



UNREAL
ENGINE

BUILD YOUR FIRST 3D GAME:

LEARN COLLISION DETECTION IN UNREAL ENGINE

STUDENT GUIDE

Activity 1

Build Your First 3D Game: Learn Collision Detection 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

In this activity, you will learn how to build a parkour course to get the player through a simple hallway, and across a treacherous void. For our purposes, parkour refers to the player running and navigating jumps of varying difficulty over a deadly void. The jumps you will be setting up will give the player a variety of challenges to overcome.

You can make the parkour challenges as easy or difficult as you like. Be mindful that difficulty does not always equal fun. You may think your level is too easy to complete, but this may be because you are the designer and have played through your level many times. Be sure to take all the feedback from your play-testers very seriously, because this will make your game more enjoyable for a wider group of players. The more play-testers you work with, the more successful your game will become.

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

You will be learning about **collision detection** and its importance in computer programming and game development. In the real world, it's easy to understand how you can sit in a chair, stand on the floor, or hit a ball because they are all physical objects. In a virtual 3D world, the concept of physical objects needs to be defined in code. The moment when two objects collide or overlap is called a collision. When objects collide, Unreal Engine can identify which objects collide and how each object should react. You will learn the critical importance of collision detection in this activity.

Additionally, this activity will give you a chance to get familiar with navigating Unreal Engine user interface in the **Viewport**.

Open up the project and let's begin!

Reviewing the Interface

The following set of activities assumes you have some familiarity with the Unreal interface, so we will only cover the areas of the UI (User Interface) that we will be using. We will also be using the default UI setup that ships with Unreal.

Locate the **Content Browser**. This can be found at the bottom of the screen. By clicking on the button indicated by the arrow (Fig. 001), you can show the sources to make navigation through the folders much easier.

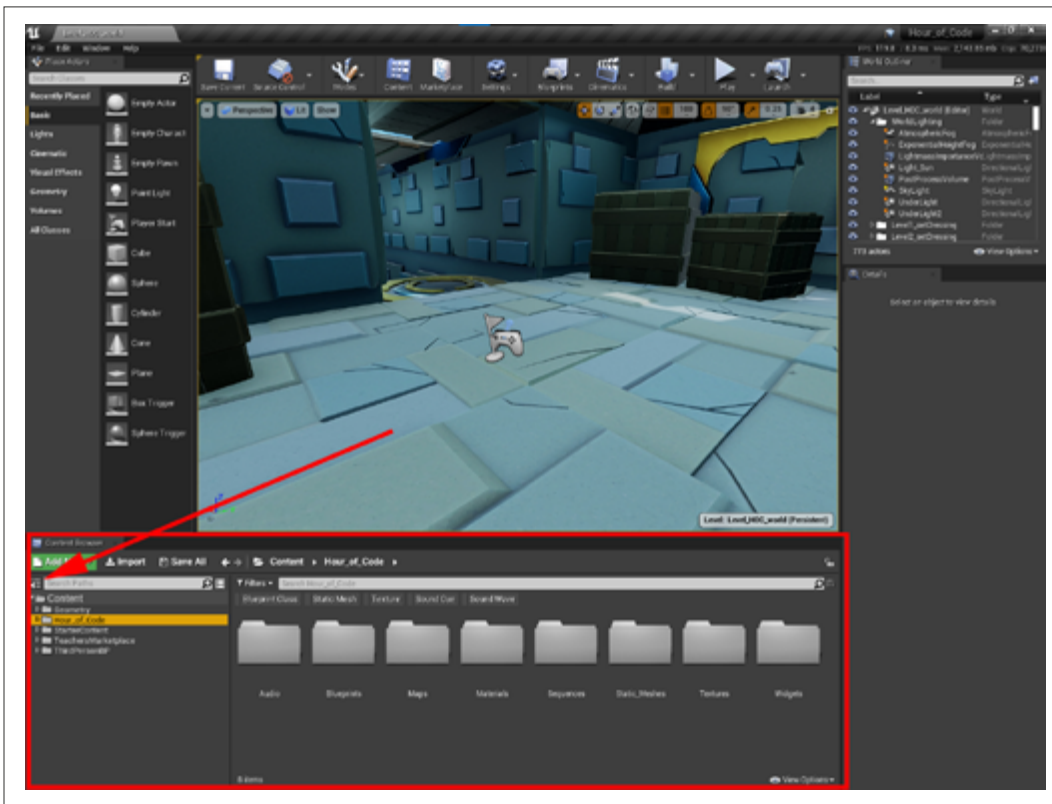


Figure 001 – The Content Browser panel.

You can also change the color of the folders to make it easier to find them. Let's change the "**Hour_of_Code**" and "**ThirdPersonBP**" to a blue color. Right-click on the folder and choose **Set Color**. Then choose a blue color from the Color Wheel.

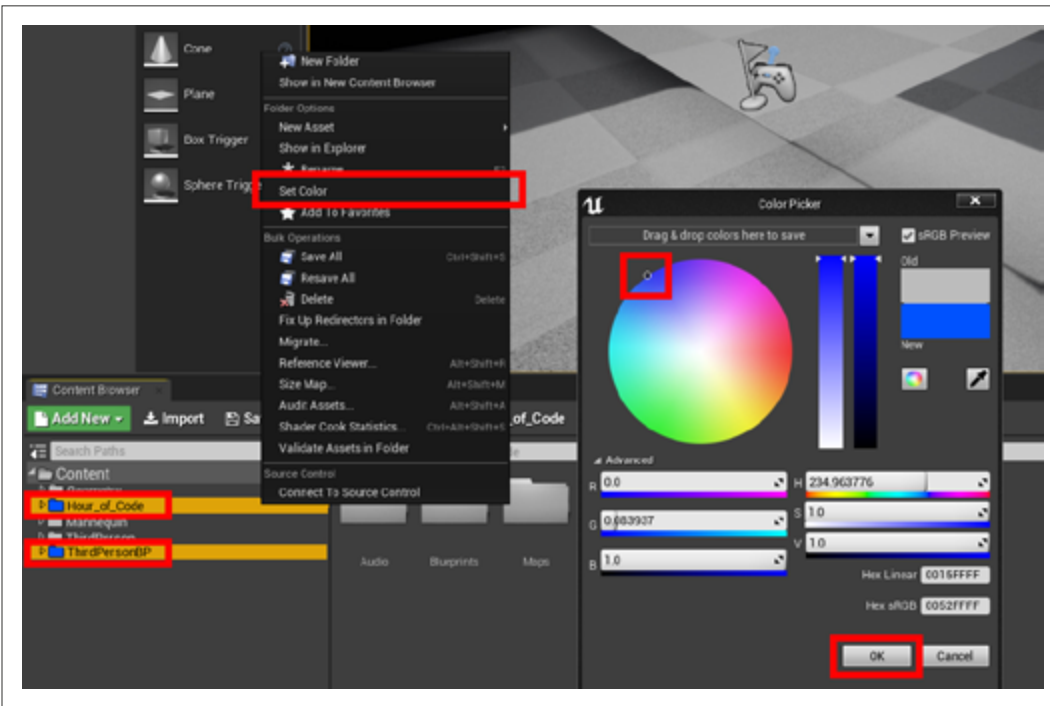


Figure 001a — Changing the Color of folders.

The **Place Actors** panel contains assets like Lights and Trigger Volumes. This can be displayed by navigating to **Windows > Place Actors**. These assets will be available in every project you create. This is different than the **Content Browser** because the **Content Browser** will also contain assets that you can import from 3D design programs like Autodesk's Maya, 3D Studio Max, and Blender.

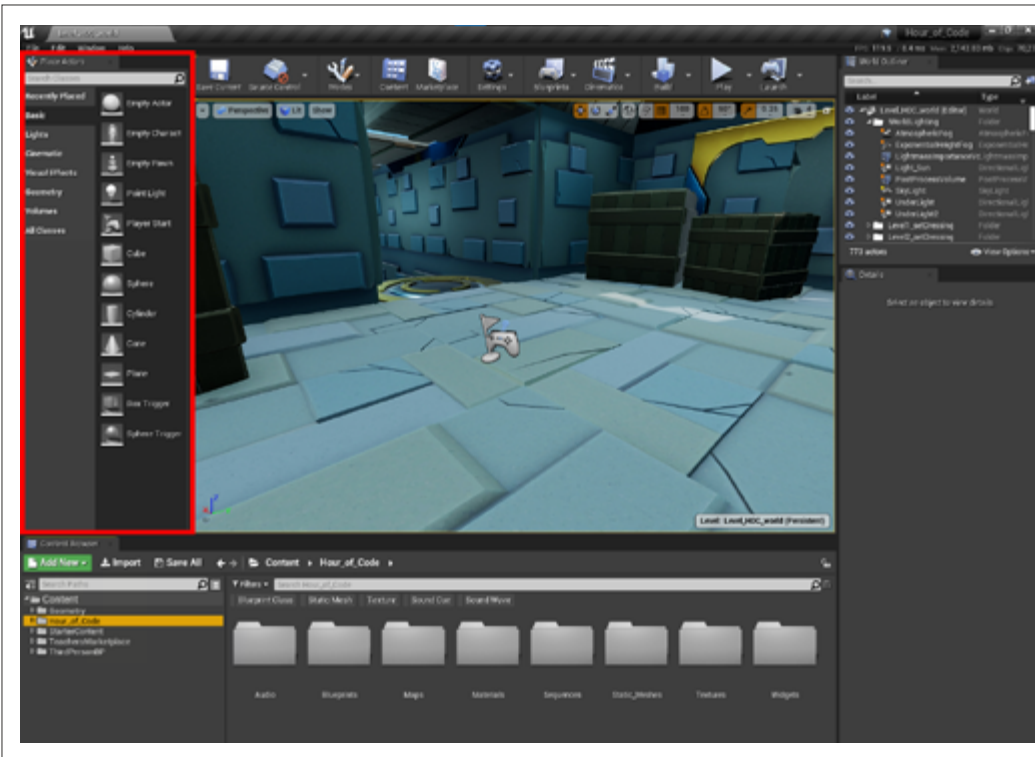


Figure 001b — The Place Actors panel.

Actors – The term “Actors” is used to identify the various objects or elements that are in a Level. While the phrase is commonly used to identify people in film and TV, it is a more general term when used in the context of your Unreal Engine projects.

We will be doing most of our work in the **Viewport**. This is the large area in the middle of the screen where you will place assets from the **Content Browser**.

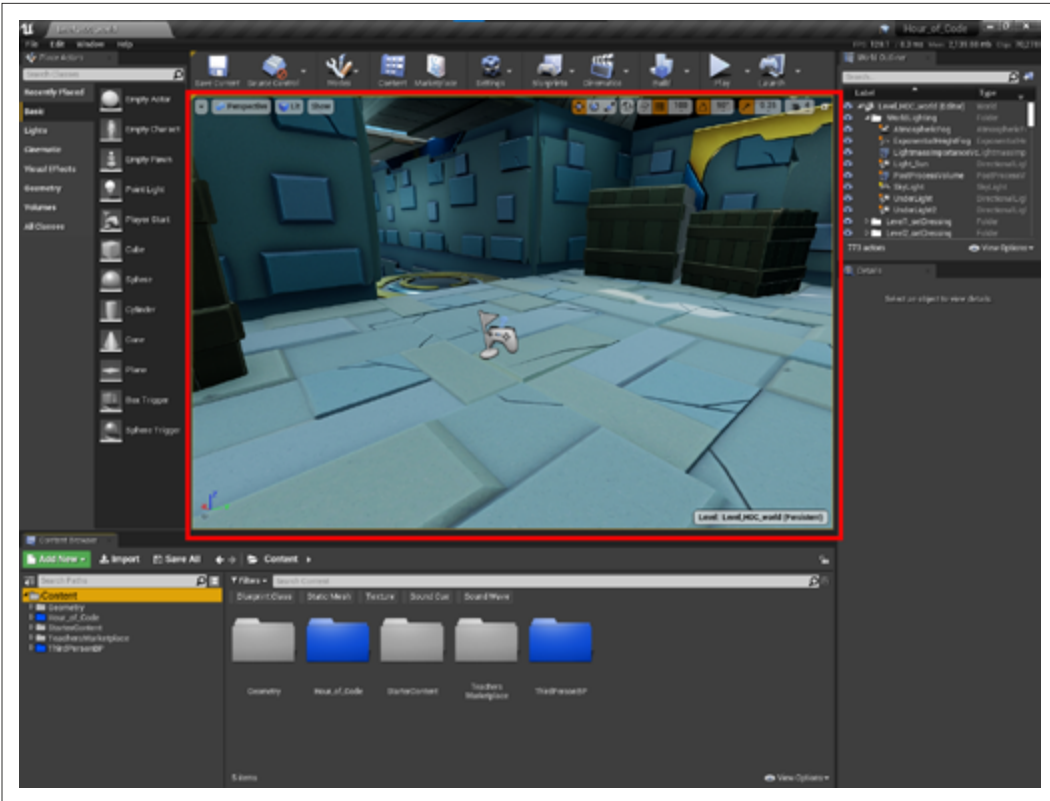


Figure 002 – The Viewport.

When navigating in the **Viewport** there are a few different ways to control the camera. First, these controls work with no other keys or buttons pressed.

- Hold **LMB (left mouse button)** and drag the mouse to move the camera forward and backward or rotate it right and left.
- Hold **RMB (right mouse button)** and drag the mouse to rotate the camera up, down, left, or right.
- If your mouse has a **scroll wheel**, you can use that to **zoom** in and out.

Next, if you have ever played a first-person game on a computer, you probably used the WASD keys to move around, and these controls will feel natural. The WASD controls can be used whenever you hold down the **RMB**. If WASD are too difficult, you can use the arrow keys as well.

Do not forget to hold down the RMB while using the following keyboard keys:

- **W** moves forward
- **S** moves backward
- **A** moves left
- **D** moves right
- **E** moves up
- **Q** moves down

You can also press the **F** key to frame any selected asset or actor. Select an asset in the viewport, then press the **F** key and see what happens. Select another asset and press the **F** key again.

Next, open 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.

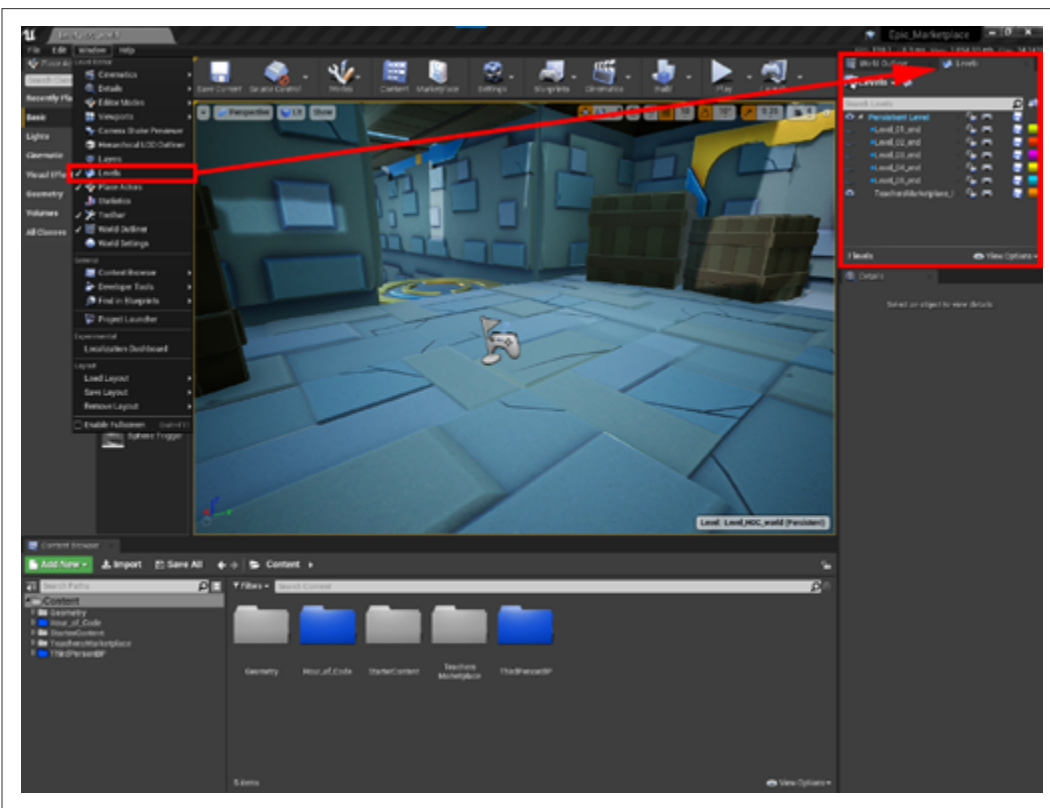



Figure 003 – Move the Levels panel next to World Outliner.

This window currently contains examples of what each activity could look and feel like. As you progress through the activities feel free to click the eyeball icon  to view the examples. We highly encourage you to work through the activities in order, as each lesson builds upon the previous.

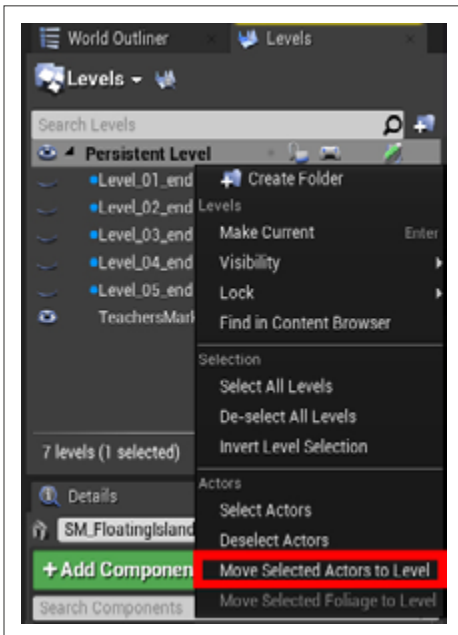


Figure 003b – Moving Actors to the persistent level.

The **Details** window, also referred to as the **Details Panel**, will show editable attributes of any Actor that is selected in the **Viewport**. Any asset placed into the **Viewport** is referred to as an **Actor**. You can think of them as similar to actors on a stage, but they are Actors in your game. Select the **PlayerStart** Actor and you will see the **Details Panel** update.

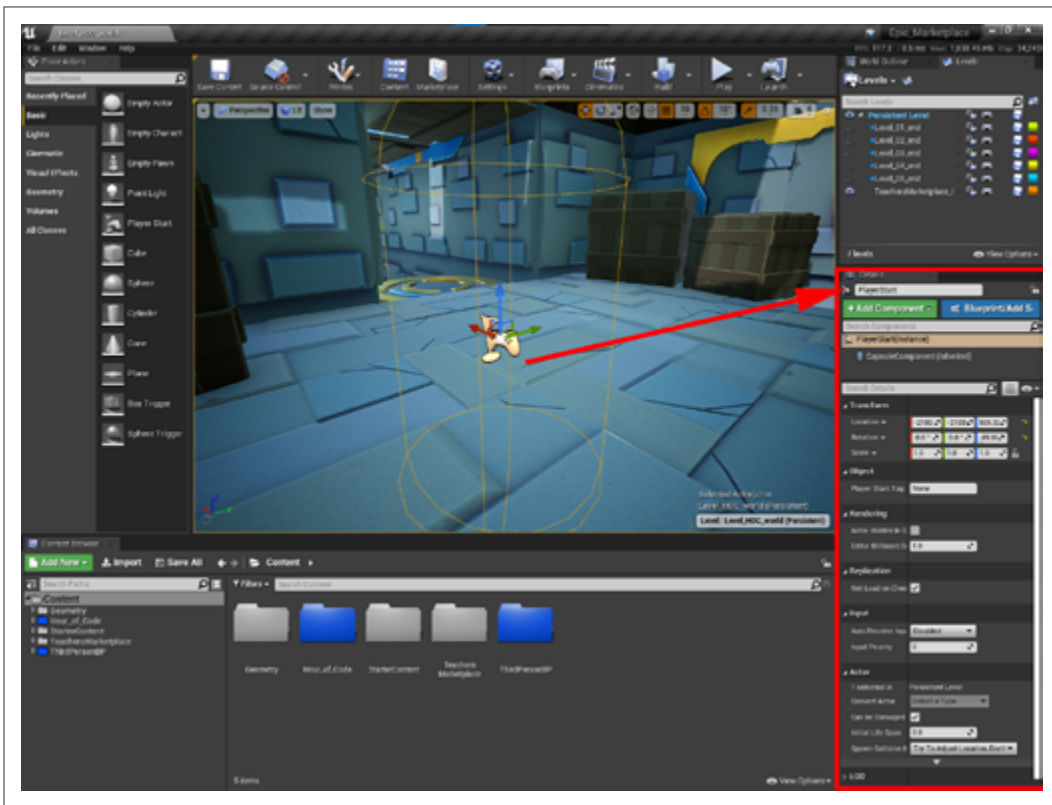


Figure 004 – The Player Start Actor location.

If you need to deselect an Actor, hold Ctrl and click the Actor.

We will also be using the **World Settings**. To open this window, click the **Settings** button at the top of the screen and select **World Settings**. This panel contains information about the game's 3D world. For these activities, we will be working within the **Persistent Level**, and we will only be editing the world settings for the **Persistent Level**.

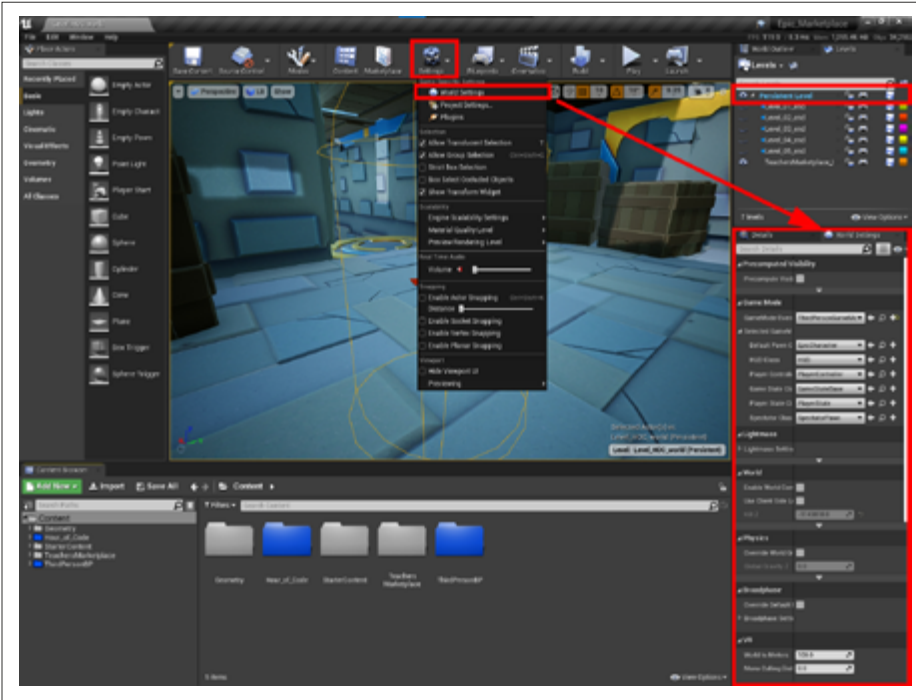


Figure 005 – The World Settings panel.

Let's Play!

Let's jump right in! Click the **Play** button at the top of the screen, or press **Alt + P**.

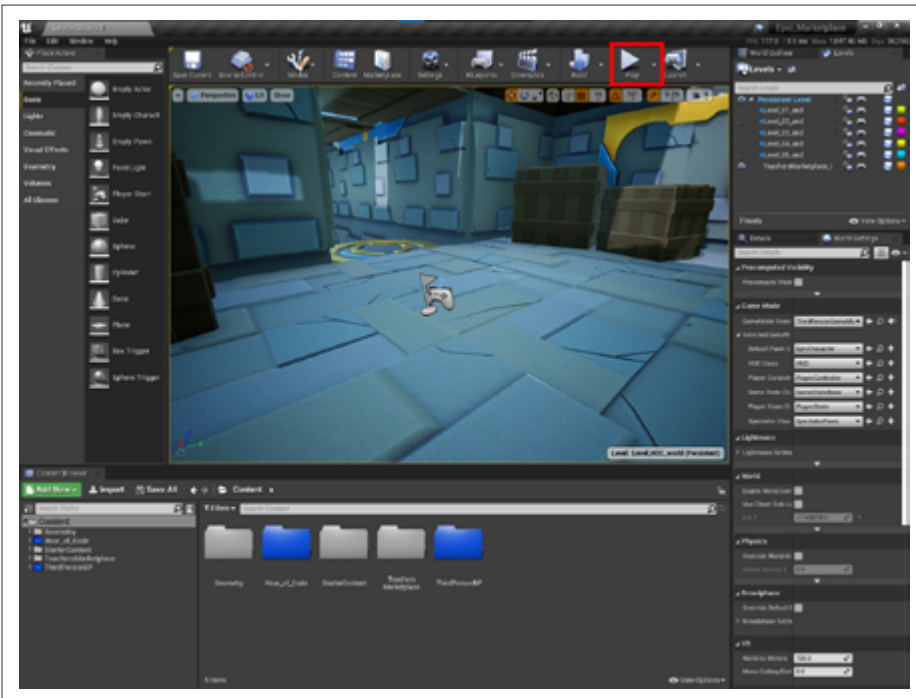


Figure 006 – Press Play to start the game.

Move your mouse over the **Viewport** and click the left mouse button. This will give you control of the player.

You can use the mouse to look around, and use the following buttons to move;

W = move forward

A = move left

S = move right

D = move backward

Space = jump

Hold Shift while moving = sprint

Try to get to the end of the tunnel. What happened?

Press the **Esc** key on the keyboard to leave the game and return to edit mode.

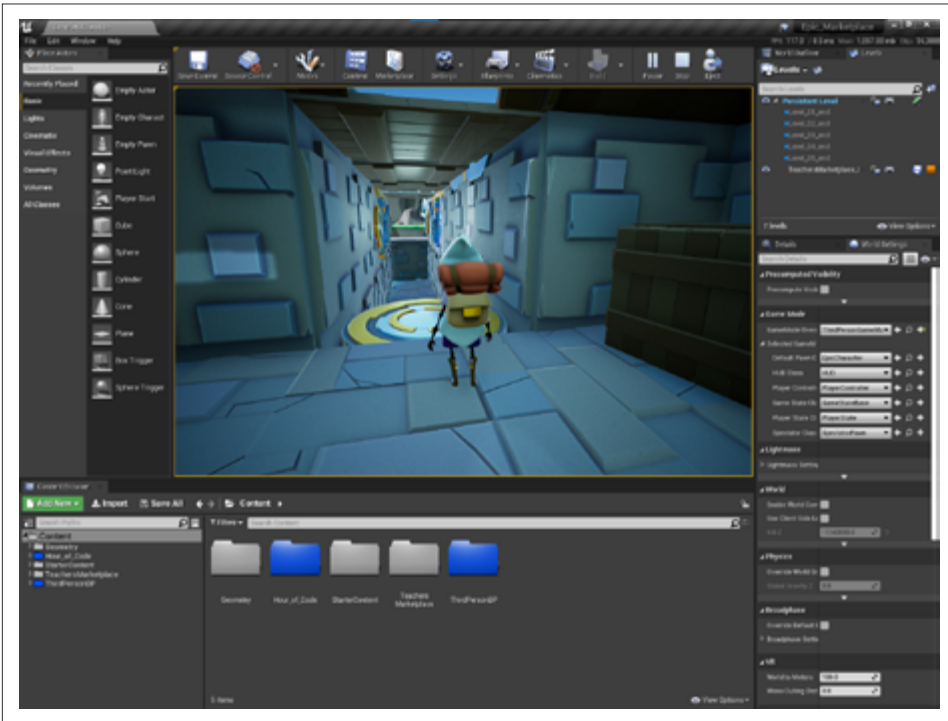


Figure 007 – Player at the first jump.

You'll notice that your character falls through the platform in the tunnel, and lands at the bottom of the pit. This is because the platform has no **collision** associated with it, it is only being drawn visibly in the world. The game doesn't yet know that the player should be able to stand on this platform. Let's create a **Collision Volume** for this asset.

Collision Detection

As you may have experienced in this test, just because an object appears to be solid doesn't mean that it will behave that way. In the physical world, it's easy to predict and observe two objects colliding. When dealing with collisions in a simulated environment, the attributes of objects need to be defined along with their positions. Math formulas in the code will calculate if two objects are overlapping, thereby causing a collision. When the collision is detected, it will perform a specific action assigned by the programmer. In our first example, the collision doesn't exist, so the player couldn't stand on the platform. To correct this, we will need to define a collision area around the platform so we can have the player stand on the platform.

Some fantastic mathematical formulas are capable of calculating collisions. Without them, the games would be pretty boring. Thank you, math!

Fixing the First Platform

To fix our problem with the platform, we will need to establish a collision area for that object. The object has a shape and texture that allow it to be visualized in the game, but this is separate from its collision properties. We need to add a collision volume as an invisible shape that is attached to the object. We can make the collision match the shape of the object or customize the shape to fit the needs of our game. Most of our work in this project will be using a simple collision shape that closely matches the object shape.

Select the platform Actor in the **Viewport** and press **Ctrl + E**, this will open the **Static Mesh Editor** for this asset.

Lost?

If you are having trouble navigating to the first platform, press the number **1** on your keyboard and then click and hold the left mouse button to look left/right and move forward/backward.

If you need to deselect an Actor, hold Ctrl and click the Actor.

NOTE: You can also open the **Static Mesh Editor** by navigating through the **Content Browser: Content > Hour_of_Code > Static_Meshes** and double-click on the **SM_Platform_M** asset.

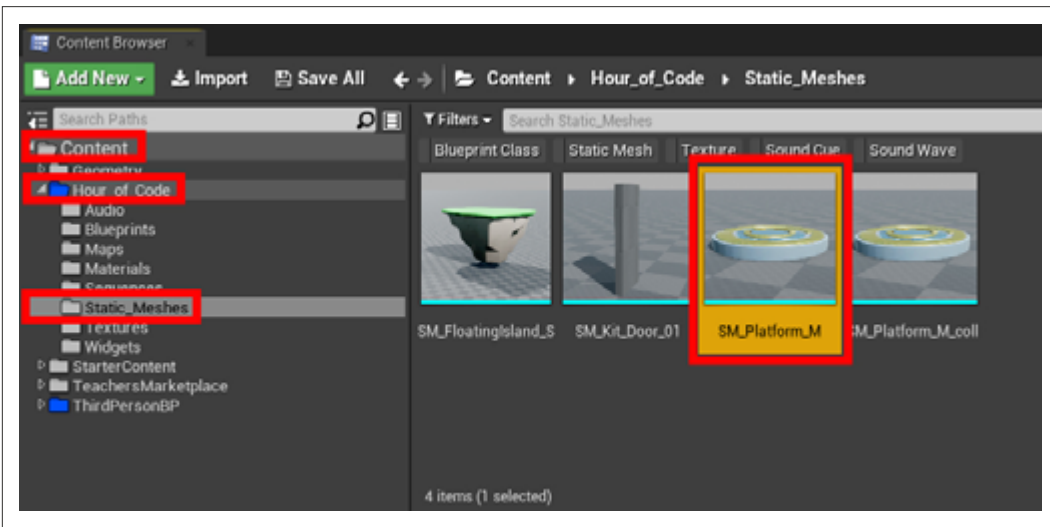


Figure 008 – Selecting the platform to add a Collision Volume.

With the **Static Mesh Editor** open, click on the **Collision** button at the top of the interface and enable **Simple Collision**. Now that we've activated collision detection, we'll need to define the collision area in the next step.

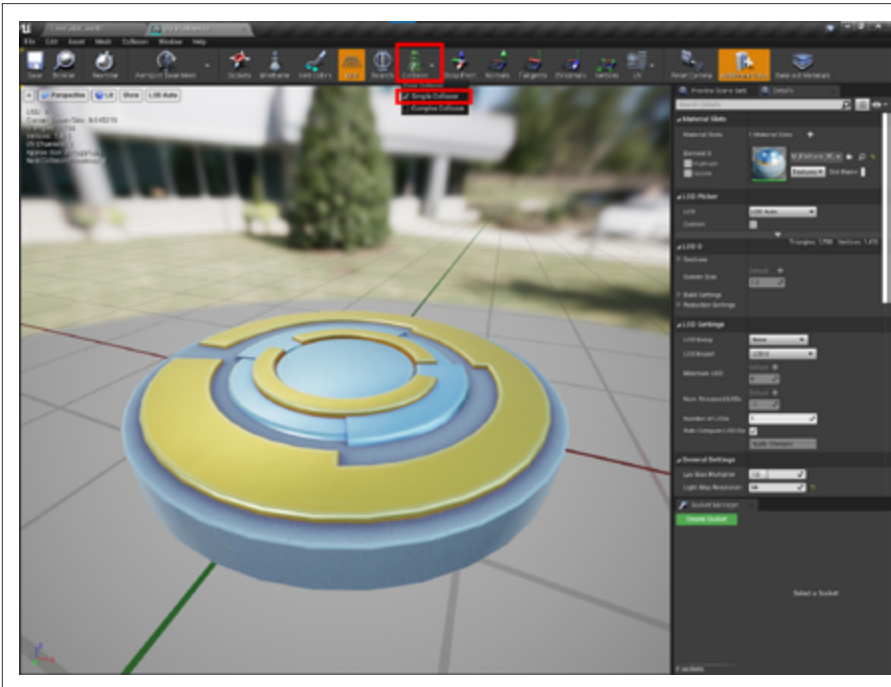


Figure 009 – Enable Simple Collision.

Next, navigate to **Collision** in the menu bar and choose the **Add 10DOP-Z Simplified Collision**. This will create a simple collision volume that fits around the platform. You will see a green shape appear around the platform. It will not fit exactly to the platform, and it does not need to, it just needs to be close enough to work for our gameplay purposes. You can always scale, rotate, and move the collision around to fit your gameplay needs.

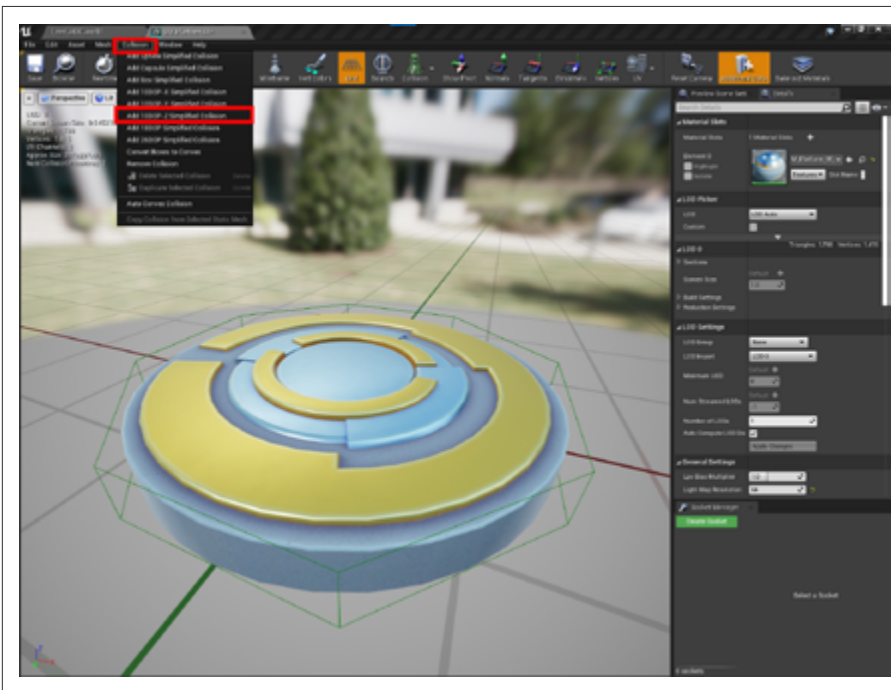


Figure 010 – Add the collision boundaries. (shown as a green outline)

Click the **Save** button in the top left to apply the changes, and then close the **Static Mesh Editor** by clicking the **X** button in the top right of the window.

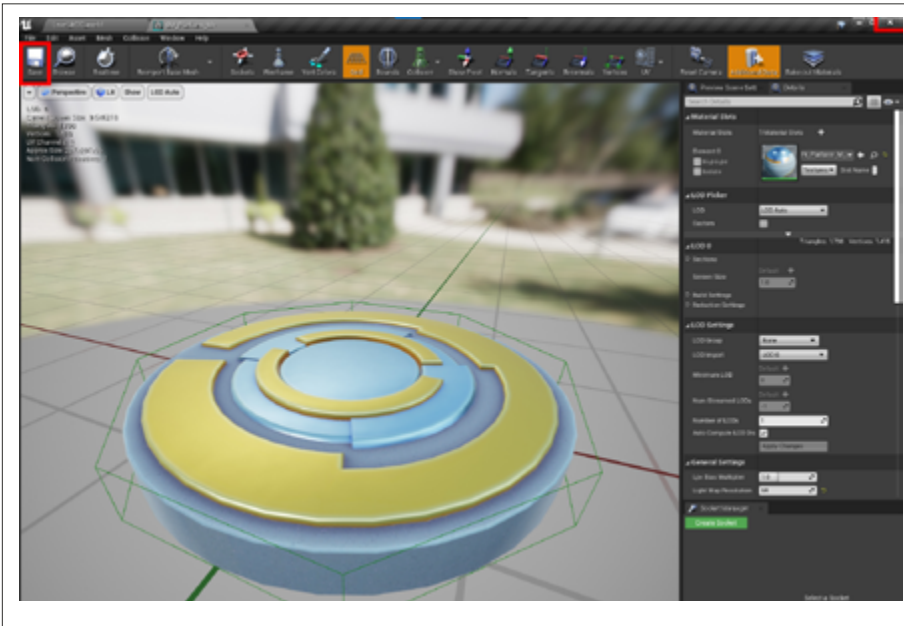


Figure 011 – Save your changes to keep the collision attached to the platform.

Review the Player Collision Definition

Let's take a moment to look at the collision on the player character. This will be important for understanding how the player's collision volumes interact with the collision volumes of objects in your level.

To get a copy of the character, navigate through the **Content Browser: Content > ThirdPersonBP > Blueprints** and then click and drag the **EpicCharacter** into an open space in the level. Then press **Alt + C**, to show the collision on all objects in the world.

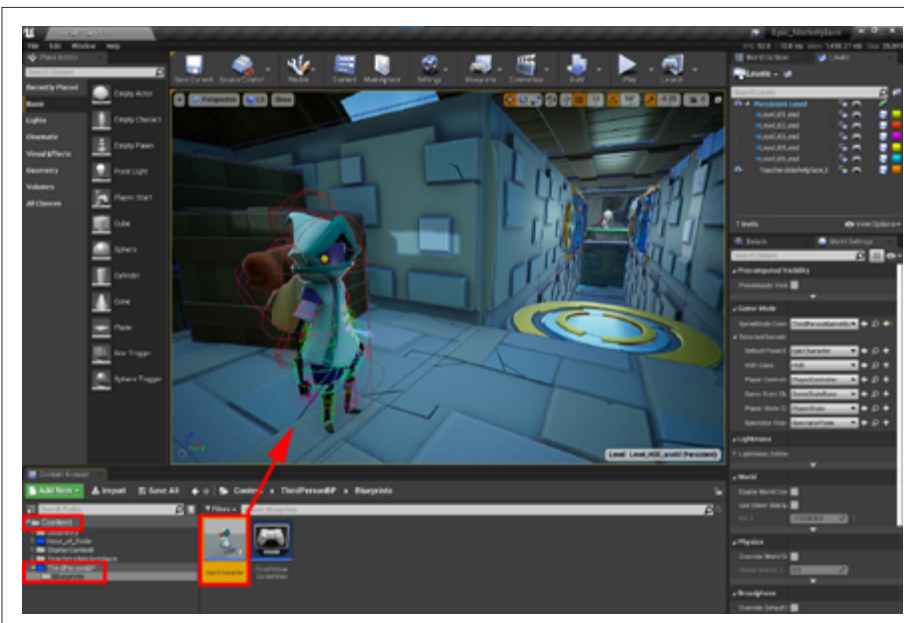


Figure 012 – View the collision volumes defined for the player character.

You will see the various collision volumes that make up the character's body sections. These collision volumes are used to determine if the character can walk on other collision volumes, and in the case of taking damage, when and where the character should be affected.

Hit-Box

Have you heard of the term "hit-box" in video games? In a fighting game, the hit-box is the defined area where the player will take damage when hit. It's also possible that a hit to the head can cause more damage than a hit to the arm. This is all done with collision volumes such as the ones we are working on in this project.

Look at the player example in Fig. 012. Can you see the collision volume shapes that surround the different parts of the body? Notice how the collision volume doesn't match the shape of the body parts exactly. If you've ever been hit by a projectile that "technically" missed your player, this is the reason.

The basic reason for these simplified collision volume shapes is time. A more exact shape would dramatically increase the computing power which would slow the responsiveness of your game. Therefore, as a Game Developer, you would have to balance precision against the most dreaded foe of all...lag!

If you look at the platform you will also see the collision you have created. Can you spot the other collision volumes for all the other Actors within the level?

Press **Alt + C** to toggle OFF the collision preview in the viewport, then select the character we just added and press **Delete** on the keyboard to delete it from the level.

Note: Collision can also be toggled by navigating to the **Show** menu in the viewport.

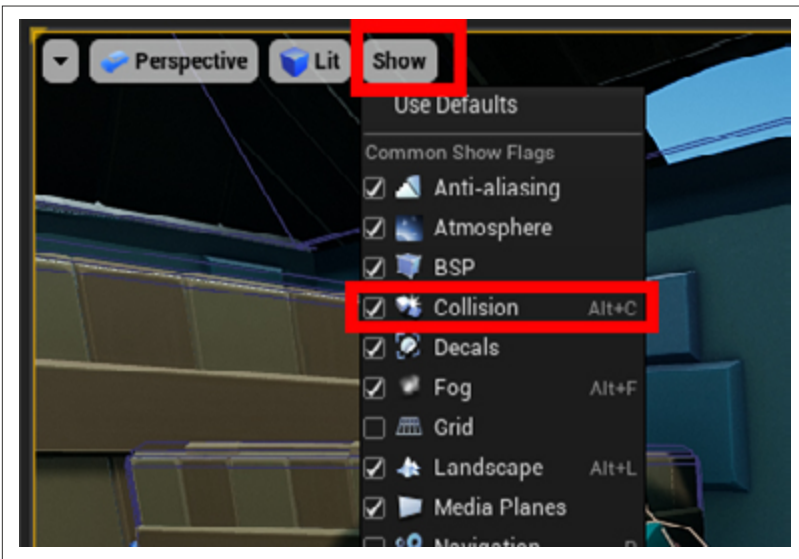


Figure 012b – Toggle Collision view from the Viewport.

Now that we have created Collision on the platform, and we have a better idea of what collision does for our player, we should playtest our platform to make sure it works. Play your level again and double-check that everything is working correctly. Does the player fall through the platform? Go back and double-check your work.

Adding More Platforms with Duplicate

Great! You should have the player landing on the first platform. Now we need to complete this hallway so the player can safely reach the other side. As a Programmer, you need to think about efficient ways to accomplish tasks. This will take less time to complete projects and will likely result in better projects. In this case, we are going to use a shortcut that quickly copies our platform so we can place the platforms needed to get across the hallway.

We can duplicate the platform a few times to create a parkour path across the pit. First, select the platform. Using the move gizmo, hold the **Alt** key, then click and drag on the **red arrow** of the gizmo to duplicate the platform down the hallway. You will need to let go of the **Alt** once you have created the duplicate, then hold it again to create a new duplicate. Once you have a few platforms in place, test your level to see if you can get across.

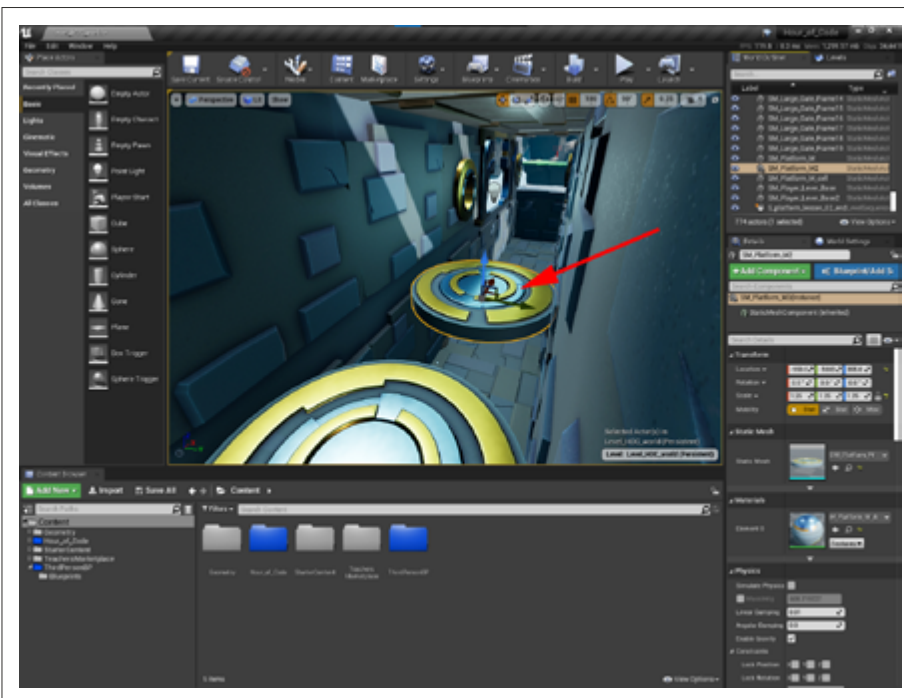


Figure 013 – The “gizmo” used to move, scale, and rotate Actors in your world.

Handling Failed Attempts

If you fell in the pit, you probably noticed that your character is unable to make it out. This is referred to as a “showstopper” or “soft lock”. The game hasn’t ended and there’s no way for it to progress.

Since the game engine doesn’t know the goal of your game, it is only doing what you tell it to. If you want the player to “die” or “restart” you must add that manually. There are many ways to tell the game what you want it to do, and once you complete this activity you will have a better understanding of how to tell the Engine to behave in a specific manner.

Showstopper Bugs

When the player falls into the pit, they are stuck and can't get out. This can be called a bug because the game can't go on.

We can address this bug by placing a **Pain Causing Volume** in the pit that will reset the player and send them back to the **Player Start**.

In the **Place Actors** tab, type "pain" into the search bar. Then drag a **Pain Causing Volume** into the pit. Once you have placed the volume in your level, press **W** to use the move gizmo and press **R** to use the scale gizmo. Place it into a location where the character's collision will touch it when they fall anywhere into the pit. "die" or "restart" you must add that manually. There are many ways to tell the game what you want it to do, and once you complete this activity you will have a better understanding of how to tell the Engine to behave in a specific manner.

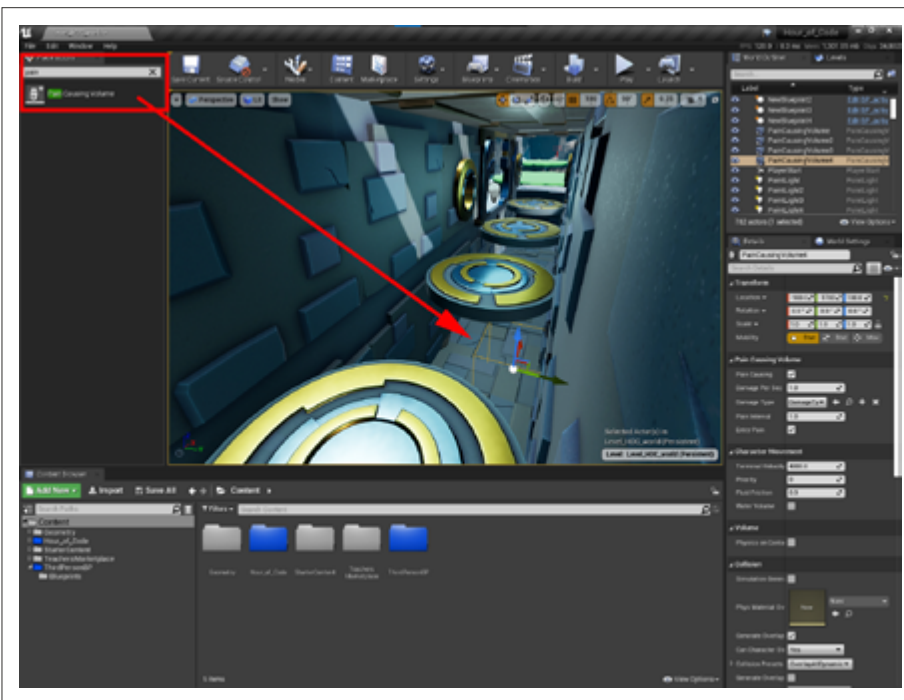



Figure 014 – Adding the Pain Causing Volume to your hallway.

If you are finding it difficult to navigate in such a small space, here are a few pro tips.

- Press **F**, to frame the selected actor in the **Viewport**.
- Click the  button in the top right of the viewport. This will open the top and side views. Click the button again to return to a single viewport layout. (Note: Use the mouse scroll wheel to zoom in/out and hold the right mouse button to move.)

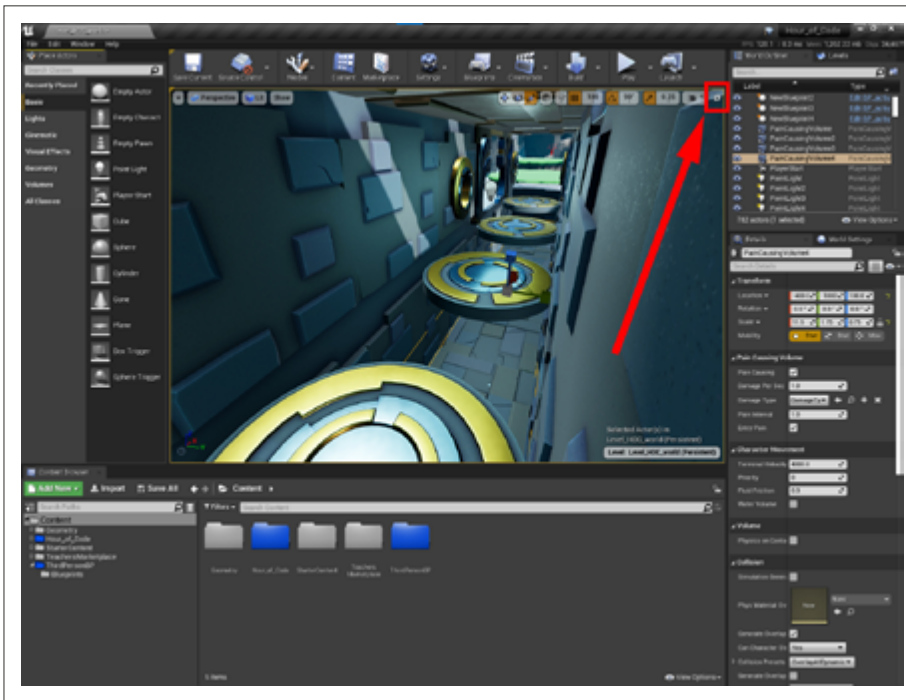


Figure 014a – Change your viewport layout to more easily align objects in 3D space.

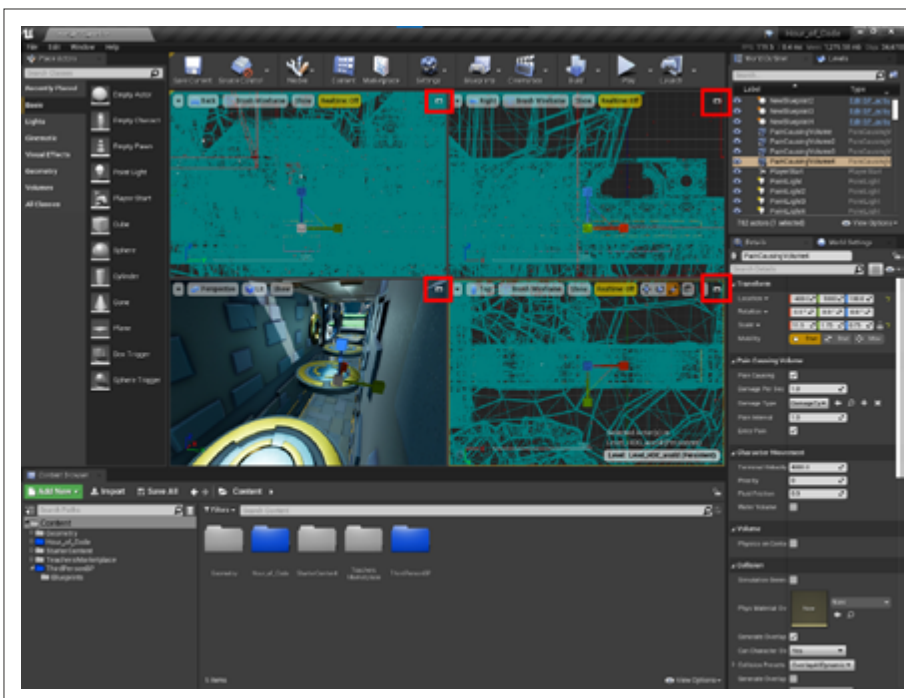


Figure 014b – Each viewport has its own toggle.

Play the game again and check that your character resets when they jump into the pit. You might need to move the **Pain Causing Volume** around a little bit to get it to work the way you want.

Detect Falling into the Void

Now that you can jump through to the end of the hallway, try jumping off the large island at the end of the level. Yeah, that seems unsafe, but is it? Go ahead, give it a try. What happens?

Press **Esc** to stop the game and press the number **1** to reset your view to Level 1.

It looks like we are missing yet another collision. In this case, the player just keeps falling. This is because the game doesn't know that you want to restart after falling a specified distance. We will address this issue by enabling the **Kill Z Bounds**, a plane that will reset the character when it touches it.

In the **World Settings** panel, scroll down to the **World** section and toggle ON **Enable World Bounds Checks**. You may have to click the down-arrow to **Show Advanced** options. Next, set the **Kill Z** to a value of **-500**.

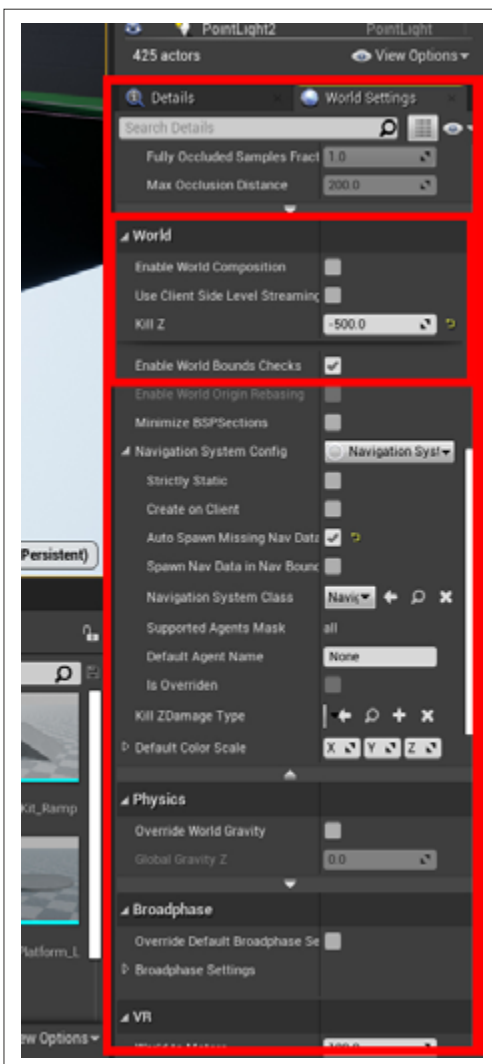


Figure 016 – Setting the Kill Z value for the world.

Play the game again, and now after falling into the void the character will respawn at the beginning of the level.

Now let's create a parkour course to reach the next location.

Build a Parkour Course to the Next Level

Now it's time for you to get creative! Once the player has made it through the hallway jumps, they see exciting new destinations that are floating in the sky. The only way to get to the next destination is a path of floating islands. This is where you come in. In this exercise, we will show you how to place a floating island in the world. Then, it will be up to you to use your skills and experience to create a path to the next destination.



Fig. 017 – Create a path of floating islands to the next destination.

You will add some small floating islands to reach the next location. They can be found in the **Content Browser** under **Content > Hour_of_Code > Static_Meshes** folder. Click and drag one of the **SM_FloatingIsland_S** assets into your scene.

Use the arrows on the Move gizmo to move this first floating platform into your desired location. If you do not see the Move gizmo, select your floating island and press the **W** key. Remember to use the icon in the top-right of the viewport to toggle the view.

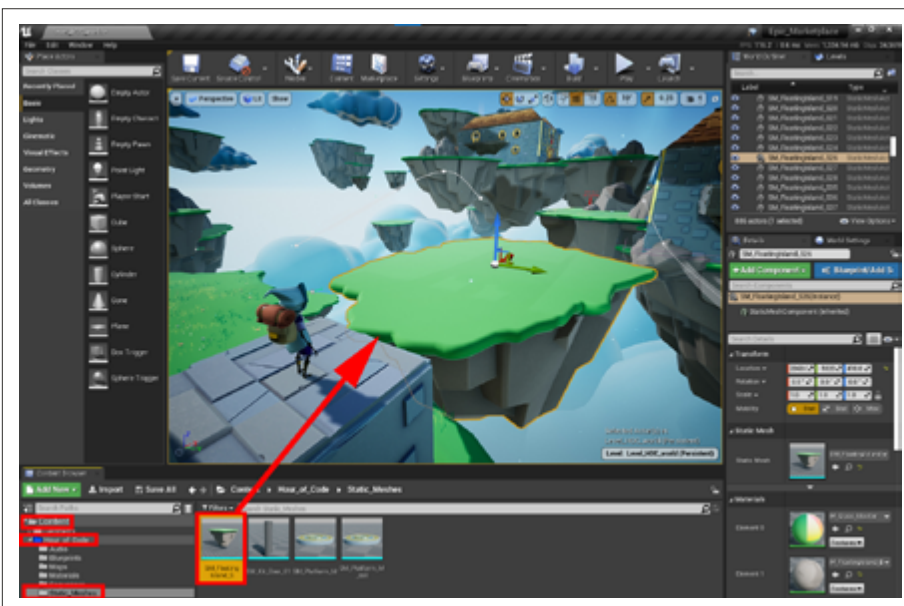


Fig. 018 – Adding a floating island to your world.

With one floating island in the scene, you can duplicate it just like you did with the platforms. Hold Alt + click and drag one of the arrows of the move gizmo to create another floating island. Create a path that will reach the far platform. It will appear similar to the image below.

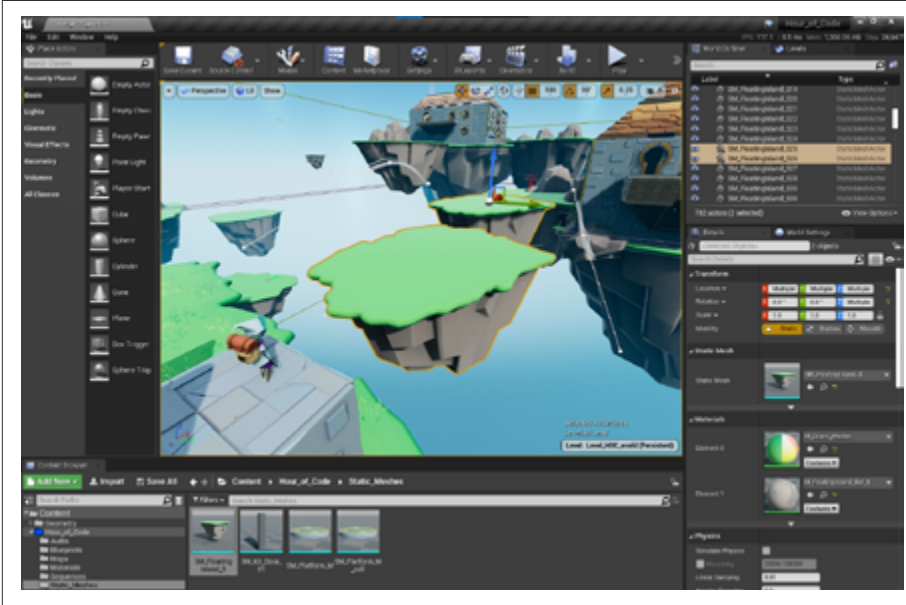


Fig. 018b – Example of floating island path to the next level.

Pro-Tip: Want to quickly test your jumps in the game? Click the down-arrow to the right of the Play button. Change the **'Spawn Player at'** option to **Current Camera Location**, then click the Play button to start your player at your current camera position. Make sure you're above solid ground before pressing Play. Don't forget to change it back to **Default Player Start** when you're done.

Again, we want to play-test this, so try and jump from one platform to the next to see if you can create a path where your players will enjoy traveling through.

Be sure to **Save** your work by navigating to **File > Save All**, or by pressing **Ctrl + Shift + S** on the keyboard.

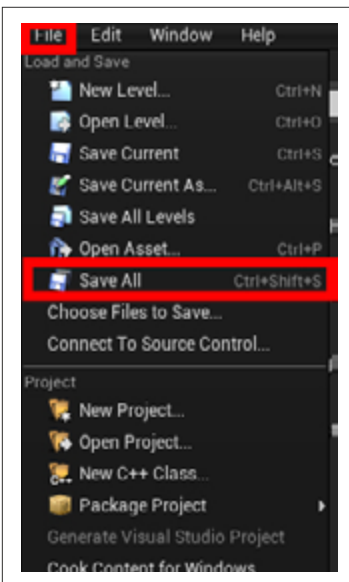


Fig. 019 – Save your work using 'Save All'.

Challenge: Can you build a set of platforms that gives the player a few different choices to get to the next section? Players enjoy having choices, it creates a more non-linear experience.

Game Design Fun: Due to the settings where collision volumes don't render visibly in-game, you can create a gameplay scenario that would never be possible in the real world. Try building an invisible bridge, however, keep in mind that players don't respond well to invisible walls. If you use collision to 'box' a player in, then you should always give the player a logical explanation of why they can't access a location. You can find Blocking Volumes in the Place Actors panel.

What's Next?

In the next Hour of Code activity, we will continue to build a more challenging game by adding moving platforms, as well as adding a checkpoint to help ease some of the frustration that comes from having to play the same section over and over.