HOUR OF CODE

UNREAL ENGINE

REACH NEW HEIGHTS WITH POWER-UPS AND COLLECTIBLES: WORKING WITH PUBLIC VARIABLES IN UNREAL ENGINE

TEACHER GUIDE

Activity 3

Reach New Heights with Power-ups and Collectibles: Working with Public Variables 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

This is the third activity in the five-part Hour of Code series for Unreal Engine. When the player reaches this level, they will have navigated our parkour challenge and defeated our moving-island mayhem. Inconceivable! The player is getting closer to finding the key and unlocking the secret door.

In this activity, you will be adding a power-up that allows the player to reach platforms that are much higher than they can reach right now. We will also look at the code that makes this power-up work, giving you a chance to make changes to improve the gameplay. Then you will add coin pickups to entice the player to explore new areas of the map.

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

In this activity, you will be introduced to **Blueprints**, the visual scripting system in Unreal Engine. You will learn how Blueprints are used to **change the attributes** of the game by manipulating the information stored in **variables**.
A variable can be any type of information that is saved in a way that it can read and modified at any time. Common uses of variables include tracking the score and the time in a game. You will be using variables to keep track of collected items and the effect of power-ups.

Preparing the Activity

If you have finished Activities 1 and 2, you can skip this section and choose to build upon the work you've already completed.

If you are starting here at Activity 3 or would like to have a fresh start, please follow these instructions for starting a new project.

Since we are starting at Activity 3, you will need to load the completed sample versions of Activities 1 and 2, 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 and 2.

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.

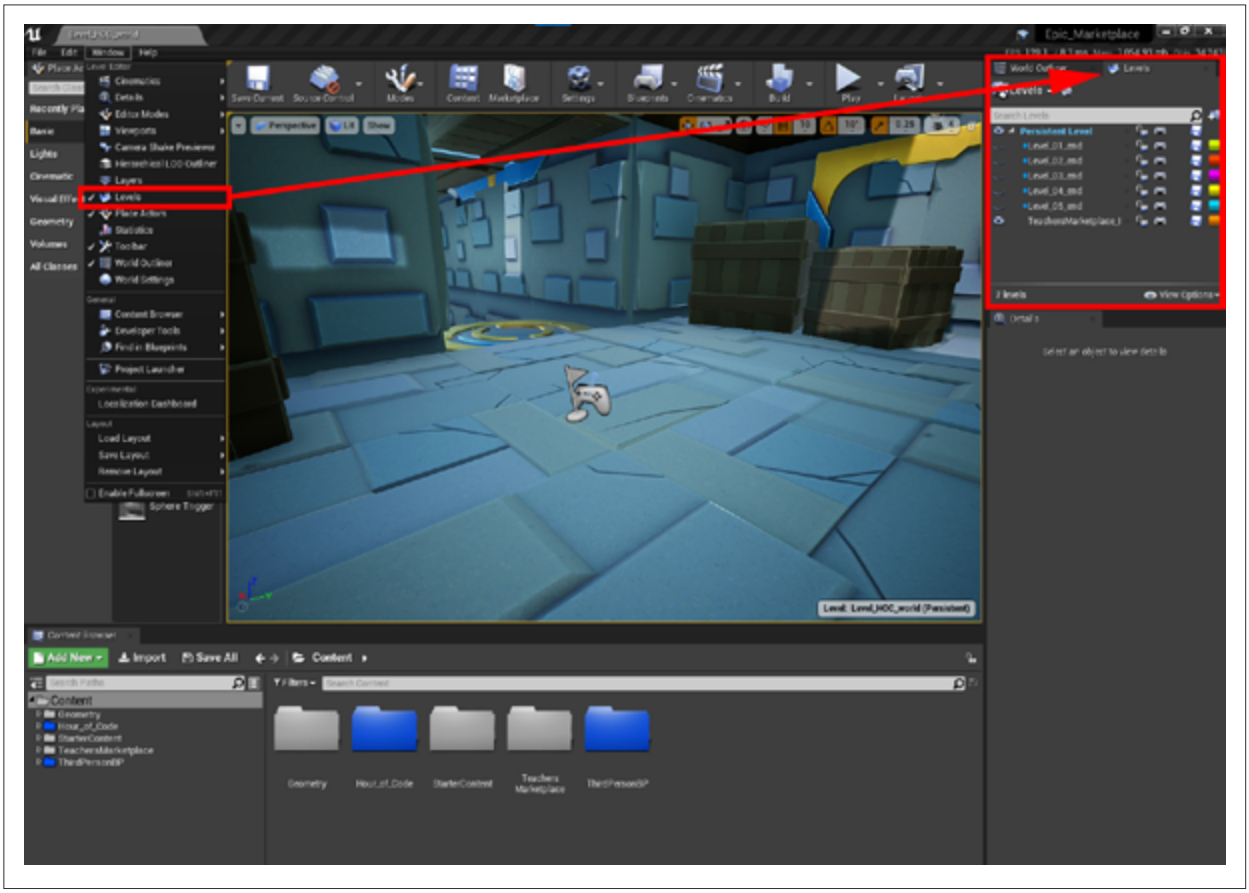

Fig. 00la – Move the Levels panel next to the World Outliner panel.

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**. You don't need to load the other levels, so you may choose to leave the **Blueprint** option checked for now.

Teacher Note

If a student joins the activity without completing the first activity, you can use these example levels. They also will allow you to break up this entire project to be taught in any order you wish, or only teach specific activities as you see fit. For example, if you are starting on activity 3, make sure the student toggles activity 1 and 2 to Always Loaded.

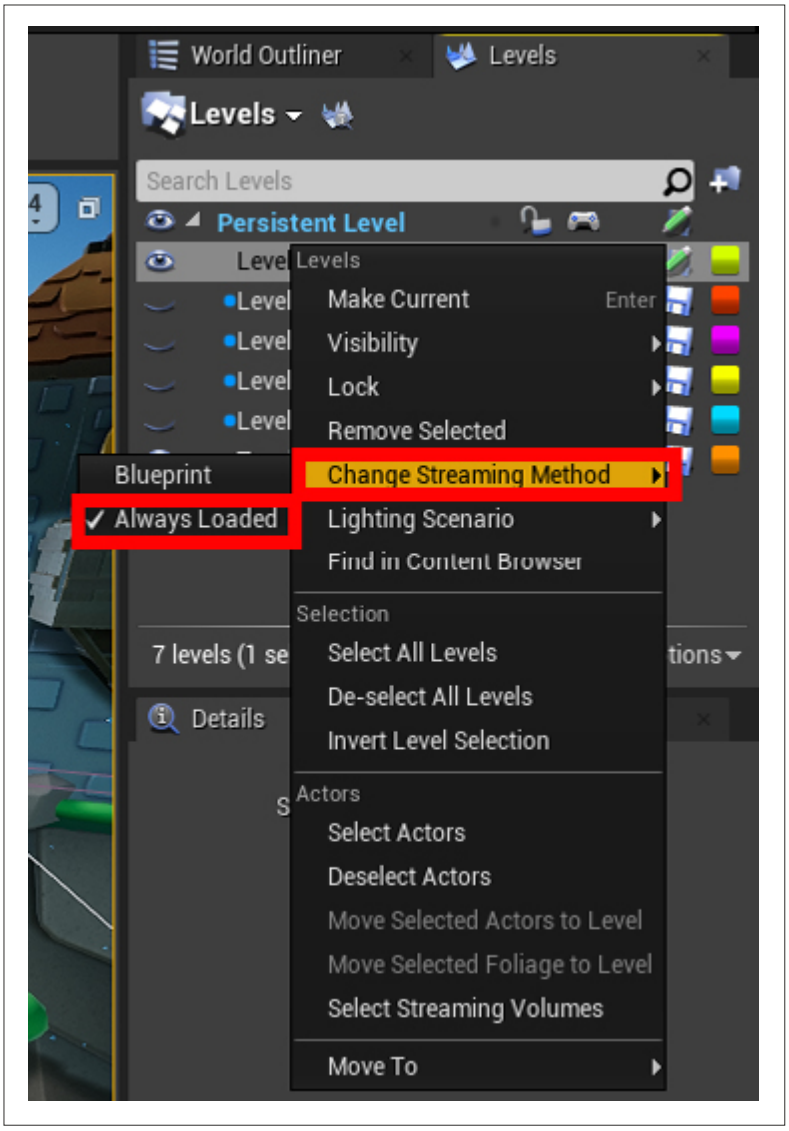

Fig. 001b - Loading the Level 1 & 2 samples if you didn't create your own.

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**.

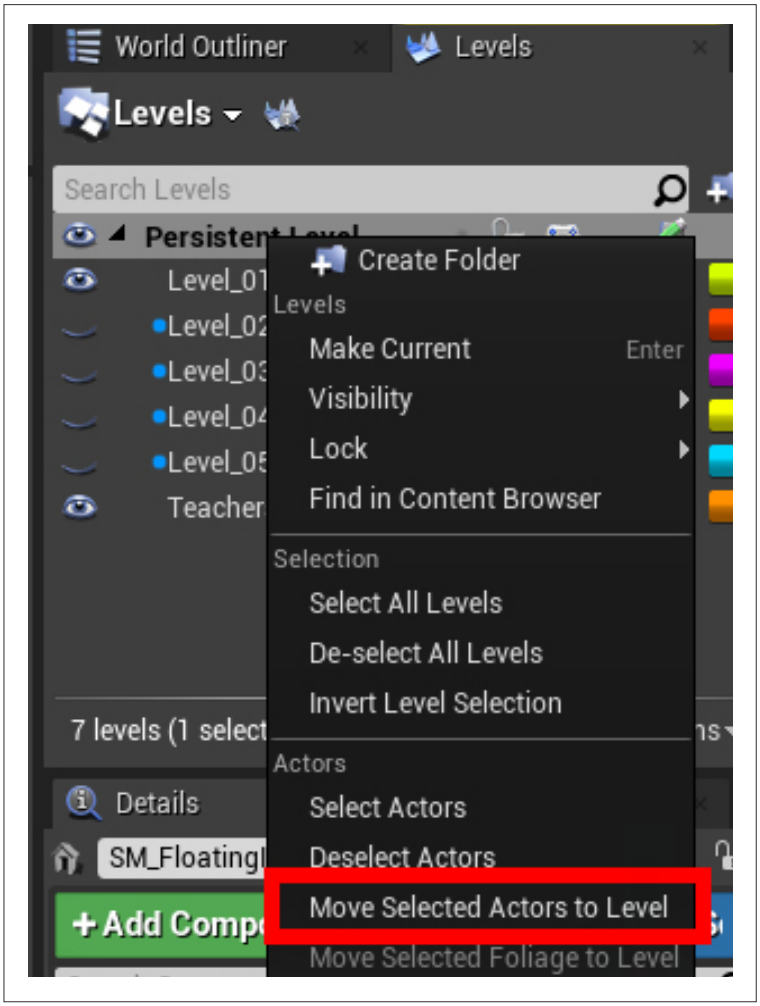

Fig. 001c – Moving actors to the Persistent Level if accidentally placed in the Level_## level.

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.

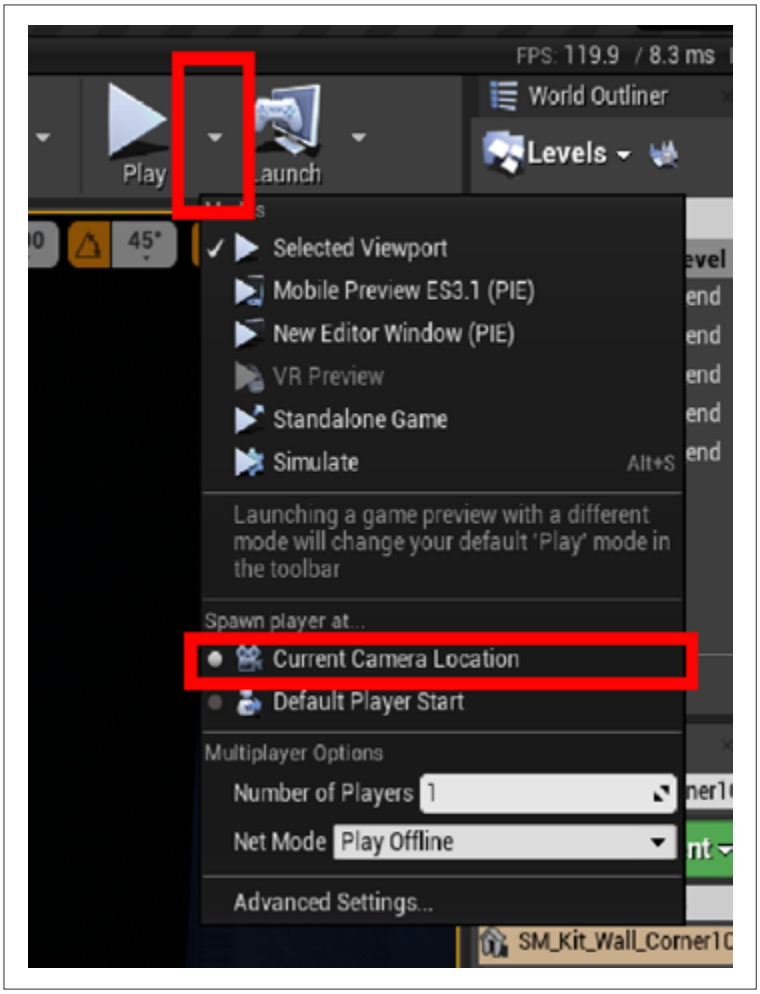

Fig. 001d – Setting Play button to start from current camera location.

Let's Play!

The use of collectible objects, such as coins, can be used to track progress and/or keep score. Coins can also be used to guide the player through certain areas and to tempt them into difficult to reach areas. These items could be optional. Collectibles could also be required, like a power-up that temporarily allows the player to get a jump-boost to help them reach an otherwise unreachable location.

Many games use these gameplay mechanics. One of the most popular examples combines mushroom power-ups and coins to aid a character in their quest to save the princess.

Some games also require a player to complete a quest by collecting a specific item, such as a key that will be used later to open a locked door. This gameplay mechanic will be covered in Activity 4.

Let's start by playing the game from our current large floating island. Click in the **Viewport** and press the number key '**3**' to move to our camera 3 bookmark. Make sure the **Current Camera Location** option is selected by navigating to the **Play** button options and pressing Play. You will notice that the next floating island is too high and out of reach.

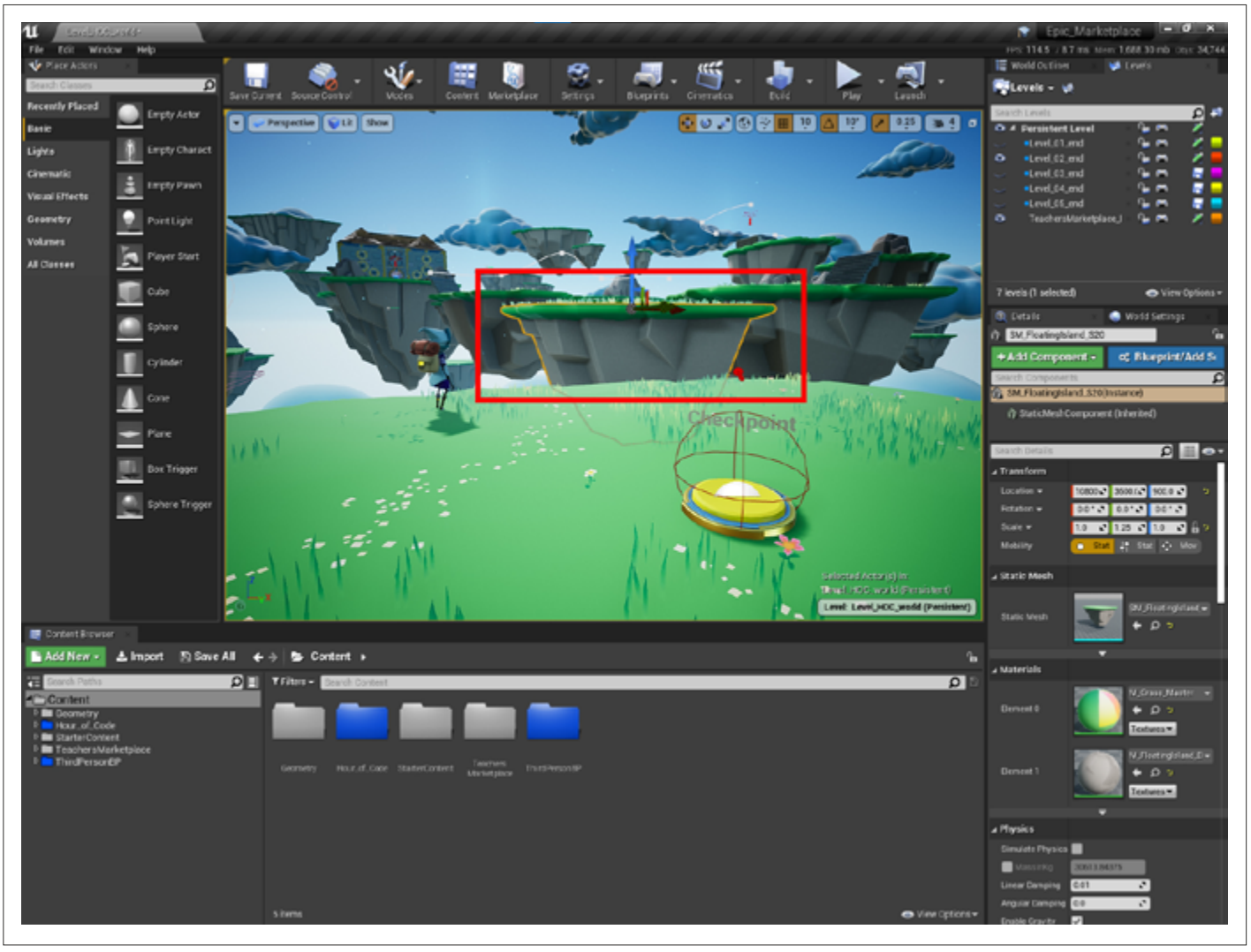

Fig 001 – The first jump in Activity 3 is out of reach.

Jump-Boost Pickup

We could create additional platforms, or we could animate a moving platform, but we have already completed that in previous activities. Therefore, in this activity, we are going to work with a simple power-up that changes the **Character Movement** variables.

For our purposes, think of a **variable** as a value, or a number, that can be changed at any time during the game to achieve the desired result. In our case, we want to temporarily change the **jump velocity**. Simply put, we want to jump higher!

In the **Content Browser** navigate to **Content > Hour_of_Code > Blueprints** and drag the **BP_jumpBoost** Blueprint into your Level onto the large floating island where the player can reach it.

Pro-Tip: Power-ups are special. Let's make them appear to be floating above the ground. After placing your power-up in the world, press the **W** key to enable the Select and translate (move) gizmo. Grab the blue arrow and drag the power-up just slightly above the ground.

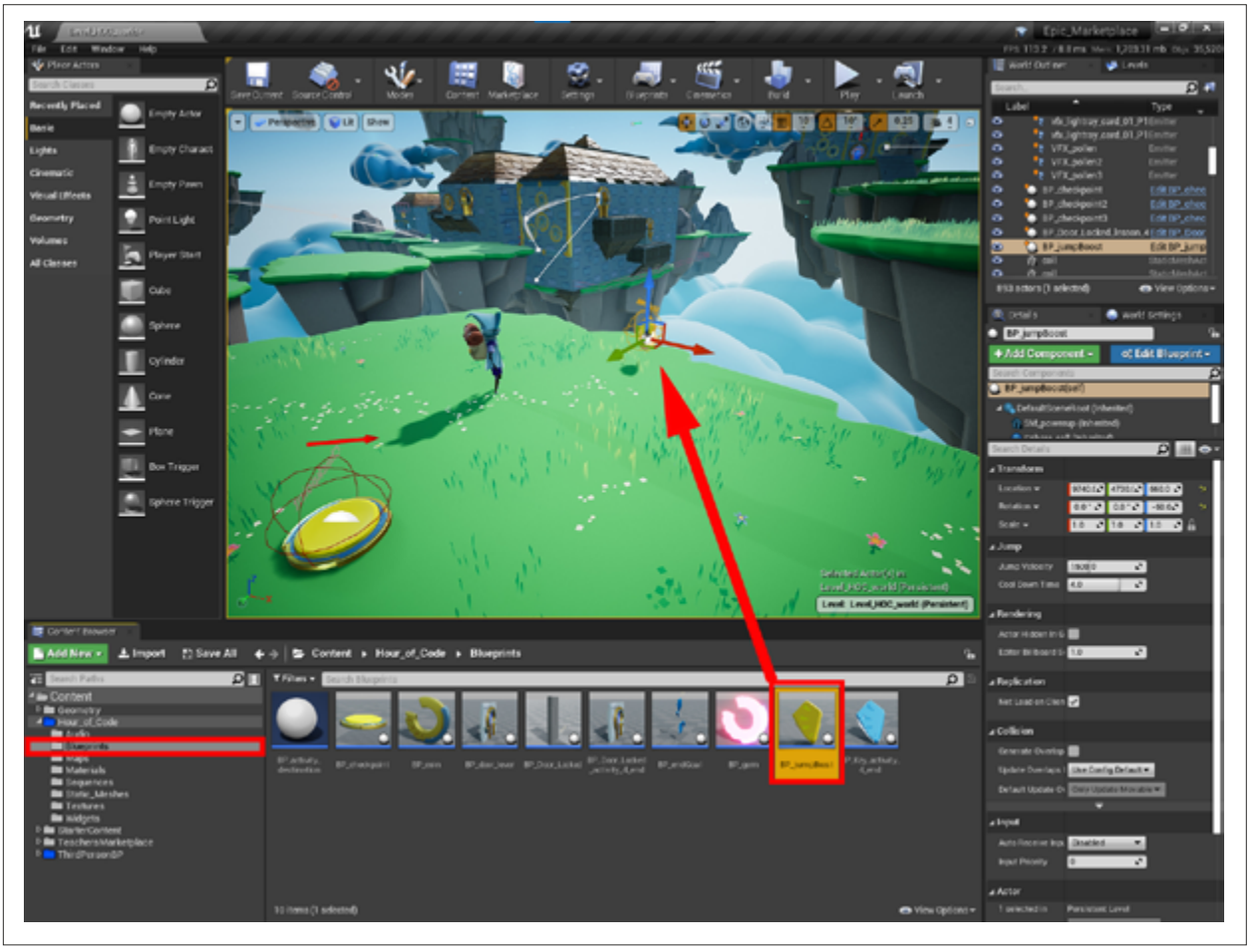

Fig 002 – Placing the Jump Boost collectible on the starting island.

First, play the game and walk slowly towards the checkpoint to avoid falling off the edge. Then walk over the jump boost. Now you can jump high enough to reach the next floating island. This power-up is temporary so be quick, it will only last 4 seconds! If you die, don't worry the power-up will respawn after 4 seconds, giving you another chance.

Customizing the Jump Boost Power-ups

Now that you've had a chance to see how this works. Let's play with some of the **Public Variables**. Select the **BP_jumpBoost** in the Level, then look in the Details panel. These **Variables** can be found under the **Jump** section. Simply click and drag on these sliders to change their value, or you can also type a value in the field. These specific **Variables** store a type of number called a **Float**. A simple way to think about a **Float** is that it's a number that contains a decimal point.

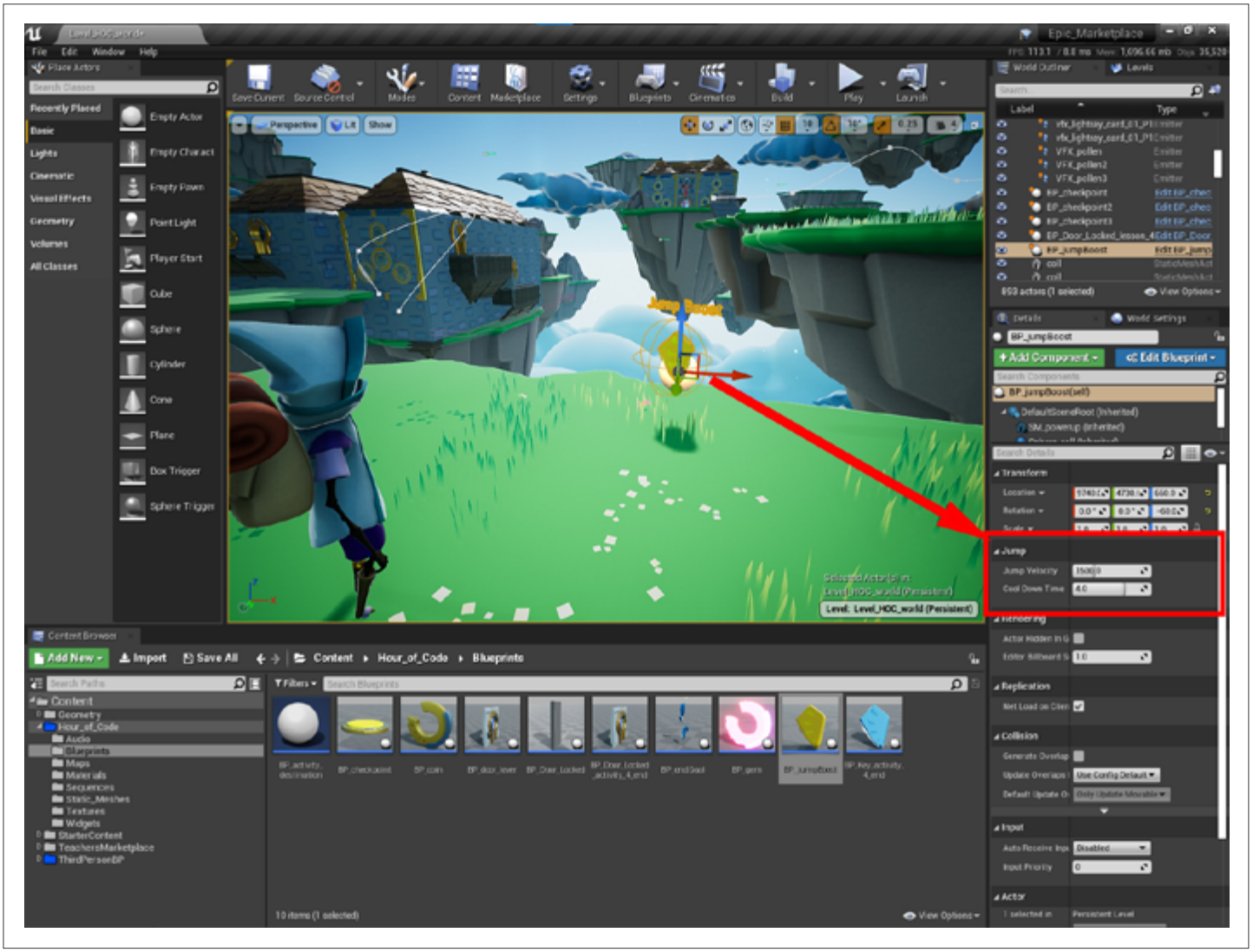

Fig 003 – The Jump variables.

By changing the **Jump Velocity**, you can make the character jump higher or reduce how high the character can jump. The **Cool Down Time** changes how long the player will have the power-up in addition to when it will respawn.

Game Design Fun: Each **BP_jumpBoost** you place in the Level can be set up to have different values. Try setting a few of these up in your level and see what it takes to get the player to jump back to the second tunnel.

Pro Tip: The little yellow arrows  next to the sliders enable you to reset them to their default values.

Looking at the Jump Boost Code

Next, let's look under the hood to see what is happening. Select the **BP_jumpBoost** in the Level and press **Ctrl + E**, or double click it in the **Content Browser** to open the Blueprint Editor. You can maximize the Blueprint Editor if it isn't already maximized by default when it was opened.

Teacher Note

Blueprints is the visual scripting language used for Unreal Engine. In this example, the purpose is to expose students to the existence of **Blueprints**. As we explain what the code is doing; Students in this activity will be changing settings of the existing Blueprints to understand the concepts presented in the activity, however, they are not expected to create their own scripts at this level of complexity.

Navigate to the **Event Graph** tab. Then locate the **Variables** section on the left-hand side of the interface and open the drop-down for the **Jump** variables category.

NOTE: You may notice a blue section at the bottom of the **Event Graph**. This can be ignored as we will be addressing this later in Activity 5.

Blueprint Basics

At a glance, **Blueprints** are blocks connected with virtual wires. The points where wires connect to blocks are called **Pins**. Let's review some important concepts used in Blueprints.

- The white pins are called Execution Pins. Think of this as the power supply and the sequence of events. Blocks will run in the order they are connected with the white wire.
- Each block can have pins on the left and right sides. Pins on the left side are used for providing the inputs that the block will need to execute. Pins on the right side are outputs that can send the result of the execution to another action or save it to a variable.
- To connect wires, click and drag from one pin onto the destination pin, and release the mouse button.
- Unreal Engine is an event-driven environment. You'll notice that the first block is activated by a specific event. That means this code sits and waits for something to happen, like the player touching a power-up, which triggers the execution of that code sequence.
- Blueprints make good use of color-coding to help quickly identify block types and compatible connections. You'll notice that only the same-colored pins are connected together.

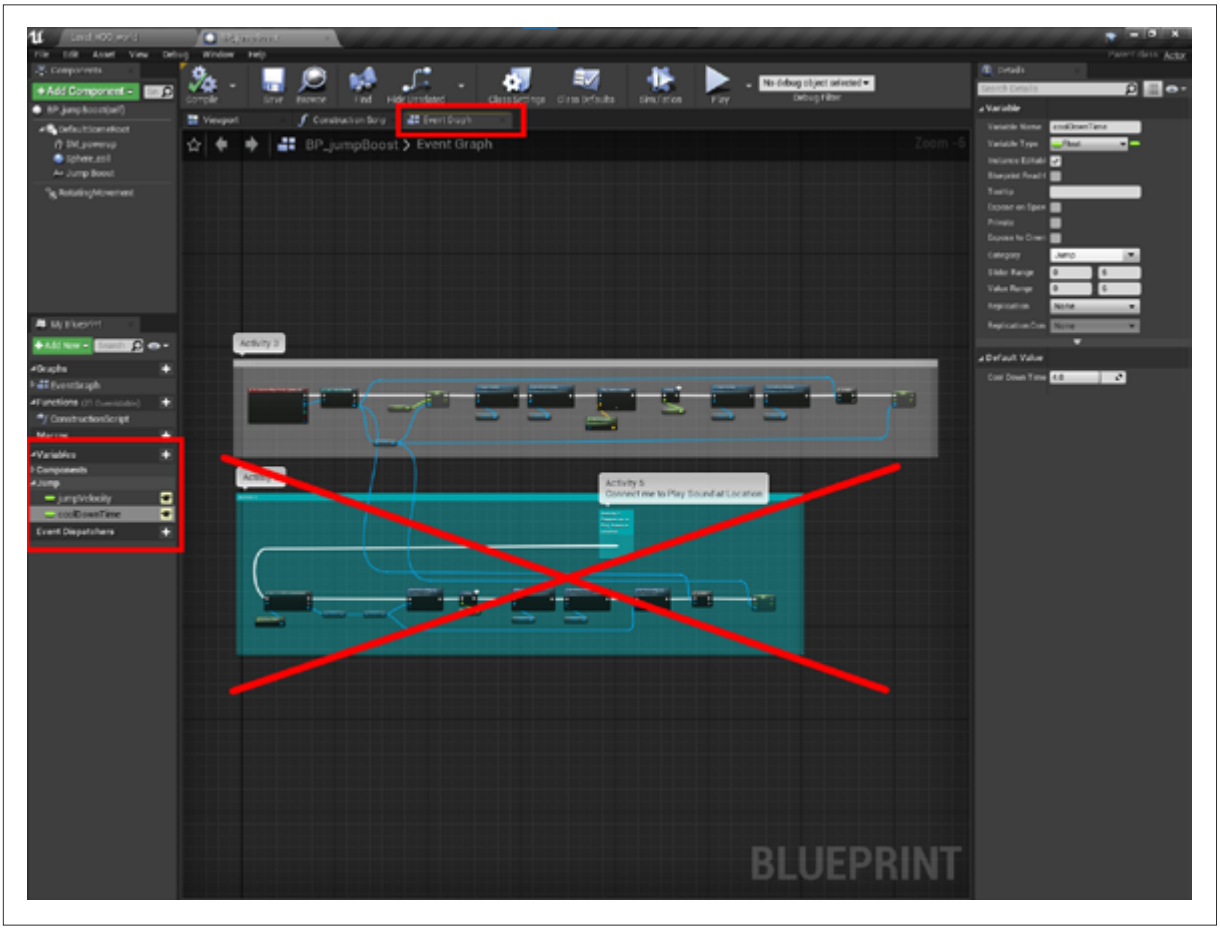

Fig 004 – The Event Graph tab in the jump boost Blueprint.

You will see that the **jumpVelocity** and **coolDownTime** have little open eye icons next to them. This is what makes them **Public Variables**. This is handy for game designers because it allows them to make changes to the code without having to open the Blueprint. This makes for faster iteration when playtesting, which you have experienced when modifying the 'jumpVelocity' and 'coolDownTime' of the jump boost power-up in your level.

Now, let's examine the code closer. Using the **scroll wheel** on your mouse, zoom in to the left-hand side of the **Activity 3** section. Use the **right mouse button** to pan within the Blueprint Editor Graph. We will break the code into 7 sections to help make this more manageable to understand. We can look at the power-up itself to get an idea of what it is made of. The image below points out the two main pieces, the **Sphere Collision**, and the **SM_Powerup**.

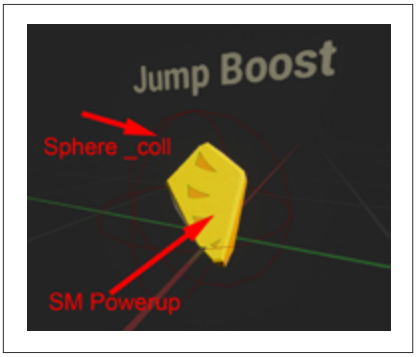

Fig 005 Jump Boost as shown in the Viewport tab in the Blueprint Editor.

1. The character **Overlaps** the **Sphere** collision, then makes a call (**cast**) to the **Character**. **This indicates that the following code will be executed once the player collides with the sphere (and thereby touching the power-up)**.

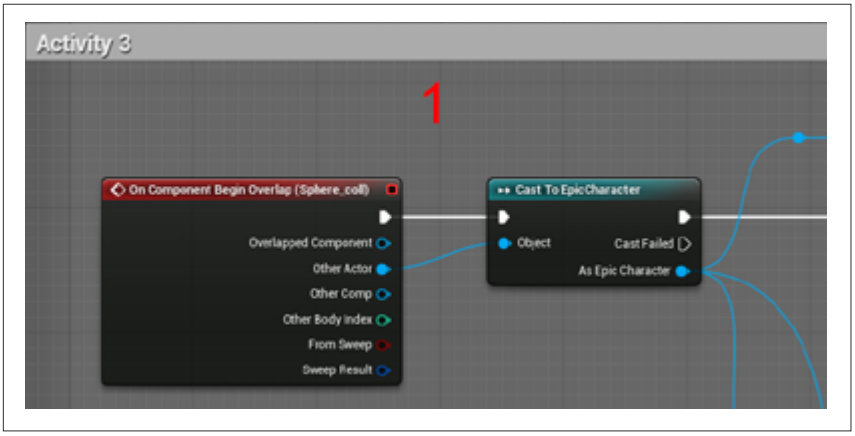

Fig 005a – The code starts when the player touches the Collision Sphere of the jump boost powerup.

2. The **Character** then changes the **Jump Velocity** value of the **Character Movement**, setting the value to a number chosen by you; the designer.

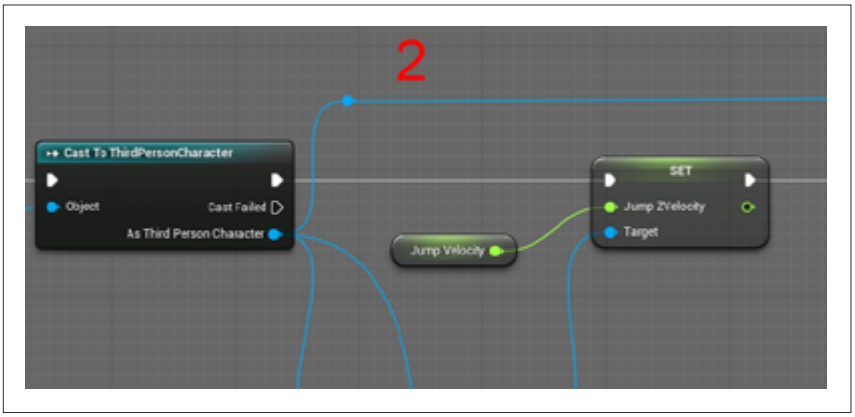

Fig 005b – Setting a new value for the player's jump velocity.

3. Next, the **SM_Powerup** is being turned invisible by toggling its visibility OFF, and the **Collision** of the **Sphere** is being set so that the character can't interact with it.

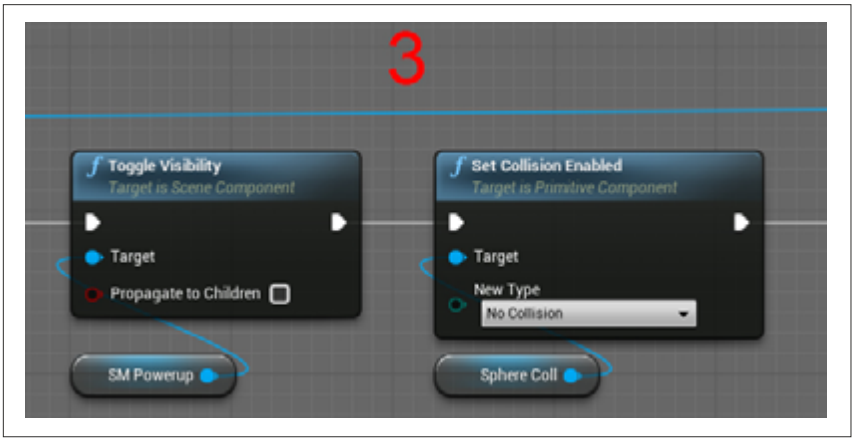

Fig 005c – Hiding the jump boost after it has been collected.

4. This section tells the game to play the pickup sound at the location of the Blueprint.

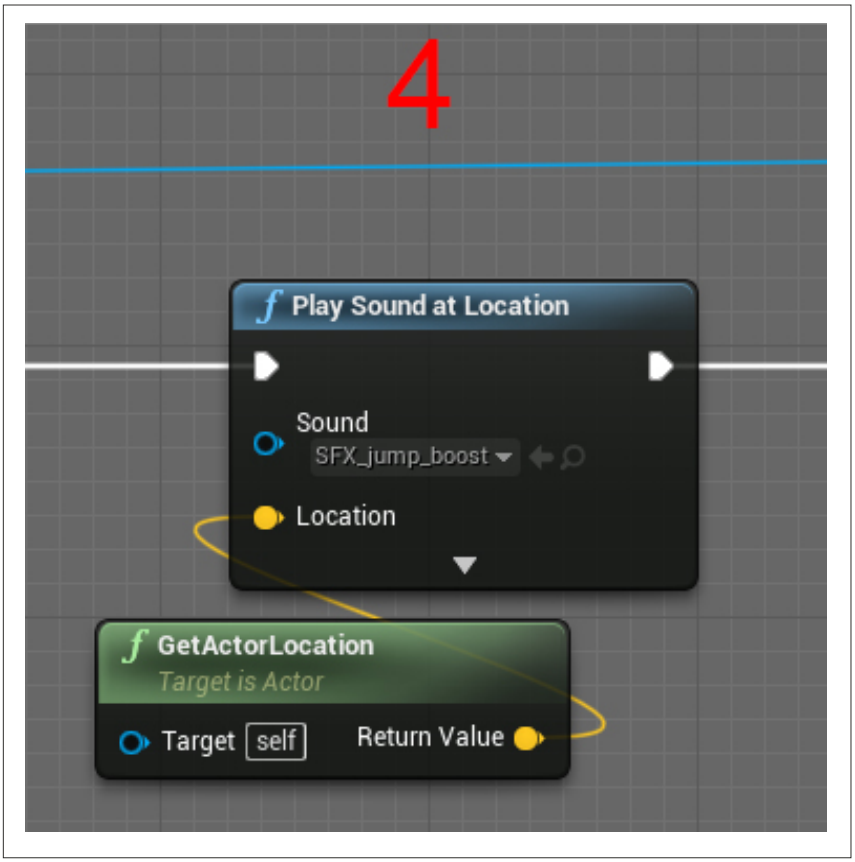

Fig 005d – Play a sound when the jump boost is collected.

5. Next, there is a delay set by the value of the **Cool Down Time variable**, Then the visibility of the **SM_Powerup** is toggled back ON, and the **collision** of the **Sphere** is re-enabled.

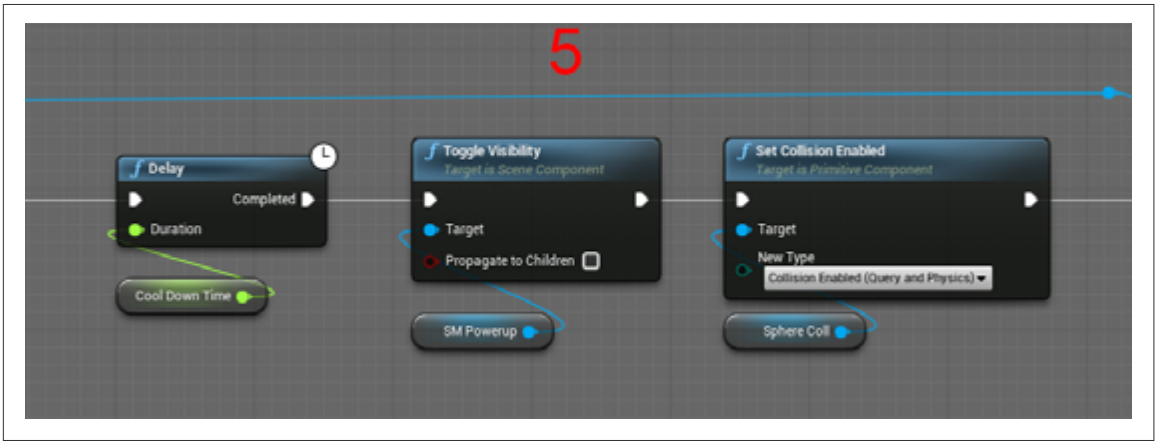

Fig 005e – Re-enable the jump boost pickup after a cool down time.

6. This node is checking to see if the **Character** is still alive, it's asking if the character **is valid**. If you follow the blue wire up and back, you will see that it is calling to the **Character**. Without this node, the game will report an error if the character dies when it reaches the last node.

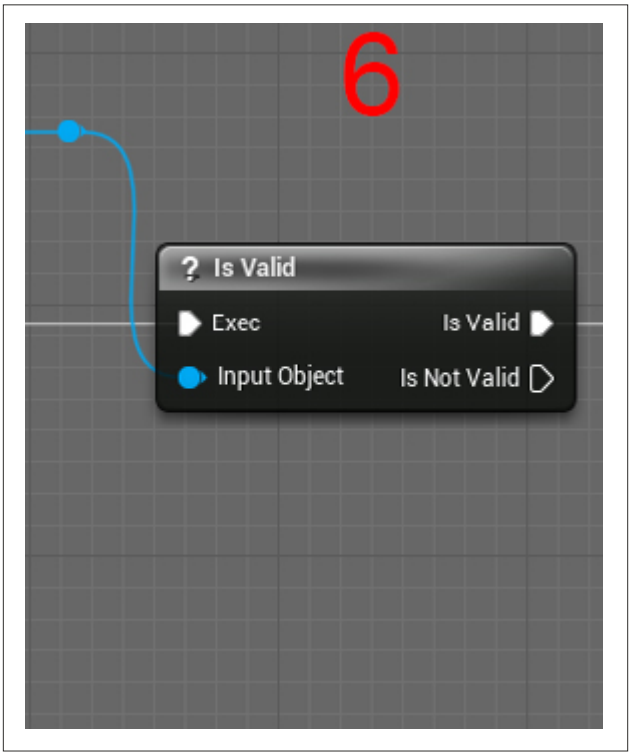

Fig 005f – Making sure the character hasn't died.

This last node is setting the **Character's Movement Jump Velocity** back to the value it was originally. Again, you can follow the blue wire back to where it is connected to the character.

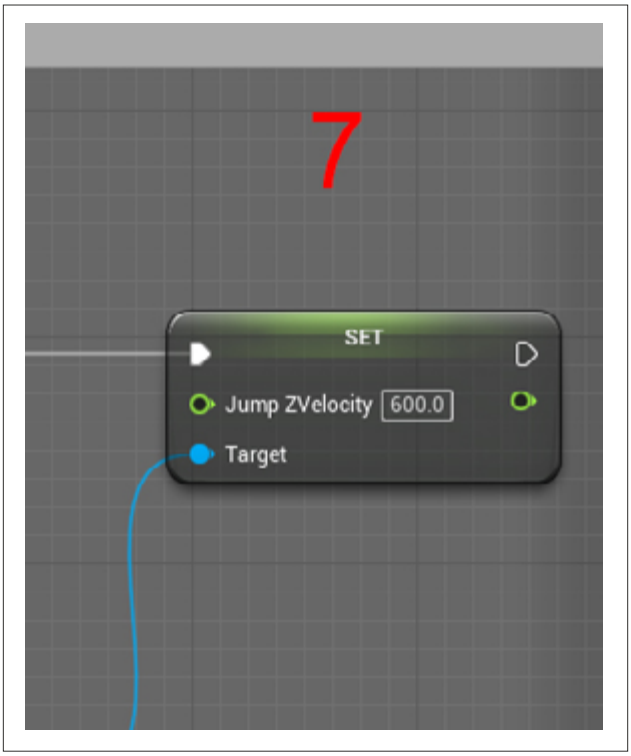

Fig 005g – Resetting the jump boost after the cool down time.

OK, that is a lot. So, let's recap in a single breath.

This Blueprint is allowing the character to jump higher for a short time, and the character is unable to pick it up again during that time. As a bonus, a sound plays to let the player know they have acquired the power-up. When the jump boost expires, the player's jump velocity will return to normal and they can collect the jump boost again.

Close the Blueprint Editor to get back to your project.

Remember to **Save All** your work.

Coin Pickups

In this section, you will add a few coins to reward the player for reaching new places.

Within the **Blueprints** folder, drag out a few copies of the BP_coin Actor and place them in your level. (You can also duplicate objects in your world by holding the ALT key then moving the object with the Move gizmo.) You can reward players, encourage them to explore, as well as lead them into danger by having shiny objects lying around your level. These coins will work perfectly as a guide to a group of even more coins up above.

You'll notice that there is another island that the player won't be able to reach unless you give them another jump boost, so be sure to help them out.

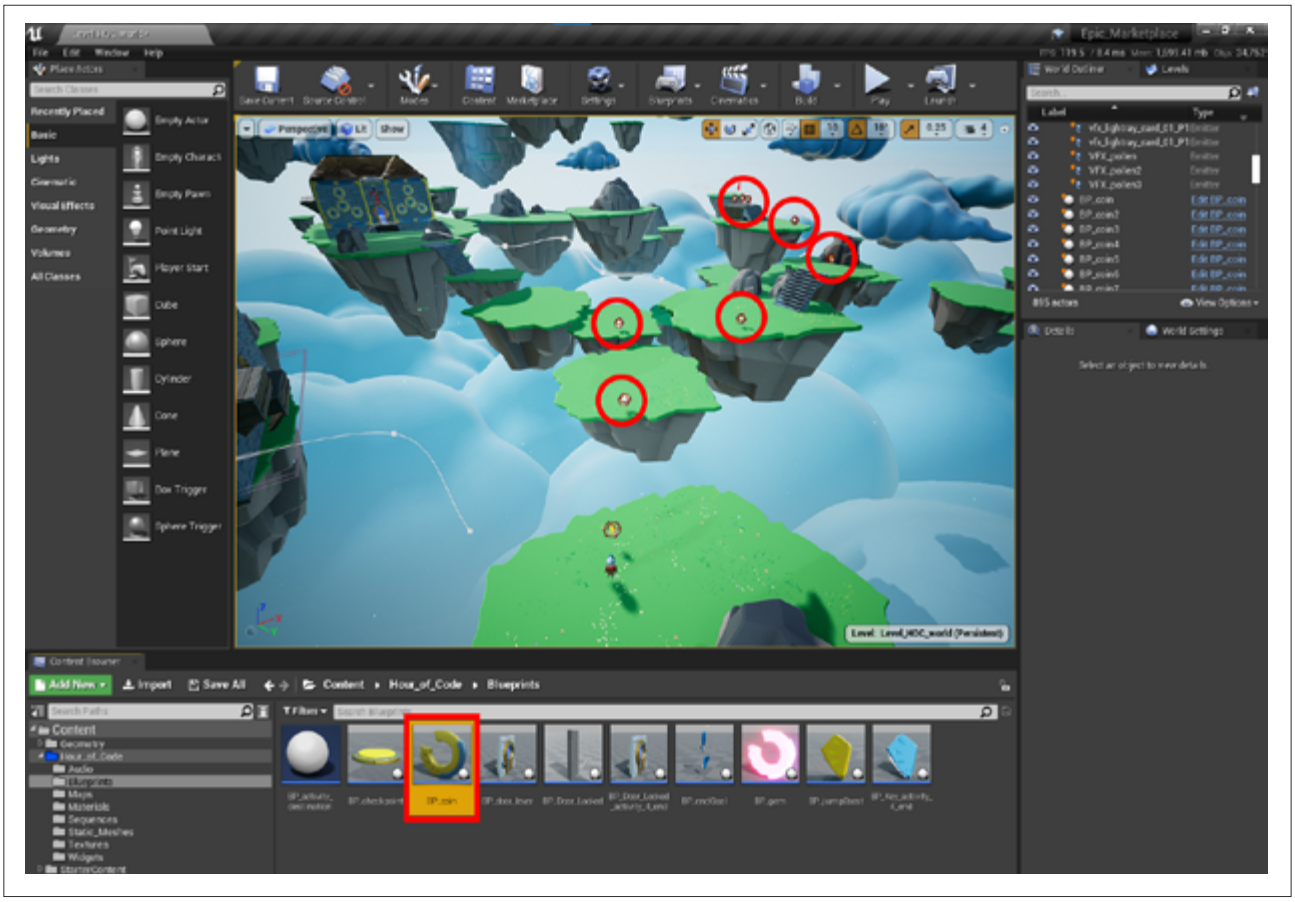

Fig 006 – Create a path of coins to guide the player.

Game Design Challenge

As we progress deeper into our game, we now present the player with multiple options of choice. As the game designer, you can use your level design and collectibles to guide the player. You may guide the user to the solution or you may choose to try tricking them into taking a detour. It's time for you to add the following items to the Level 3 area of this map.

Adding Coins – Add more coins to your level. Think about using coins to guide the player. Maybe, you can try to lead the player up to the large structure in front of them? They'll get to a locked door only to realize that they have to go back to find the key.

Add Jump Boost – A jump boost is needed to make the first jump, but that should already be in place. If the player navigates to the right, then there is another jump that needs a jump boost power-up. Place this additional jump boost powerup so the player can make the jump. This will be a great place to put the secret key. (In Activity 5)

Remember to ask your peers to playtest. Watch them play, but don't guide them or speak while they are playing. Are they following your intended path? Are they confused? Are they stuck? If your play-testers are having trouble getting to specific locations, you should try to correct it. If you've added extra complexity, it may be worthwhile to remove it. If the path is critical to the gameplay, you can work on improvements before your next play-tester attempts to try your game.

Up for a challenge?

Open the **BP_coin** and see if you can figure out how it works. Why would this Blueprint cast to the **Game** and not to the **Character**? You will also notice there is a section labeled Activity 5 inside the Coin Blueprint. The **Activity 5** section is where we will be hooking up the **Heads-Up Display (HUD)**.

Teacher Note:
This Blueprint is cast to the **Game**, because if the character dies, the coin counter will reset to 0, and the coins are set to be destroyed after they are picked up. Thus, the player can't pick them up again after the character respawns.

Remember to **Save All** your work.

The next activity will introduce a conditional, the key item, which will allow you to unlock the door, enabling your players to complete the game.