HOUR OF CODE

UNREAL ENGINE

POLISH AND PUBLISH YOUR GAME:
WORKING WITH WIDGETS AND THE HEADS-UP DISPLAY IN UNREAL ENGINE

STUDENT GUIDE

Activity 5

Polish and Publish Your Game: Working with Widgets and the Heads-up Display in Unreal Engine

---

Overview

Unreal Engine is an immersive 3D game engine that powers some of the most popular video games in the world. While some games require teams of professionals to produce a final product, you can get started with no experience. Rather than focusing solely on basic concepts of computer programming, you'll jump straight into building a video game. This series of activities is designed to guide you through the process of creating a 3D video game while highlighting the important computing concepts along the way. We hope the promise and excitement of building a video game will give you the context and motivation to learn essential computer programming concepts.

This entire program contains five (5) Hour of Code activities that combine all the concepts and instructions you need to complete a 3D video game that you can play and share with friends. The activities and included project files are designed to be done in order from beginning to end or you can select any single activity to complete individually. Each activity holds exciting new challenges and discoveries that unlock the power of game development using Unreal Engine.

About this Activity

What are some of your favorite games? Is the information about the game displayed to the player in a way that lets you understand what is happening? Are you aware of where your score and health are located? Can you visually see what items you have in your inventory? If you have special abilities, is there a way for you to see whether they are currently available or if they are in a cool-down period? These are all elements of the User Interface (UI) or Heads-up Display (HUD). When implemented effectively, they add to the positive user experience in the game.

It's your turn to become a Game Developer and use Unreal Engine; one of the industry-standard tools for Game Development. To begin adding hud functionality to your game, You will add UI and HUD elements to enhance the player experience. Also, you will get to modify your game to make it more your own. Finally, you will package (publish) your game so you can share it with the world!

In this activity, you will be adding some polish to your game.

We will focus on the following aspects:

- Heads-up Display (HUD)
    - Coin and Gem counter
    - Key icon
    - Jump Boost icon
- Packaging the game to be played on Windows PCs.
    - Adding a way for the player to close the game once it has been completed.

- Connecting existing code.
    - The code provided will give you a starting point for further independent research; it is intended to give you a starting point as you begin to create your games.

Adding and completing the above functionality will make the game more engaging as well as helping players understand what is going on.

This last push will take us deeper into the code and will be the most rewarding. You are 80% of the way to having a working level, and this last 20% will make your game stand out. These "quality of life" additions require more playtesting, so be prepared to engage your peers in the process!

Getting Started

If you have not downloaded **Unreal Engine** and the **Hour of Code Project**, see the Getting Started Guide to do so. If you have, open the project and begin!

Programming Concepts

We will be introducing a new type of Blueprint in this activity, the **Widget** Blueprint. Think of a **Widget** as a HUD (Heads-up Display) element. They are displayed on-screen for the player to see, and because they are a type of **Blueprint**, they can contain code that changes what they display.

Most of the code and widgets in this activity have all been created in advance, so you will only need to change a few settings to make them show up on-screen. To learn more about **Widgets** and how to create them, check out the "Your First Hour with UMG" course on Unreal Online Learning.

The complex task of creating a game is made up of smaller, much simpler tasks. By breaking all these tasks up into manageable parts, creating a game becomes much easier. Technology is constantly upgrading and adapting, and by keeping up with it as it changes, you will become more marketable as you progress through your career.

Again, this last push will make it worth the time. Your skillset is about to become more robust, and you will have new tools to help you become a more successful designer.

Preparing the Activity

If you have completed Activity 1, 2, 3, & 4 already, you can skip this section if you would like to build on the work you've already completed.

If you are starting here at Activity 5 or would like to have a fresh start, please follow these instructions for starting a new project.

Since we are starting at Activity 5, you will need to load the completed sample versions of Activities 1, 2, 3, and 4, to start your game from the beginning. Completed samples for each level are included in the sample project. Follow the steps below to load the content from levels 1, 2, 3, and 4.

The completed levels are found in the **Levels** panel. This can be displayed by navigating to **Windows > Levels**. Click and drag the window by the tab to dock it next to the **World Outliner** so we can use it later.

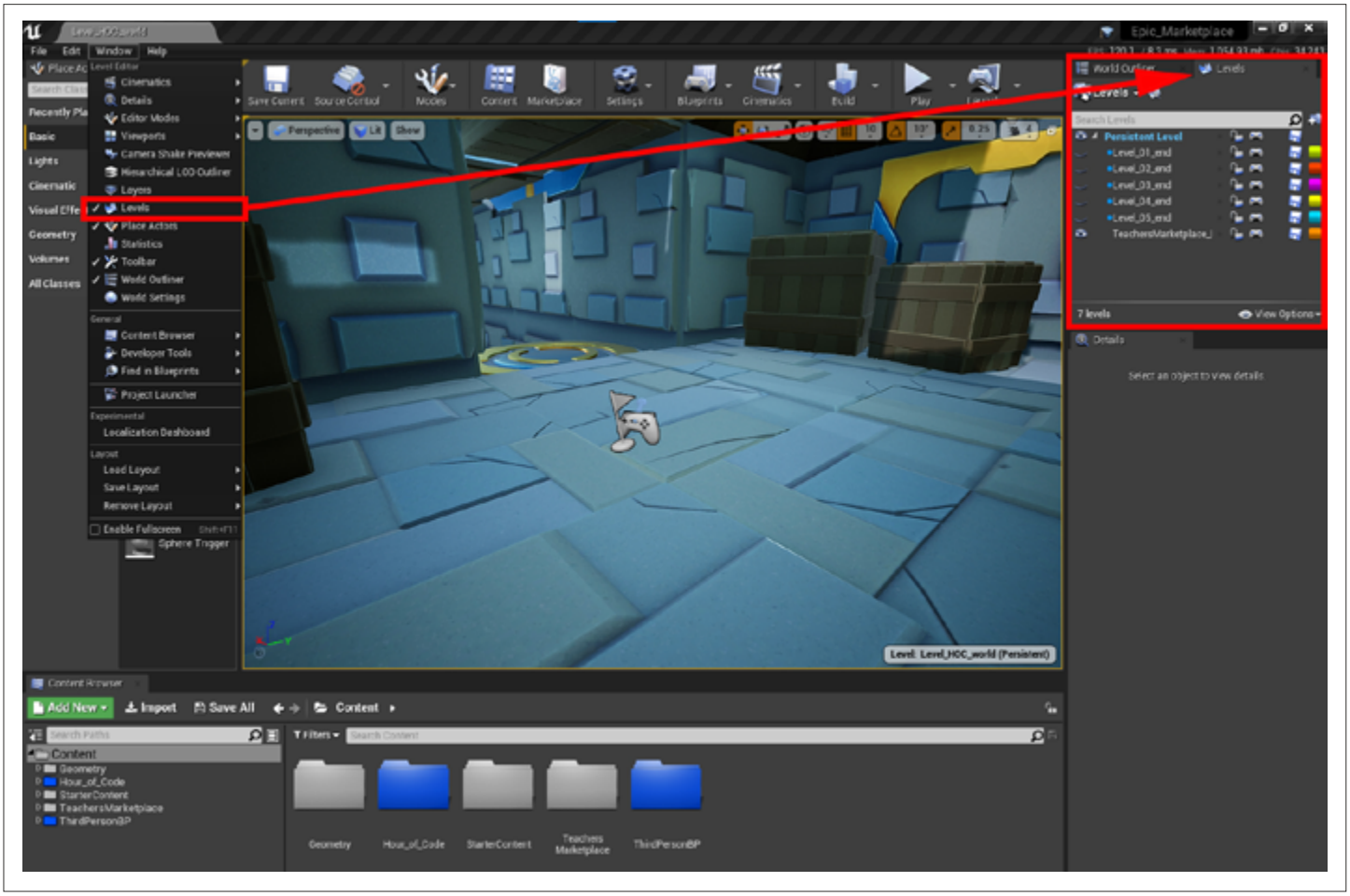

Fig. 001a – Move the Levels panel next to the World Outliner panel.

Load the completed level by navigating to the **Levels** panel and **right-clicking** on **Level_01_End** and choosing **Change Streaming Method > Always Loaded**. This will load the example level when you play the game. Repeat this step for **Level_02_End**, **Level_03_End** and **Level_04_End**. You don't need to load the other levels, so you may choose to leave the **Blueprint** option checked for now.

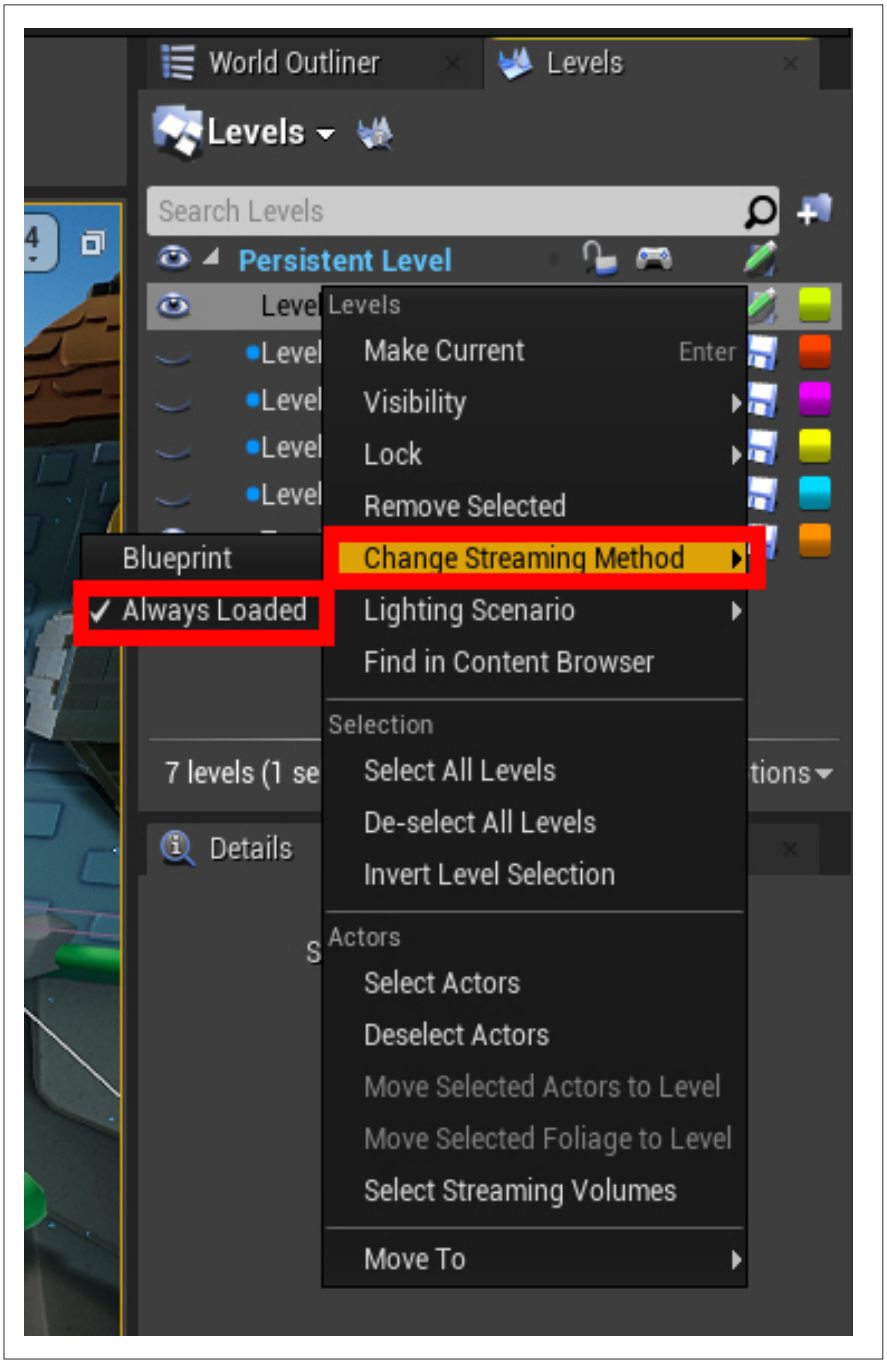

Fig. 00lb - Loading the Level l, 2, 3, & 4 samples if you didn't create your own.

NOTE: Make sure **Persistent Level** is shown in **bold** text. If you double-click one of the other levels, i.e. **Level_01_end**, that level becomes active. This means that when an actor is added to the **Viewport** it will be added to the **Level_01_end level**. The other levels will still be visible, but not available for editing in the **Viewport**. Within these activities we will only be adding assets to the **Persistent Level**. The level name that is in **bold text** is the active level.

Troubleshooting

If an actor is added to any level other than the **Persistent Level**, simply select the actor, and **right mouse click Persistent Level** and choose **Move Selected Actors to Level**.

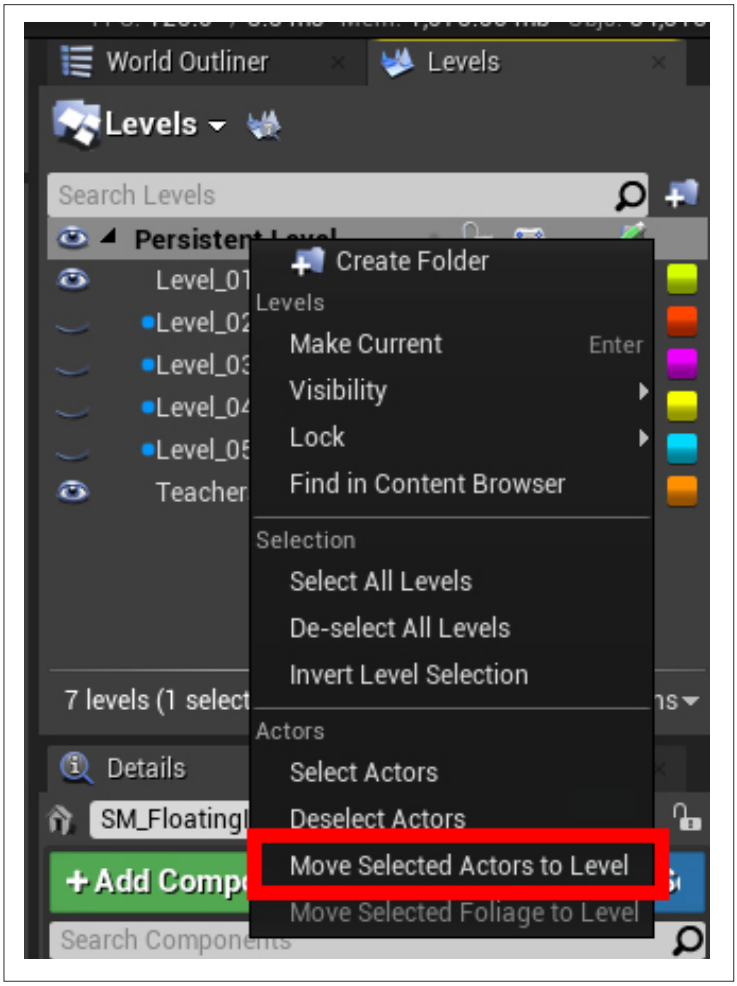

Fig. 001c – Moving actors to the Persistent Level if accidentally placed in the Level_## level.

Time-saving Tips

We have added a few helpful things to the project. Take a moment to review these tips.

Camera Bookmarks

Camera bookmarks are useful for quickly changing your location in the editor. To see how they work, first click anywhere inside the **Viewport**, then press the number **1** or **2** at the top of your keyboard. You will notice that the camera will jump to specific locations.

1 = beginning location of activity 1
2 = beginning location of activity 2

This will allow you to quickly move around your level without having to manually navigate from one place to another. Buttons 1 – 7 are assigned to important bookmarks for this course, and you can assign your own camera bookmarks by pressing the **Ctrl + any number** at the top of your keyboard. Try using numbers 8 – 0.

Your Turn: Set another camera bookmark somewhere in the level using **Ctrl + 8**. To check if it worked, you can revisit other camera bookmarks by pressing a number between 1 and 7. Now press 8 on the keyboard and it should bring you to the bookmark you created. Did it work?

Play From Current Location

Did you know that you can start the game from where the camera is currently located? To set this up, simply open the drop-down menu next to the **Play** button and choose the **Current Camera Location** option. This will save you a lot of time when playtesting your levels. Just be aware that if you start the game above a void, your player will fall into the void and respawn.

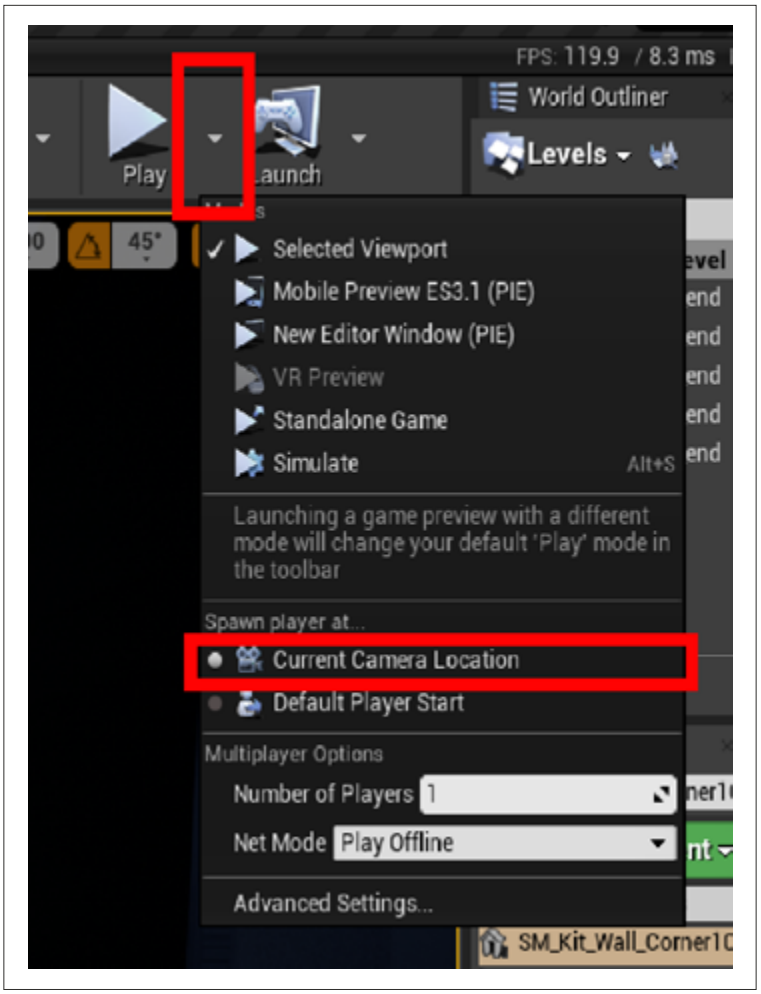

Fig. 00Id – Setting Play button to start from current camera location.

Let's Play!

Let's place a key in front of the locked door so we can start to explore the information that can be displayed on the HUD.

To jump to the final locked door, you can click in the **Viewport** and press the number **5** on the keyboard to jump to the Camera Bookmark. To find the key, open the **Content Browser** and navigate to the **Content > Hour_of_Code > Blueprints** folder. Drag the **BP_Key_Activity_4_end** into the level front of the door. (We are placing the key in front of the door to make testing more convenient. When designing the final game you'll eliminate extra keys and find a nice place to hide the key.)

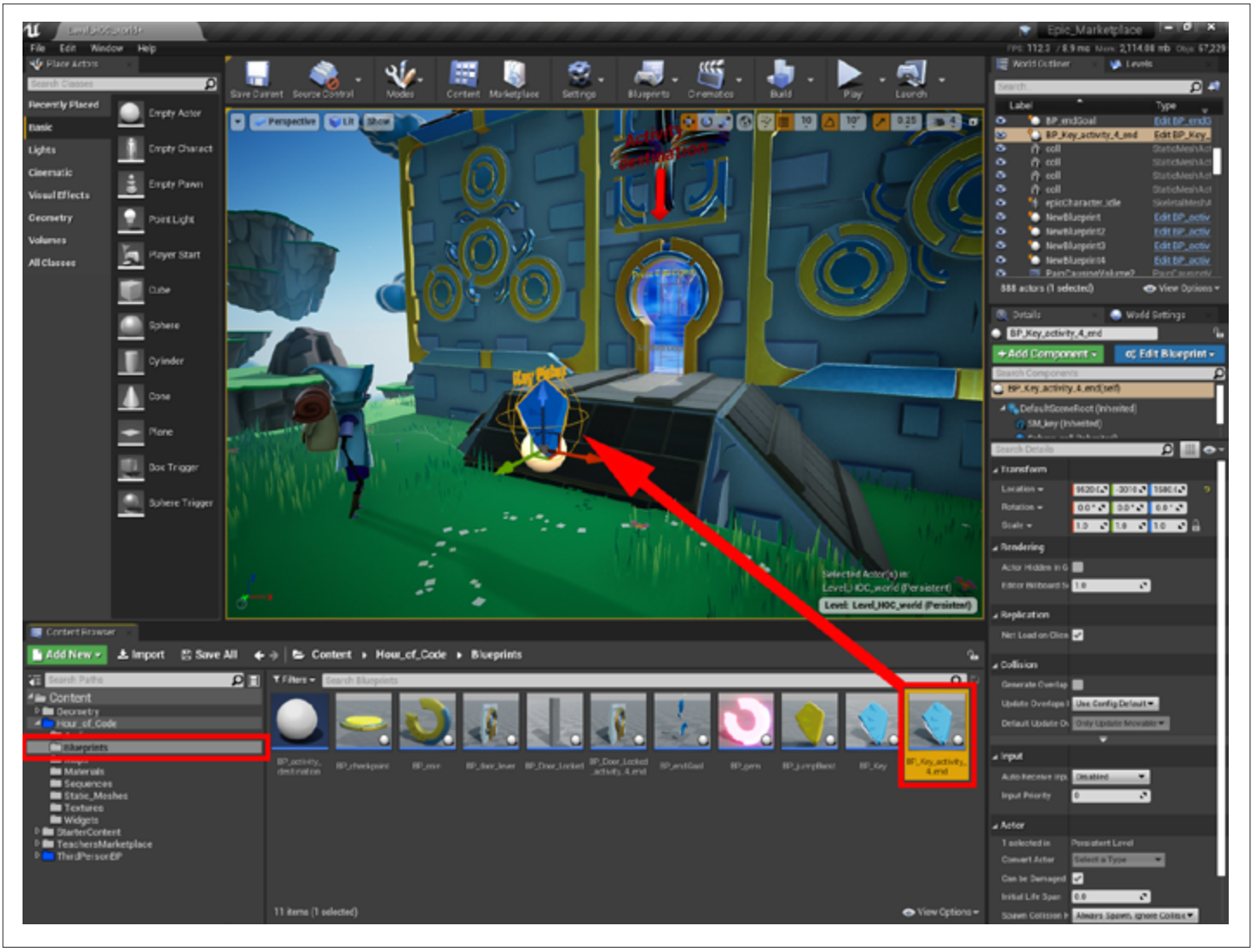

Fig 001 – Add the Key to your level.

To play the game from our current location, make sure to select the **Current Camera Location** from the dropdown arrow next to the Play button, then click Play. Pick up the key, walk to the door, and press the **E** key on the keyboard. The door will open, and you will see the Key icon appear in the bottom-left area of the screen. Then, exit the game by pressing **Esc** on the keyboard.

Fig 001 – Add the Key to your level.

To understand what is happening we need to examine the door Blueprint. Select the door in the **Viewport** and press **Ctrl + E** to open the Blueprint Editor. Or double click the **BP_Door_Locked_Activity_4_End** Blueprint in the Blueprint folder.

With the Blueprint Editor open, click on the **Event Graph** tab if it's not open already. You will see a yellow area in the bottom right. Zoom in to that area using the **scroll wheel** on the mouse. You can pan by holding the right mouse button while dragging.

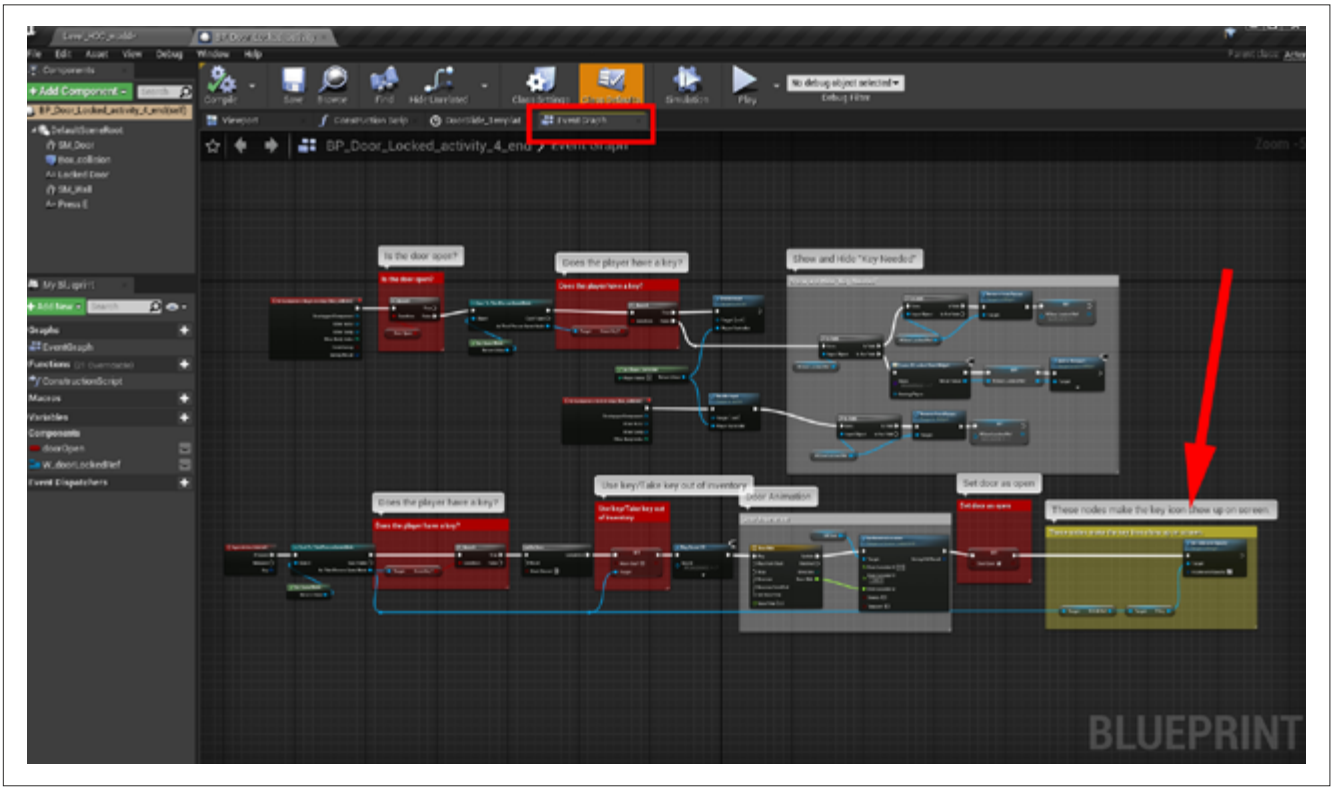

Fig 002 – The part of the Blueprint showing the key icon.

Inside this area, you will see 3 nodes.

1. A reference to the HUD Widget (W HUD Ref)
2. The texture for the key icon (T Key)
3. A way to control the opacity and color of the texture (Set Color and Opacity)

By clicking on the checkerboard, you can open the Color Picker.

Warning: Be sure to close the Color Picker, even if you don't make any changes.

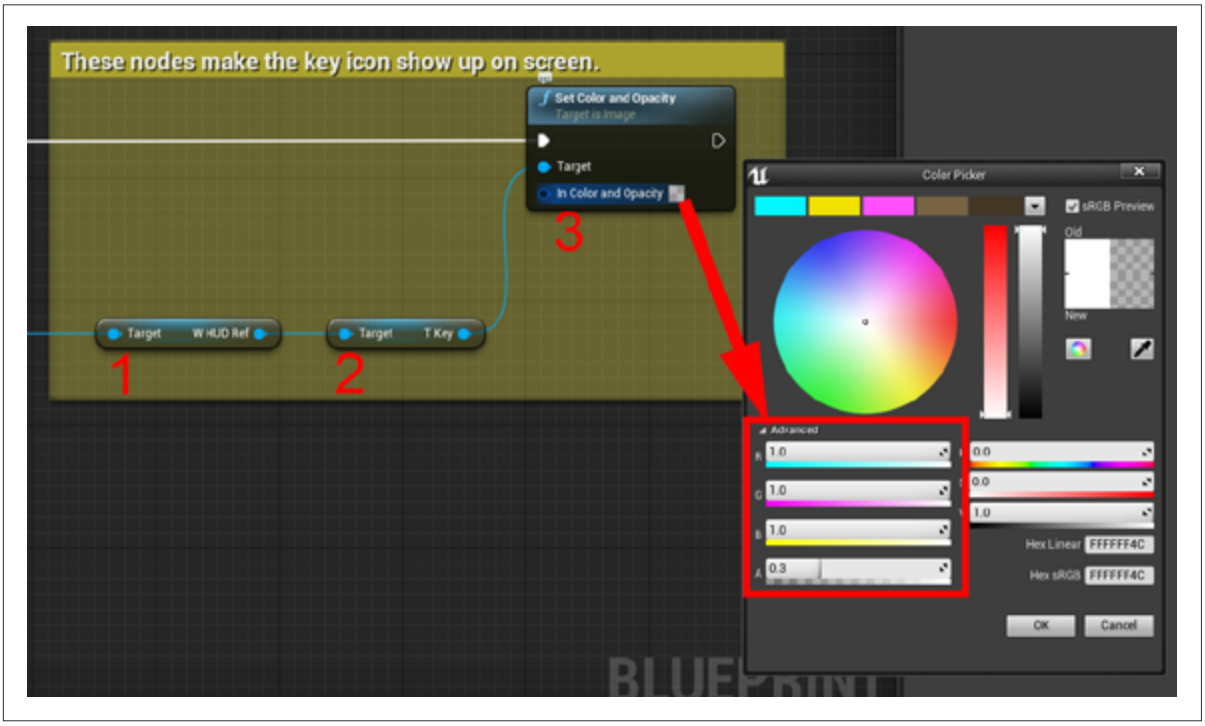

*Fig 003 – Selecting a **Color** and **Opacity** with the Color Picker.*

Follow the blue wire back to its origin (to the left) and you will see that the information for the HUD is being stored in the **ThirdPersonGameMode**.

If you completed Activity 2, you may recall that we talked about how Blueprints can **Cast** to the **Game** to get information. In this case, the door is **casting** to the **Game Mode** to get information about the **HUD**.

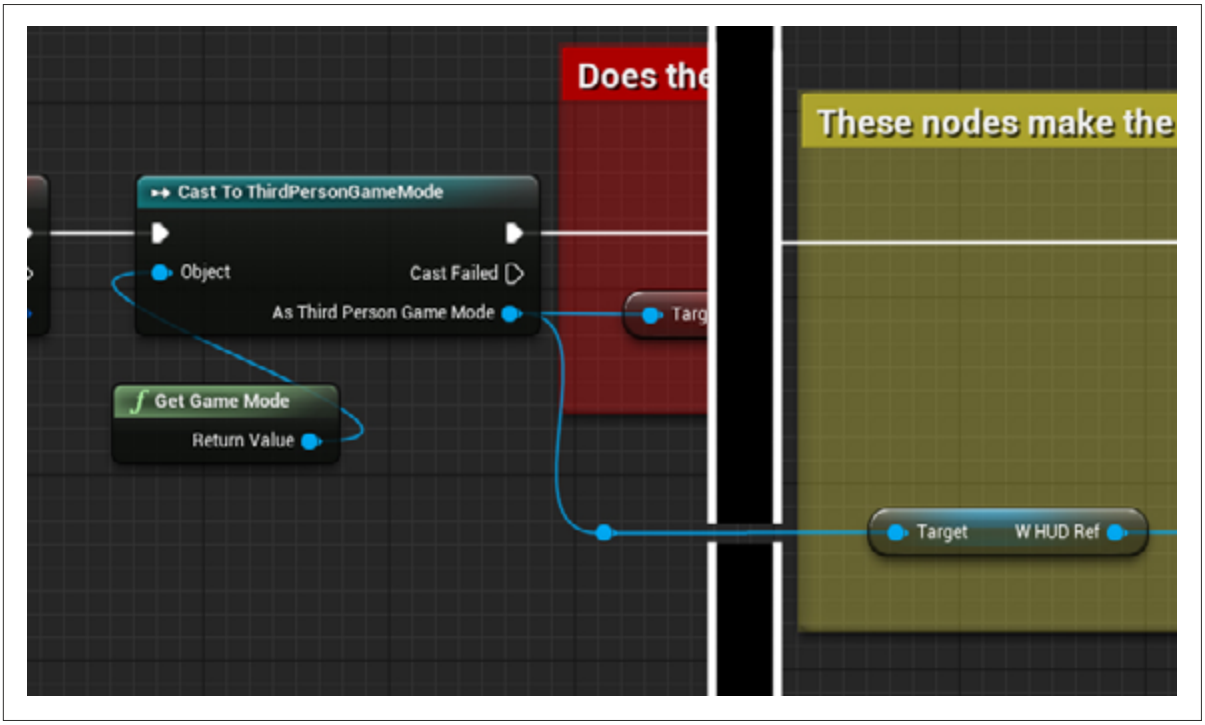

Fig 004 – Casting to the Game Mode allows the data to be used any time during the game.

This tells us that the **game** is handling the **HUD** display. To see how the **game** is doing this, you can open the Game Mode Blueprint. In the **Content Browser** navigate **Content > ThirdPersonBP > Blueprints** and open the **ThirdPersonGameMode** by double-clicking on it.

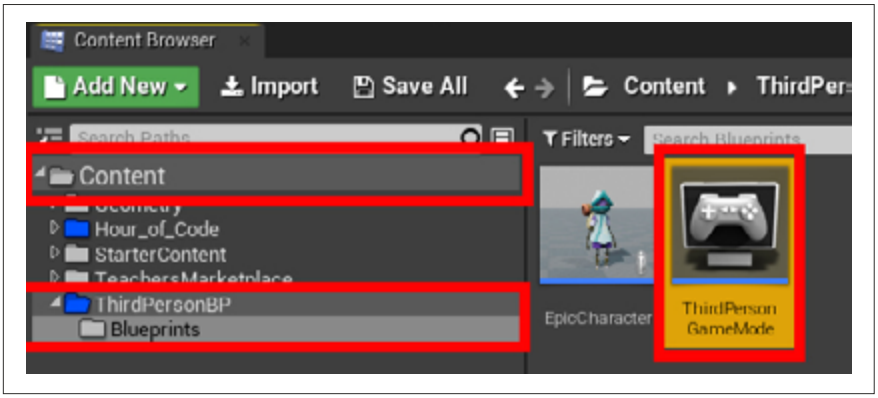

Fig 005 – Access the ThirdPerson GameMode Blueprint.

With the **ThirdPerson GameMode** Blueprint open, click the **Event Graph** tab if it is not already selected. Locate the blue area labeled **Display HUD**. Let's review the first three nodes on the left side of that area. These nodes:

1. Create the **HUD** widget. **(Create W HUD Widget)**
2. Set a variable for the **HUD**. **(SET W HUD Ref)**
3. Display the **HUD** to the screen. **(Add to Viewport)**

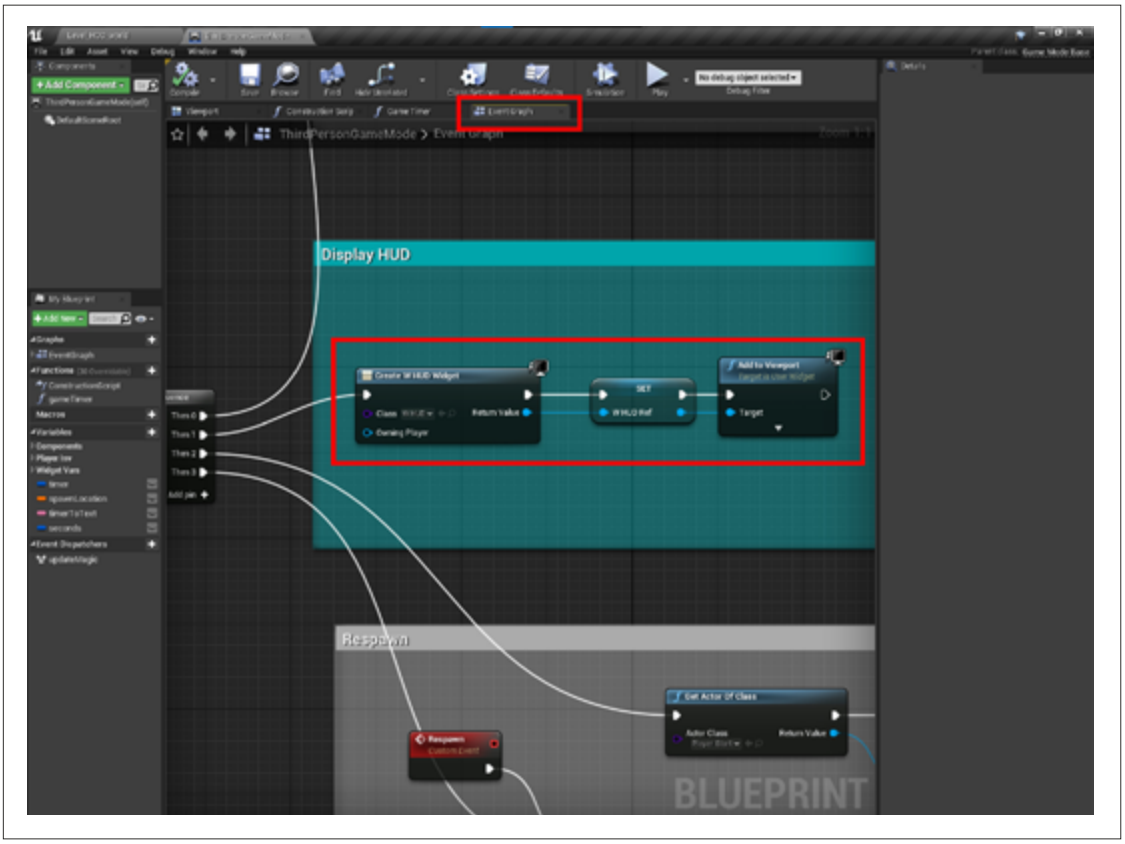

Fig 006 – HUD controls in Blueprint Event Graph.

Close the Blueprint Editor by clicking the **X** in the upper-right.

Working with Widgets

Let's get the HUD Widget open. In the **Content Browser**, navigate to **Hour_of_Code > Widgets**, and double-click the **W_HUD**.

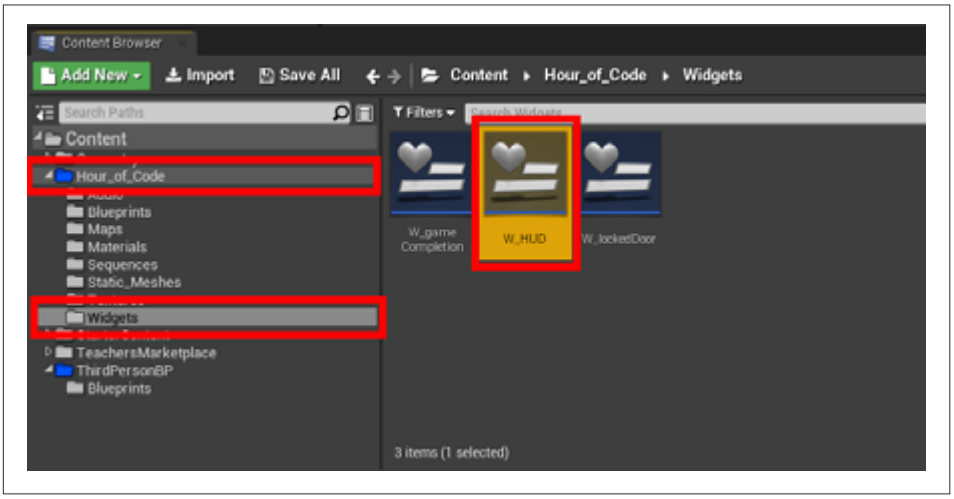

Fig 007 – The W_HUD item in the Widgets folder.

Overview of Widgets

Widget Blueprints are a little different than the previous Blueprints we have worked with. So, let's look at the areas that we will be working with.

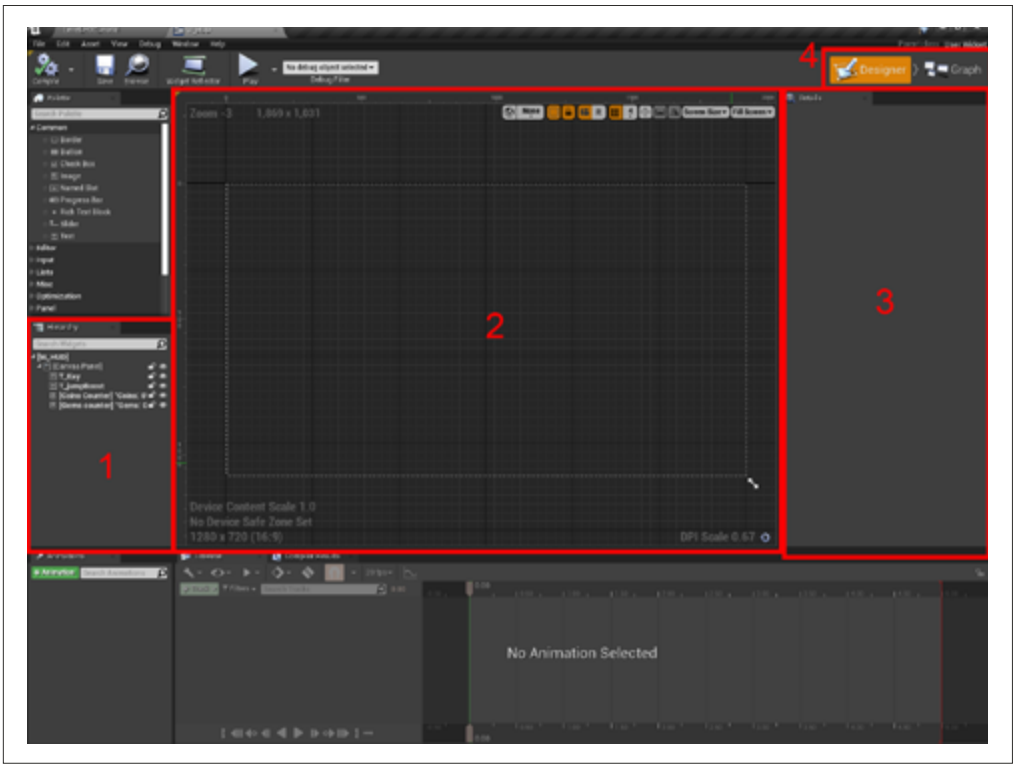

Fig 008 – The Widget designer

1. **Hierarchy** – This area is a list of all the elements that are being displayed on the screen. Currently, all the elements have been set to be invisible, this is why you haven't seen them on-screen during activities 1 – 4. That's about
to change.
2. **Visual Designer** – This is a representation of the game screen, where we have placed the key icon.
3. **Details** – This panel will update when you select elements, like the key icon, in the **Hierarchy** or **Visual Designer**.
4. **Editor Mode** – This will switch between **Designer** mode and **Graph** mode.

The Key Icon

In this section we are going to:
- Let the player know they can pick up a key in this game.
- Make the key icon opaque when the character picks up the key.

1. In the **Hierarchy** panel, select **T_key**.
2. In the **Details** panel, scroll down and find the **Color and Opacity** section (You may need to click the arrow to the left of the **Color and Opacity** label to expand the view) and change the **A** (alpha) to 0.3; you will see the faded **key** icon appear.
3. Then, click the **Compile** button at the top-left area of the screen.

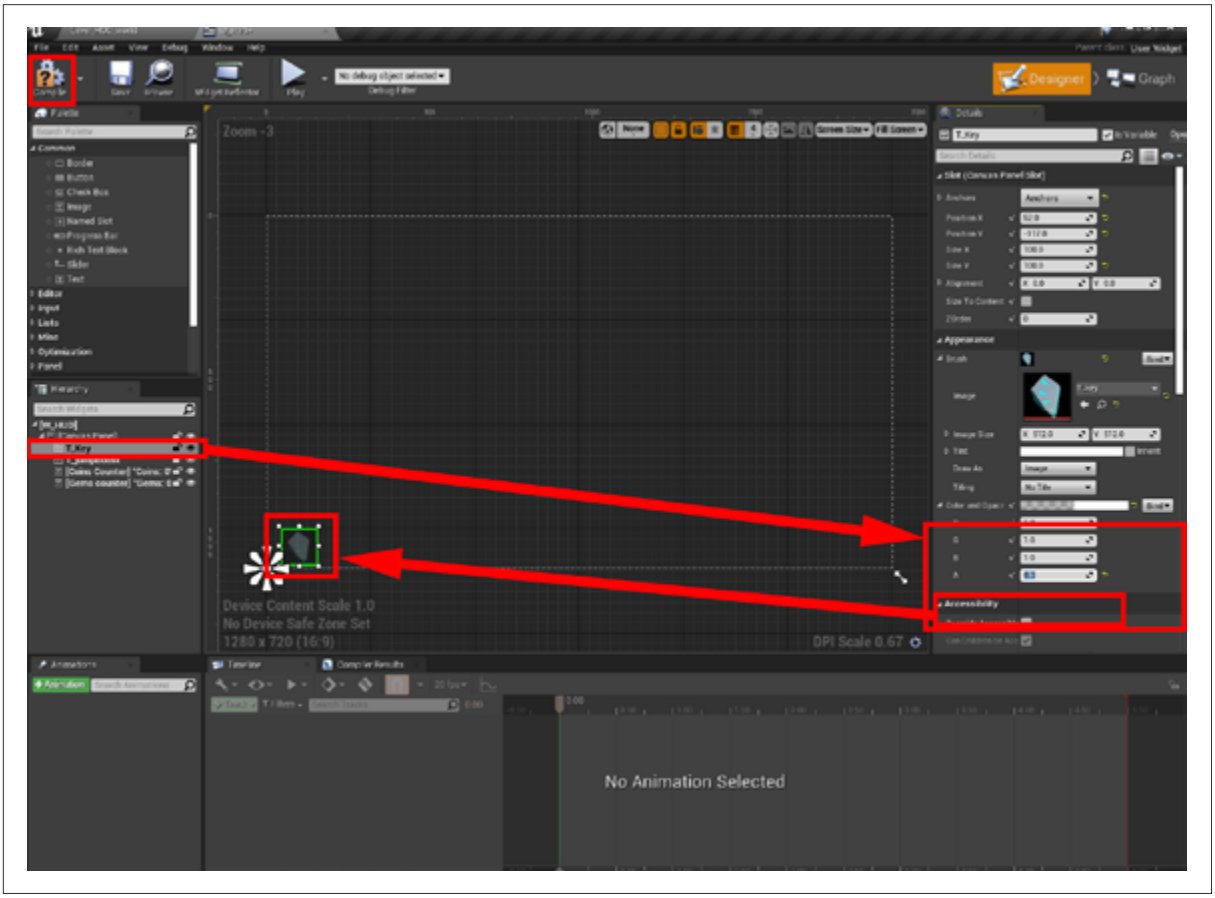

Fig 009 – Setting the Alpha value of the key HUD icon to make it visible.

Now when you play the game, the key icon can be seen on the screen. Close the Widget Editor by clicking the **X** in the upper-right corner, then click the Play button in the Unreal Editor to play your level.

Fig 010 – The semi-transparent key HUD icon in the lower left.

Next, we will make the **key icon** opaque when the **character** picks it up. This will be controlled by the **key** Blueprint. Whether you have been following along from Activity 4 or starting here in Activity 5, you will need to open the **key** Blueprint we have prepared for you.

1. Navigate to **Hour_of_Code > Blueprints** and open **BP_Key_activity_4_end** by double-clicking on it.

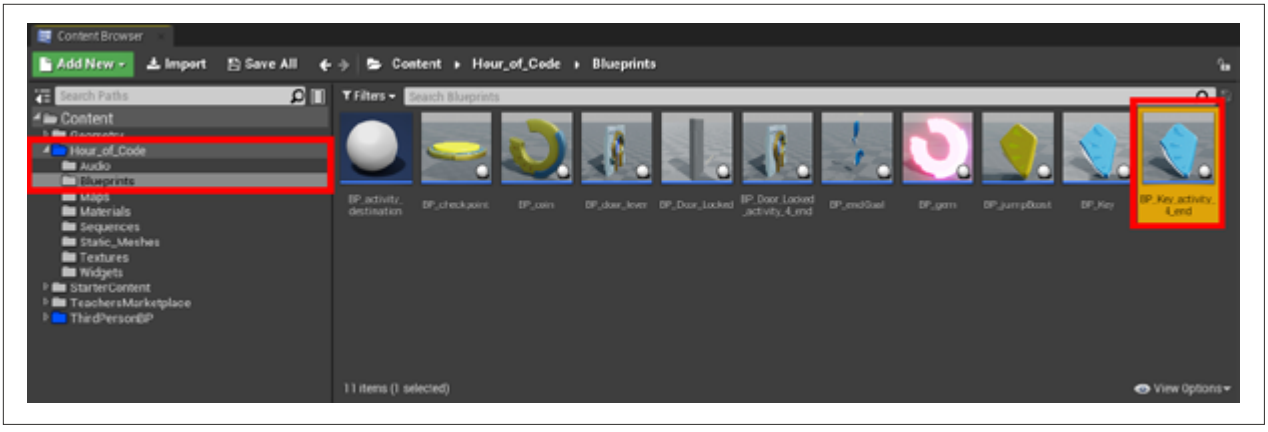

Fig 011 – Open the BP_Key_activity_4_end Blueprint.

2. Click the **Event Graph** tab if it is not already selected.
3. You will see a blue area labeled **Activity 5 Make the key icon opaque**. (If you have been following from Activity 4, you can move to the **NOTE** below.)

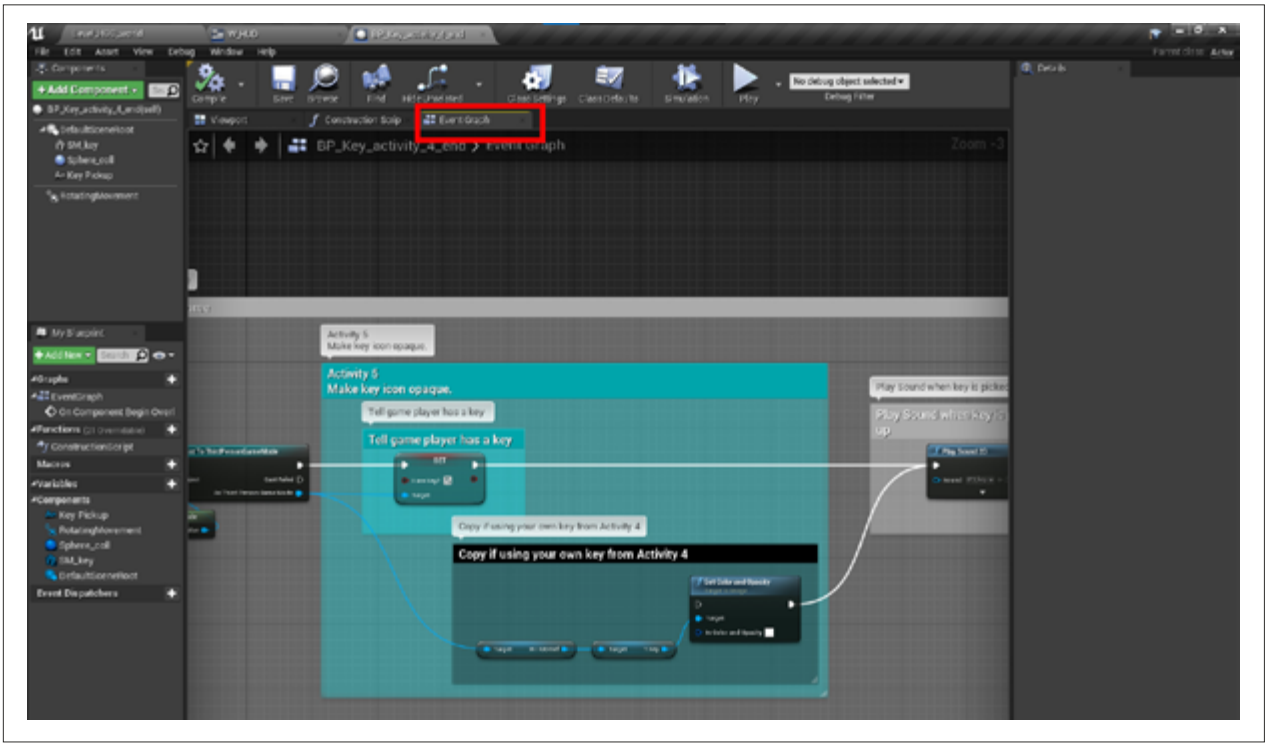

Fig 012 – View the Event Graph tab.

4. Next, drag a wire from the **Set (Have Key?)** execution pin, to the **Set Color and Opacity** node.

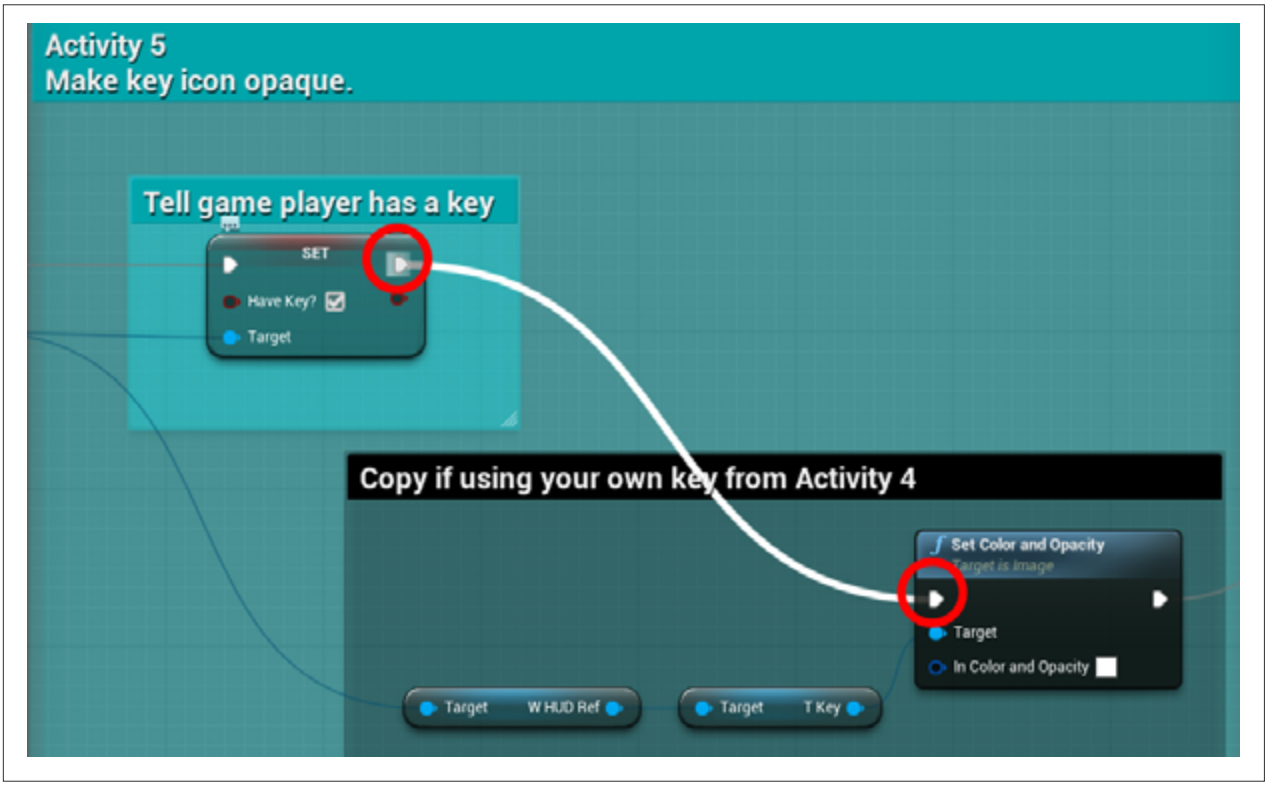

Fig 013 – Connect the execution pins between the key and the color and opacity nodes.

NOTE: If you built a Key Blueprint in Activity 4, you will have to follow this step to update it. If you would like to skip this step, simply use the **BP_Key_Activity_4_End** Blueprint provided with this project.

To update your custom key Blueprint, select these 3 nodes by using shift+click or clicking and dragging to create a selection box. Copy the 3 selected nodes and paste them into your key Blueprint. Then connect the nodes as seen below.

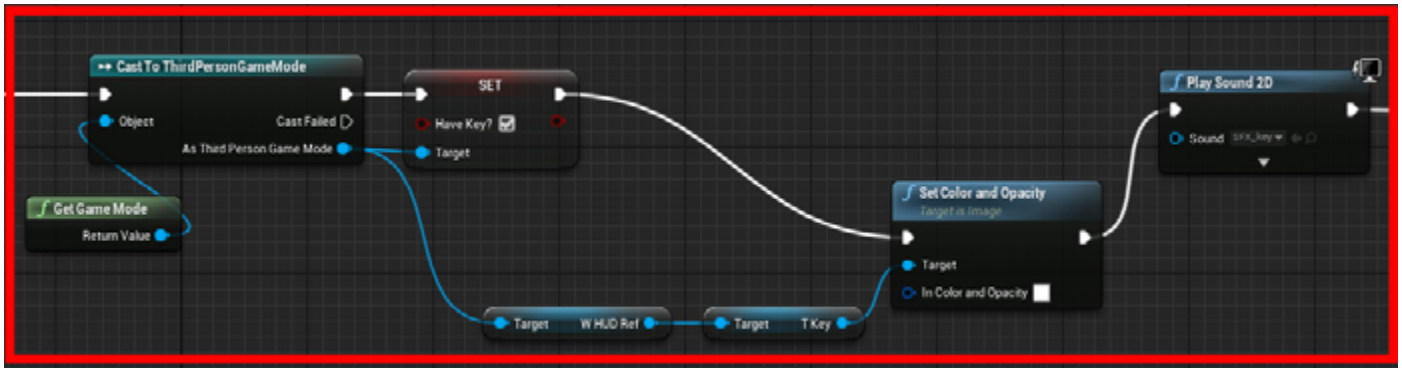

Fig 013a – Copy and paste the key color and opacity settings.

5. Click the **Compile** button to update the changes to your Blueprint, then close the Blueprint Editor.

Now, play the game and collect the key. You will now see that the key icon becomes opaque. Next, open the door and you will see that the key icon becomes transparent again.

Fig 014 – Key HUD icon is opaque when the key is collected.

Let's review what is happening.

1. The **Game Mode** displays the **HUD** on the screen when the game begins, and the **key** icon is initially drawn at 30% opacity.
2. When the **character** touches the key, it tells the Game Mode to draw the key icon at 100% opacity.
3. When the player opens the **door**, it tells the **Game Mode** to draw the **key** icon at 30% opacity.

Well done, that was a lot of coding. Next, we will set the Jump Boost icon to behave the same way.

Remember to **Save All** your work.

Jump Boost

In this section, we will be working on 3 different Blueprints.

1. **W_HUD (**in **Content > Hour_of_Code > Widgets)**
2. **BP_jumpBoost** (in **Content > Hour_of_Code > Blueprints)**
3. **EpicCharacter (**in **Content > ThirdPersonBP > Blueprints)**

The Jump Boost Collectible

First, drag a **BP_jumpBoost** into the level. (Press **W** to activate the Move gizmo to move the object as needed.)

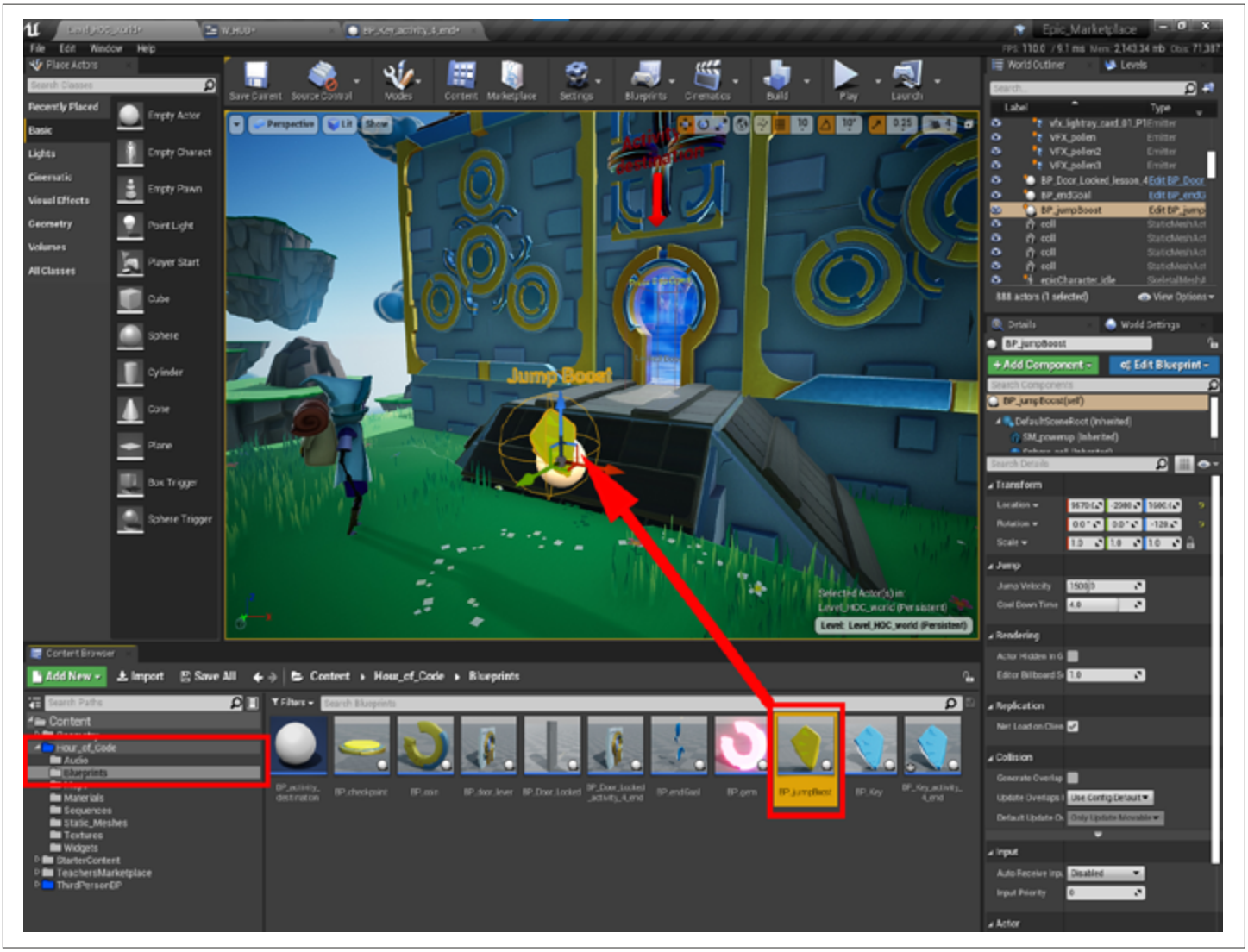

Fig 015 – Add the jump boost Blueprint to the level.

The HUD Blueprint

Now open the **W_HUD** Blueprint. Navigate to **Content Browser > Hour_of_code > Widgets** and double-click **W_HUD**.

1. In the **Hierarchy** panel, select **T_jumpBoost**.
2. In the **Details** panel, scroll down and find the **Color and Opacity** section, change the **A** (alpha) to **0.3**; you will see the **jump boost** icon appear.
3. Then, click the **Compile** button at the top-left area of the screen and close the Widget Editor.

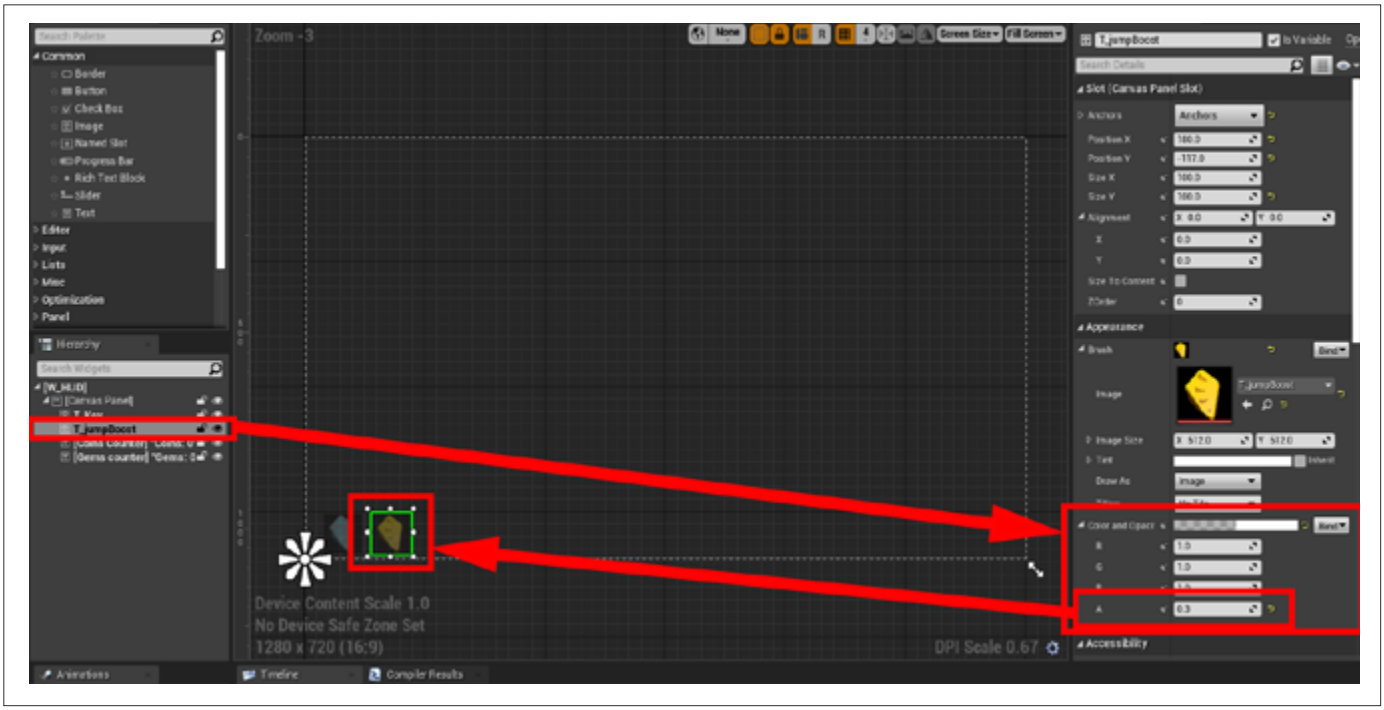

Fig 016 – Notice the jump boost icon on the HUD.

Play the game to see that the **jump boost** icon appears on the screen. When you collect the Jump Boost, you'll notice that the icon doesn't change. We'll address that next in the Blueprint.

Fig 017 – The semi-transparent jump boost icon before collecting the jump boost.

Remember to **Save All** your work. (File > Save All)

The Jump Boost Blueprint

Next, double-click the **BP_jumpBoost** blueprint to open it.

1. Click the **Event Graph** tab if it is not already open.
2. You will notice a blue section labeled Activity 5, and inside is another section labeled "**Connect me to PlaySound at Location**".
3. Connect the execution pin from **Play Sound at Location** to the reroute node in this section. See Fig 018.

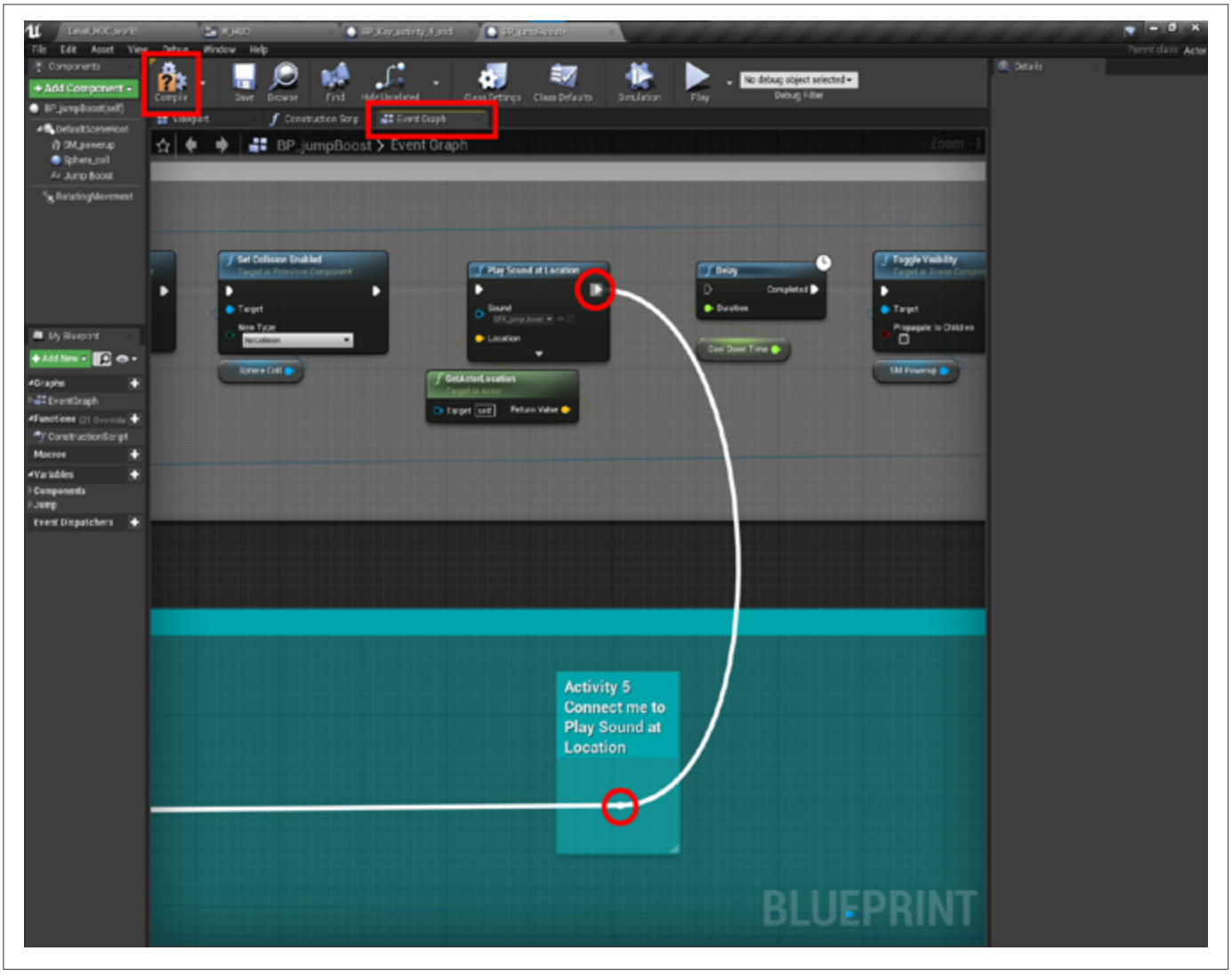

Fig 018 – Connecting the play sound into the Blueprint.

You may notice that redirecting this wire from the execution pin will cut off the remaining nodes in the above section. It's OK! We have included a copy of those nodes in this newly connection section for convenience.

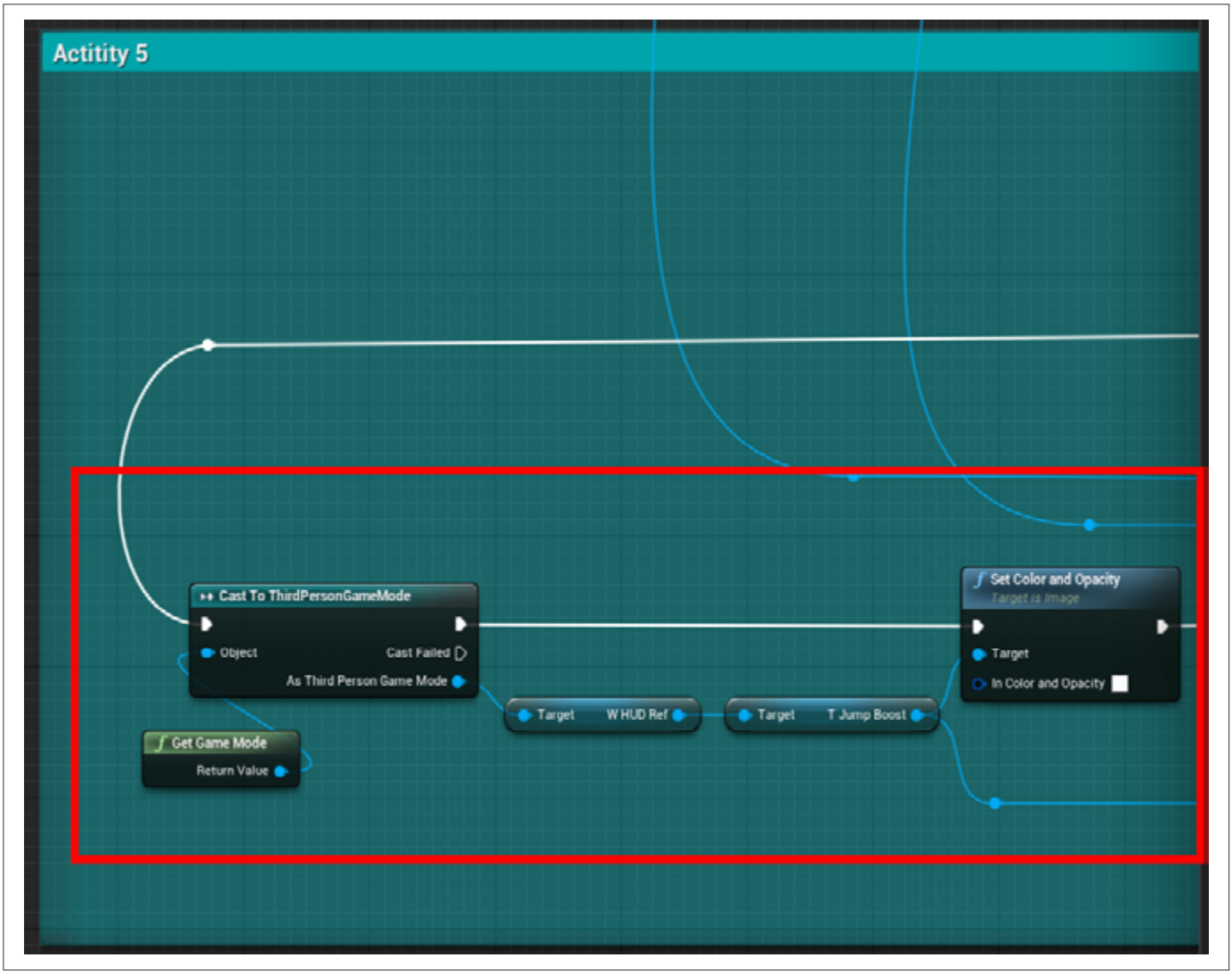

Fig 019 – Set the color and opacity of the jump boost HUD icon.

Play the game, collect the **jump boost**. The **jump boost** will now change opacity when the character touches it and will change again at 4 seconds (default value is 4) when the **cool down timer** expires.

Programmer's Eye for Detail

Sometimes being a programmer could require some detective skills. Being curious and observant could lead you to solve problems.

Did you notice the **bug** we just introduced?

If you didn't notice it, play the game again and jump off into the void. This time watch the **jump boost** icon on the screen. You will notice that the **jump boost** icon doesn't reset when the character dies. This is because the **character** is **colliding** with the **Kill Z** plane, which we talked about in **Activity 1**. The **Kill Z** plane is causing damage and kills the player, but it does it a little differently than the **Pain Causing Volume** we introduced in **Activity 1**.

This is a good example of what happens when working on a game. Often, a designer will implement a feature, and notice they have created a **bug**. We will address this **bug** in the next section.

Remember to **Save All** your work.

EpicCharacter Blueprint

Navigate to **Content > ThirdPersonBP > Blueprints** and open **EpicCharacter** by double-clicking on it.

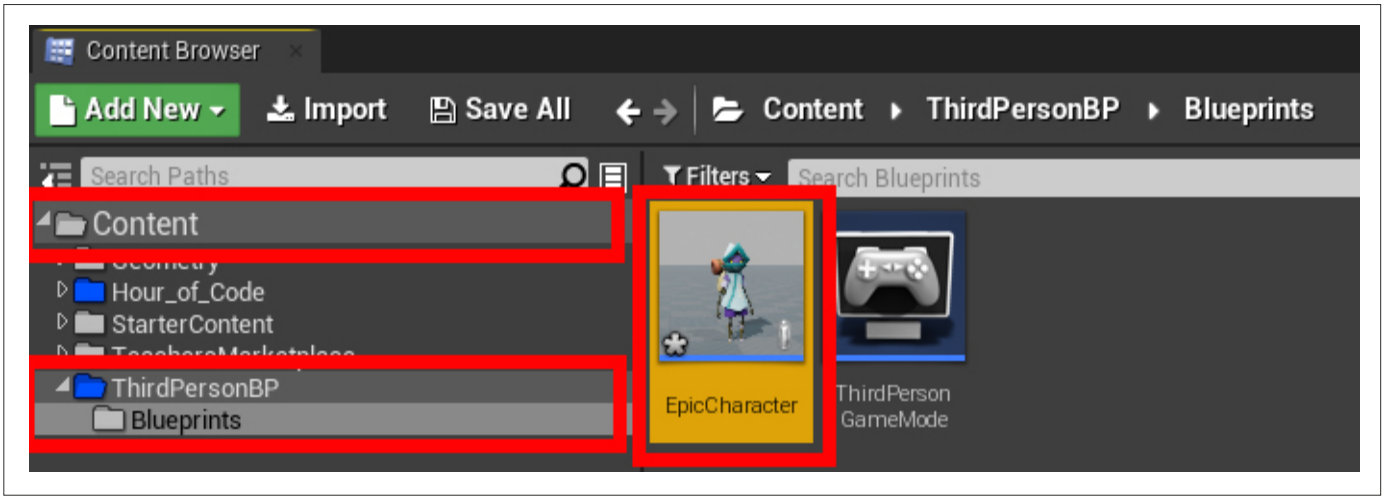

Fig 020 – Open the EpicCharacter Blueprint

Click on the **Event Graph** tab if it's not selected already. You'll see a few blue areas. Locate the area labeled **Activity 5**. Zoom in to that area and locate the red **hitKillZ** node.

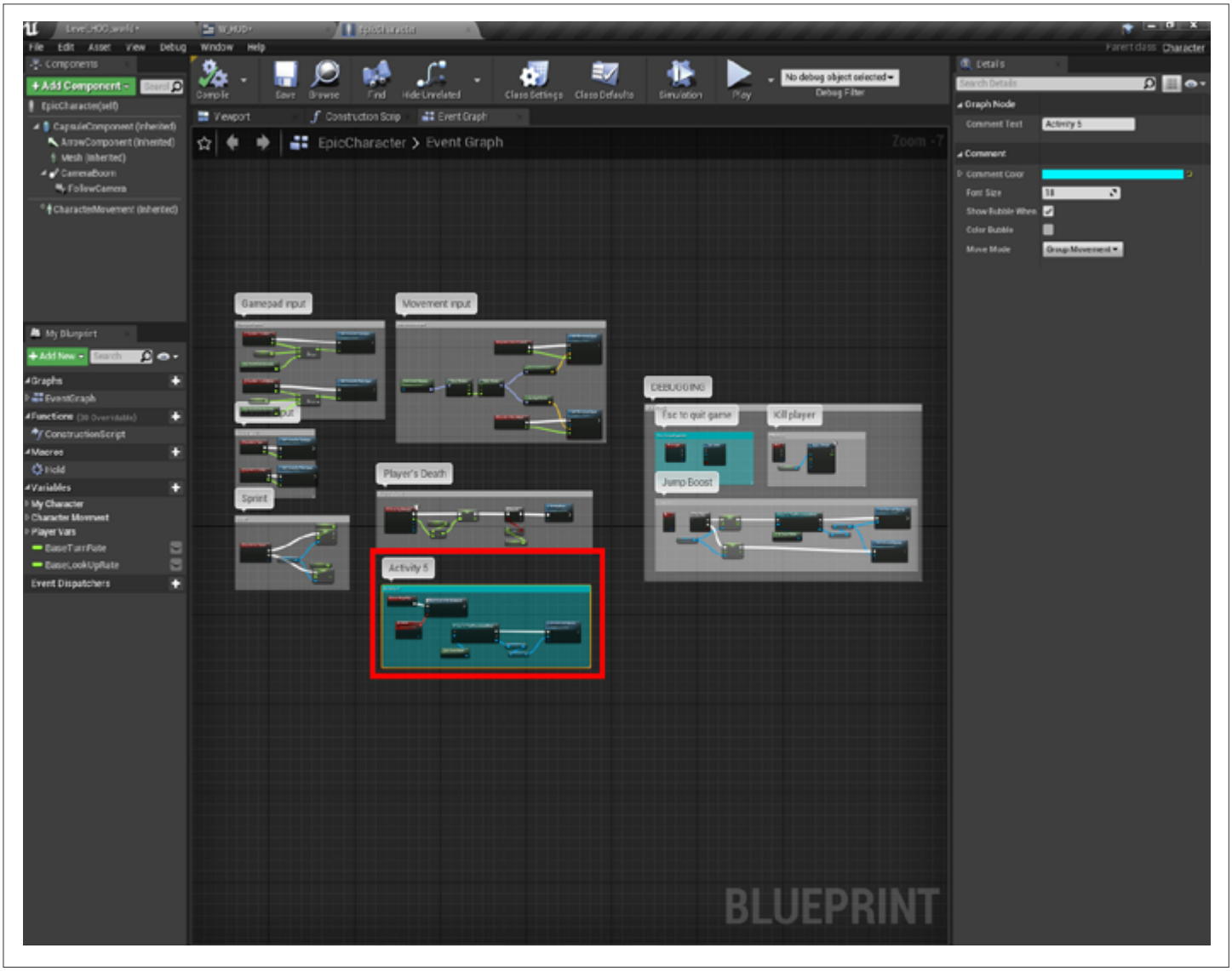

Fig 021 – Activity 5 section of the EpicCharacter Blueprint

Connect the execution pin of the **hitKillZ** node, to the **Cast to ThirdPersonGameMode** node. See Fig 022.

Then, click the **compile** button. Now when the **character** is destroyed, it will also reset the opacity of the **jump boost** icon.

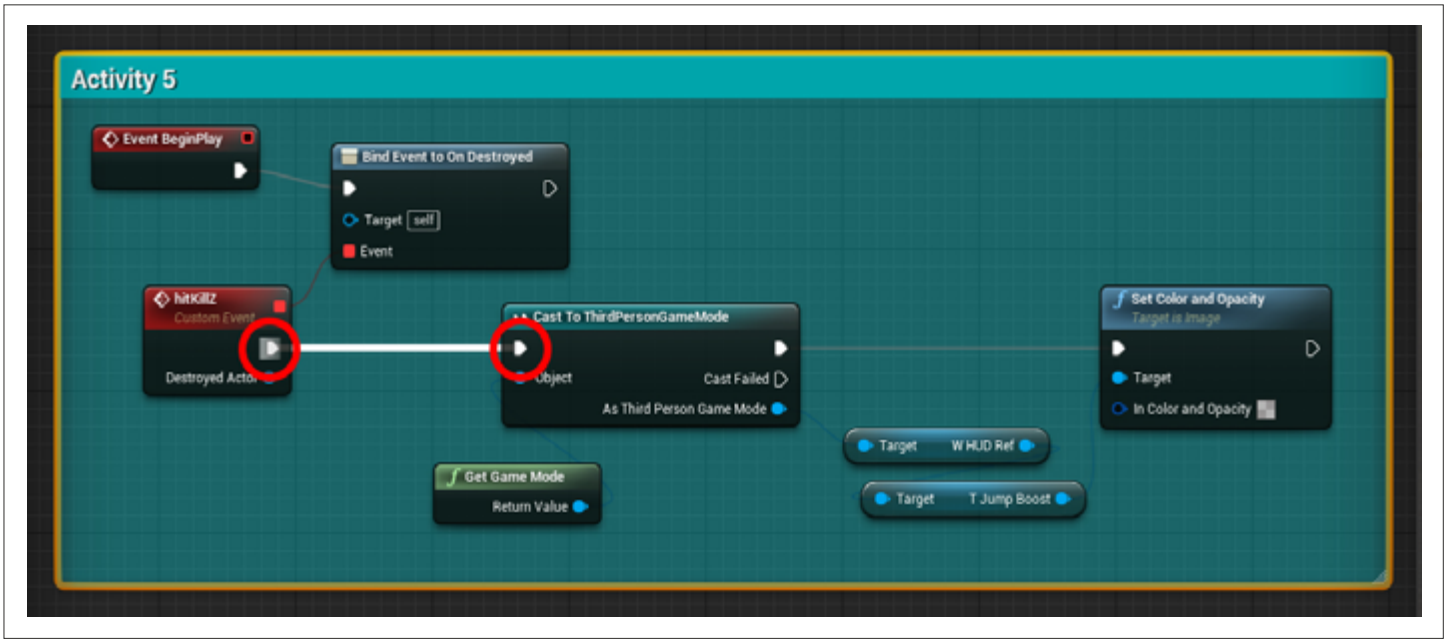

Fig 022 – Connect the hitKillZ with ThirdPersonGameMode to know when the player dies from falling.

It's time to playtest this to make sure that it is working correctly. Pick up the **jump boost** and fall into the void. When the character dies, the **jump boost** icon will reset.

Remember to Save All your work.

Coins and Gems

In this section, we are going to reveal the coin and gem counters within the **W_HUD** Blueprint.

Before we get to the code section, we want to add a coin and a gem where we can easily access them. In the **Content Browser**, navigate to **Content > Hour_of_Code > Blueprints** folder and place a **BP_coin** and a **BP_gem** into your level. Remember, you can press **W** to activate the Move gizmo to reposition your objects.

The **coins** will represent the common pickups, these will be easy for the player to get to. They will also be used to lead the player.

The **gems** will be uncommon and will require more skill to obtain. They will also be used to reward the player for overcoming the more difficult challenges.

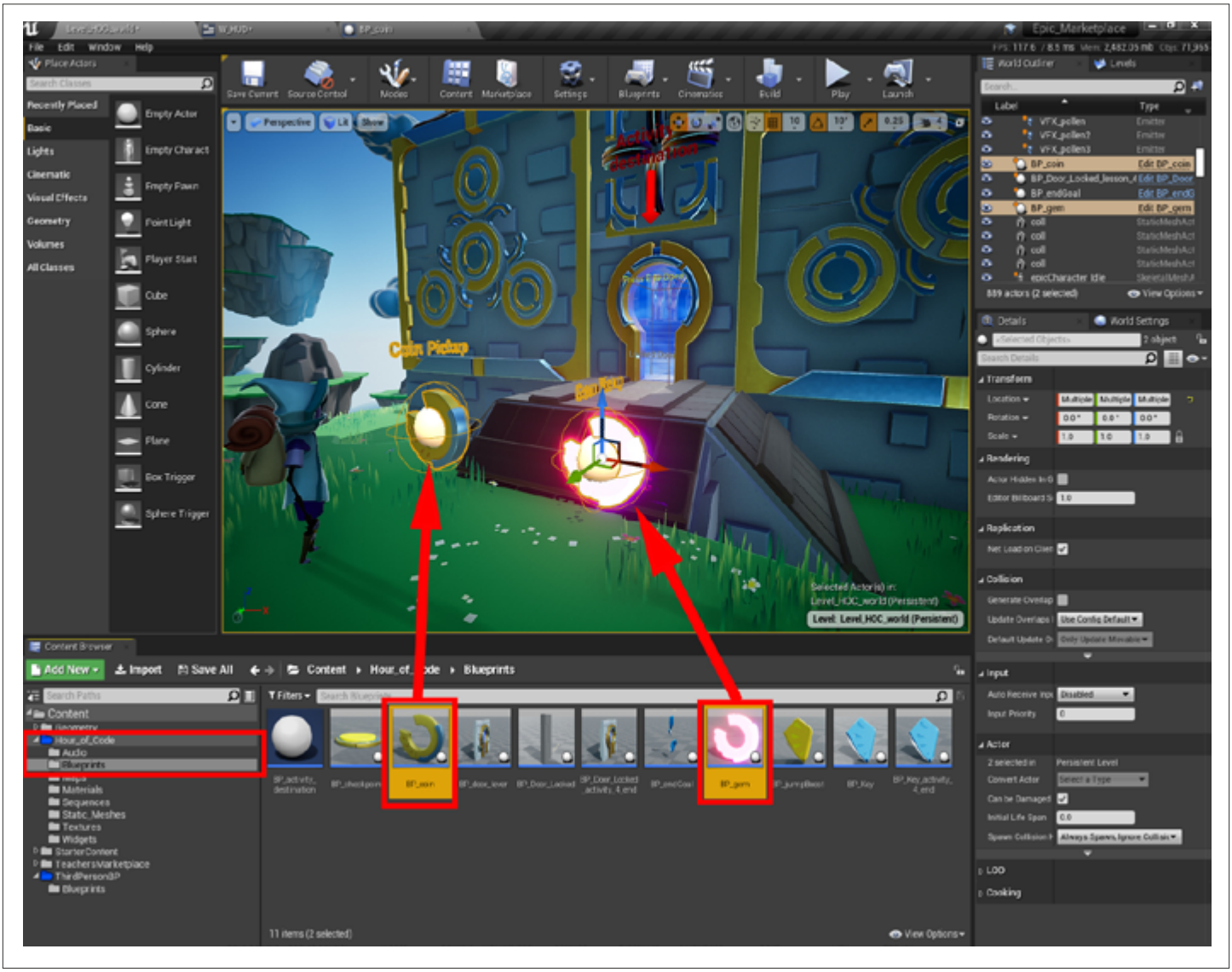

Fig 023 – Adding a Coin and a Gem to the Level.

1. In the **Content Browser**, navigate to **Content > Hour_of_Code > Widgets** and open the **W_HUD** Blueprint.
2. In the **Hierarchy** panel select the **Coins Counter** and the **Gems Counter** (to select both, you can click on Coins Counter and hold shift while clicking the Gems Counter).
3. In the **details** panel, change the A (alpha) value of **Color and Opacity to 1**.
4. Then click the **compile** button.

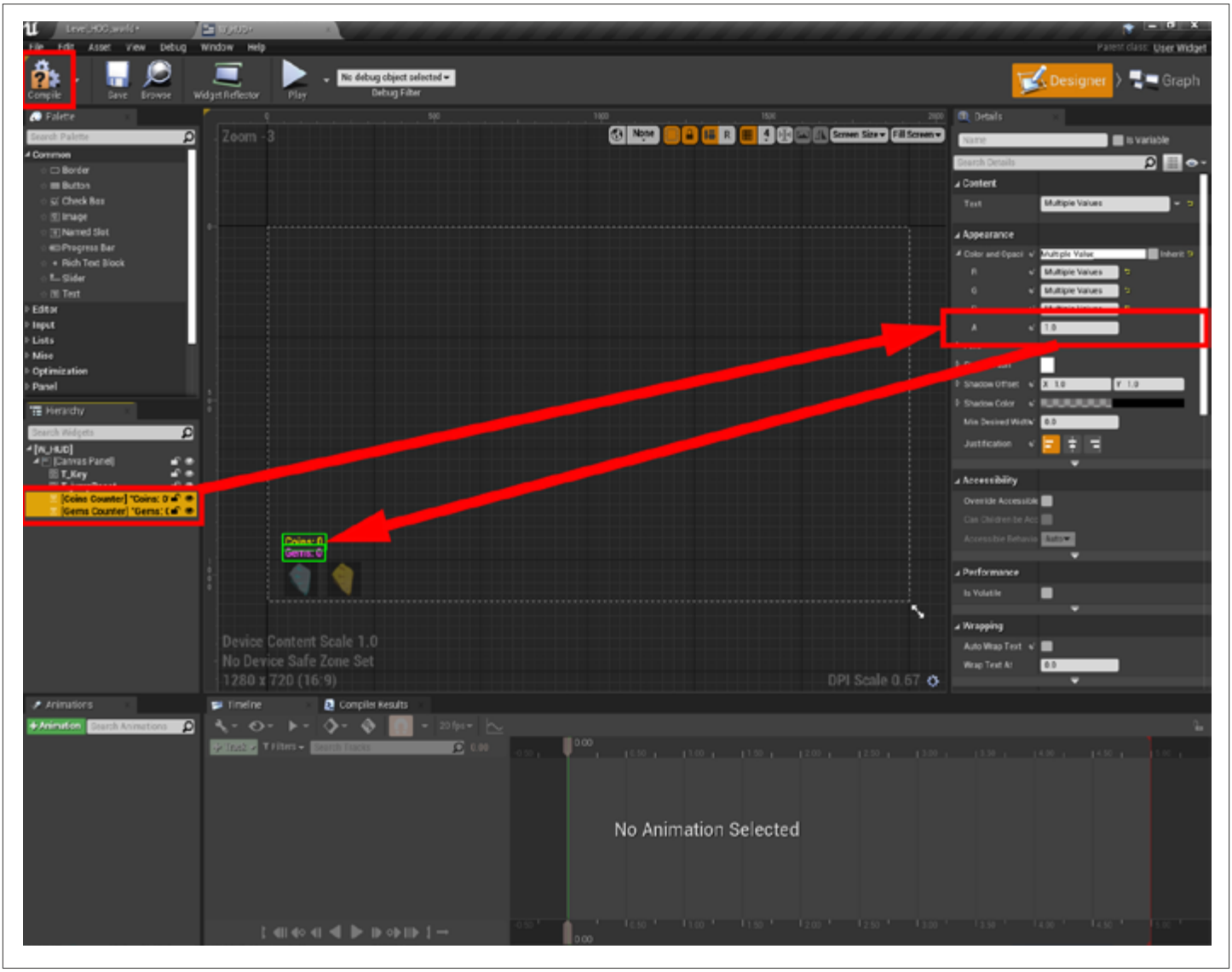

Fig 024 – Coins counter and Gems counter widgets.

Now play the game and pick up the coin and the **gem**. Watch the counters in the bottom-left area of the screen.

Did you notice the **bug**?

The coin counter didn't change. This is because the **BP_coin** Blueprint is missing a connection.

In the **Content Browser** navigate to **Content > Hour_of_Code > Blueprints**. Open the **BP_coin** and click on the **event graph** if it is not already selected and look at the code. Can you figure out what needs to be connected?

1. Locate the blue area labeled **Activity 5**.
2. Connect the execution pin of the **Set (Coin Count)** node to the **Update Coins** event node.
3. Then, **compile** the code and close the Blueprint editor.

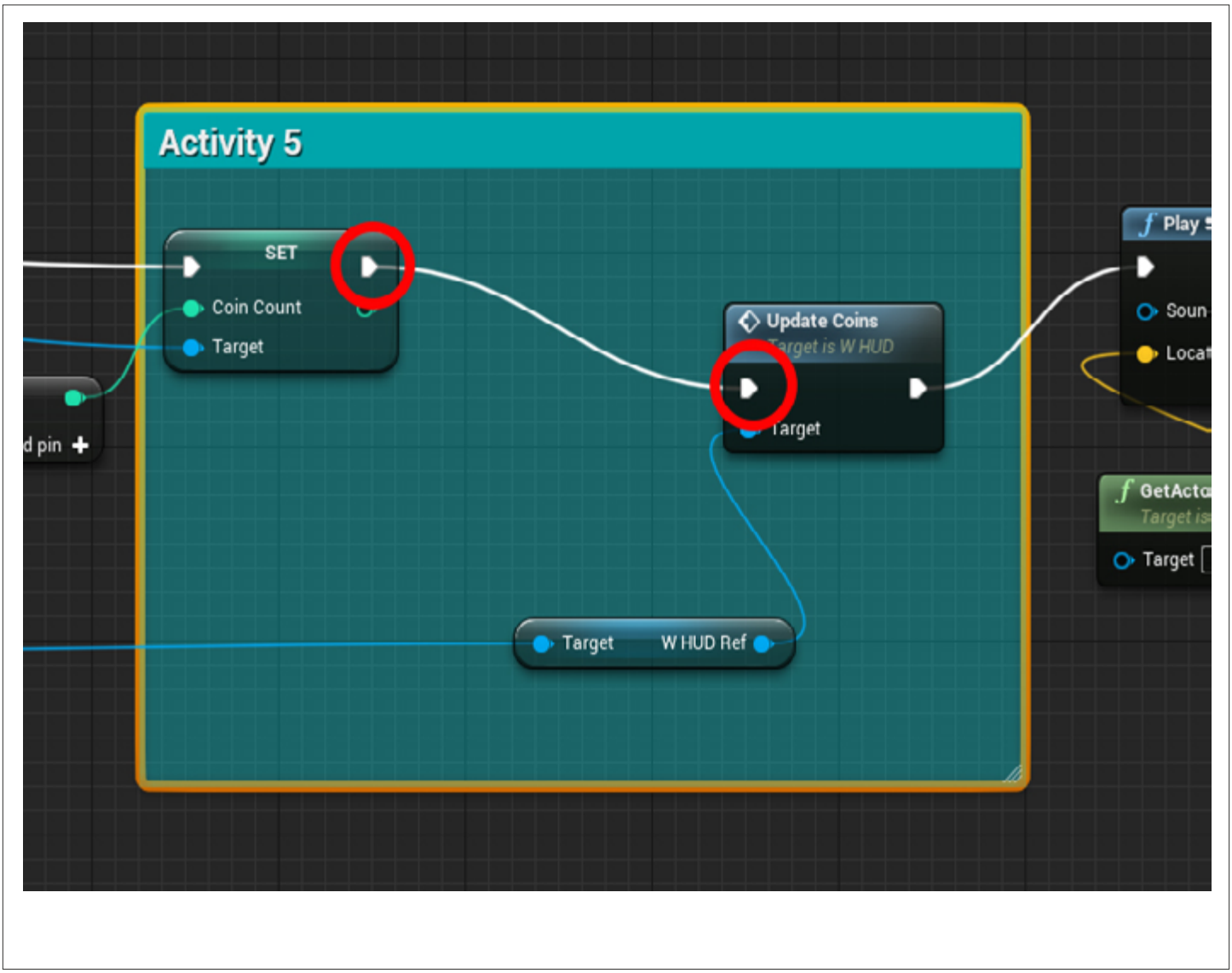

Fig 025 – Connecting the Set (Coin Count) node to the Update Coins node.

Playtest again, and you should notice that the HUD updates correctly. Great work!

Fig 026 – Collecting Coins and Gems.

Extension Activity

If you've followed along from the first activity, you have added floating islands, moving platforms, coins, gems, keys, and power-ups! Now, take some time to build the game using your skill and creativity. Feel free to make the game your own by moving or adding items. Maybe you should hide the key, so it is not in plain sight. Make sure to place gems in harder to reach places as they are intended to be a bonus collectible. If you're feeling especially daring, modify the Blueprints further to make the game more interesting. Have some fun. It will be great to share your original modified level with your peers when you are finished.

Keeping Score

Before we package up our game, let's take a quick look at how our game is keeping score. Open the **W_gameCompletion** Blueprint located in the **Content > Hour_of_Code > Widgets** folder of the **Content Browser**.

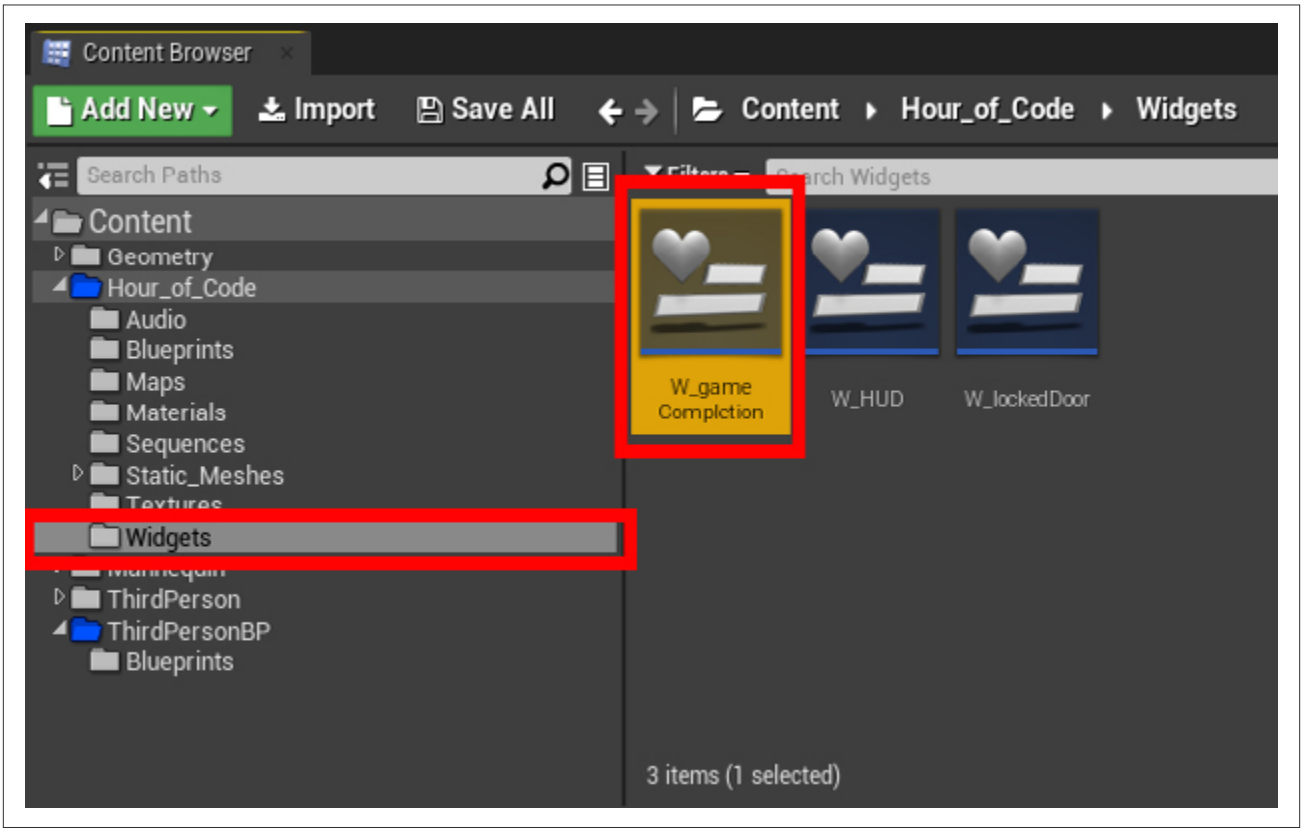

Fig 033 – The Game Completion Blueprint.

In the **Designer** view, you can see how each of the widgets is laid out. Now, select **time** in the **hierarchy** panel. Then, look at the top of the **details panel**, you will see a checkbox that says **is Variable**. This means that this widget can be used as a variable, and can be given information just like the **coins** and **gems**.

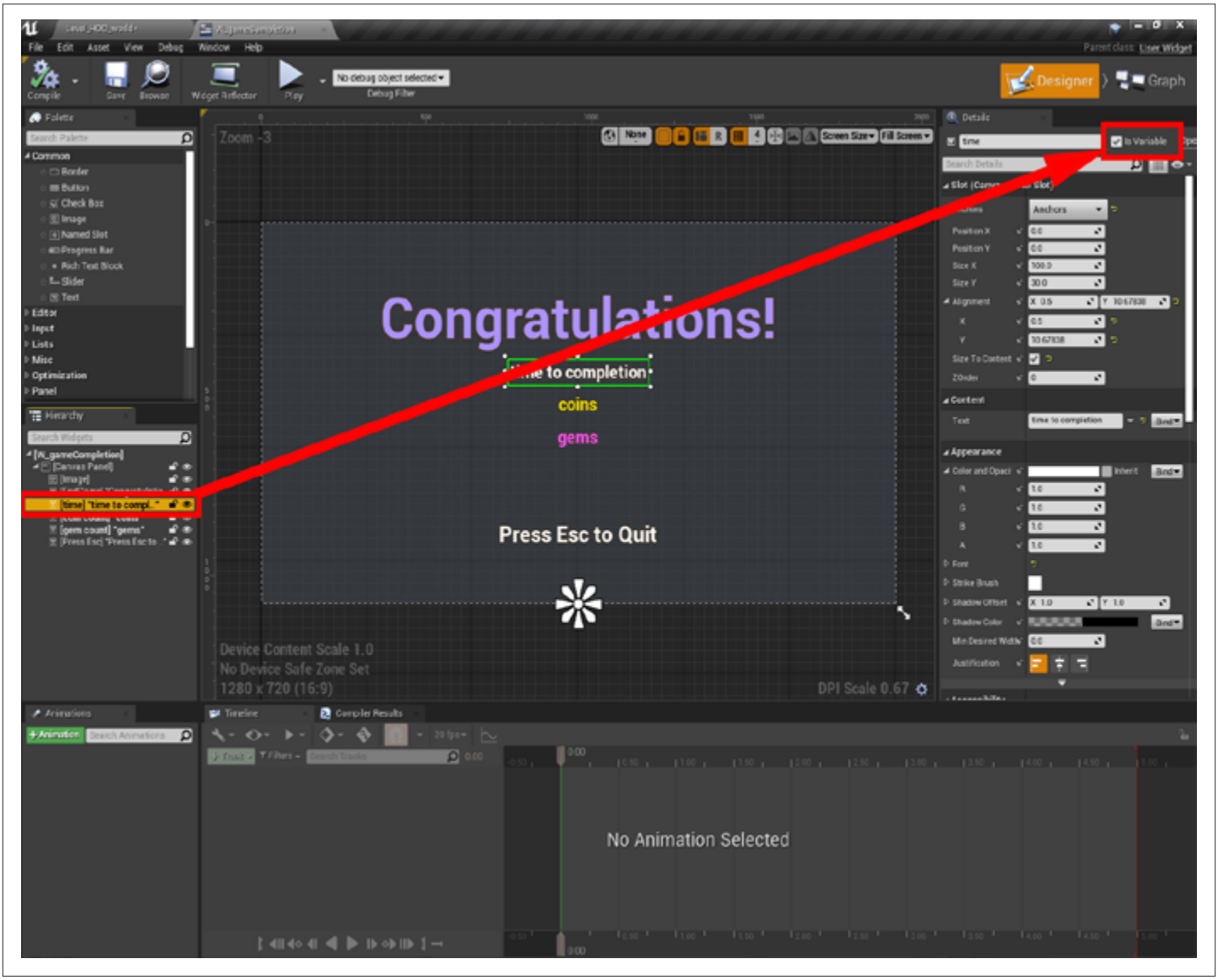

Fig 034 – The Variable checkbox for the time Widget.

Click the **graph** button. In the **graph**, you will see 3 sections. The top section is reporting the **time** it took to complete the game. The middle section reports the number of **coins** the player collected. And the bottom section reports the number of **gems** the player collected.

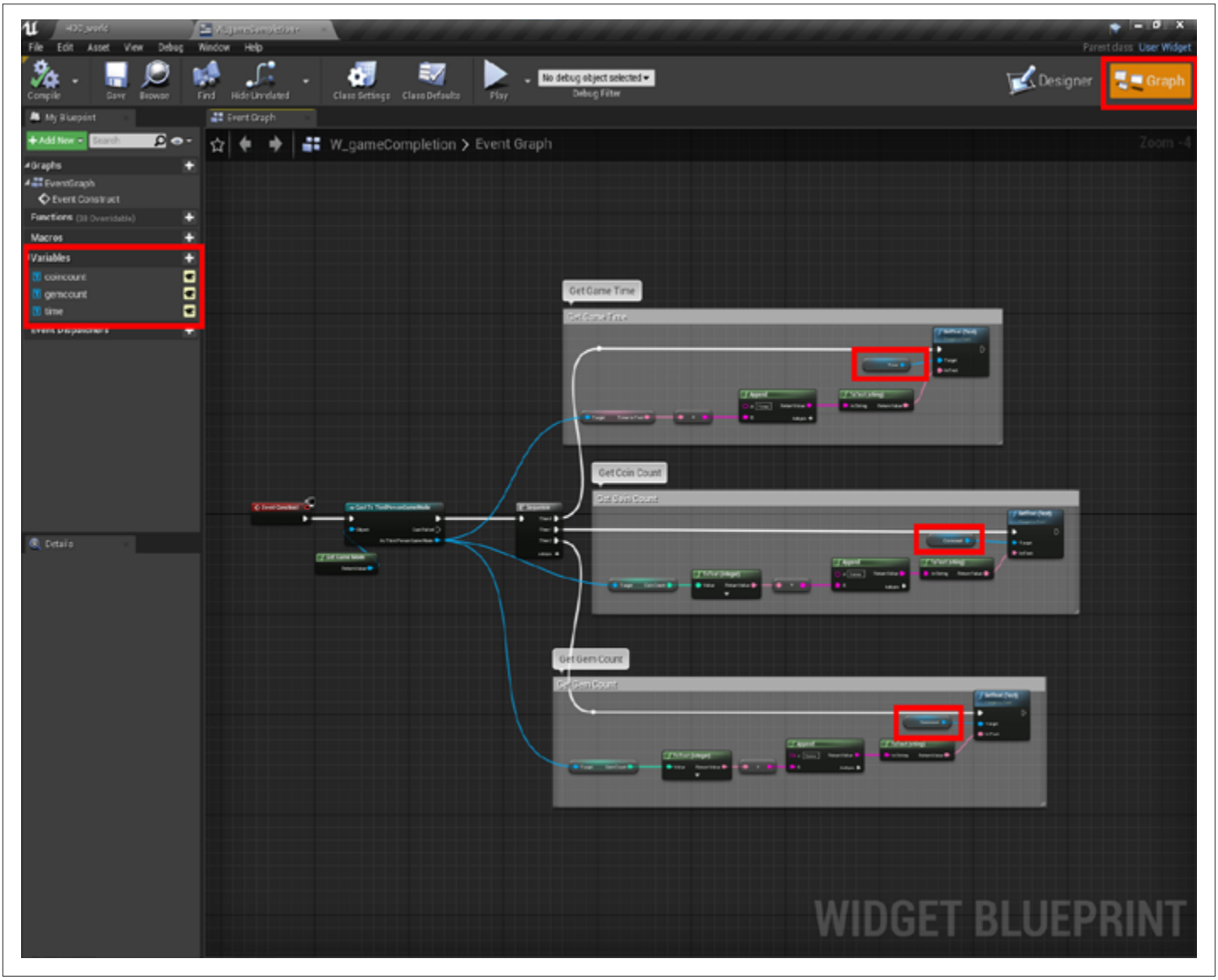

Fig 035 – Coins, Gems, and time on the Game Completion Widget Blueprint Event Graph.

By looking at this graph you can also see how the variables are being used as targets of the **SetText** node. Now follow the blue wires back, you can see that the **game mode** is where the data is being called from.

Remember that **casting** is one way that Blueprints can talk to each other. Here the **HUD** Blueprint is **casting** to the **game mode** to get information about the **coins**, **gems**, and the **time** it takes for the player to complete the level.

Even though we didn't change anything in these Blueprints remember to **Save All** your work. It will be important for the last section.

Packaging the game

With all the work you have completed in this course, let's package up the game so you can add it to your portfolio or just send it to your friends.

We will need to make sure that they can quit the game when they want to though. To do this, open the **EpicCharacter** blueprint from the **Content > ThirdPersonBP > Blueprints** folder of the **Content Browser**.

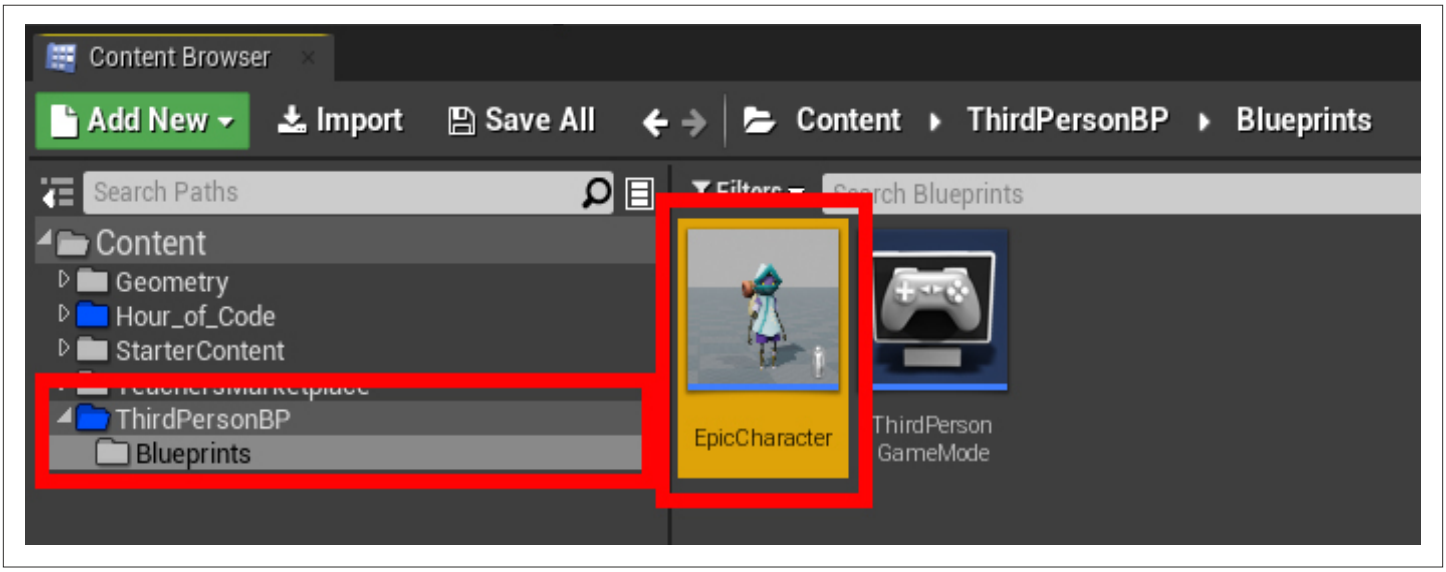

Fig 036 – The Epic Character Blueprint.

Locate the blue area labeled **Esc to quit** inside the **DEBUGGING** area. Then connect the **pressed** execution pin to the **Quit Game** node. Then **compile** and close the Blueprint Editor. **Save All** of your work.

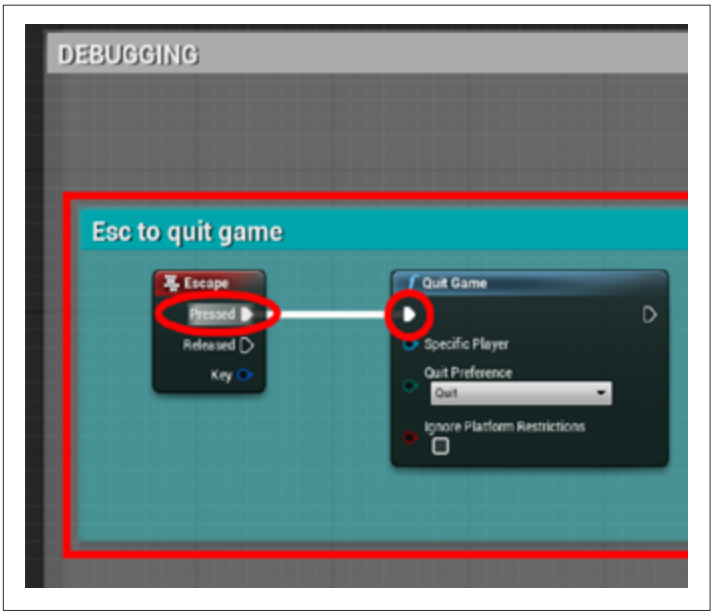

Fig 037 – Esc to quit game nodes.

You may need to install Visual Studio to package your game. This link will take you to where you can download Visual Studio.
https://visualstudio.microsoft.com/downloads/

Then follow this quick tutorial to get it set up for the packaging of your game.
https://docs.unrealengine.com/en-US/Programming/Development/VisualStudioSetup/index.html

Setting Up Visual Studio for Unreal Engine
You will want to make sure to install these 2 features of Visual Studio to package your game.

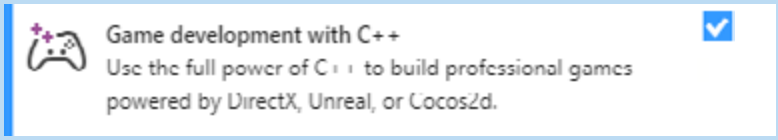

Fig 038 – Visual Studio Game development with C++ feature.

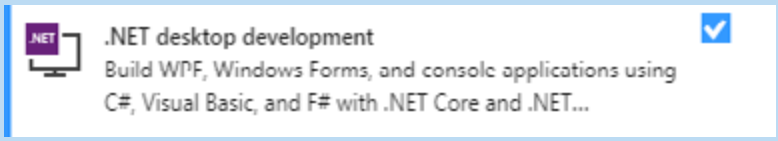

Fig 039 – Visual Studio .NET development feature.

This step will require a reboot, so be sure to save all your work.

Once it is downloaded and installed, you can package the game.

Go to **File > Package Project > Windows (64bit)**, then choose the location where you wish to save your packaged project.

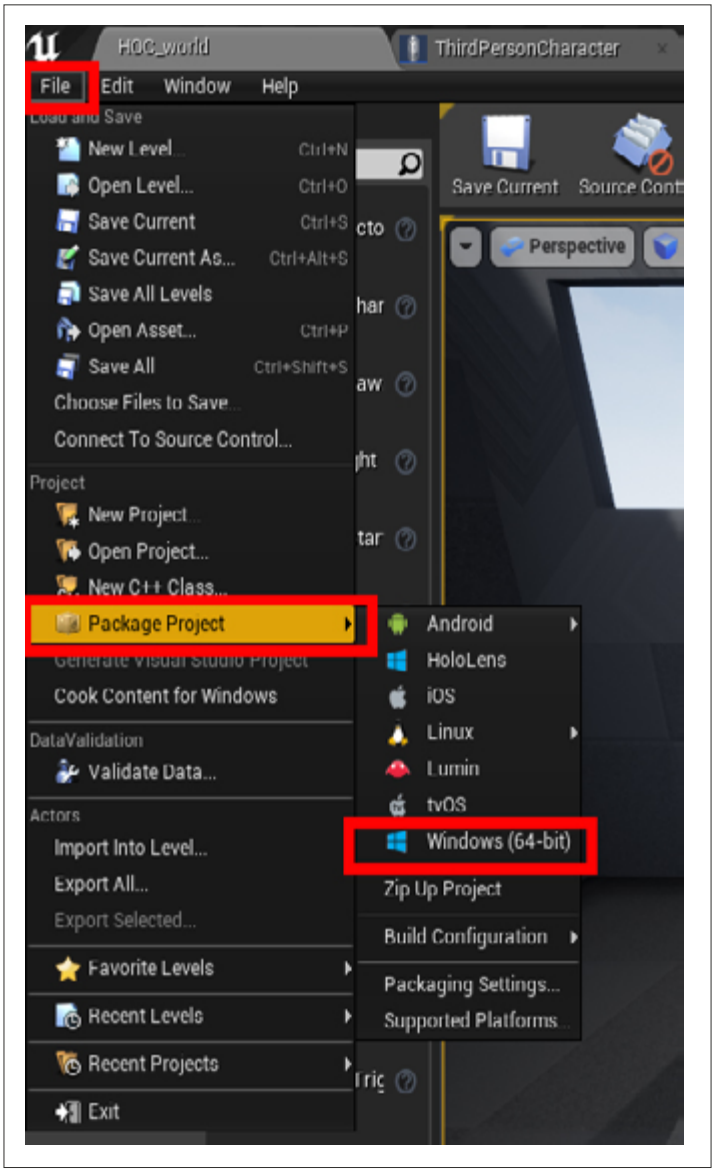

Fig 037 – Esc to quit game nodes.

Once the game has been packaged up you will see a folder called **WindowsNoEditor**. You can rename this folder if you wish. Simply open that folder and double-click on the **Hour_of_code.exe** to play the game.

Moving Forward

Congratulations! You have completed this series of activities. You now have enough knowledge to know what sorts of questions to ask as you further your game development career. Keep this project for your notes and refer to it as needed.

We have one last treat for you. Open the **levels** window and set all the levels to **always loaded** and play the game. This will give you a chance to see how we designed the level. Enjoy!