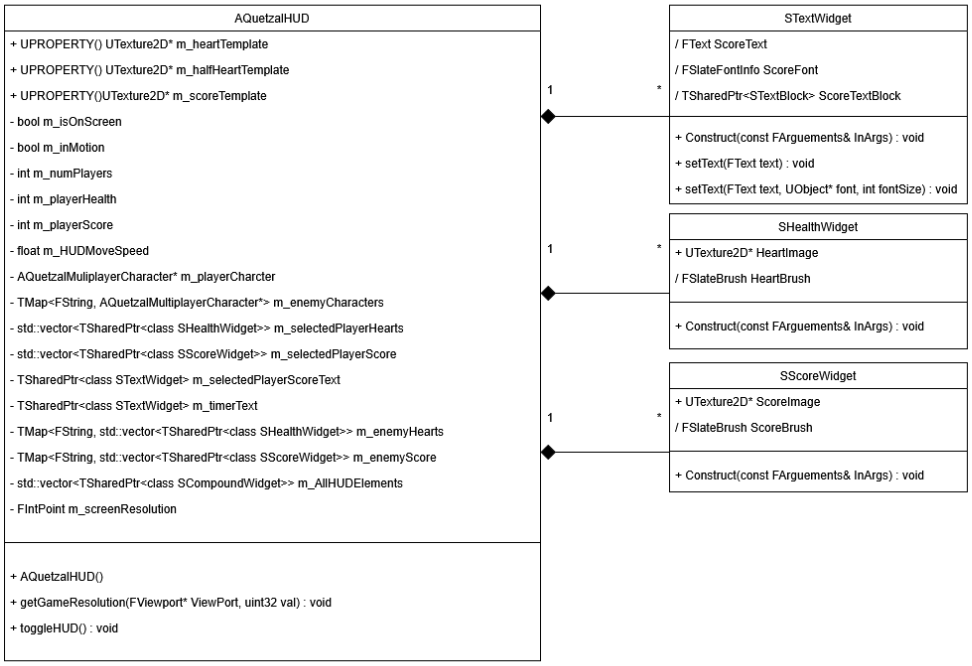
High-Level Design of varying systems in the game.

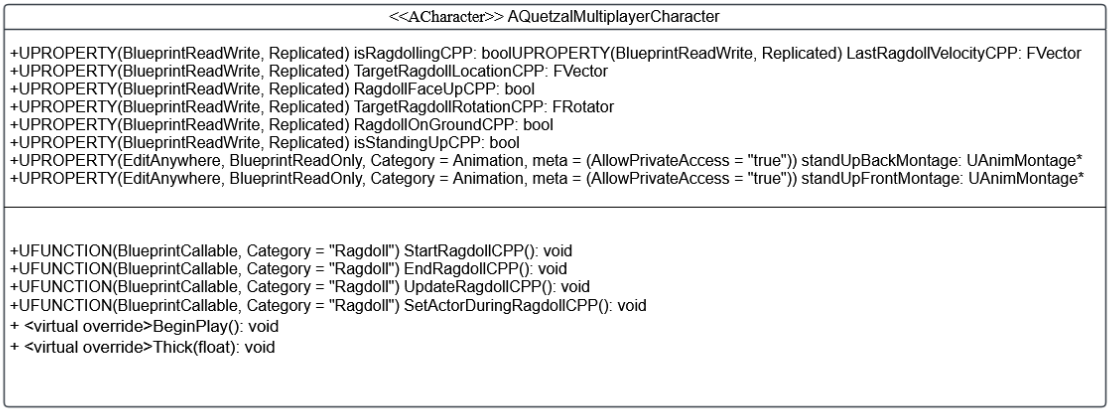
- The Character Controller will handle the input from the user that will control the player.
- The HUD will keep the players informed of the current health and score of the players using values from Character and Character State.
- Weapons will interact with the player by activating equipped weapons or receiving damage from weapons.
- Depending on the Character State the player can enter into a Ragdoll state.
- At the center is the Character interacting with all systems in some varying ways.

4.2 Mid-Level Design of the HUD



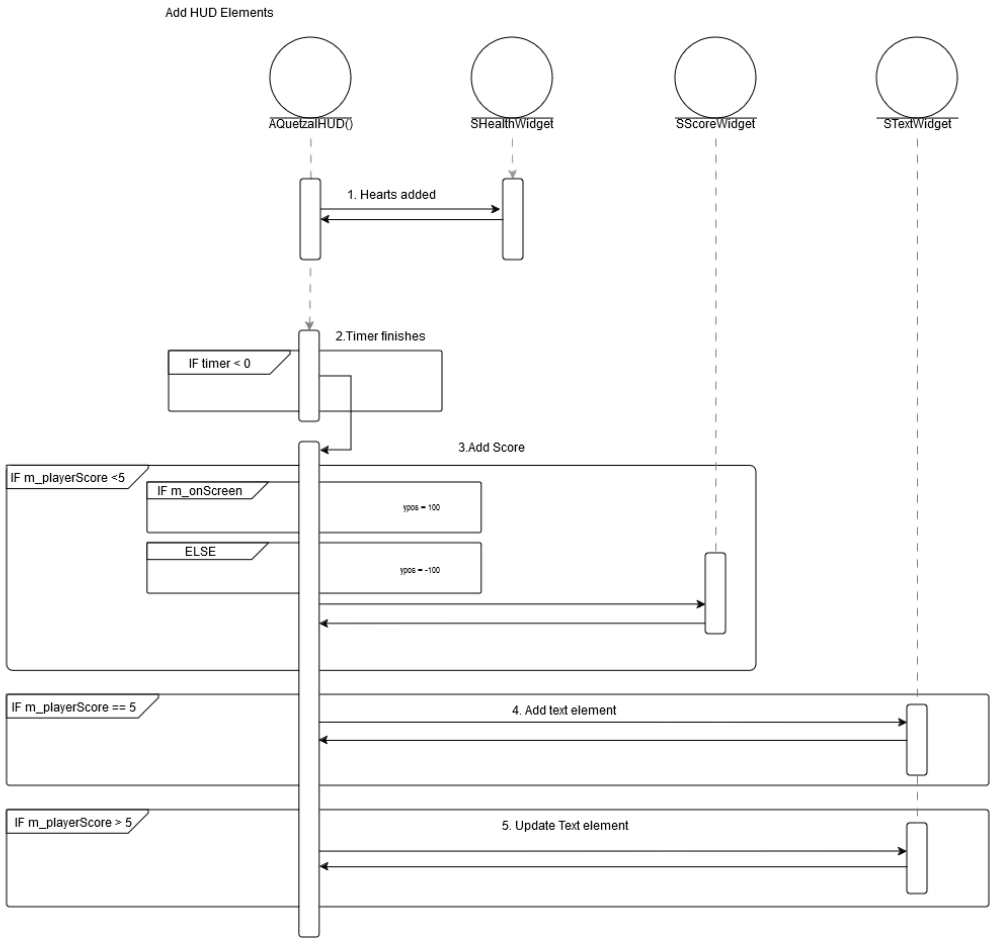
4.3 Detailed Class Design of the HUD





5 Process View of the HUD

Add HUD Elements



The HUD elements are all made through Slate widgets created in AQuetzalHUD BeginPlay() function.

Process Steps:

1. **Hearts added**
    - The QuetzalHUD BeginPlay() function will create all the player and Enemy hearts. The main player's hearts are the biggest on their screen.

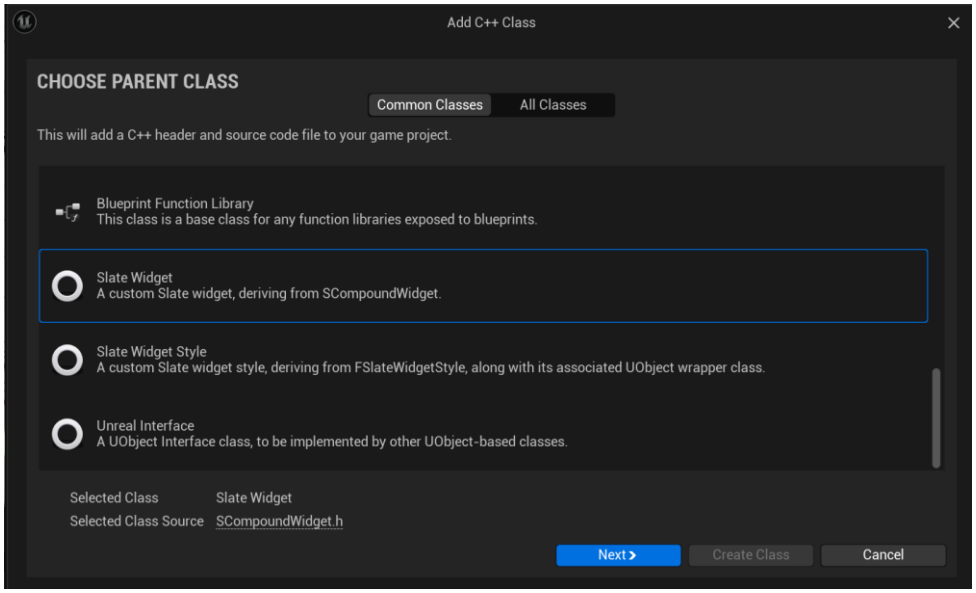
- This function checks how many players there are through the GetWorld()->GetNumControllers() function and calculates how many sets of hearts it needs to show.
- Right now, the hearts won't be updated but each set of hearts will correspond to each player.

2. **Timer finishes**
    - To demonstrate the add score function, when the timer gets triggered, it will call the addPlayerScore() function.
3. **Add Score**
    - The addPlayerScore() function will add score bottle tally icons on screen until the player's score reaches 5.
4. **Add Text Element**
    - When the player's score hits 6, it will remove 4 of the 5 tallied score and instead count up with a text counter.
        - Here m_selectedPlayerScoreText gets created and will get set with each added score
        - The font and font size will also be set here through the m_font variable initialized in the QuetzalHUD Blueprint
5. **Update Text Element**
    - For every score added past 6, the on screen text will update with the score of the player.
        - This will be done for each player but currently only is functional for the main player.

6 Use Case View

If you wanted to create new screen widgets:

- Create a new C++ class with a SCompoundWidget parent



- Set up the .h file with all the variables you want (in this case we just want a brush to display the image on screen)
- Also set up the SLATE_ARGUEMENT to set what your Widget will take in when created (in this case we want to take in a texture to render on screen)

```
12
13    #pragma once
14
15    #include "CoreMinimal.h"
16    #include "Widgets/SCompoundWidget.h"
17
18    /**
19     *
20     */
21    class QUETZALMULTIPLAYER_API SHealthWidget : public SCompoundWidget
22    {
23    public:
24        SLATE_BEGIN_ARGS(SHealthWidget)
25            :_HeartImage(nullptr)
26        {}
27            SLATE_ARGUMENT(UTexture2D*, HeartImage)
28
29        SLATE_END_ARGS()
30
31        /** Constructs this widget with InArgs */
32        void Construct(const FArguments& InArgs);
33
34    protected:
35        FSlateBrush HeartBrush;
36    };
37
```

- In the .cpp file set the resource we will be using (in this case use the parameter we take in to set the brush to the texture we will select)

- Set the size of the image to the width and height and set the draw type as an image
- Finally, to set up the widget we will create a child slot and create a new Slate Image using our brush (here you can also add an offset and many other functionalities but for our case, we just want an image)

```
BEGIN_SLATE_FUNCTION_BUILD_OPTIMIZATION
void SHealthWidget::Construct(const FArguments& InArgs)
{
    //This creates sets the brush that is established in the Blueprint for QuetzalHud
    HeartBrush.SetResourceObject(InArgs._HeartImage);
    HeartBrush.ImageSize.X = InArgs._HeartImage->GetSurfaceWidth();
    HeartBrush.ImageSize.Y = InArgs._HeartImage->GetSurfaceHeight();
    HeartBrush.DrawAs = ESlateBrushDrawType::Image;


    //This creates new UI elements with the brush initialized above
    //Most (if not all) of the animations will be done through QuetzalHUD.cpp where the HUD elements are created
    ChildSlot
    [
        SNew(SImage)
            .Image(&HeartBrush)
    ];

}
END_SLATE_FUNCTION_BUILD_OPTIMIZATION

```

- In our custom HUD class, create a UPROPERTY UTexture2D* variables to store the images we will create and display on screen.

```
UCLASS()
class QUETZALMULTIPLAYER_API AQuetzalHUD : public AHUD
{
    GENERATED_BODY()

public:

    AQuetzalHUD();
    //HUD Element Templates

    /** Heart Template. */
    UPROPERTY(EditAnywhere, Category = "UI Assets")
    class UTexture2D* m_heartTemplate;

    /** Half Heart Template. */
    UPROPERTY(EditAnywhere, Category = "UI Assets")
    class UTexture2D* m_halfHeartTemplate;

    /** Score Template. */
    UPROPERTY(EditAnywhere, Category = "UI Assets")
    class UTexture2D* m_scoreTemplate;

    /** Font Template */
    UPROPERTY(EditAnywhere, Category = "UI Assets")
    class UObject* m_font;
    //TODO Make custom Font or find one more suited for our game

```

- For each heart we want to make we need to:
    - Set the image from a template

- Set the content scale using the aspect ratio to ensure HUD elements don't stretch when the aspect ratio changes (see getGameResolution function),
- Set the transform position
- Add the hearts to a vector of the player hearts (to be able to modify them if the player takes damage)
- Add the hearts to a list of all the heart elements (to be able to move them off screen when needed)

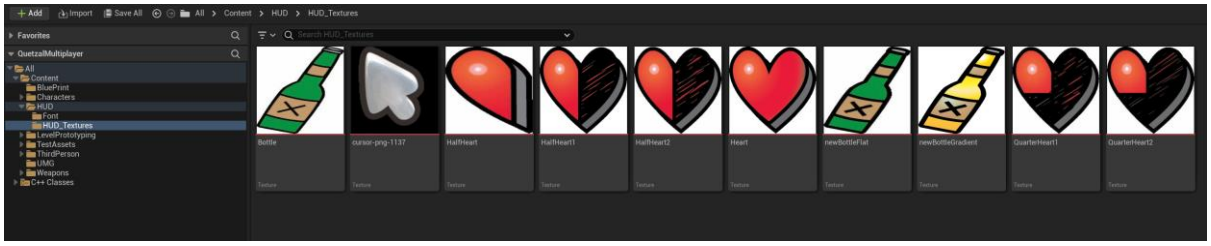
```
//Initializing Player's Hearts
TSharedPtr<class SHealthWidget> HealthWidget;

for (int i = 0; i < MAX_HEARTS; i++)
{
    HealthWidget = SNew(SHealthWidget).HeartImage(m_heartTemplate);                                    //Setting the heart image from blueprint
    HealthWidget->SetContentScale(FVector2D(1 / aspectRatio, 1));                                      //Setting the heart size
    FSlateRenderTransform transHearts(INITIAL_SCALE, FVector2D(INITIAL_POS.X + DISTANCE_BETWEEN_HEARTS * i, INITIAL_POS.Y));  //Setting the transform pos of the heart
    HealthWidget->SetRenderTransform(transHearts);                                                     //Setting the heart pos
    m_selectedPlayersHearts.push_back(HealthWidget);                                                   //Pushing the hearts into a vector of all the player's hearts
    m_AllHeartElements.push_back(HealthWidget);                                                        //Pushing the hearts into a vector of all hearts
}
```

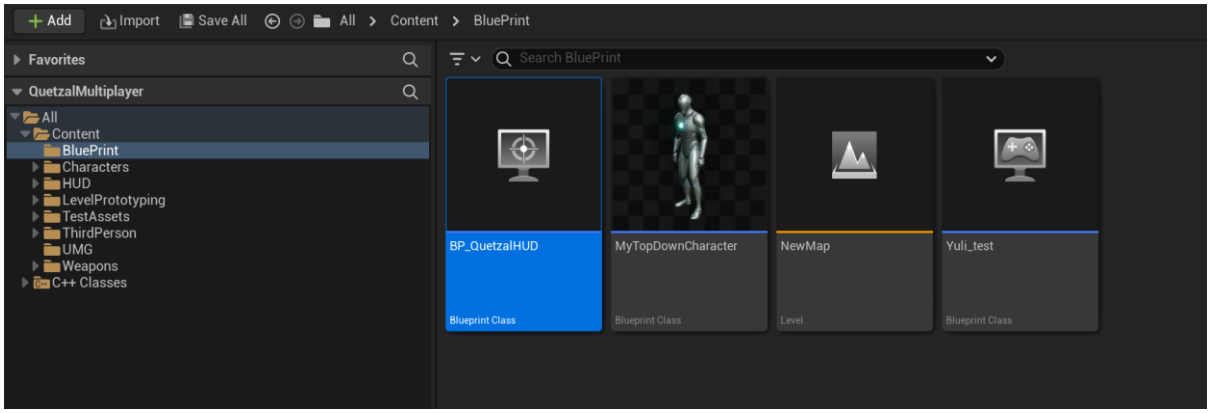
- All widgets must be initialized through the game's viewport

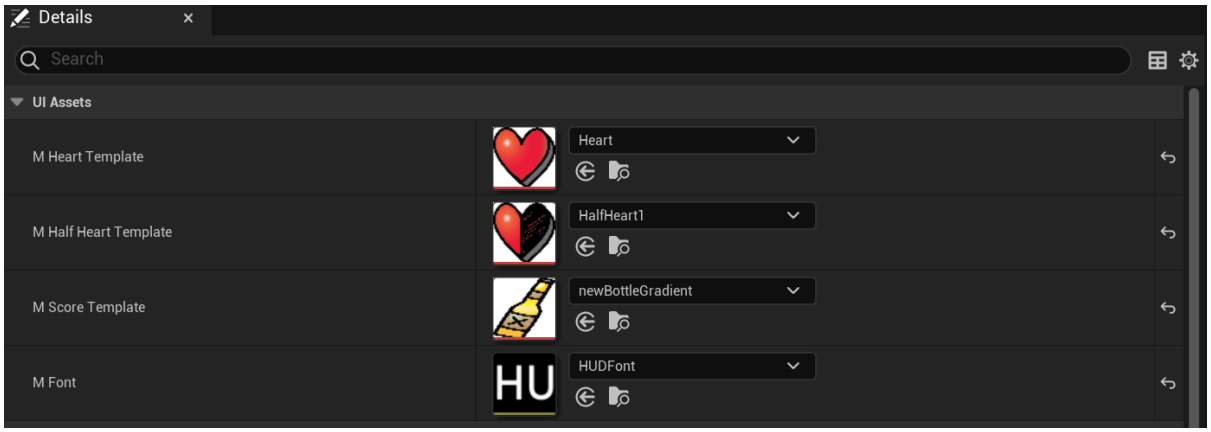
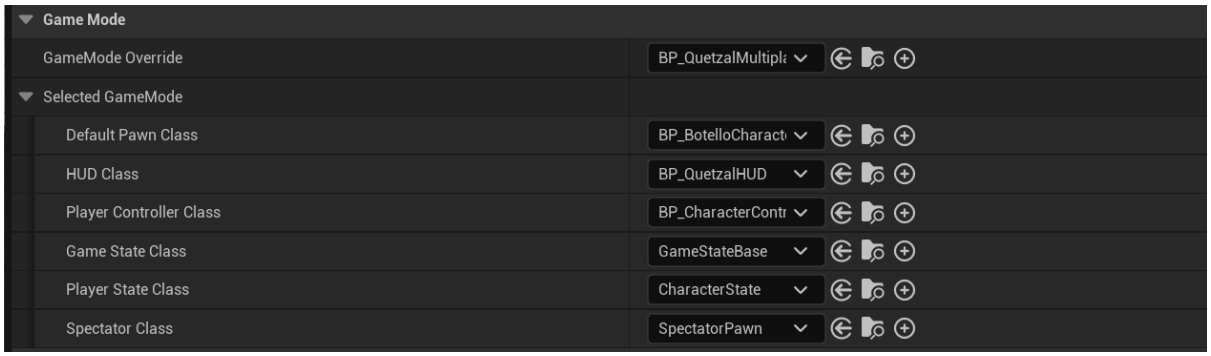
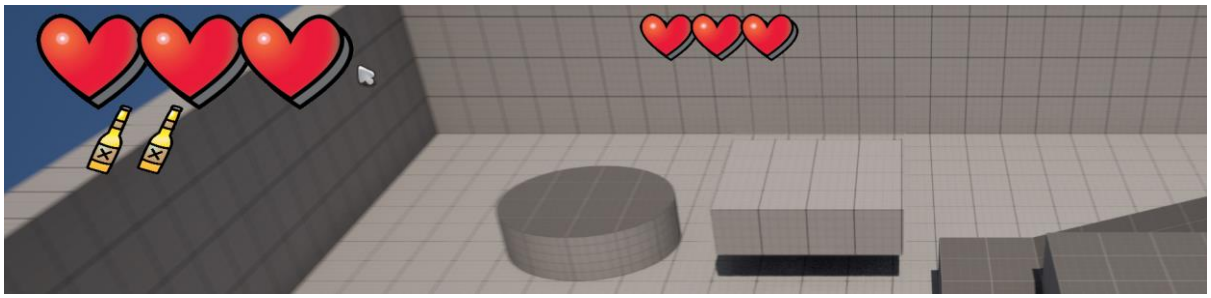
```
//Adding all content to the viewport (this viewport function needs to be called for each HUD object that gets added)
for (auto widget : m_AllHeartElements)
{
    viewPort->AddViewportWidgetContent(widget.ToSharedRef());
}
```

- Import your HUD assets



- In our BP_QuetzalHUD, select the variables created in our custom HUD.h file and set the textures we made to our texture variables.





- Finally set the custom HUD to the used HUD class in your Game Mode override