HOUR OF CODE

UNREAL ENGINE

THE KEY TO UNLOCKING THE FINAL AREA:
WORKING WITH CONDITIONAL STATEMENTS IN UNREAL ENGINE

STUDENT GUIDE

Activity 4

The Key to Unlocking the Final Area: Working with Conditional Statements 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

We have created jumping challenges, moving platforms, collectibles, power-ups, and now it's time to introduce the final challenge. The secret to completing the game is hidden behind a locked door in a fortress located high up in the sky.

You'll be using the Blueprint visual scripting system to create the secret key and hide it in your world. Armed with the Blueprints you've already created; the floating islands, collectible coins, jump boost power-ups, and a secret key, you'll be able to test your skills at game design. When the player finds the key, unlocks the door, and reaches the end goal, they'll see their score. Players can compete for the best score!

At the end of this activity, you'll be able to play your game from beginning to end.

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!

In this lesson, you'll learn how to work with **conditional statements**. The condition is finding a key to open the door. Once inside, you can complete the game.
Good Luck!

Programming Concepts

You will be digging deeper into the Blueprint visual scripting system of Unreal Engine. In this activity, you will be creating a secret key that will be used to unlock the final level. You will achieve this by implementing Boolean variables and a Conditional Statement; a concept in programming where code logic will only be executed when certain conditions are met.

When building your Blueprint asset, you will experience the process of solving a complex problem by using code to break it down into a series of smaller simple steps. You can connect these steps in a logical flow to complete your goal. Programming teaches the valuable skill of breaking big problems into a series of simple steps. As you master this process, you'll become a better problem solver and you'll be empowered to create anything you can imagine!

Preparing the Activity

If you have completed Activities 1, 2, and 3, you can skip this section and choose to build on the work you've already achieved.

If you are starting here at Activity 4 or would like to have a fresh start, please follow these instructions for starting a new project.

Since we are starting at Activity 4, you will need to load the completed sample versions of Activities 1, 2, and 3, 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, and 3. 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.

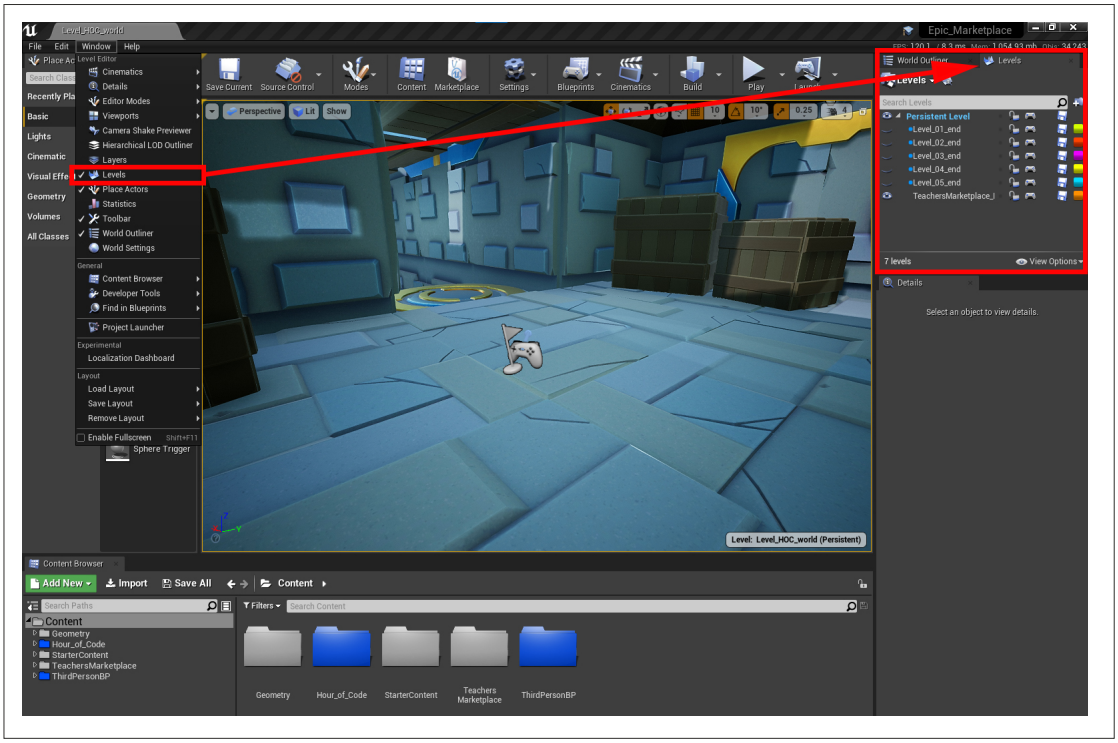


Fig. 001a – Move the Levels panel next to the World Outliner panel.

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** and **Level_03_End**. You don't need to load the other levels, so you may choose to leave the **Blueprint** option checked for now.

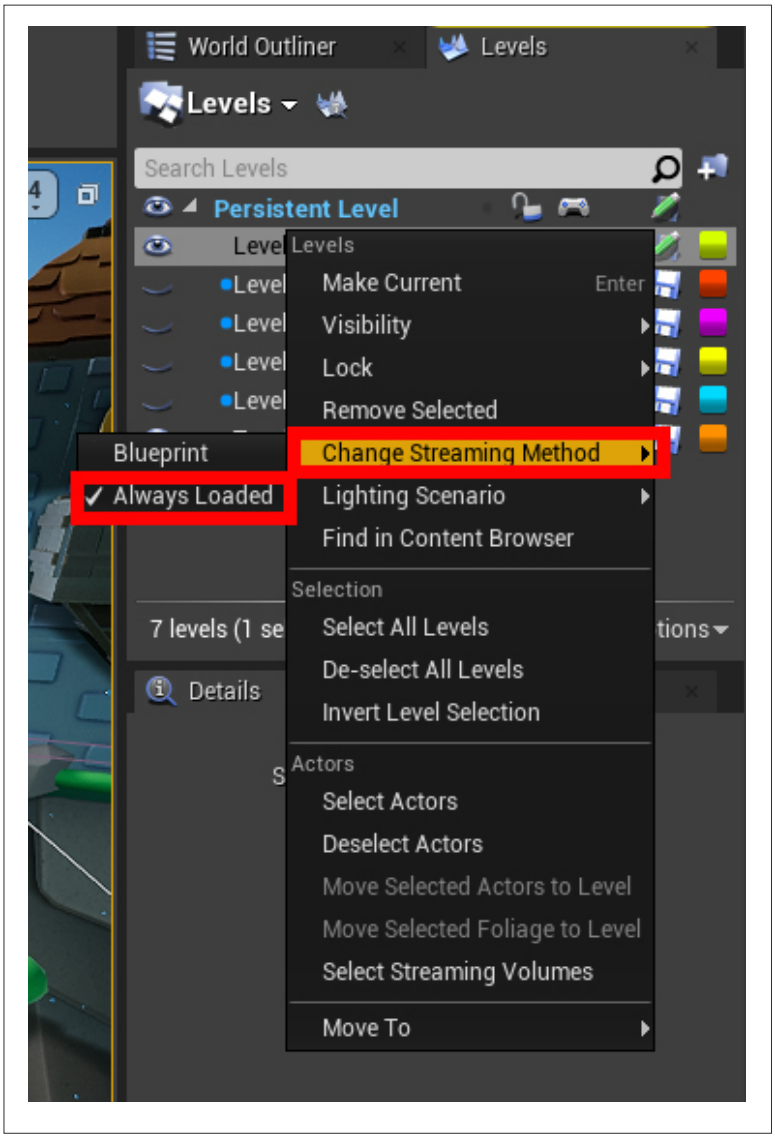


Fig. 001b - Loading the Level 1, 2, & 3 samples if you didn't create your own.

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**.

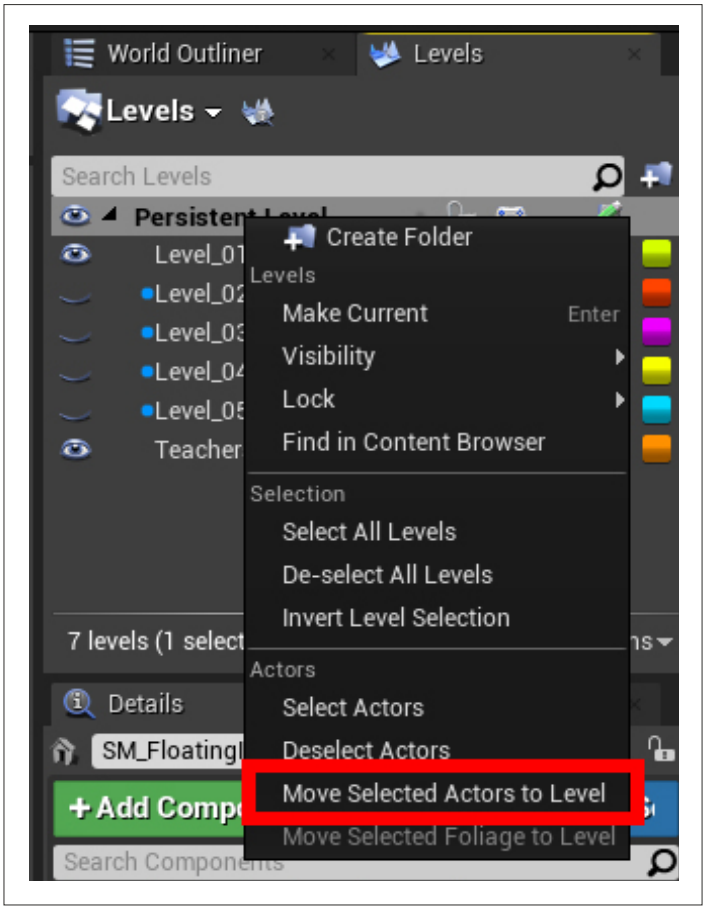


Fig. 001c – Moving actors to the Persistent Level if accidentally placed in the Level_## level.

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.

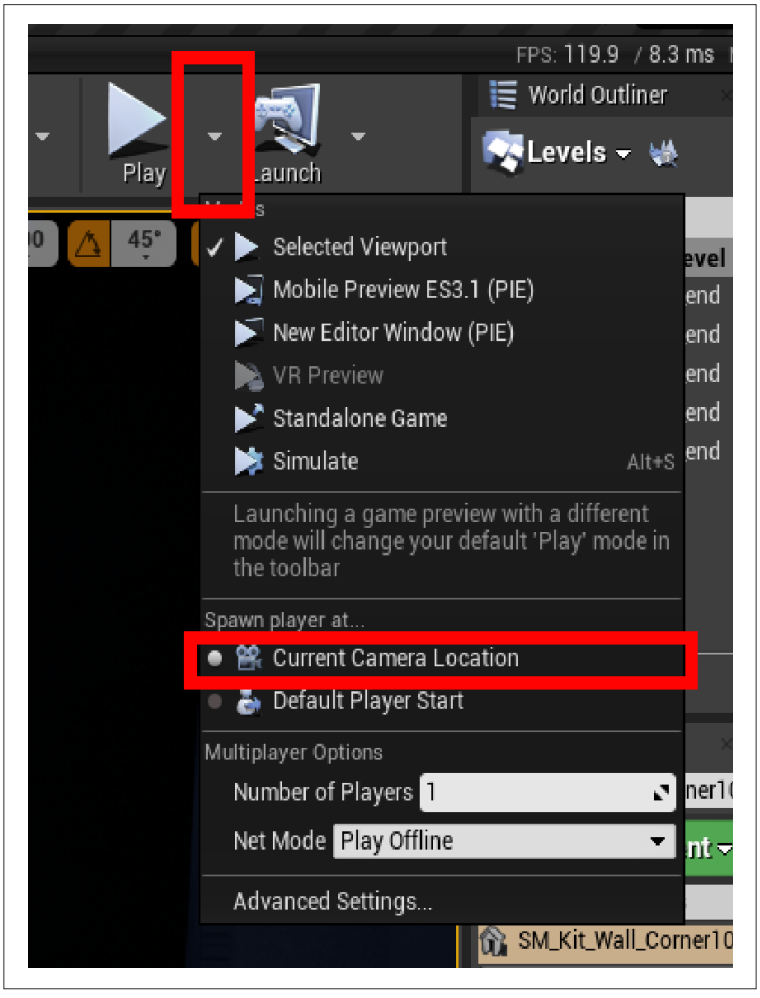


Fig. 00ld – Setting Play button to start from current camera location.

This series of lessons has been designed to introduce students to computer science concepts in the context of using the industry-standard game development tool, Unreal Engine.

Let's Play!

If you play the game and reach the door to the large building, you will notice that you require a key to open it.

Let's start the game at the door so you can test it out. Click in the **Viewport** and press the number **5** key to move to the camera 5 bookmark. Make sure the **Current Camera Location** option is selected in the Play button options and then press Play.

Walk up to the door. It should display the text, "Key Needed" and prevent you from opening the door.

Stop your gameplay here so we can get into the editor and make some magic happen!

Conditional Statements

A **conditional** statement checks to see if something is **true** or **false**. More commonly they are called **if/then statements** and they are also known as **if/then/else statements**. Simply put, they work like this.

If the **key** has been picked up = **True**
If the **key** has not been picked up = **False**

In Blueprints these are also known as a **Branch**, the branch node looks like this.

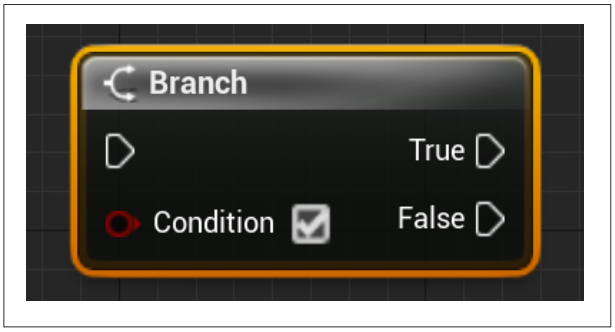
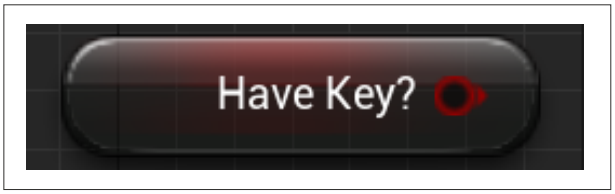


Fig. 001

What is a Boolean?

In Programming, the Condition can be stored as a **Boolean**, or **bool** for short. The data can be stored in one of two states, it is either **ON** (true) or **OFF** (false). We will use this stored data later in step 14. **Bools** are red, making them easy to see at a glance in your Blueprints.



What exactly are Blueprints anyway?

The main thing we will be doing in this level is building the "key" **Blueprint**. You can think of a **Blueprint** as a suitcase that contains objects as well as code, and that code can talk with various parts of your game.

For example, your suitcase can contain your cell phone, and your cell phone can talk with a GPS satellite. This would be super handy if the airline lost your suitcase because you can use the code from the GPS to talk to your cell phone and find your suitcase. Which as it turns out, landed before you did, and is on the baggage carousel behind you.

If you need to brush up on **Blueprints**, just skip back to the previous Hour of Code activities. If you feel like you are getting overwhelmed, talk with your peers to troubleshoot as you go along. Game creation is a team effort and should never be done alone. If your peers are asking you for help, be kind, and assist them with the help they ask for. This will help solidify your understanding and ensure they assist you in the future.

This next section has 19 steps. Think about what we are trying to accomplish and follow each step closely to understand how we are building our solution.

To get an idea of how to hook up a **Boolean**, navigate to the **Content > Hour_of_Code > Blueprints** folder in the **Content Browser** and open **BP_Door_Locked** by double-clicking on it.

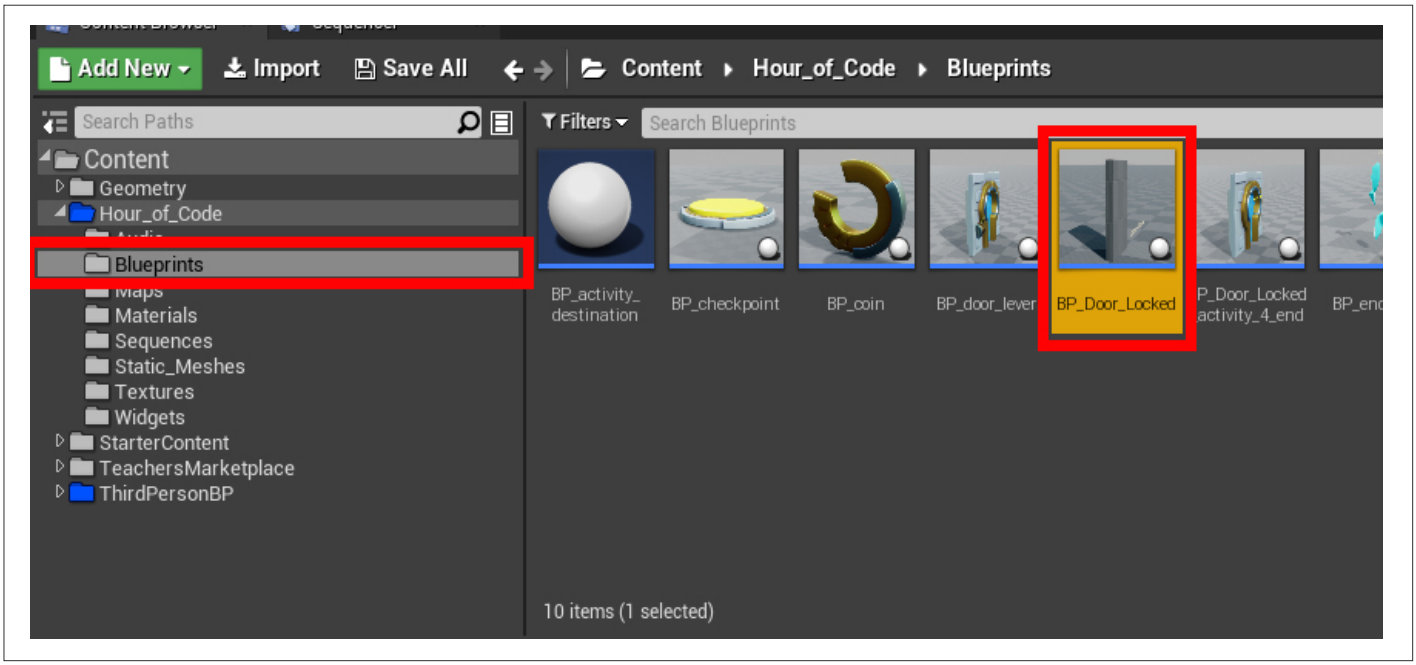


Fig 026 – The BP_Door_Locked Blueprint

Click on the **Event Graph** tab. You will see a yellow comment box area labeled **Activity 4**. **Currently, the Target is not connected to the branch. With Blueprints, you can click and drag to connect nodes when building your code. Try it now!**

Inside this yellow area simply drag a wire from the **Have Key**? **Boolean** pin to the **Branch Condition** pin.

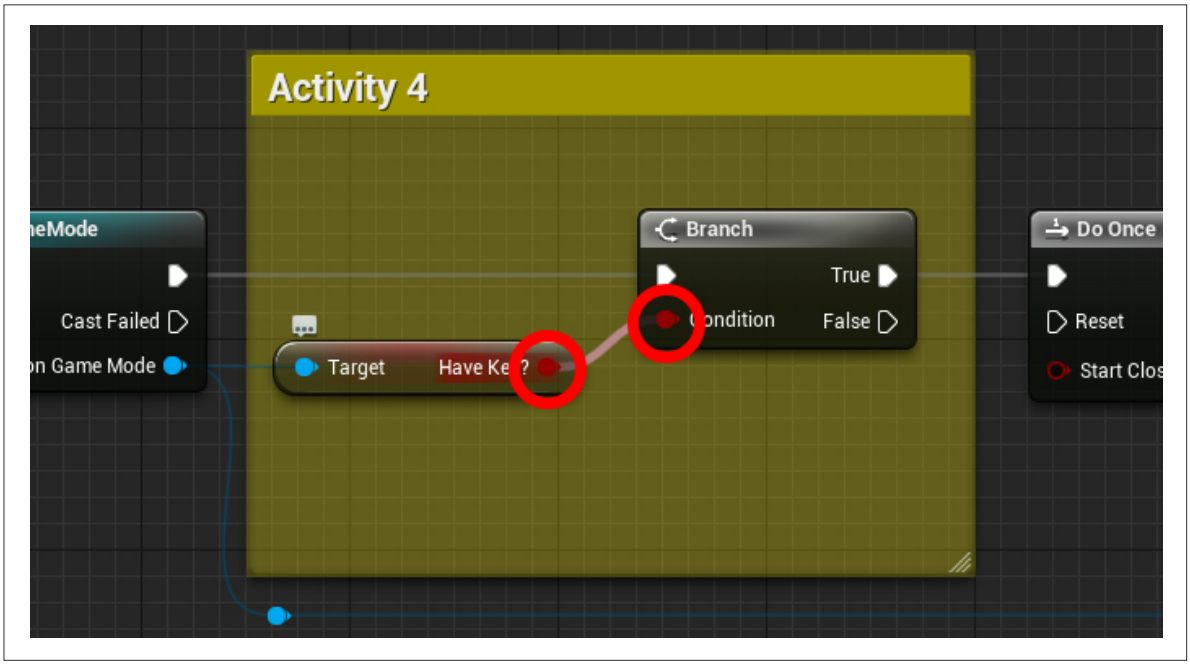


Fig 025 – Connect the "Have Key?" Boolean to the Branch Condition.

The Blueprint will now check the condition, "Does the player have a key?" This can be either True or False. Does that sound simple?

Let's create a key from scratch that will open the door.

1. To build a Blueprint Actor from scratch navigate to **Content > Hour_of_Code > Blueprints** folder, and **right-click** in an empty space. Then choose **Blueprint Class**.

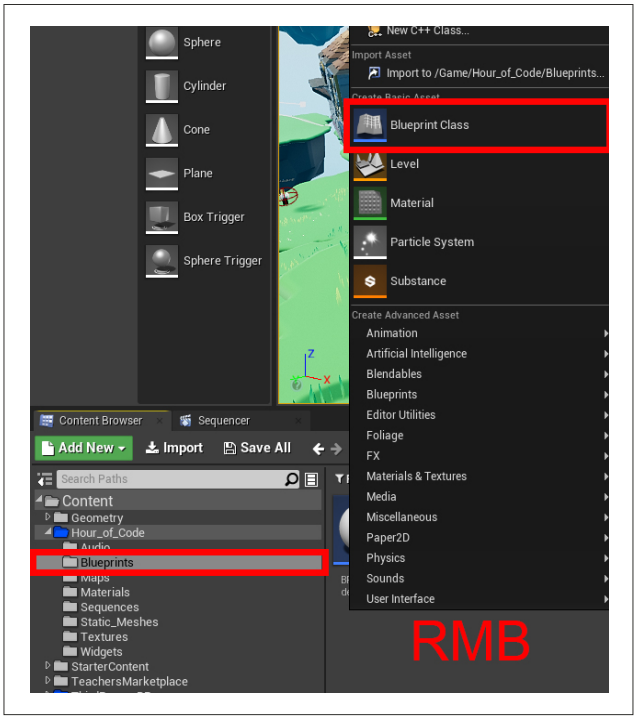


Fig 002 – Right-click the Blueprints area to create a new Blueprint Class.

2. Select an Actor class from the top of the list.

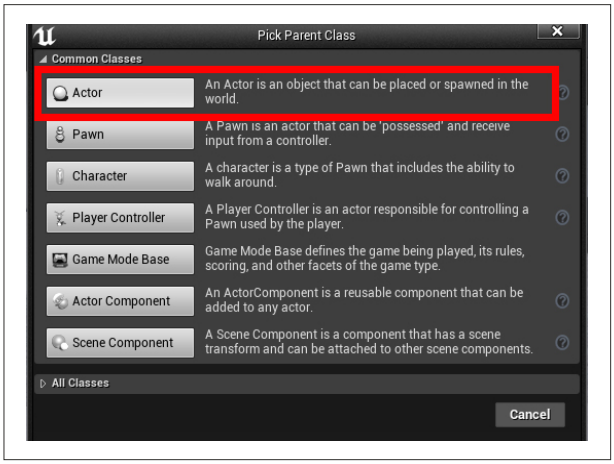


Fig 003 – Select "Actor" as the Blueprint parent class.

3. Name this new blueprint **BP_key**.

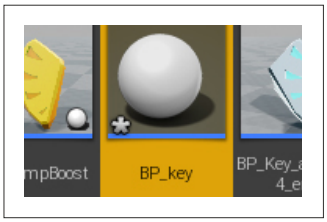


Fig 004 – Your new key Blueprint.

4. Double-click **BP_key** to open the Blueprint. Click the **Viewport** tab if it doesn't automatically load. From here, we will be assembling the parts the player will be able to see and interact with.

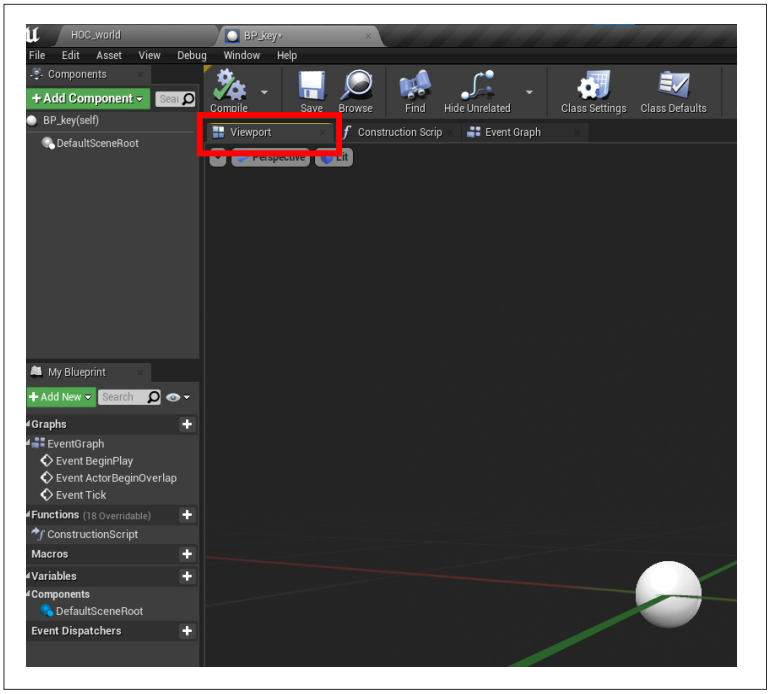


Fig 005 – Select the Viewport tab for the key Blueprint.

5. We will need to add a few **Components** to the viewport, so the player knows what they will be picking up, and so the **Character** has something to interact with. The first piece we will add is a **Collision Sphere Component**. The collision sphere component will define the **hit area** for the key. When the player is in the hit area, it will activate the collision detection. Click the green **Add Component** button at the top left corner of the interface, and search for "**sphere collision.**" Then click the **Sphere Collision** option from the drop-down. You'll see a wireframe sphere appear in the viewport.

Fig 006 – Select a Sphere Collision.

6. Then click on the **DefaultSceneRoot** in the components tab, this will ensure the next piece we add doesn't become a child of the **Sphere Collision**.

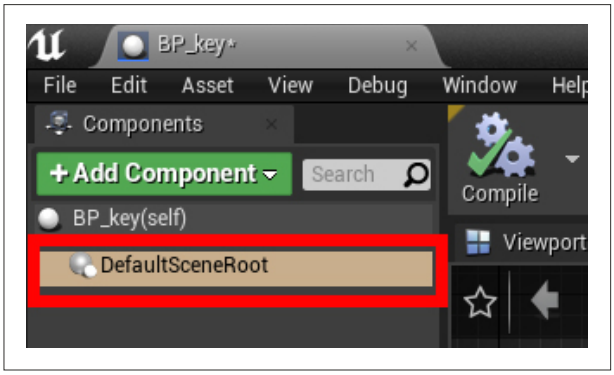


Fig 006b – Click on DefaultSceneRoot when adding components.

Once again, click **Add Component** and search for "**Static Mesh**." Be careful to choose the **Static Mesh** that is located at the bottom of the list.

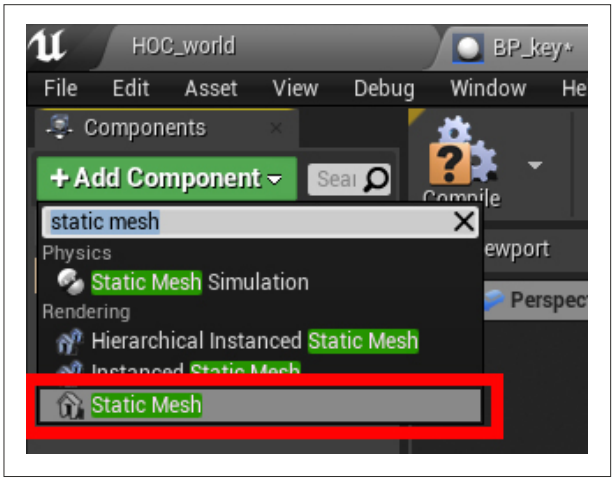


Fig 007 – Select the "Static Mesh" from the bottom of the list.

7. Now select the Static Mesh in the Component panel. Navigate to the Details panel which is located on the right-hand side of your screen, and find the Static Mesh section. Click the drop-down that says None, and search for “key”. Then choose the SM_key. You can move the key so it sits in the middle of the sphere if you would like.

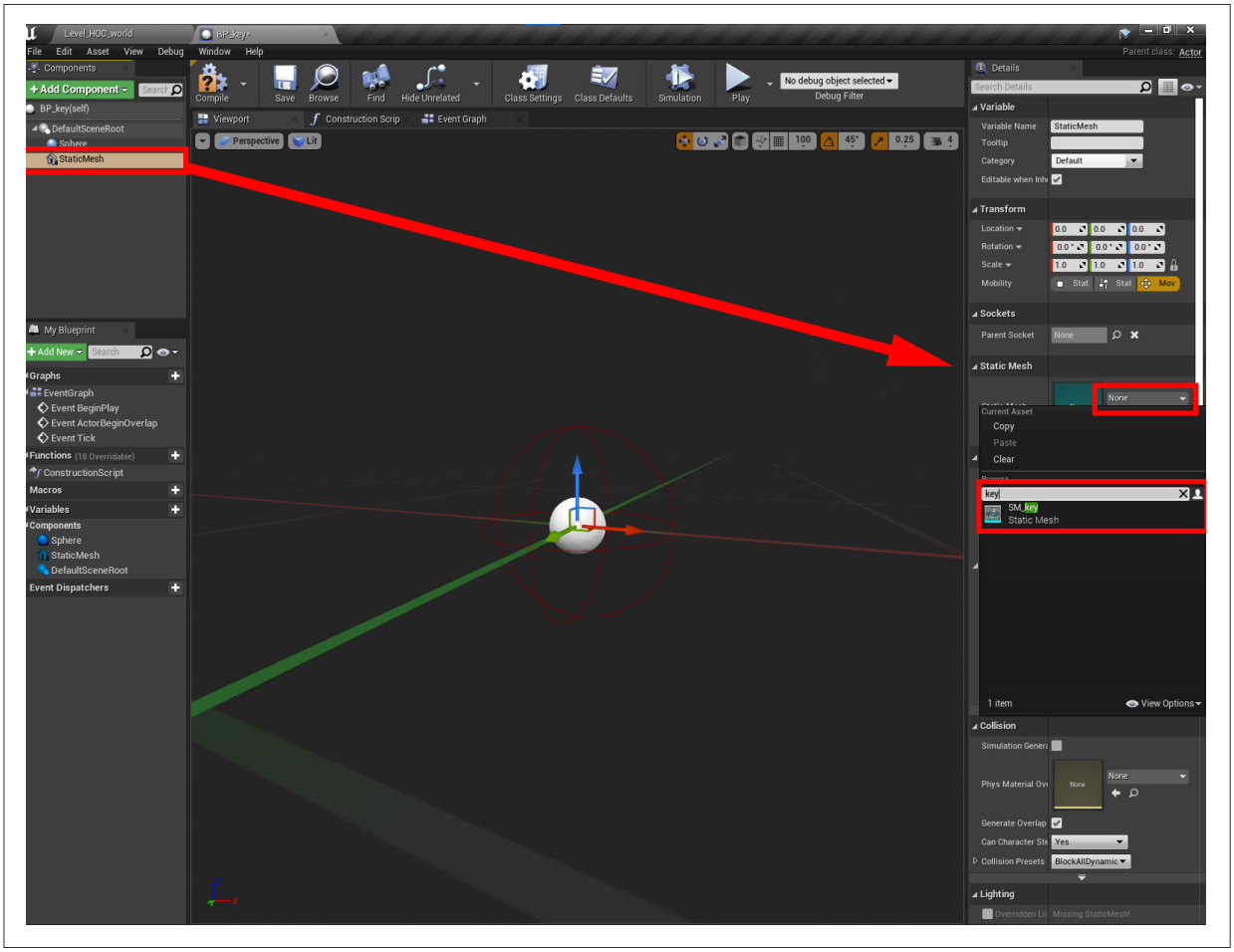


Fig 008 – Select the key as the static mesh.

8. At this point, we should do some housekeeping. It’s ALWAYS a good idea to follow best practices that will aid you in the future. By naming our assets it will be easy to keep track of them. Select one in the content panel and press **F2** to rename it. Rename the **Sphere** to **Sphere_coll**, because this is a collision sphere. Then rename the **Static Mesh** to **SM_key**, because it is a key. The Static Mesh (SM) is a 3-dimensional object that does not change shape.

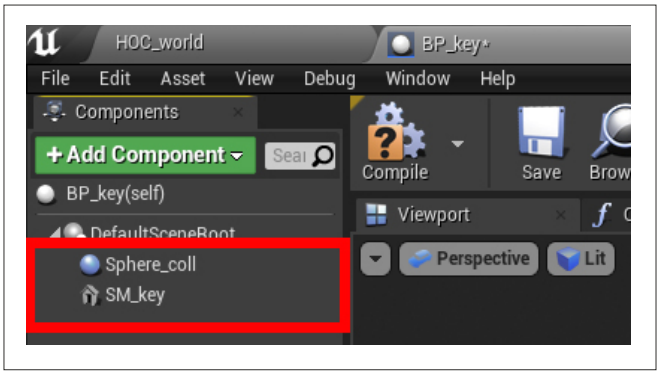


Fig 009 – Creating the collision sphere and the key.

9. Let's Save our work by clicking the Save button at the top of the interface.

Fig 010

10. Click the **Event Graph** tab, this is where we will be writing the code to make this Blueprint work. You will see 3 nodes in the graph. Select all 3 and press the **Delete** key on the keyboard. We will not need them.

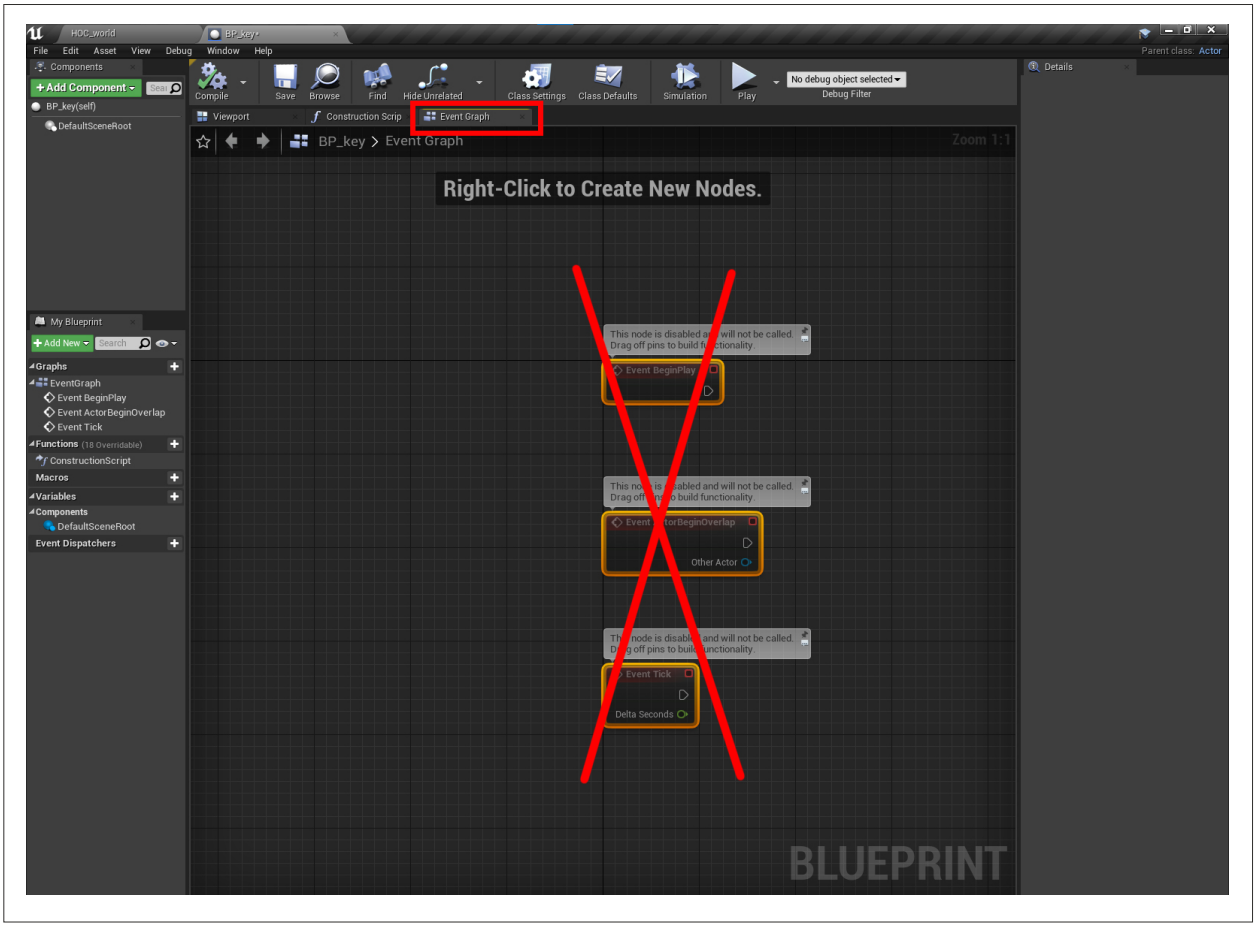


Fig 008 – Select the key as the static mesh.

11. Select the **Sphere_coll** in the components tab, and scroll to the bottom of the **Details** panel. Click the second green button in the **Events** section labeled **On Component Begin Overlap**. This will create an **Event** node that will trigger when the **Character** touches the **Sphere_coll**. (In other words, this is what will execute when the player picks up the key.)

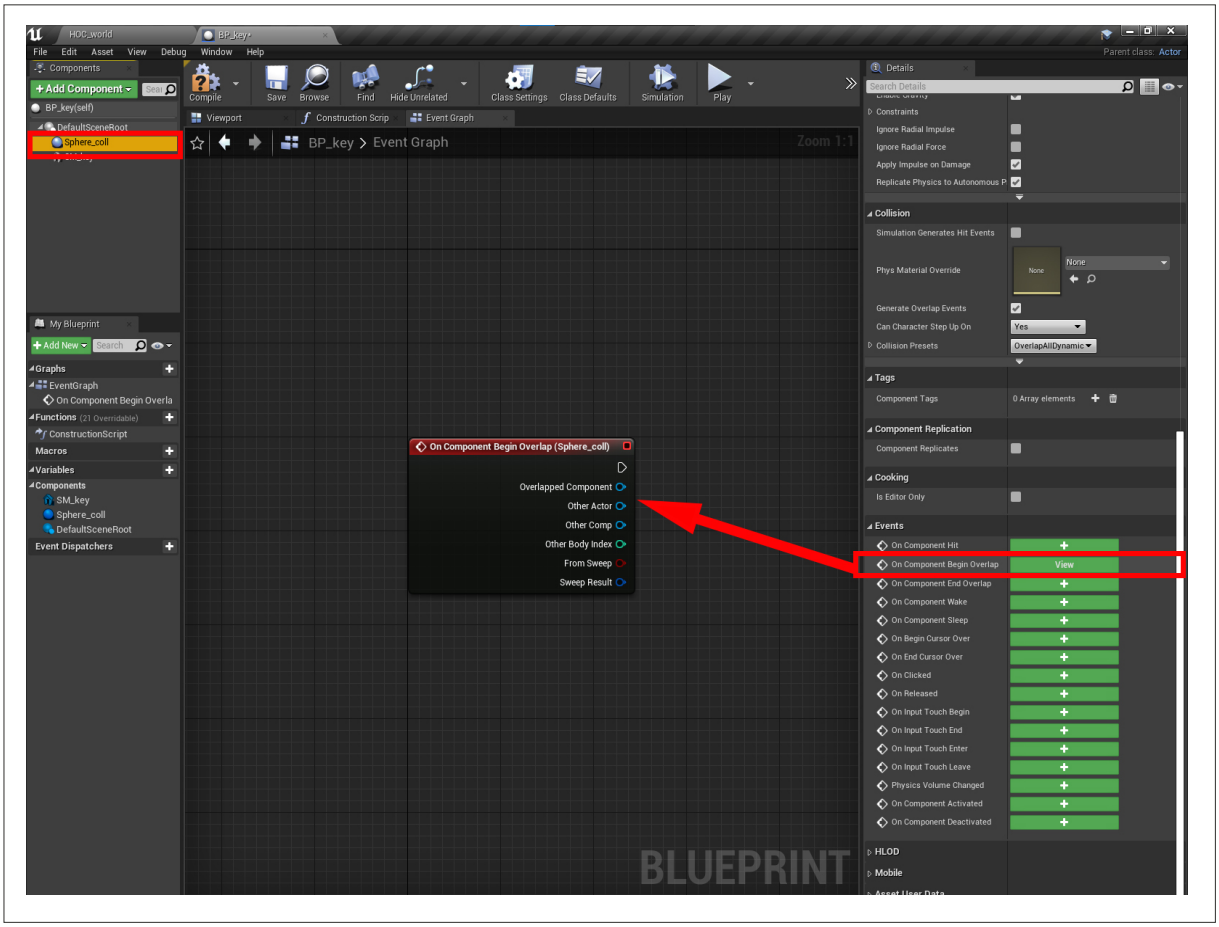


Fig 012 – Add the event to detect when the player touches the key.

12. We need to tell this Blueprint to talk to the game, to accomplish this we need to **Cast** to the game. Let's begin by clicking and dragging a wire from the execution pin to the blank space in the Event Graph background and let go to bring up the **Execution actions**. Type "**cast to third person game mode**" in the search field. Select Cast to **ThirdPersonGameMode**, this will create a new node.

    TIP: Use the scroll wheel to zoom in or out and click and drag the right mouse button to move your view.

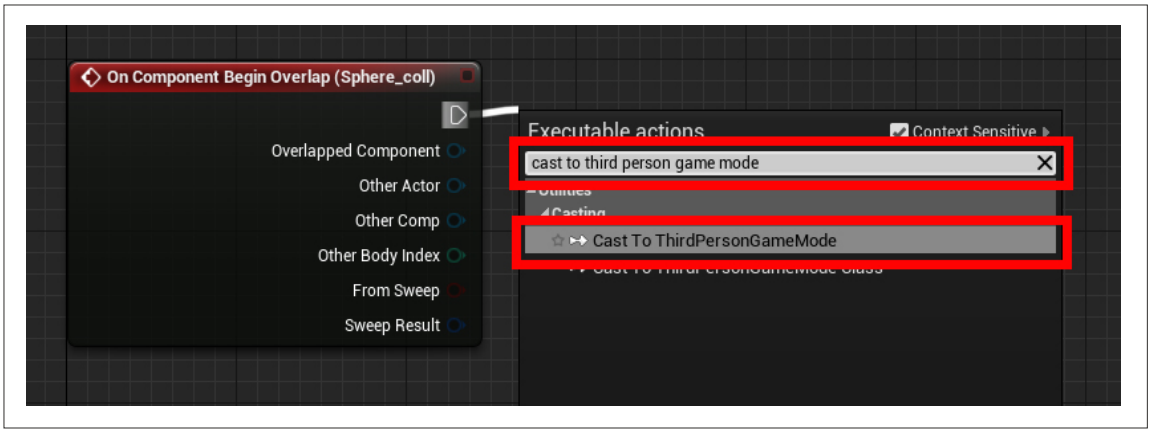


Fig 013 – The first action will be to communicate with the game.

13. Drag a wire from the **Object** pin and search for "get game mode". Choose **Get Game Mode**, this will create another new node.

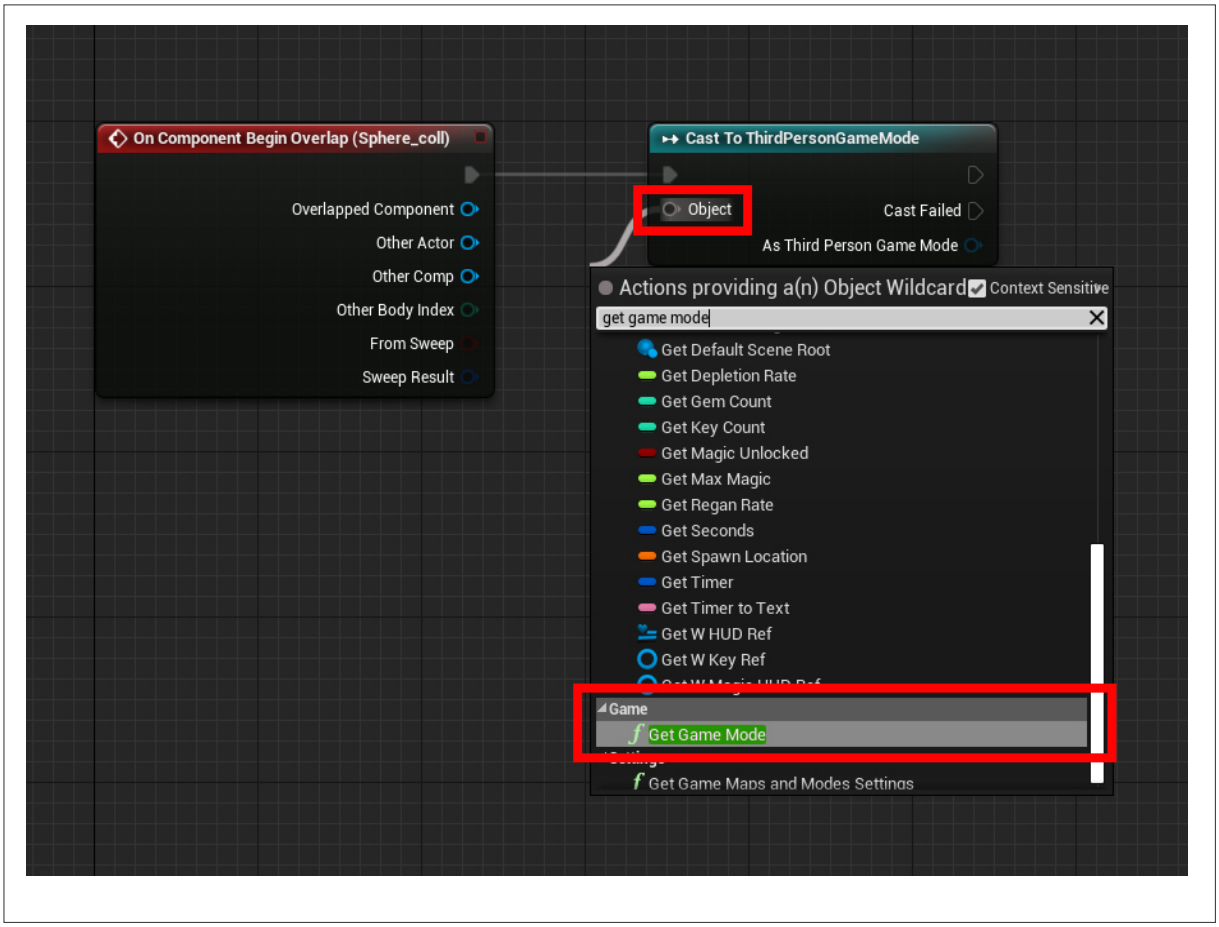


*Fig 014 – Define **Get Game Mode** for the Object property.*

14. Now that this Blueprint is talking to the game, the game needs to know what to do. We are going to tell the game to add a key to the character's inventory. To achieve this, pull a wire from the bottom blue pin As **Third Person Game Mode** from **Cast to ThirdPersonGameMode** node and search "**set have key.**" Choose **Set Have Key**?

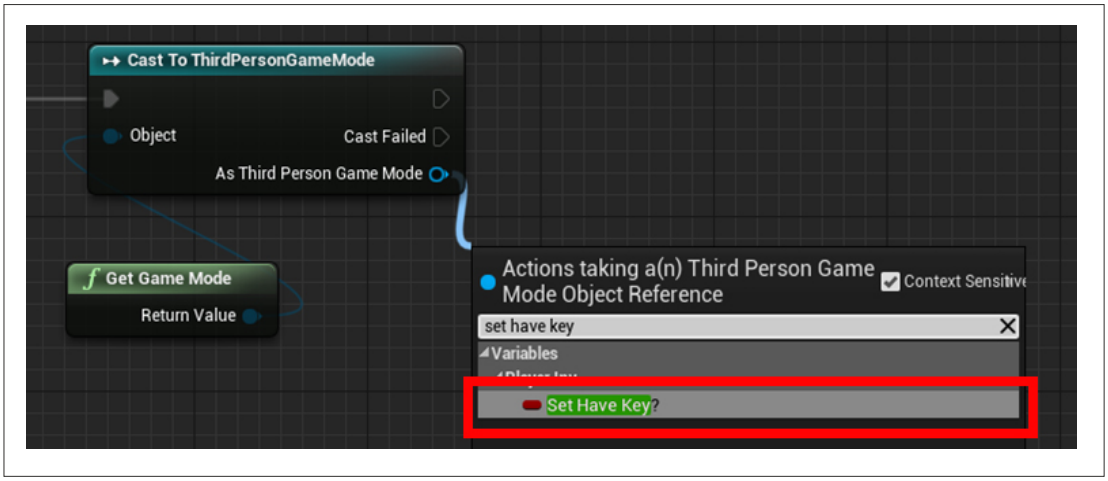


Fig 015 – Setting the Have Key Variable.

15. Connect the execution pins  from **Cast to ThirdPersonGameMode** to the SET node(see below), and toggle the checkbox **ON**. We want to **set** this **Boolean** to **true**.

    The question is "Does the player have the key?"
    We are setting the answer to yes (**true**) with this **boolean** node.

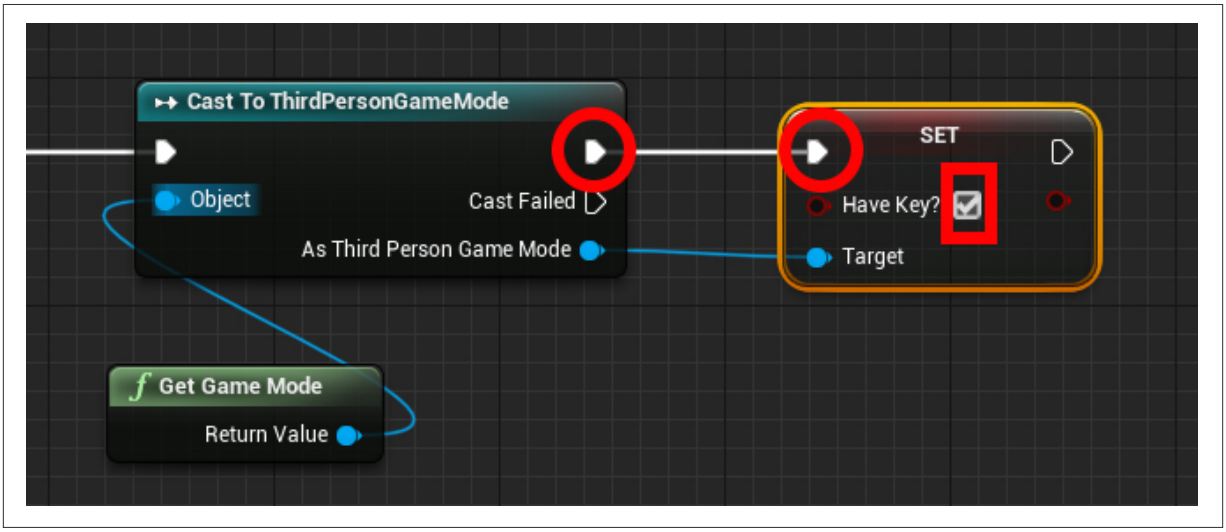


Fig 016 – Set the Have Key Variable to True.

16. Let's add a sound so the player knows they have picked up the key. Pull off the execution pin from SET and search "play sound" and choose **Play Sound 2D**.

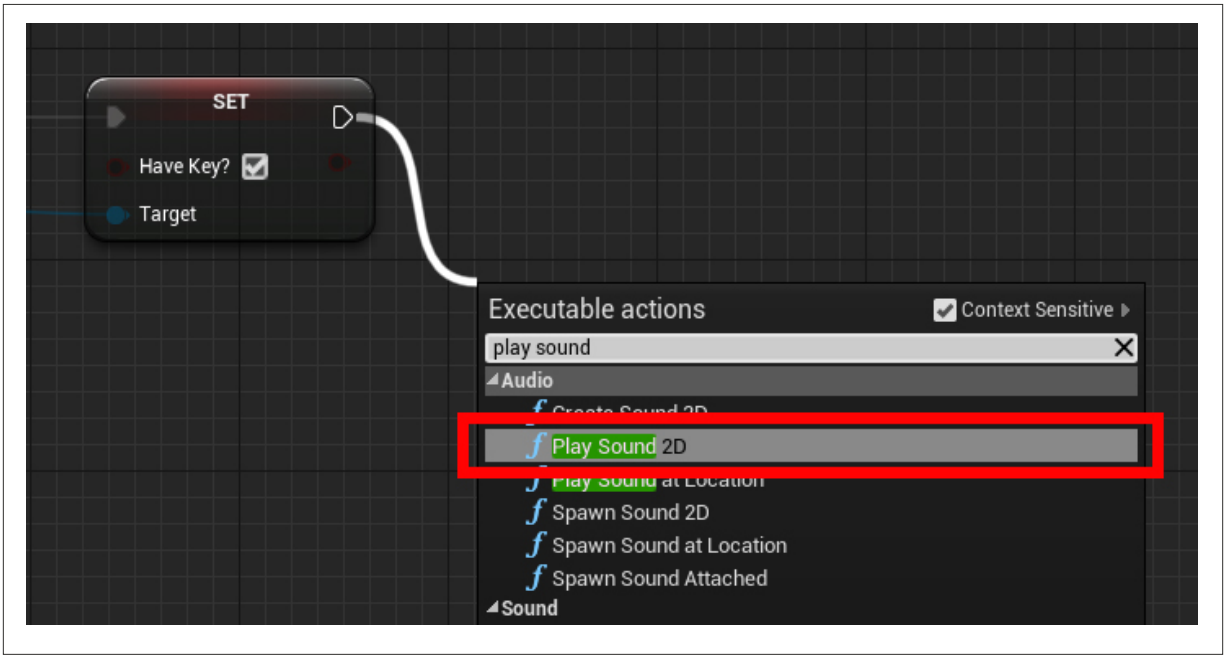


Fig 017 – Next, play a sound to signal that the key was picked up.

17. We need to choose a sound to play. Click the **Select Asset** drop-down, search "key" and choose **SFX_key**.

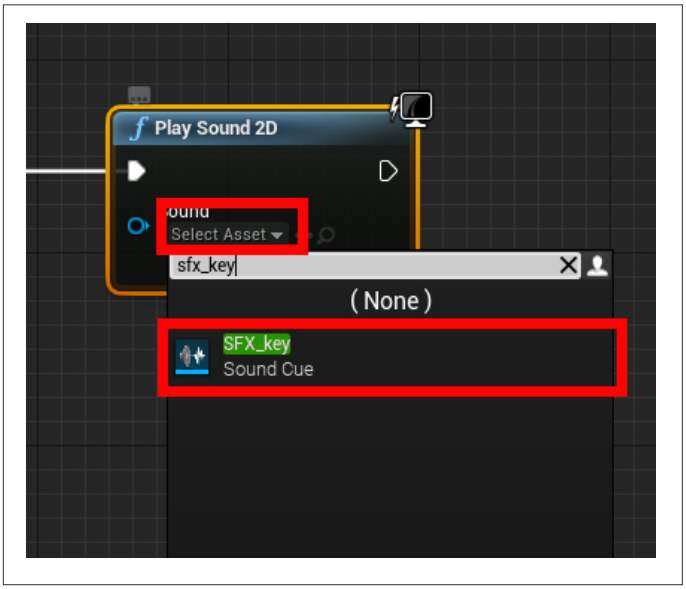


Fig 018 – Select the sound to play when the key is picked up.

18. The last node we need to add will tell the Blueprint to destroy itself. After all, the character has touched the key to collect it, therefore it does not need to be in the level anymore. Drag a wire from the execution pin of Play Sound 2D and type "**destroy actor**". Choose **Destroy Actor**.

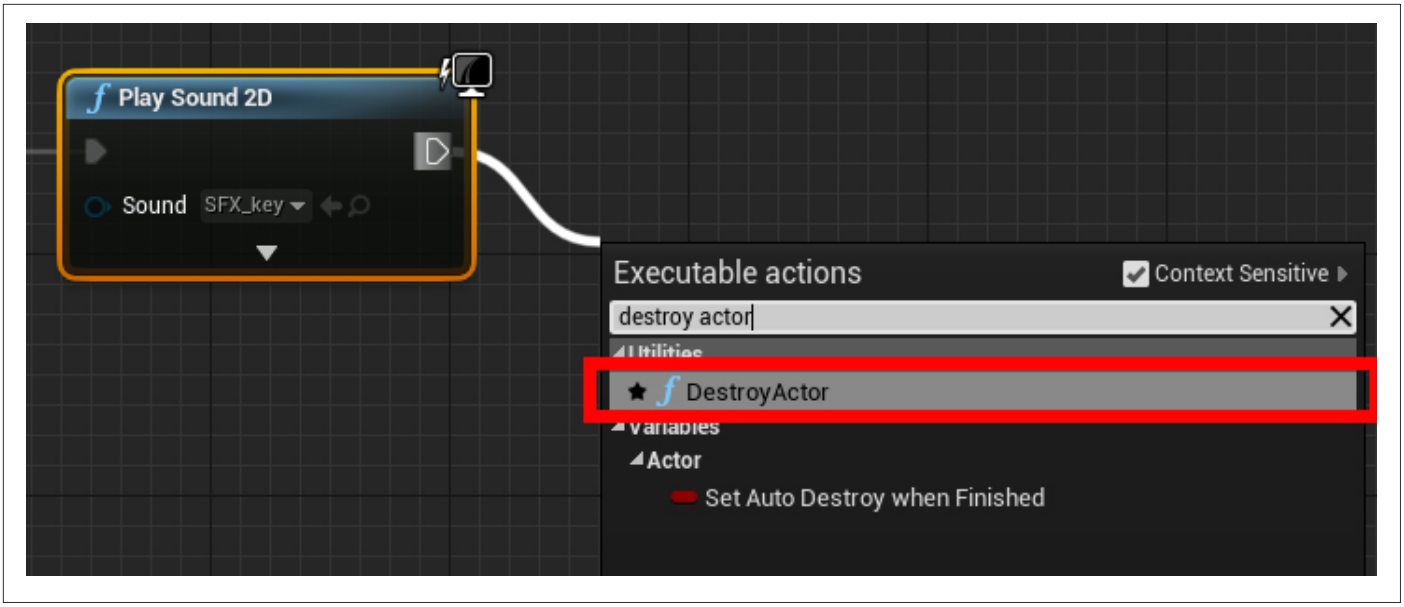
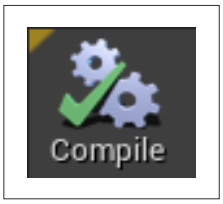


Fig 019 – Finally, use Destroy Actor to remove the key from the game when it is collected.

19. To complete this we will need to **Compile** the code and **Save** all the work we have completed. When you click the **Compile** button, the "?" should change to a green checkmark.

Look at the image below and cross-reference it with your work.



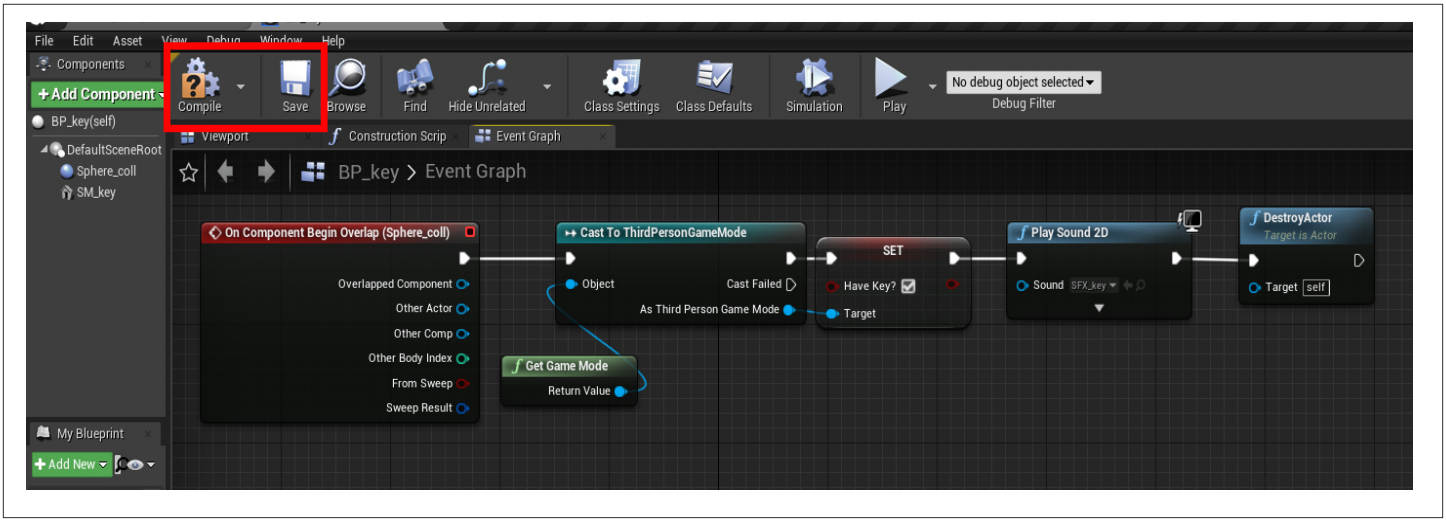


Fig 020 – Save and Compile the Blueprint.

Now, it's time to playtest your Blueprint.

Make sure you have compiled and saved your Blueprint and then click the X in the upper-right corner to close the Blueprint editor. (Make sure you are closing the window with the Blueprint editor, so you don't accidentally close your entire project.)

Now, you should be back to your world editor. Click in the Viewport and press the number 5 to jump to camera bookmark 5. In the Content Browser, navigate to Content > Hour_of_Code > Blueprints. Drag a copy of your **BP_key** into your level next to the door on the final island. Press W to activate the Move gizmo. Drag the blue arrow up to position the key floating above the ground.

Press the Play button to test your key.

To fully test your key, DO NOT walk over it as soon as you start playing the game.

Fig 021 – Adding the key to your game for testing.

First, walk up to the door, you will see a message that tells the player “Key Needed”. This is because you have yet to pick up the key.

Fig 022 – The “Key Needed” message should appear when trying to open the door without a key.

Second, pick up the key. You should hear a metallic key sound, and the key will disappear.
NOTE: Be sure your speakers are working correctly.

Next, walk up to the door. The "Key Needed" message should **NOT** show up on the screen.

Last, press the **E** key on your keyboard and the door will open, and a key icon should show up on the bottom left-hand side of the screen. Don't worry about this icon, it's just a hint of what we will be covering in Activity 5.

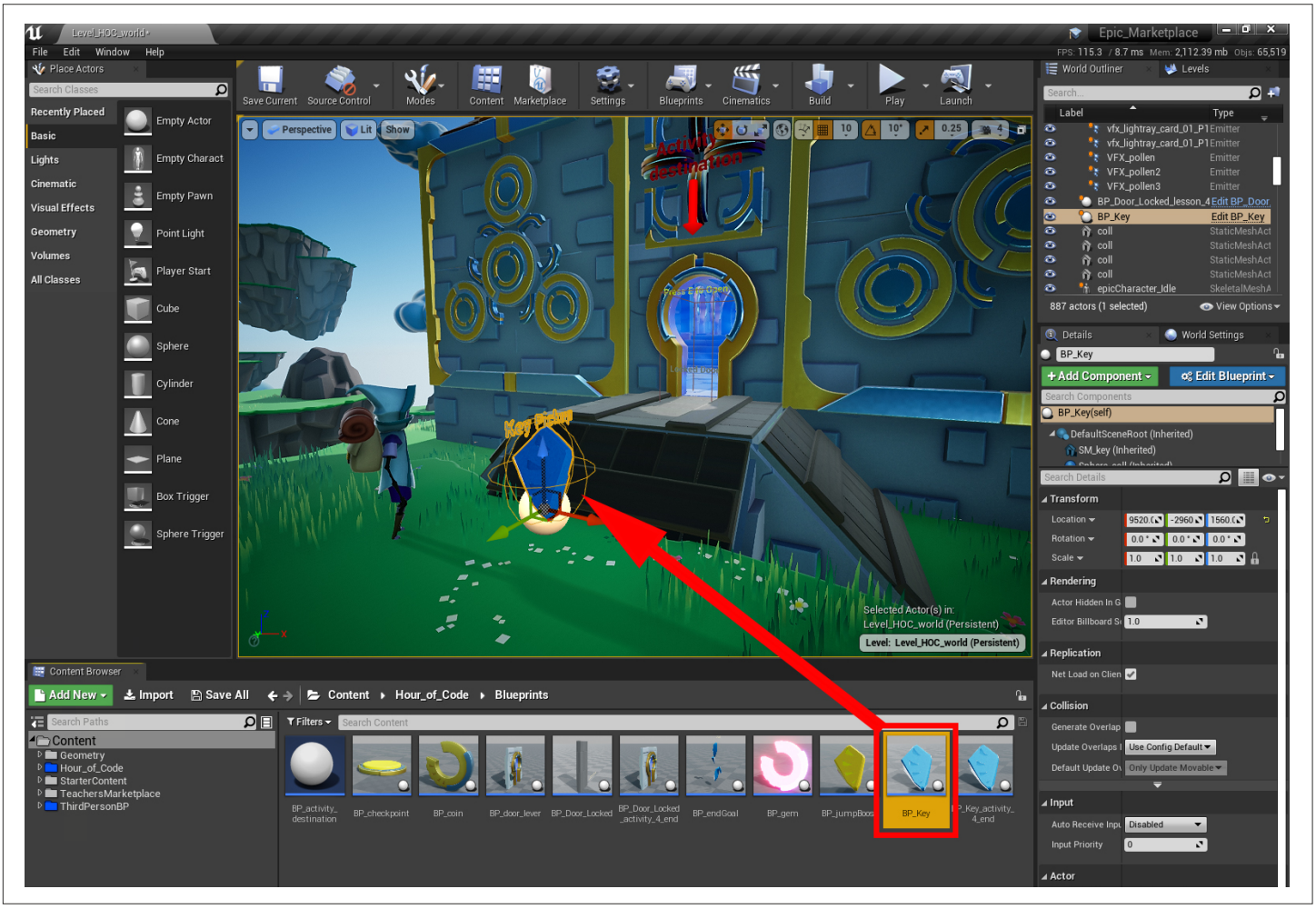


Fig 023 – The door can open when the player has the key. An icon will appear on the screen, which is a hint for what you can do in the next activity.

Now that the door is open, we have one last task. We need to add the end goal. Inside the **Blueprints** folder, you will find **BP_endGoal**. Drag it into your level to the top of the ramp at the back of the building.

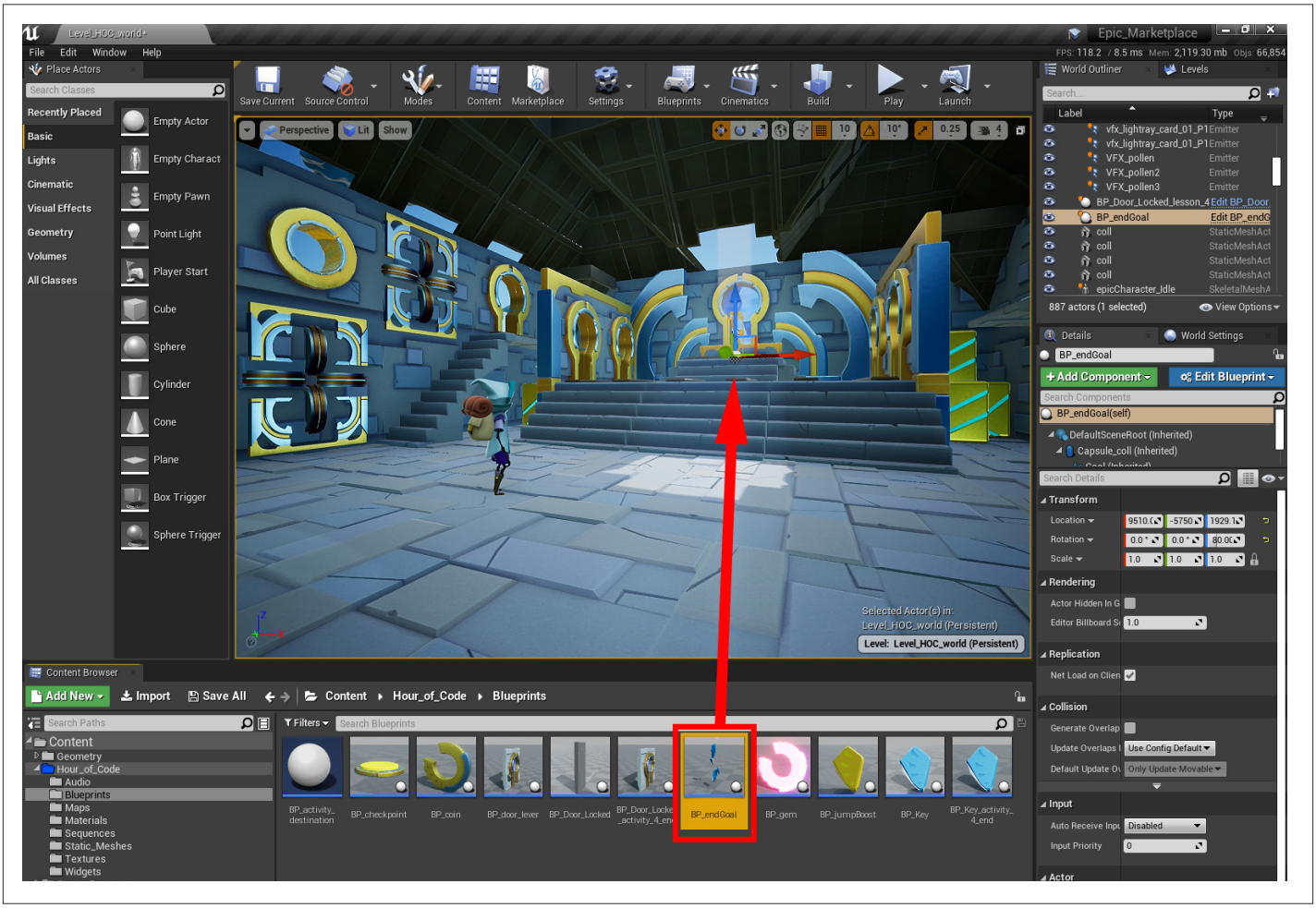


Fig 027 – Add the End Goal inside the locked room.

With the goal in place, you can play your game from beginning to end. You'll notice that once you touch the end goal the player is rewarded with various stats, such as how many coins and gems they collected, in addition to how long it took to complete the level.

To play your game from the beginning, click the dropdown arrow on the **Play** button and select the **Default Player Start** option. Now, clicking the **Play** button should start you at the beginning.

Remember to **Save All** your work.

Game Design Challenge

Now that you have a working game, try adding more coins and moving the key to a better location. Can you make the game more interesting and challenging for players? Have some people test your new game and give you feedback.

Up for a challenge?

Now that you have had a chance to see how **Booleans** work, open the **BP_Door_Locked_activity_4_end** blueprint from the **Content > Hour_of_Code > Blueprints** folder in the **Content Browser**. You will find quite a few **Boolean** nodes throughout this Blueprint that have been marked in red. Can you find where the door is looking to see if the player has a key? How about the section that tells the key icon to show up on the screen?

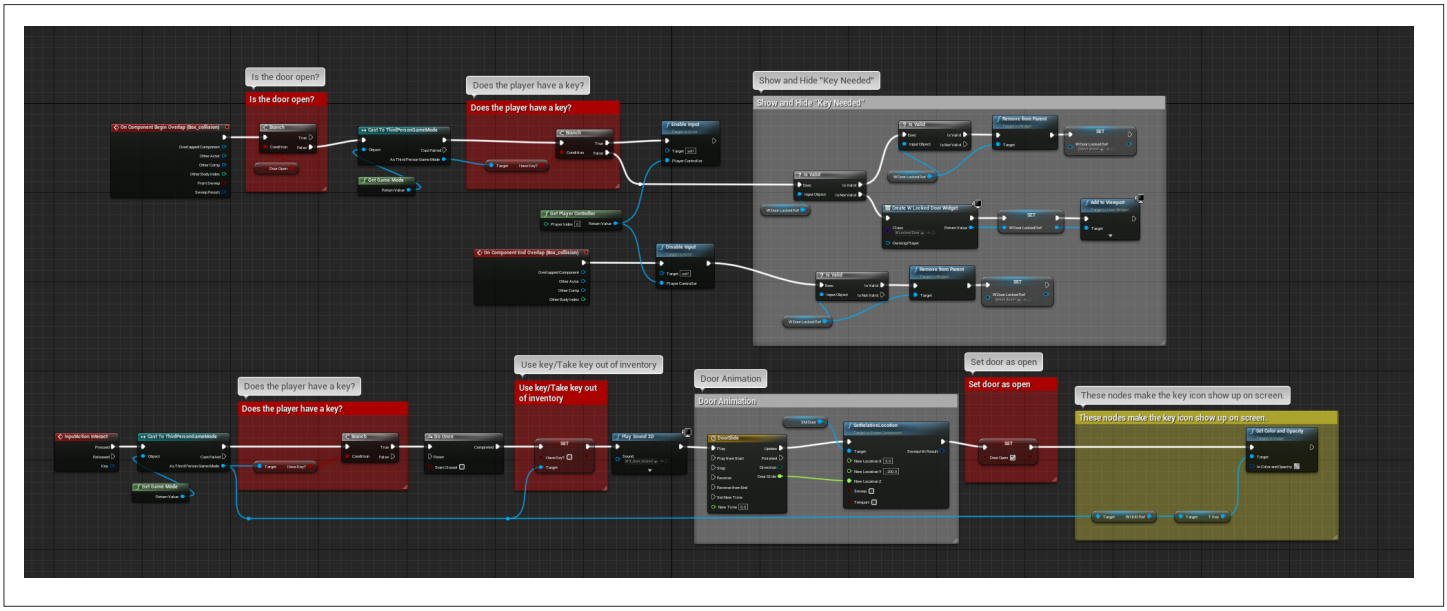


Fig 024 – Examine the other Booleans used in this Blueprint.

In our final Hour of Code activity, we will cover how to make various sections of the heads up display (HUD) visible. Both the key icon you saw earlier, and the end game screen are part of the HUD. We will also package the game so you can send it to other people to play or add it to your portfolio when looking for jobs.