Unity Interface Overview

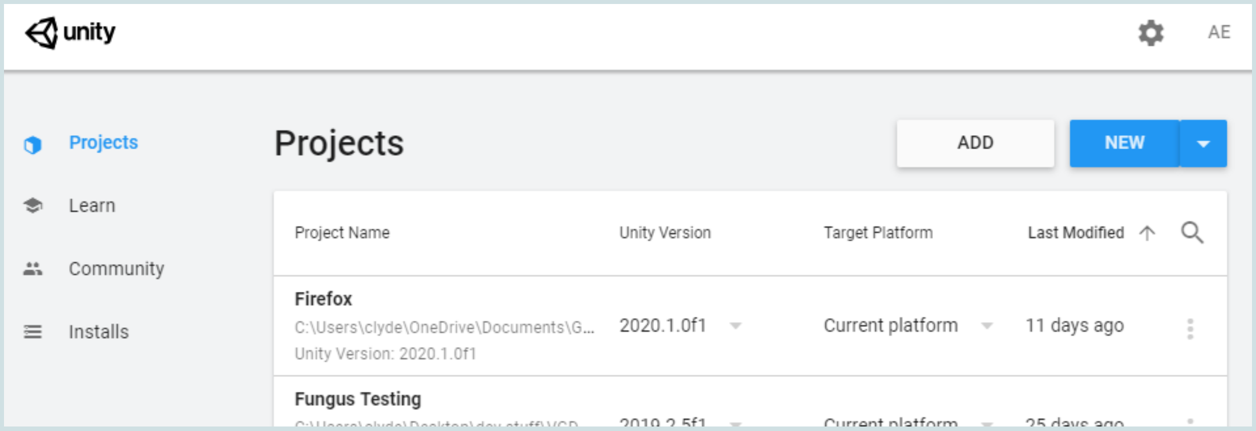
By Adam Eckert

This overview is designed to give you a basic familiarity with the Unity Interface. This is an important skill set for design team members, as knowing the fundamentals of Unity allows you to implement levels, narrative segments, and test out tweaks and changes for yourself!

If you do not have Unity/Unity Hub installed, please ask an officer for direction.

Unity Hub/Opening a Project

Once you have downloaded the “Unity Interface Tutorial” sample project (link) extract the contents of the zip file and open up Unity Hub.
Click the ”Add” button pictured below, and navigate in the pop-up window to wherever you extracted the file. Select the folder called “Unity Interface Tutorial” and press the **Select Folder** button.



Project Name	Unity Version	Target Platform	Last Modified
Firefox C:\Users\clyde\OneDrive\Documents\G... Unity Version: 2020.1.0f1	2020.1.0f1	Current platform	11 days ago
Fungus Testing	2019.2.5f1	Current platform	25 days ago

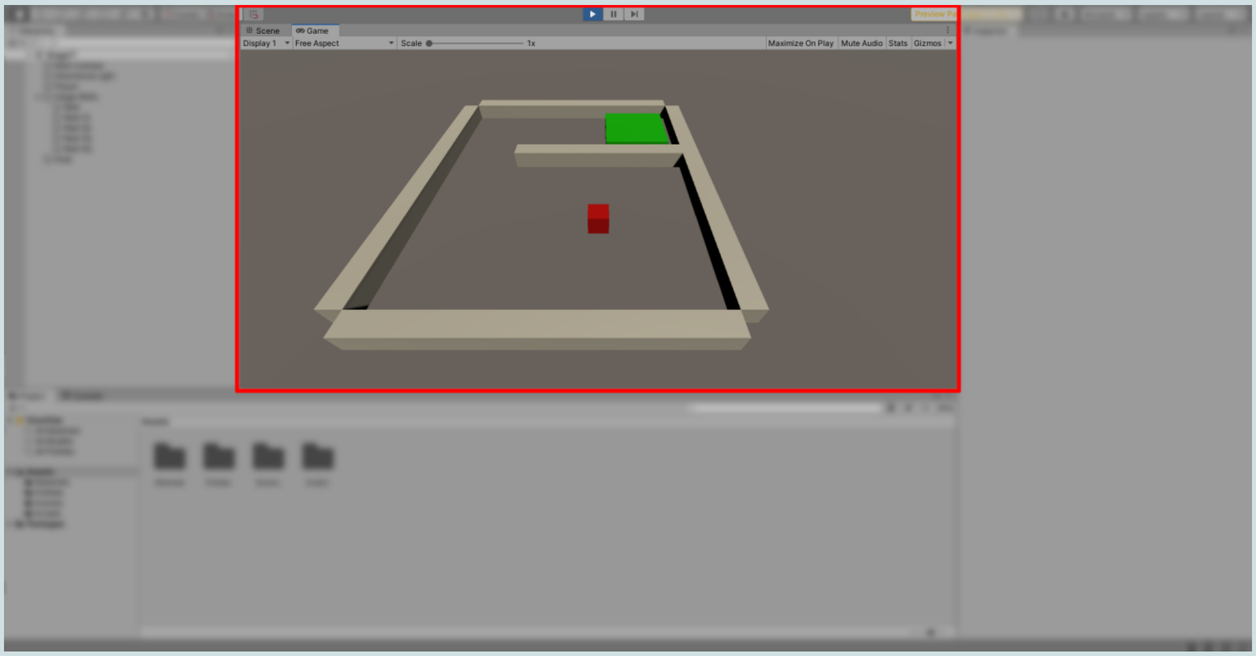
Once this is done, the project should appear in your Unity Hub Projects list. Click on the name of the project to open it in Unity. (This may take a little while if you have a slow computer.)

Unity Interface

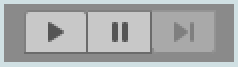
Upon opening a project in Unity, you will be met with a number of different panels, which we will go through one-by-one in the following sections.

Note that this is written for the default layout. Users can drag panels around to change the layout however they like, and there are options to load and save layouts in the top right corner of the interface.

Running the Game



First things, first, let’s show you how to actually run the game itself! In the very middle of the toolbar at the top of the Unity interface, there are three buttons, **Play**, **Pause**, and **Step**, as pictured:

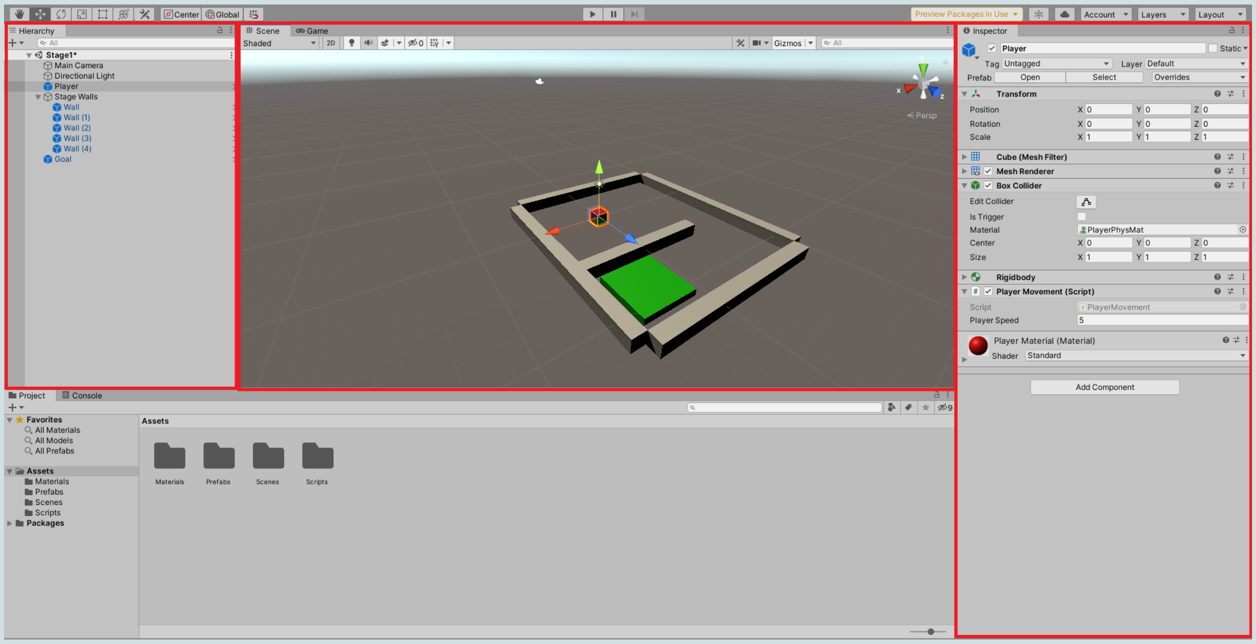


Pressing the play button will run the game. While the game is running, Unity is in **Play Mode**. In Play Mode, the game is run in the **Game View**, which automatically replaces the Scene View in the default layout.

To pause the game, press the pause button. While paused, pressing the step button will advance the game by one frame at a time. To exit Play Mode (which stops running the game) press the play button again.

While running the sample project that accompanies this overview, you can use the Arrow Keys or WASD to move the red cube around. Upon reaching the green square, the scene is reloaded and the square moves back to its original position.

Editing Game Elements

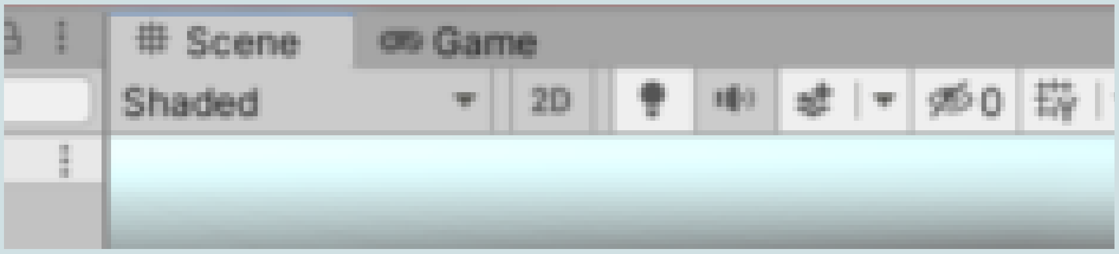


Player
Tag Untagged Layer Default
Prefab Open Select Overrides
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Cube (Mesh Filter)
Mesh Renderer
Box Collider
Edit Collider
Is Trigger
Material PlayerPhysMat
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Rigidbody
Player Movement (Script)
Script PlayerMovement
Player Speed 5
Player Material (Material)
Shader Standard

Scene

While not in Play Mode, the **Scene View** is present instead of the Game View in the upper middle area of the screen. This view allows you to fly around the game world, see invisible objects, and select and edit elements of the game.

You can select whether the Scene or Game view is displayed using the tabs in the top left corner of the panel, pictured below:



Moving in Scene View

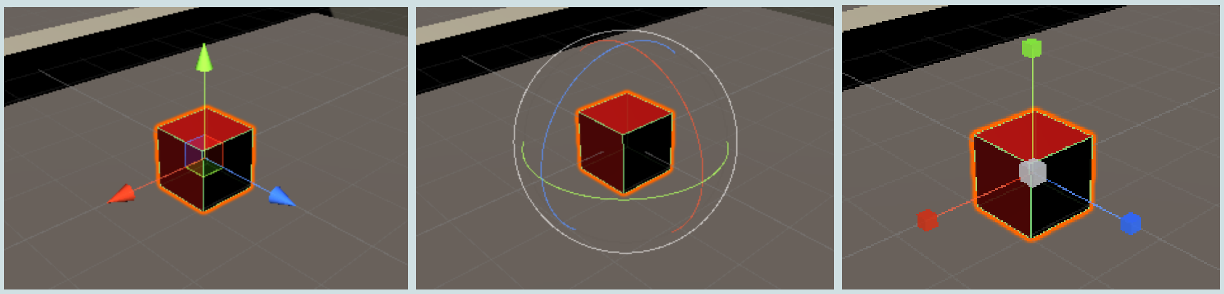
To fly around in the Scene view, just move your mouse anywhere inside it and press & hold the right mouse button. **While the right mouse button is held down**, you can look around by moving the mouse, and move using the keyboard:

- **W, A, S,** and **D** move the camera **forwards/backwards** and **left/right**.
- **E** and **Q** move the camera **up/down**.
- Holding **Shift** while moving will make you **move faster**.

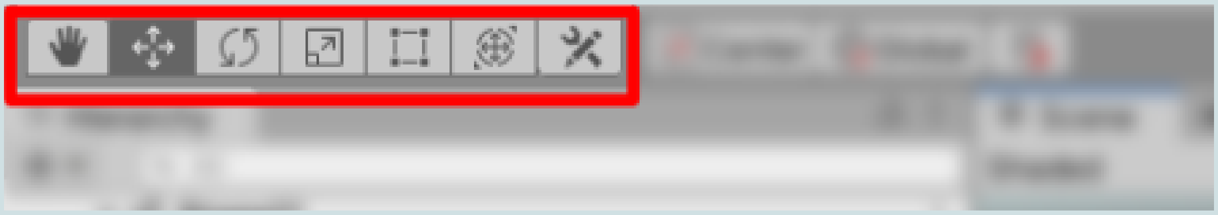
Try flying around for a bit!

Selecting Objects

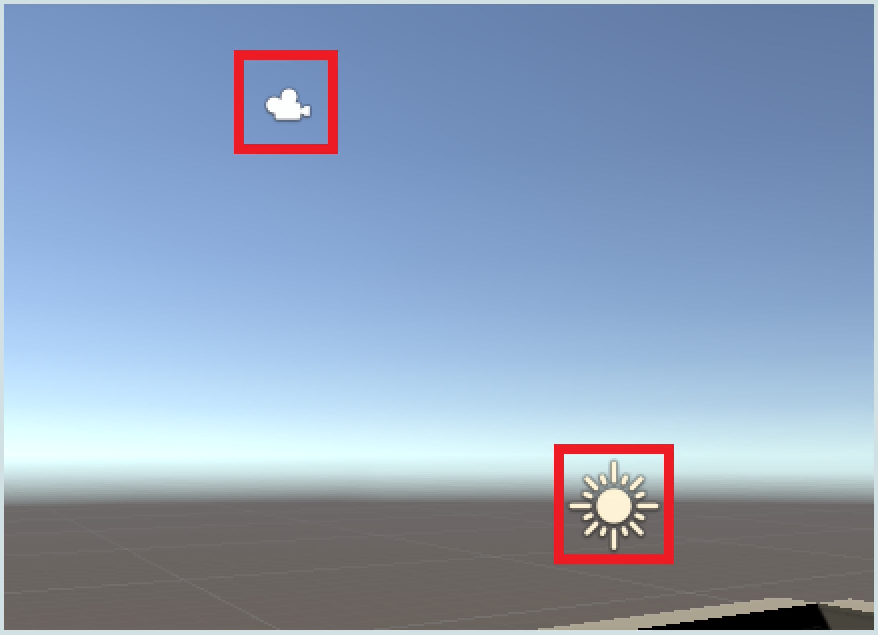
While in the scene view, you can left click on any objects to select them. Selected objects will have an orange highlight, and some tools will appear that allow you to edit either the **position, rotation,** and **scale** of the object, as pictured:



You can select which one of these tools you want to see by pressing **W (position), E (rotation),** and **R (scale)**.
You can also use this set of buttons (pictured below) to select one of these tools (and a few others) or to see which one is currently selected. They are found in the top left corner of the Unity editor.



Some objects are represented by 2D symbols in the Scene View, such as the **Camera** and **Directional Light** objects pictured below. These objects aren't visible in the Game View, and generally serve other functions. These two are automatically created in any new Unity scene, as they play the important role of defining the vantage point of the player when running the game and providing a light source so the player can see game objects.

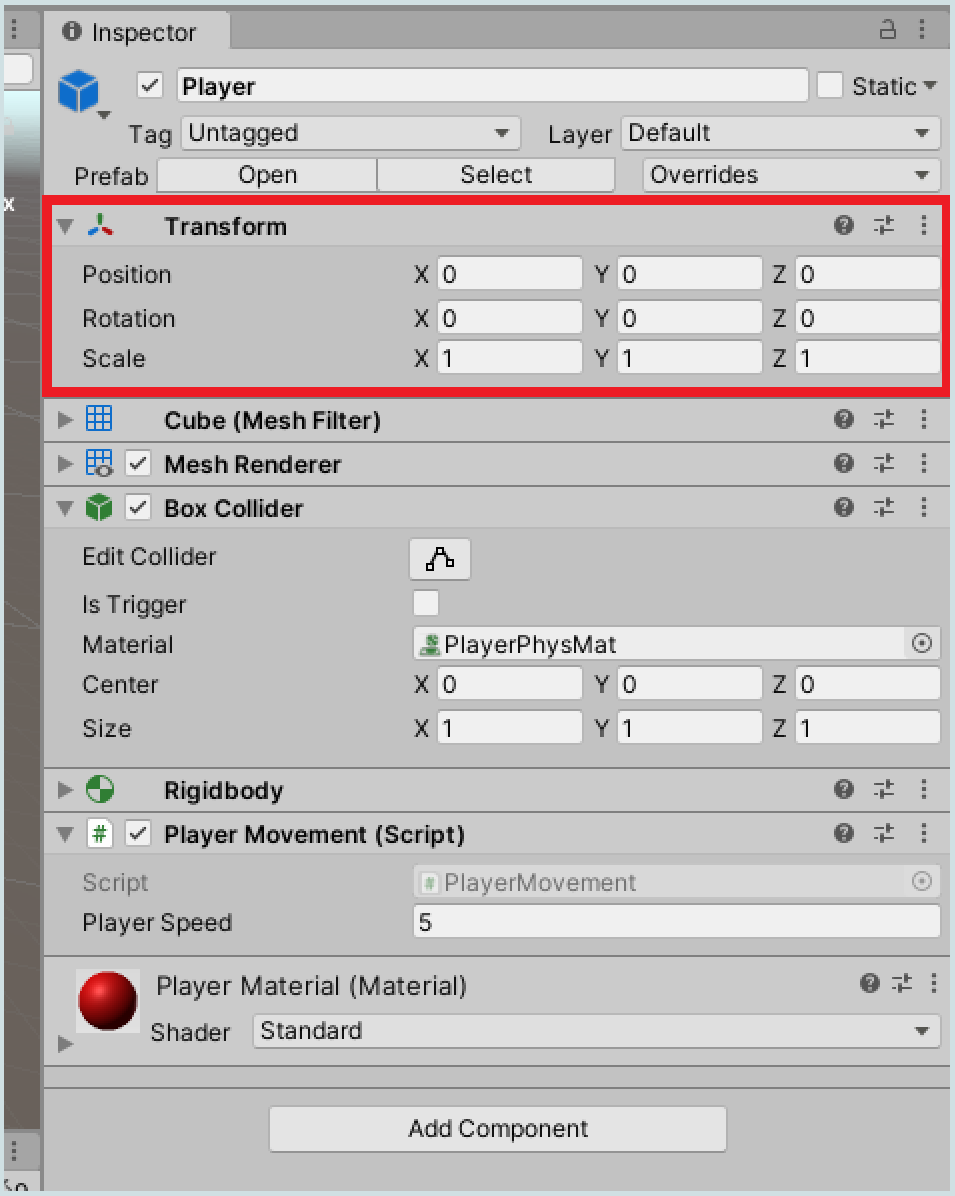


There are many uses for objects that serve more abstract functions in the game world. Unity comes with many objects prebuilt, though users can make their own as well..

Inspector

When an object is selected, you may notice that the panel to the right, known as the **Inspector**, changes. This panel displays information about *anything* that is selected in Unity. This includes both game objects as well as assets selected in the Project Panel (which we will get to later.)

In the image below, I have selected the player object, and the inspector is showing all of the **components** (the labeled collapsable sections) of this object, along with their values:



Inspector

✓ Player | ☐ Static

Tag: Untagged | Layer: Default

Prefab: Open | Select | Overrides

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Cube (Mesh Filter)

✓ **Mesh Renderer**

✓ **Box Collider**

Edit Collider

Is Trigger: ☐

Material: PlayerPhysMat

	X	Y	Z
Center	0	0	0
Size	1	1	1

Rigidbody

✓ **Player Movement (Script)**

Script: PlayerMovement

Player Speed: 5

Player Material (Material)

Shader: Standard

Add Component

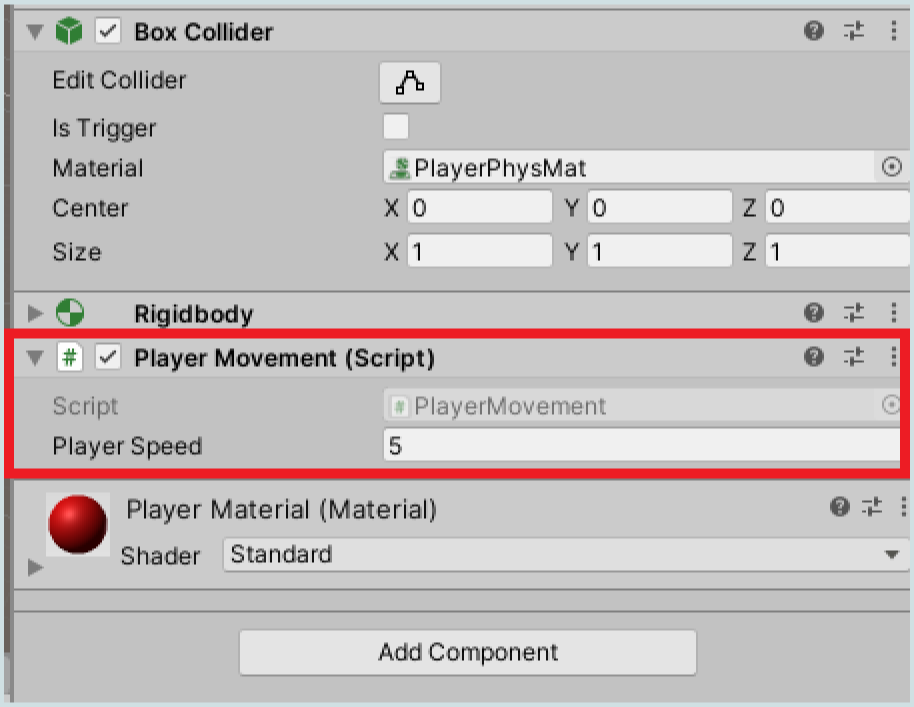
This may look overwhelming, but don't worry; many of the elements the inspector shows are, to designers at least, irrelevant most of the time. For now, just notice the **Transform** component highlighted in the image, which contains the Position, Rotation, and Scale of the object on each axis. **Every** object has a Transform, even the more abstract ones that may not be affected by what some of its Transform values are.

In any case, you can use the Inspector to edit these values without manually moving them around in the Scene View. To edit them, **click on the text box you want to edit and type your desired value**, or, alternatively, **click and hold the label of the text box you want to edit and drag the cursor left or right**.

Editing at Runtime

An important (and very useful feature) of Unity is the ability to edit attributes while the game is running. This allows you to very quickly and easily refine values to be exactly what you want without having to stop and restart the game. **However, any changes you make while the game is running will revert back once you exit Play Mode**.

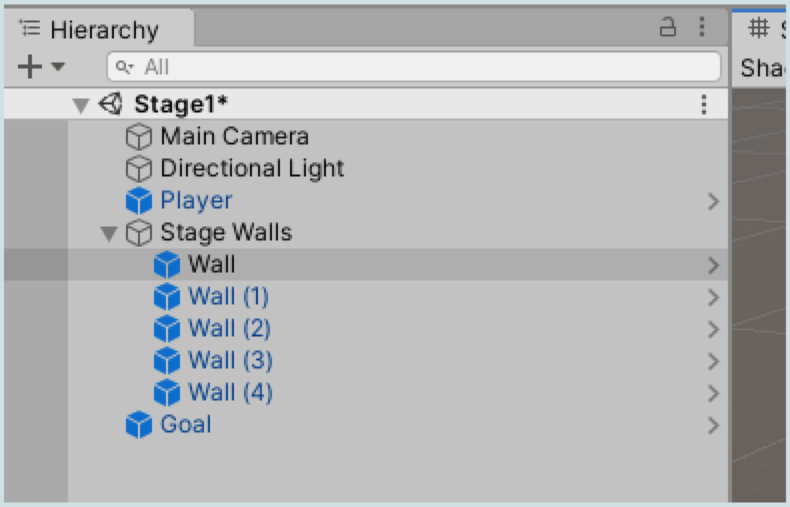
You can try out this functionality with the Player Speed value of the Player Movement script, attached to the Player object, pictured below:



Box Collider
Edit Collider
Is Trigger
Material PlayerPhysMat
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
Rigidbody
Player Movement (Script)
Script PlayerMovement
Player Speed 5
Player Material (Material)
Shader Standard

Hierarchy

Another thing you may have noticed is the panel to the left highlighting an item when it is selected in the Scene View. This is the **Hierarchy Panel**, and it displays a list of every object currently present in the game. You can select objects in this view as well, and any selected objects will be highlighted in the Scene View and visible in the Inspector.



Object Parenting

In the image above, you'll notice that all the Wall objects are listed together, slightly indented, under the "Stage Walls" object. This object is what's known as an "empty" game object (it only has Transform attributes) and I created it to keep the Hierarchy organized.

Organization

If you click the arrow to the left of this object, the list of Wall objects in the Hierarchy will disappear. Reclicking this arrow, or selecting a wall object via the Scene View, will cause this list to reappear. This structure is referred to as **Object Parenting**, where the Stage Walls object is the **parent** and all of the Wall objects are **children**. As I said before, I did this to keep the Hierarchy a little more approachable with a collapsable list, but this functionality has other very important uses too!

Local and Global Space

If you select the Stage Walls object and edit its Transform in the Scene View or Inspector, you'll notice that **all of the child objects will move along with it**. What's more, even though these wall objects moved, a look at their Transforms reveals that **their position values didn't actually change**.

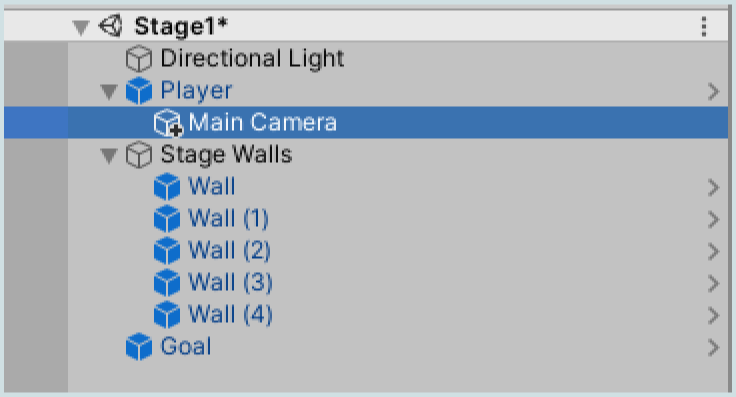
This is because child objects exist in **Local Space,** as opposed to **Global Space**. Any object that has no parent is positioned using Unity's global coordinate system - if their Transform has an X value of 4, then their X position is equal to 4.

Child objects, however, are positioned relative to their parents. If their Transform's X value says 4, then their X position is 4 greater than that of their parent object.

How to Create Parent-Child Relationships

One great use for this feature is to make a camera that follows the player. Currently, when running the sample project, the camera object stays still as the player moves around. If we want it to follow the player, we can make the camera object a child of the Player object!

We can do this using the Hierarchy. Simply click and drag the Main Camera object onto the Player object. Make sure that the Player object is lightly highlighted in gray when you let go of the mouse button. When you are done, Main Camera should be indented underneath the player, like this:



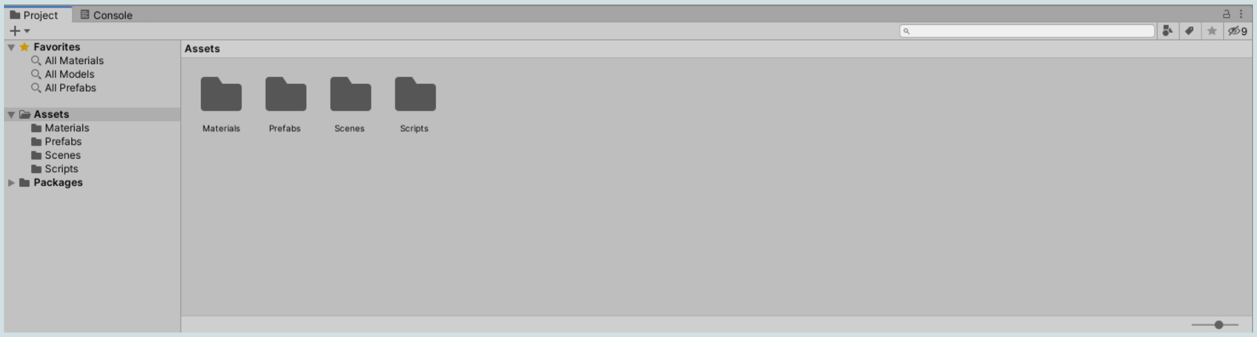
Now, when you run the game, the Main Camera will follow the player. Test it for yourself!

One last note, this parent-child structure can be nested; that is to say, you can give child objects their own children, and so on and so forth, and they each will be positioned relative to their immediate parent.

Understanding Project Structure

Project Panel

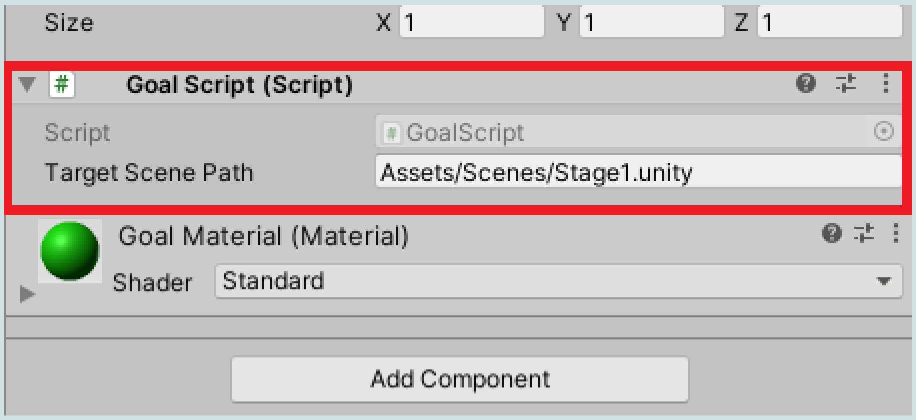
Now it's time to take a step back and look at how what we've done so far fits into the bigger picture. At the bottom of the screen, you will see a panel that looks like this:



As you can tell, this panel is structured very similarly to a regular file explorer. This panel allows you to browse the contents of the game's project folder; specifically, we are interested in the

Assets folder. The Assets folder contains all of the assets in the Unity project. This includes any file, like images, scripts, and more.

If you look at the Goal object, you'll notice that the Goal Script has a value that points to the location of a scene asset. Depending on the component, assets of any type can be used to fill a value in the Inspector.



Size X 1 Y 1 Z 1

Goal Script (Script)

Script: GoalScript

Target Scene Path: Assets/Scenes/Stage1.unity

Goal Material (Material)

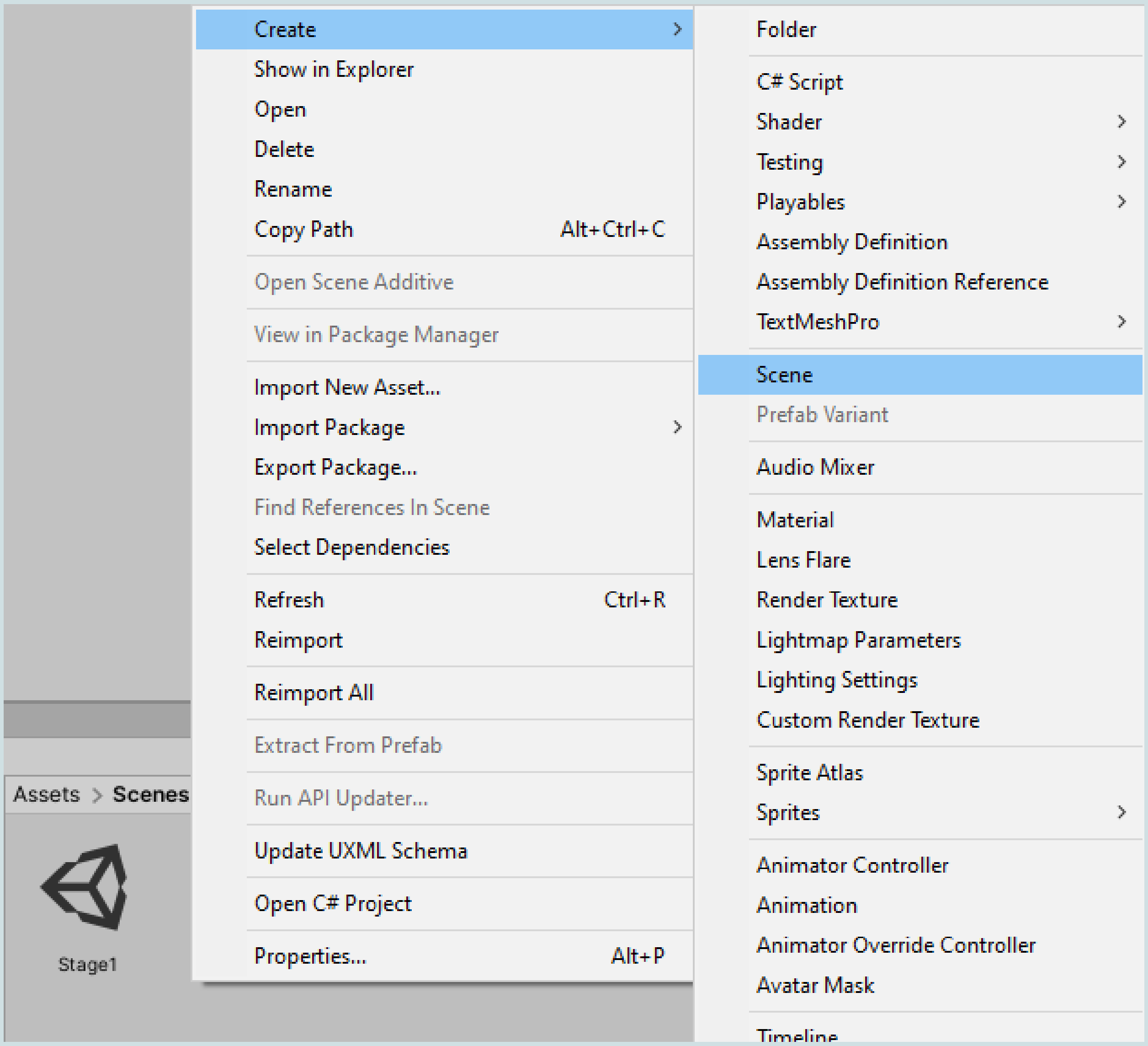
Shader: Standard

In this case, the value refers to which scene should be loaded when the player reaches the goal. Throughout this section, I will take you through the steps of making a second level for the sample project so we can plug that scene into the script for the goal object.

Scenes

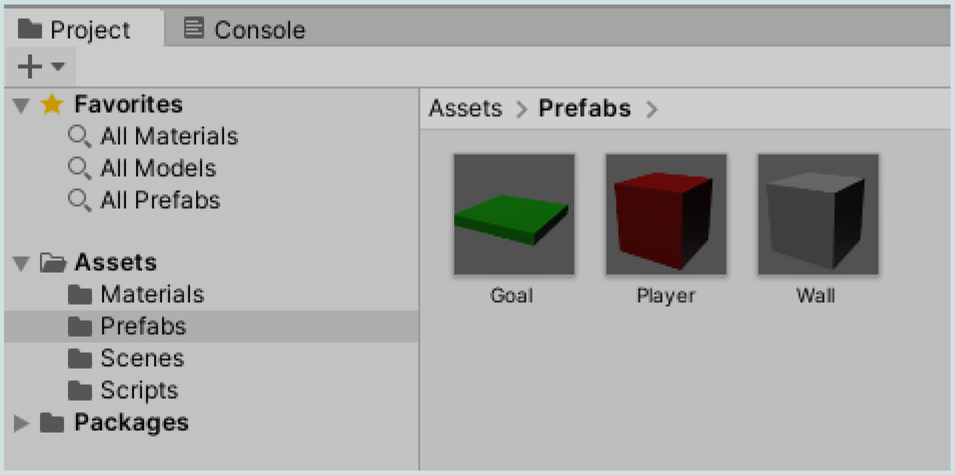
Up to this point, you have been working within a **Unity scene** called "Stage1". Unity scenes are essentially distinct spaces in which a game takes place. For instance, a game like *Super Mario Bros.* might have a scene for its main menu, a scene for each level, and scenes for its game over and level transition screens. If you have experience with Gamemaker, a Unity scene is very similar to a GM room.

In the sample project, navigating to the Scenes folder should show you Stage1, the scene you currently have open. To make another, right click in the empty space next to Stage1 and select **Create > Scene** and name it "Stage2". There is still work to do in Stage1, so do not open it yet.



Prefabs

Now that we have a new scene, we have to prepare some objects to fill it with. Go ahead and navigate to the **Prefabs folder** in the Project Panel.



Here, there are three different prefabs that came with the sample project: the goal, the player, and the wall. Upon selecting one of them, its components and attributes will appear in the Inspector. As you may have guessed, **a prefab is like a blueprint for a game object**.

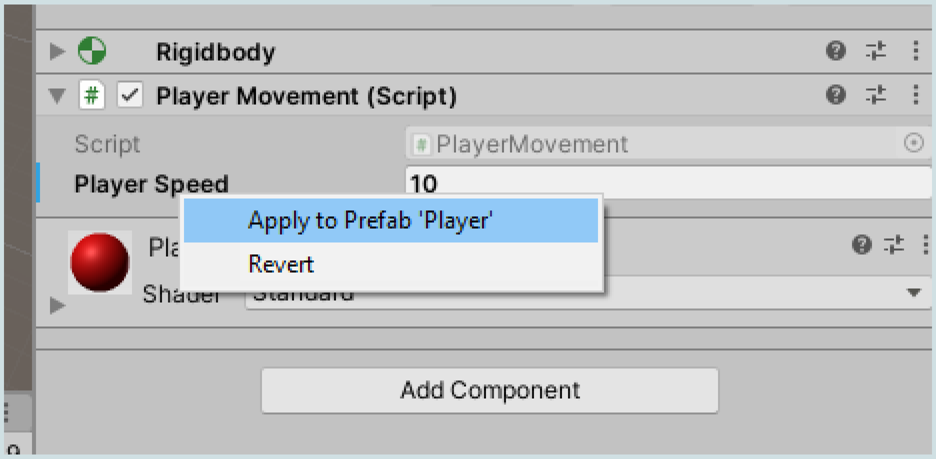
You can click and drag any prefab into the Hierarchy or Scene View, and an **instance of that prefab** will be created in the scene. This means that all the values you see in the Inspector for that game object will default to whatever value its corresponding prefab says it should be. If a game object is an instance of a prefab, it will have a blue cube to the left of its name in the Hierarchy.

If you change the values in an instance (like how we changed the player's speed earlier) it *will not* change the value stored in the prefab, instead making that value show up in bold in the Inspector.
Conversely, if you change the values in a prefab, it *WILL* change the default value in *EVERY INSTANCE OF THAT PREFAB, IN EVERY SCENE!!!*

As such, it is **very important** to double check whether you have selected a *prefab from the project panel* or an *instance of a prefab from the scene* whenever you make changes.

Applying Changes to Prefabs

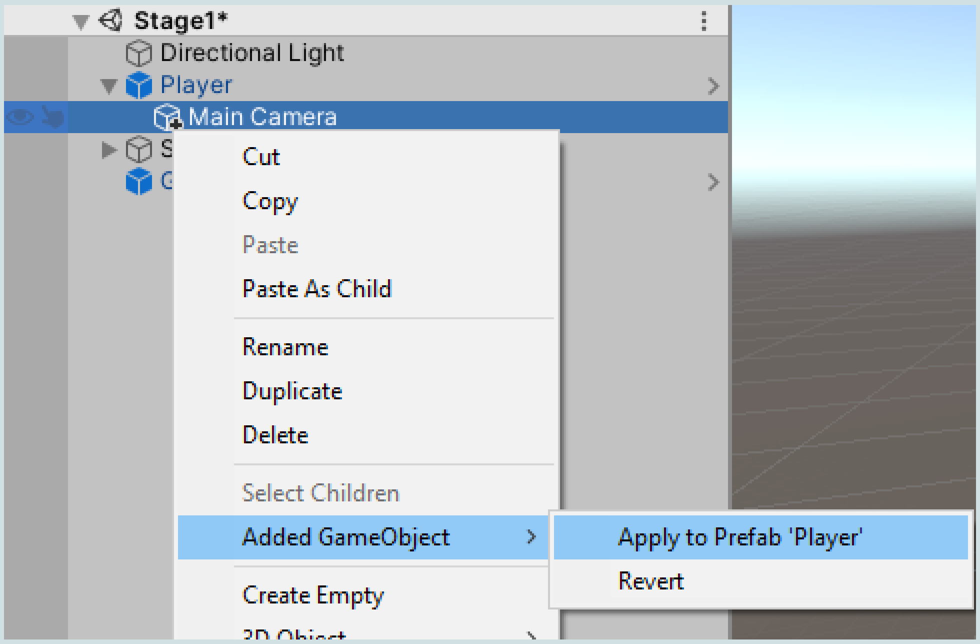
However, if you have made changes to an instance, you can still apply those changes to its prefab if you wish. For instance, let's apply changes we've made to the Player object in Stage1 to the Player prefab. In the image below, I've changed Player Speed in my instance to 10. To apply that to the prefab, all I have to do is **Right Click > Apply to Prefab 'Player'** on the name of the value I want to apply. You can also select **Revert** to undo whatever changes you made to the instance.



Applying Children to Prefabs

That's not all! Remember how we made the Main Camera object a child of the Player object? If we apply that change, each instance of the Player prefab we create will come with its own Main Camera child object out of the box.

To do this, right click on the ***child object*** in the Hierarchy and select **Added GameObject > Apply to Prefab 'Player'**.



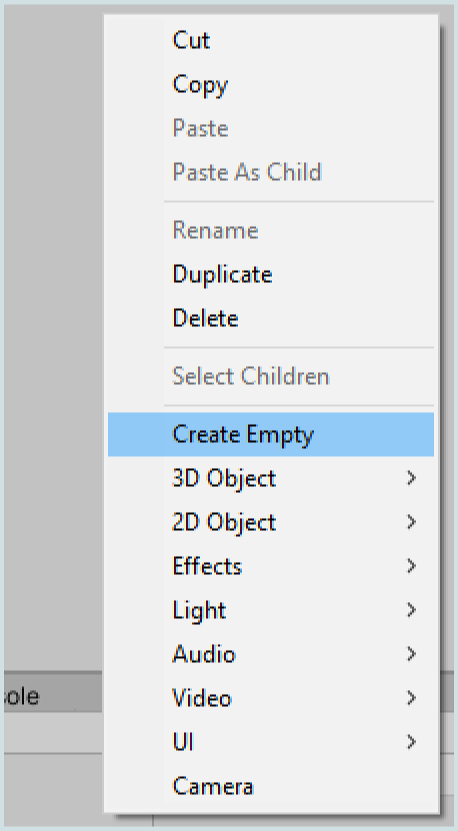
Creating Instances and Other Game Objects

Now it's time to take these prefabs and fill up our Stage2 scene! You should know everything you need to lay down some walls, place a goal, and put the player in there. Feel free to reference the previous sections or ask an officer if you need any help!

Be sure to delete the Main Camera object that Unity created automatically in the scene, since we already have a camera on the Player prefab.

Once that's done, all that's left in this scene is to organize the Hierarchy a little. Remember the "Stage Walls" object in Stage1 of the sample project? We're going to make one for Stage2 as well!

If you **Right Click > Create Empty** in any blank space in the Hierarchy, you will create a game object with no components other than its Transform. This context menu also allows you to create many other types of game objects, which you can explore for yourself if you wish. As a designer, though, you will mostly just be placing prefabs in the scene.

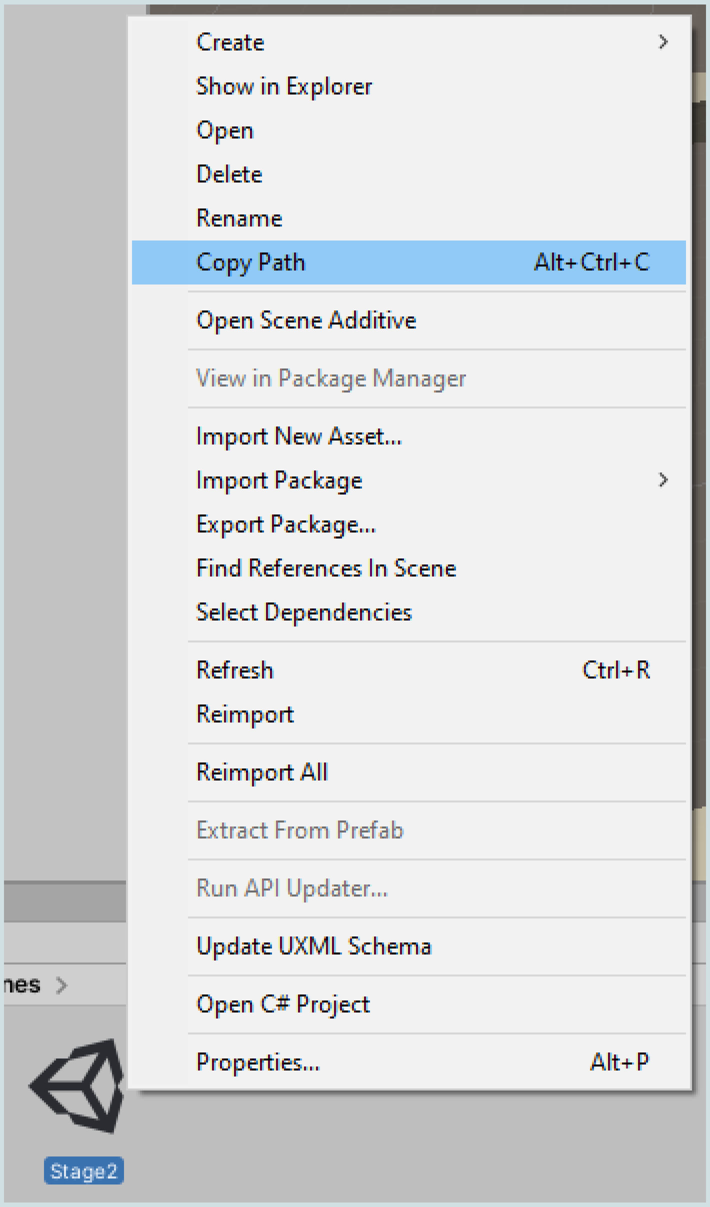


Rename it "Stage Walls" at the top of the Inspector, and set its X, Y, and Z positions all to zero (not required but good practice.) Now, you can select all the Wall instances at once and make them children of the Stage Walls object.

Congratulations! Your level is complete, and your Hierarchy is nice and organized!

Finishing Up

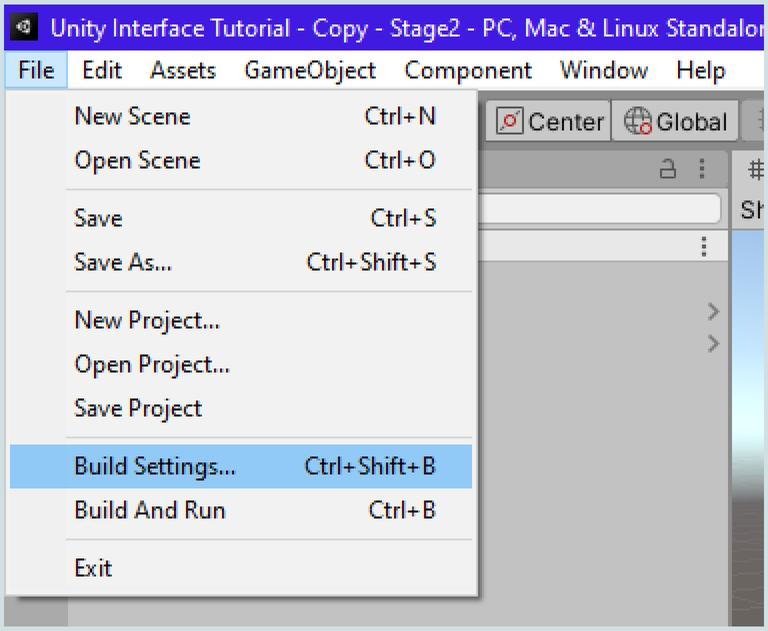
All that's left is to wire your levels together. Go to Stage1 and open up its Goal instance. Usually, when dealing with a script or a prefab, you can just drag the asset straight from the Project Panel into the value you want to assign in the Inspector. In this case, though, the Script is looking for the file path that leads to the scene it takes us to next, so we need to get the path for Stage2. To do this, just **Right Click > Copy Path** on Stage2 in the Project Panel, and paste the value into the Inspector for the Goal instance. (**It will probably not work yet. See note below.**)



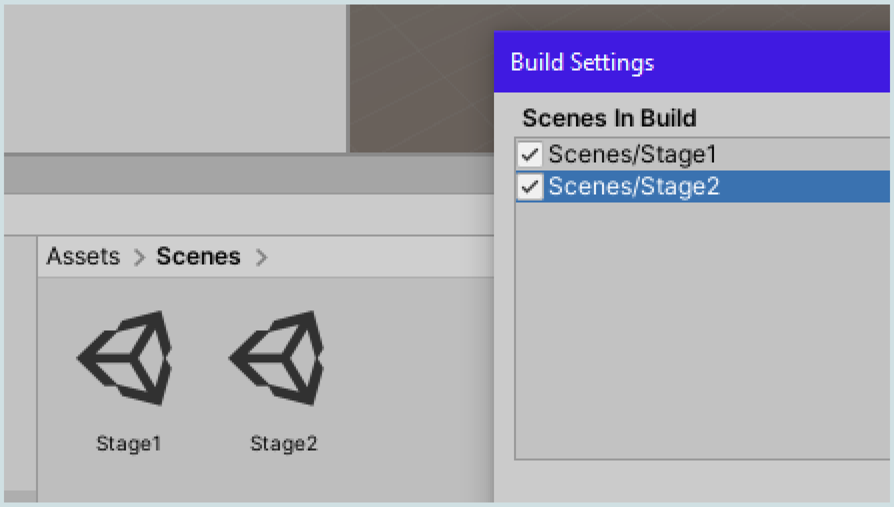
And now you have a working game! ...In theory. In practice, the Goal probably did not take you anywhere when you reached it. This has to do with the Build Settings, which are not really relevant to the Design team. You can stop reading this guide now if you want, but if you want to do this final (and very easy) step, read ahead.

Actually Finishing Up

In the top left corner, select **File > Build Settings**.



Drag both of your scenes from the Scenes folder into the big box at the top of the Build Settings window, as pictured:



Simply close out of the Build Settings window. Do not press Build or Build and run. When you press the play button to test the game, your scenes should load in perfectly!