Keaton Smith
February 10, 2023

# Basic Game Development in Unity

## Introduction:

This wiki article will equip you with all of the knowledge you need to get started developing games in Unity-3d. Even if you are a complete beginner, after reading this article you will be well on your way to designing your own games.

## Table of Contents:



## 1. Required Materials

**To get started using Unity, first you will need to download the software which can be found for free here**. You will also want to have an integrated development environment (IDE) for programming. Common IDEs include Microsoft Visual Studio, or if you want a more minimalistic looking IDE try Atom. To build any scripts you write you will also need to download the appropriate libraries. You may also need to download winrar here for unzipping any downloads.

## 2. Unity Editor UI

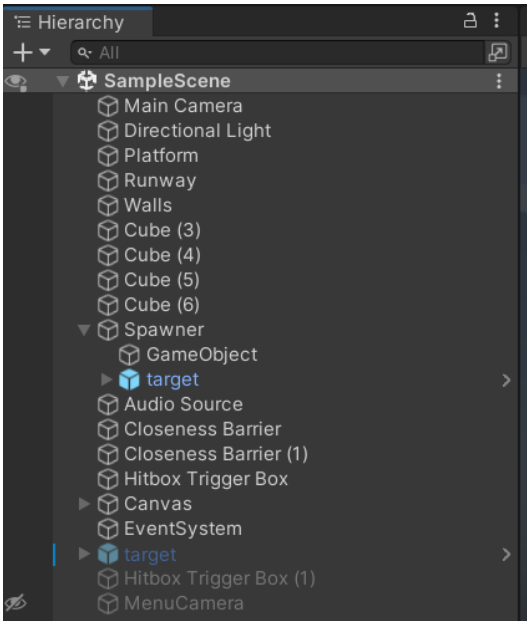
This section covers the basic modules within the editor including:

- Hierarchy Menu
- Inspector
- Assets
- Scenes
- Layers
- Navigating the Scene and Game Window
- Pause and Play
- Console

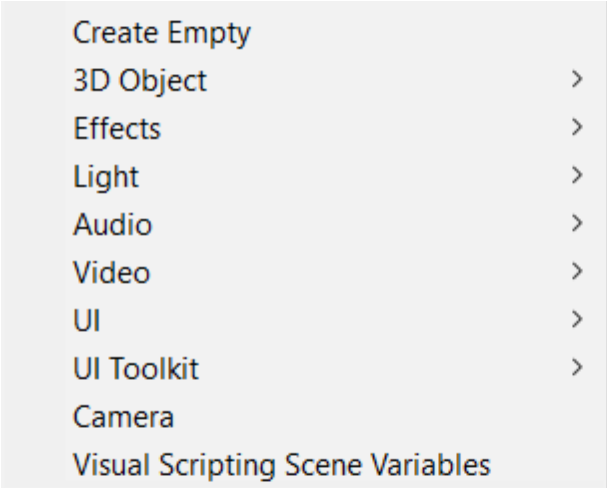
### Hierarchy Menu

The Hierarchy window in Unity can be found on the left side of the Unity editor and contains all of the scenes and GameObjects that make up your project. GameObjects are organized into a

tree structure, with parent GameObjects listed above their children. This feature makes it easy to create, delete, and rename GameObjects, as well as to change their hierarchy and parenting relationships.



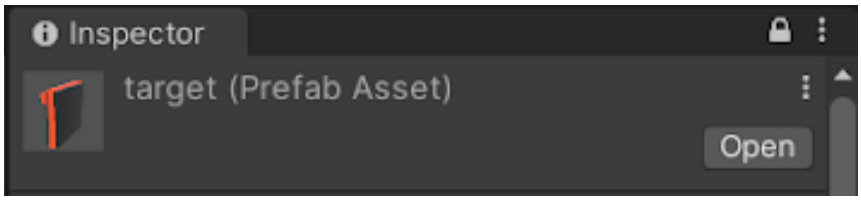

In the picture above you can see that “target” is a child object of Spawner. By right clicking in the empty space you can add a new scene or GameObject. Types of GameObjects vary greatly as shown in the picture below. They can be used to create in-game objects, lighting, effects, cameras and more. If you just want to write a script for your game you can simply create an empty GameObject and attach a new C# asset to the object (more on that later).



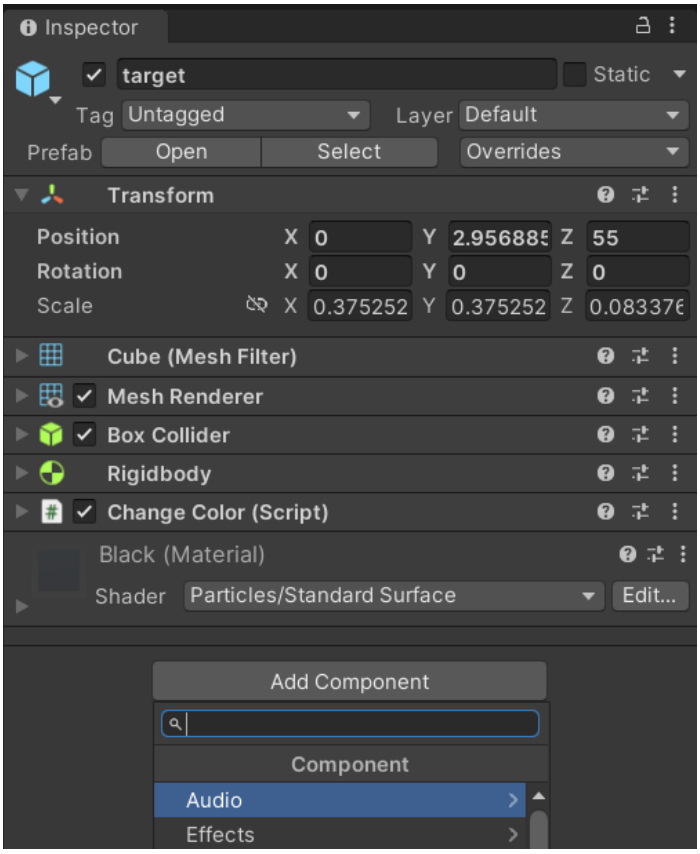

**Inspector**

The Inspector window can be found on the right side of the Unity editor whenever you select a GameObject from the Hierarchy window or an Asset. Inside you will see all of the properties and components of that object or asset. This is where you can manipulate the position, rotation, scale, appearance, and other properties of in-game objects. Additionally this is where you can

add scripts, audio sources, colliders and other components (used to manipulate the appearance and behavior of a GameObject) to that object.




Click the lock icon in the top right of the Inspector window to allow yourself to drag assets into the components section of the inspector window relating to a GameObject that you have selected in the hierarchy field on the left of the screen. Alternatively, you can scroll to the bottom of the Inspector window and click “Add Component.” This will pull up a menu from which you can search for your desired component.

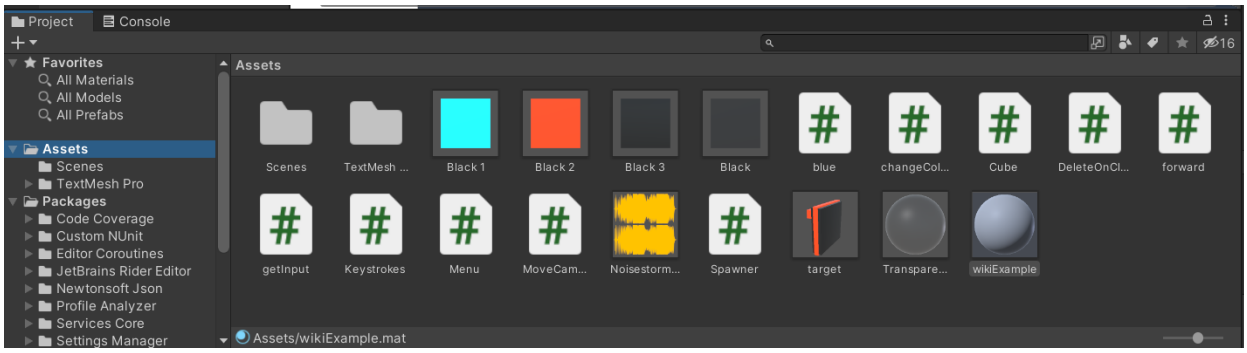



You can click the check box on the left of certain components to toggle them on or off. You can also click the arrow to the left of each component to see and modify all parameters related to that component (e.g. in the picture above the Transform component is opened, allowing us to easily modify the position, rotation and scale of the selected GameObject).

**Assets**

The Assets window can be found at the bottom of the Unity editor by selecting **Project > Assets** as shown in the picture below. Inside this window you will see all of the assets and files in your project. This includes textures, audio files, models, scripts, and prefabs. From this window you

can easily organize your project assets with folders as well as create, delete and rename assets.




## Layers

Layers are used to group GameObjects, allowing you to manipulate the behaviors of a group of objects all at once. This could be used to control which objects are visible and which objects will interact with Unity's physics engine. Each object in the scene can be assigned to a specific layer using the following steps:

1. Select the object you want to assign to a layer.
2. In the Inspector window, find the Layer drop-down menu at the top.
3. Click on the desired layer in the drop-down menu.

Alternatively, you could use the following code (in C#) to assign a GameObject to a specific layer:

*gameObject.layer = LayerMask.NameToLayer("Name of Layer");*

To implement this you would need to include the script asset as a component of the appropriate GameObject and include this line in the void Start() function and ensure you have "*Using UnityEngine;*" at the stop of your script.

To add a new layer to your game within the Unity editor, navigate to Edit in the top left part of the screen and select: Project Settings > Tags and Layers. There you will see a list of built-in layers as well as the option of adding your own.

Here is an example of how you might make use of this feature in your game. Let's say you want your player to be able to pass through certain objects but not others. To do this, create a custom layer for the objects that have hard boundaries. For each GameObject in this class, assign them to the new layer as previously discussed (you can do this quickly for multiple objects using the CTRL + click shortcut). Lastly you will need to add a script to the player object and make sure that collision is only set for objects in this new layer. In Unity 3d, one way to accomplish this would be to use the UnityEngine Raycasting. Here is an example script that checks if the camera is pointed at an object in the layer.

```
using UnityEngine;

public class WeaponController : MonoBehaviour
{
    public LayerMask targetLayer;

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            // Fire a ray from the camera's position in the direction of the mouse
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;

            // Check if the ray hit an object in the "Target" layer
            if (Physics.Raycast(ray, out hit, 100f, targetLayer))
            {
                // Do something when the object is hit
                Debug.Log("Hit object in target layer");
            }
        }
    }
}
```

Keep in mind that this feature could be helpful for a wide range of use cases.

**Scenes**

Scenes can be thought of as different levels or locations within the game. As such, a Unity project can contain multiple scenes; each scene will have its own separate file containing the objects, assets, and other files that belong to that scene. It is common practice to have a "cut-scene" or "loading screen" to make the transition between different scenes seem fluid. To add a new scene to your Unity project, navigate to File in the upper-left corner of the editor and select "New Scene" from the drop-down menu. You can then rename the scene and save it to the Assets folder of your project by clicking File > Save Scene, or by using the shortcut CTRL + s after selecting the scene in the Hierarchy window. You can also add new objects to this scene as described previously.

To transition from one scene to another you can use the SceneManger class built-in to the UnityEngine. To do so, follow these steps:

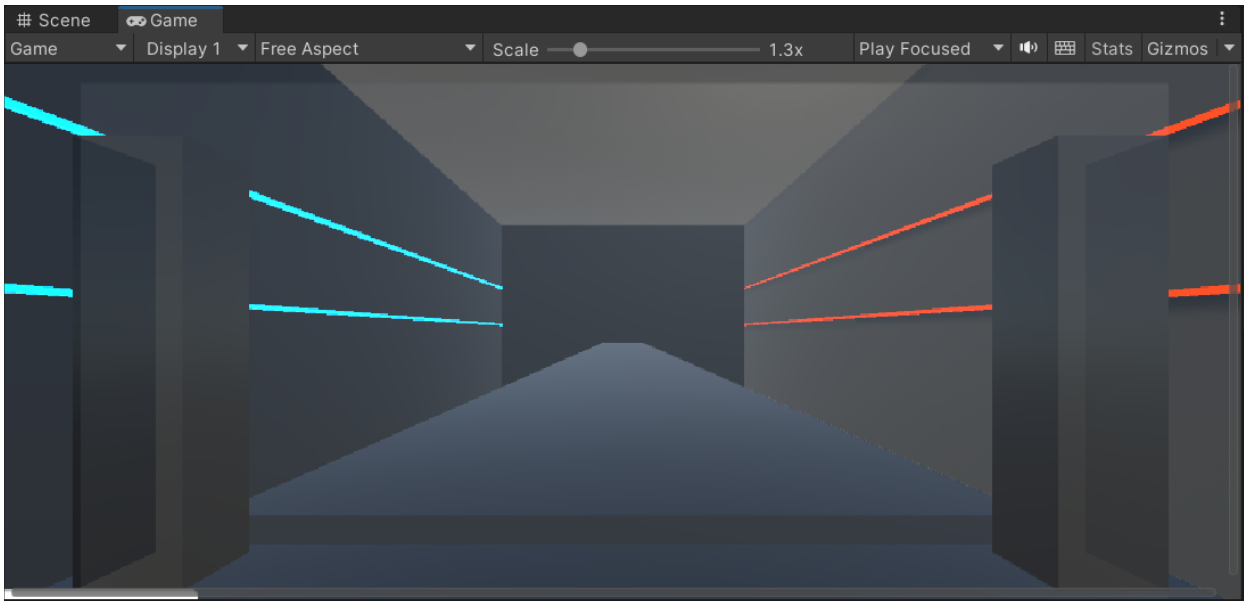
1. Navigate to File > Build Settings.
2. Click "Add Open Scenes" to add all of the scenes in your project to the build list.
3. Specify the starting scene by clicking on it in the build list and then clicking "Set as Start Scene."
   Note: You can also specify the starting scene in a script using:
   *using UnityEngine.SceneManagement;* at the top of your script, and:
   *SceneManager.LoadScene("MainScene");* in void Start()
4. Next, to transition from one scene to another, simply use the LoadScene function again but with the name of the scene you want to load, like so:
   *SceneManager.LoadScene("SceneName");*

Incorporating multiple scenes into your game will add depth, novelty, immersion, and expansiveness that is hard to capture with a single scene.  Beyond that, it will help you organize your game into subsections, making it easier to develop and debug, as each segment will have fewer elements and be less computationally demanding to run.

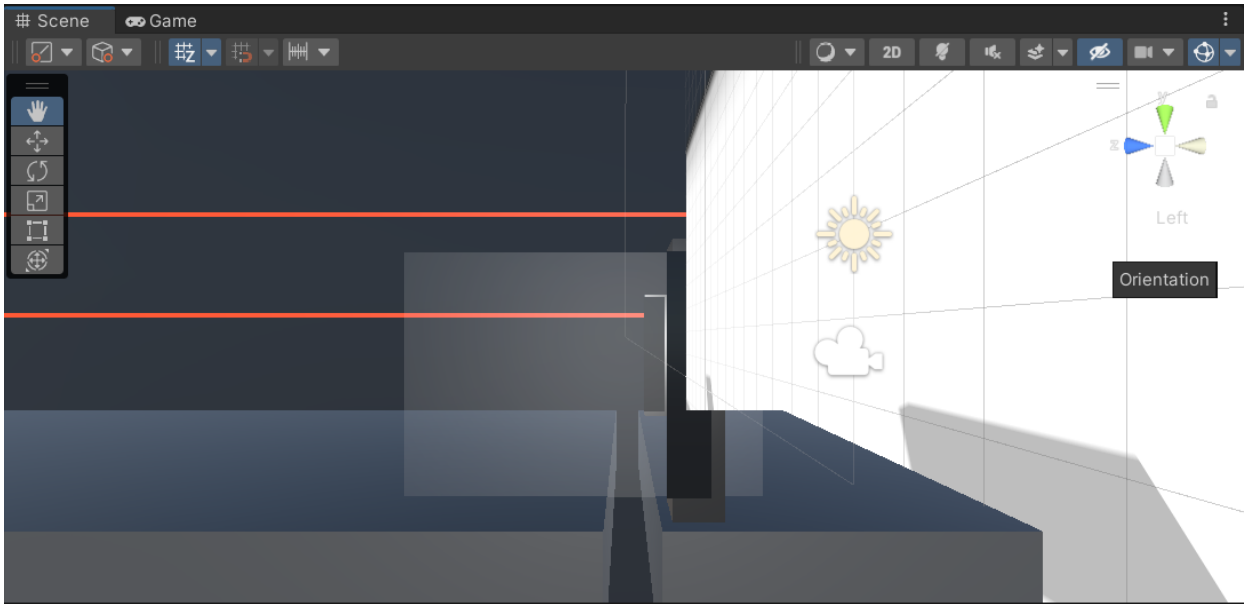
**Scene and Game Window**

The Scene and Game displays are used to create and test your game, respectively.  You can toggle between the two views by selecting the desired one in the top left of the window as shown in the picture below.

Game View



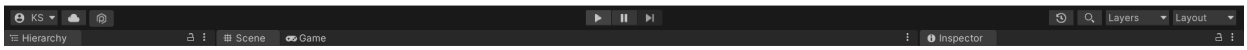

Scene View



The scene view is used to create your player environment and is where most of the game development should take place (in conjunction with coding and play testing). You can change the point of view by altering the orientation in the upper-right. To interact with the scene (and objects within it), use the tools on the left side of the Scene window. These tools are used primarily to select objects and alter their position, rotation, and scale (size).
Note: All of these alterations can be achieved using the Transform tool at the bottom. Hover your cursor over one of the tools to see what type of tool it is; click on it to begin using it.

**Pause and Play**

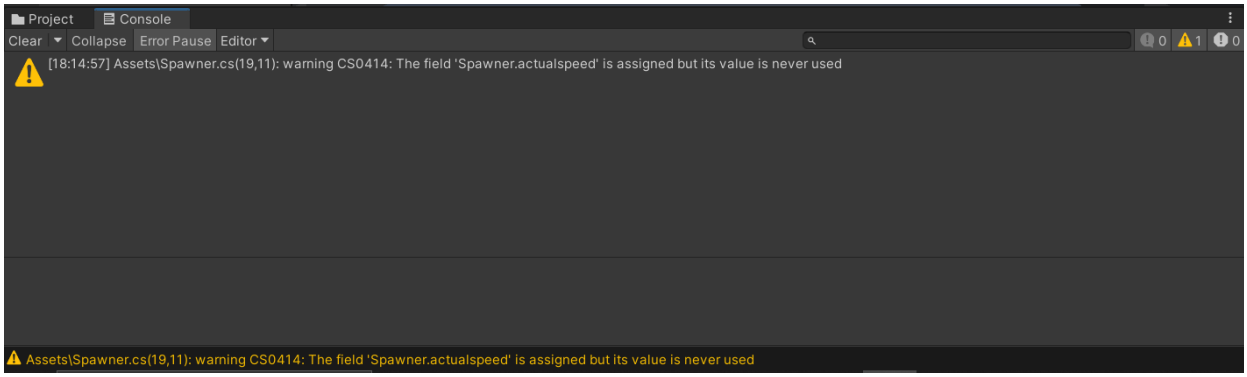
To start your game, click the play button at the top-center part of the screen. You will know the game is running if the play button is highlighted in blue. Pressing it again will stop the game. To pause or unpause the game click the pause button to the right of the play button. The button to the right of the pause button will make the game step forward by one frame (if the game is already running and paused). This can be especially useful for evaluating what is happening inside of your game on small time scales.




Note: When you press play the Scene/Game window will automatically switch to the Game window. If you want to see what is happening from a different point of view you can switch back to the Scene view while the game is running.

**Console**

The Console can be seen by selecting Console (instead of Project) at the bottom of the Unity editor (top left of the picture below).



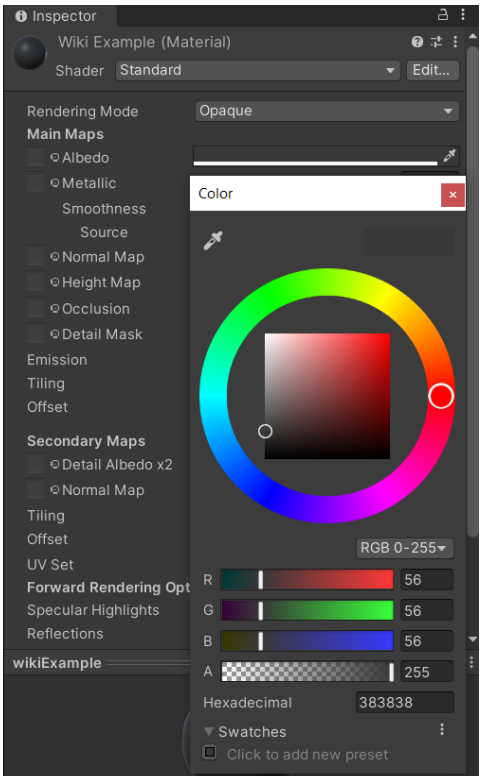

This window is highly useful for debugging your scripts. To write to the console in your code use:
> **Debug.Log("Your message goes here.");**
In the above picture you can see the warning: "...Assets\Spawner.cs(19,11): …". This gives you the particular script, line (19), and character index (11), to which the warning refers.

## 3. Objects and Components

This section will cover the basics of creating objects, materials, and prefabs. It will also show you how to create an environment for the game to exist in. To demonstrate, here is an example in which I create a "target" prefab (an instance of which is shown in the picture below) that the player(s) will interact with.

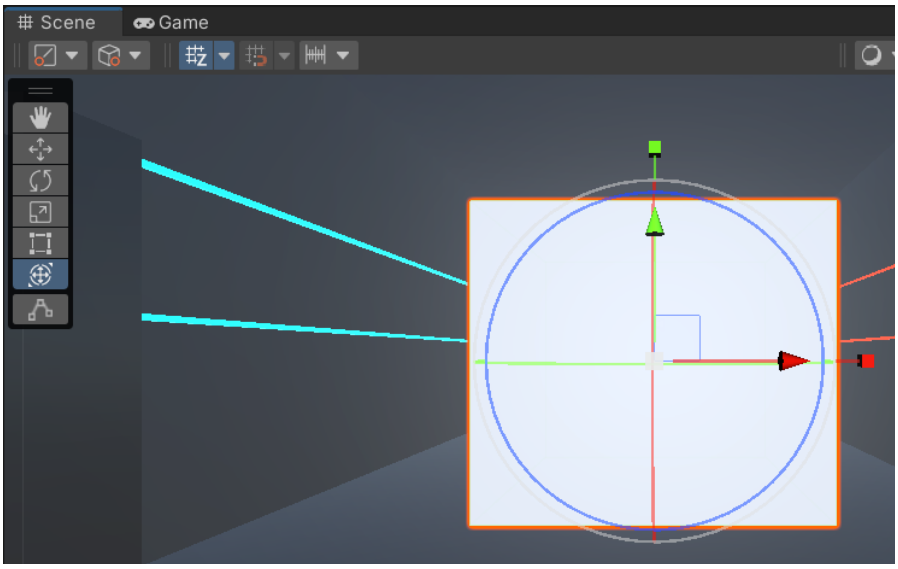
**Prefab Example**

1. Right click in the Hierarchy window > 3D Object > Cube
2. In the Inspector window change the x, y and z scale to 0.4 in the Transform component.
3. Right click in the Assets window > Create > Material
4. Select the Material in the Assets window > In the Inspector window, find Main Maps > Albedo
5. Click on the white color to change the Material's color as shown below:




6. Create a second material but with the color orange instead of gray.
7. Right click on the GameObject > 3D Object > Cube.
   This will create a child GameObject of the original cube. Repeat so that you have three total cubes (two of which are child objects of the original).
8. Select each child cube and, from the Assets window, drag the orange material into the Inspector window.
9. Using either the Transform component within the Inspector window or the Transform Tool in the Scene view, resize the two new objects such that one creates a border around the original cube and the other gives the object a distinguishable orientation. To resize each

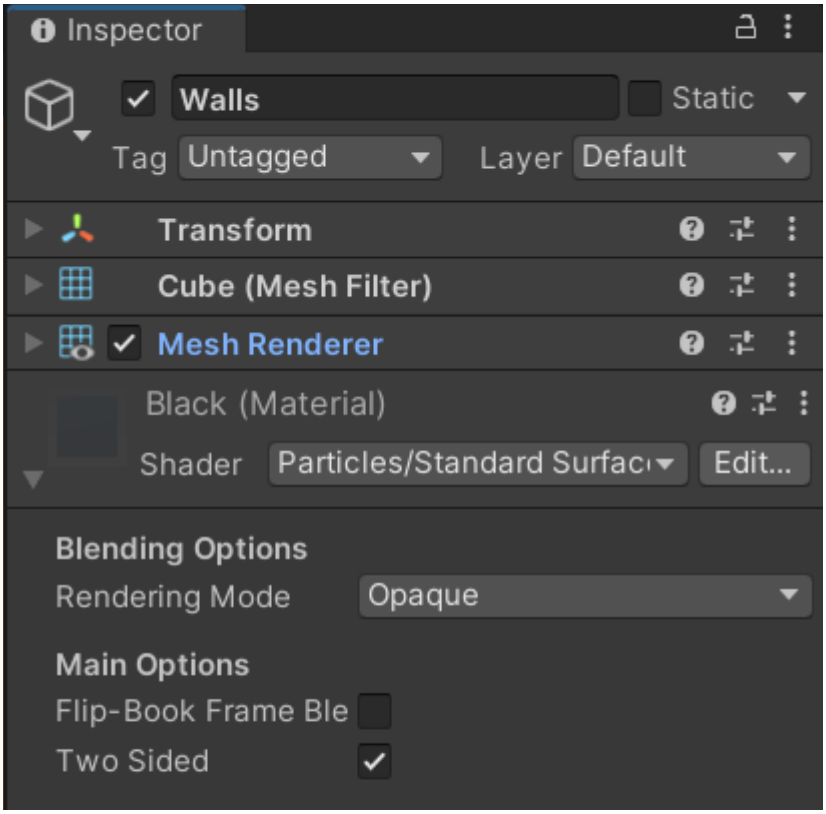
object inside the Scene view, simply select the object from the Hierarchy window or, using the View tool in the Scene view, click on the object. The Scene view is covered more extensively in a later section, but for now, you can select Scene view tools from the menu shown on the left in the picture below. If you are unsure which tool is which, simply hover your mouse over one of them and the name of the tool will be displayed. Next, select the Transform tool and click and drag to resize. If you click and drag on one of the (red or green) arrows you can translate the position of the object; if you click and drag on one of the (red or green) squares you can resize the object along the corresponding axis.




10. Once you are satisfied with the size and appearance of your target object, click and drag the original GameObject from the Hierarchy window into the Assets window. This will create a prefab automatically that you can instantiate and manipulate with scripts later on.
11. Unless you want to create a variant of your prefab object, you are free to delete the GameObject from the Hierarchy window.
    Note: For more information on how to instantiate/spawn your newly created prefab inside your game, refer to the scripting section of this article. If you want different instances of your prefab object to have different orientations or positions, this can be modified in your code later; you do not need to create a copy of your prefab with a different starting rotation/position.

**Player Environment**

To create a box for the player to exist in, add a new GameObject cube to your scene. Navigate to the Inspector window and increase the scale of the cube (units are in meters). Next, add a material to the cube and select Particles/Standard Surface as the shader. Finally, check the "Two Sided" box under "Main Options." This will effectively hollow the cube so that there is space inside of it. This space will serve as your player environment. The Inspector window can be seen below.




## 4. Lighting

This section will cover the basics of creating lighting objects within your game environment. To set up your light source follow the steps below.

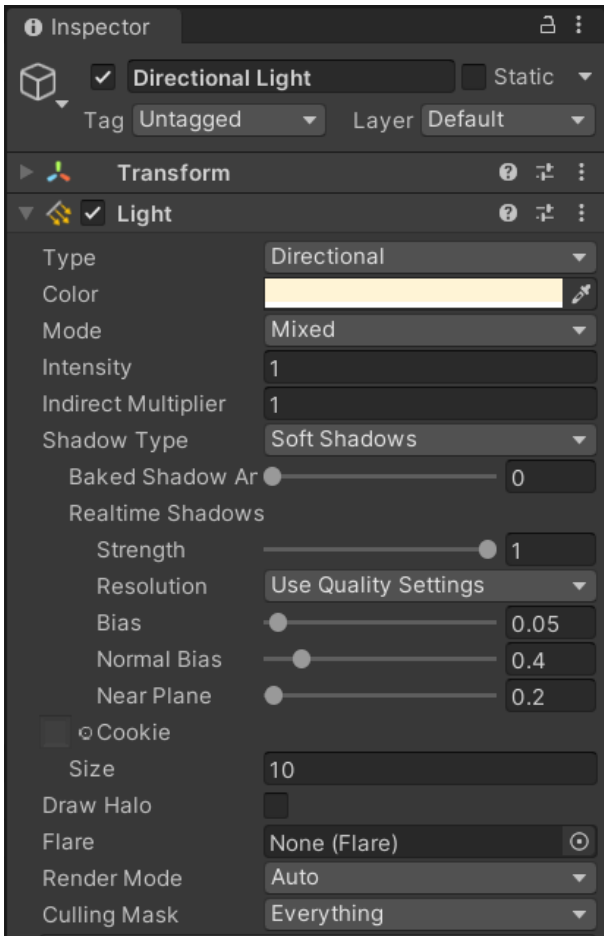
1. Right click in the Hierarchy window > Light > (choose your lighting type). The lighting types are as shown:


Here is a short description of the different lighting types:

1. Directional Light: This type emits light in a specific direction, simulating a distant light source like the sun.
2. Point Light: This type emits light in all directions from a single point, which is useful for lighting a scene.
3. Spotlight: This type emits light in a cone shape from a single point.
4. Area Light: This type emits light from a rectangle, simulating a large light source like a window or a projector.

5. Reflection Probe: These are objects which act like mirrors. This can also be useful for replicating reflective surfaces such as puddles.
6. Light Probe Group: This is a feature that allows you to effectively produce indirect lighting by capturing other scattered light.

There are a number of parameters you can alter in the Inspector window as shown in the picture below. Most notable are Intensity and color.




## 5. Music

This section will show you how you can add music to your game by following the steps below.

1. First you will need to add your song to the Assets folder. This can be done by simply dragging and dropping your (lossy or lossless) audio file from your Windows folder into the Asset window.
2. Right click in the Hierarchy window > Audio > Audio Source
3. Select your Audio Source GameObject and drag your audio file into the Inspector window
4. Now, as long as the Audio Source is active, the song will begin playing whenever you start the game.

Note: To trigger the start of the song with an in-game event, you will need to set up a trigger. Refer to the Trigger section of this article (here) to learn more about triggers.

## 6. Sprites

This section will teach you to display images on the screen in your game.

1. First you will need to drag the media (as an example we will use a .png file) from your Windows folder to the Assets window in the Unity editor.
2. Next you will need to create an empty GameObject in the Hierarchy window.
3. Give the new GameObject a name like “Image Display.”
4. Select the GameObject and navigate to the Inspector window.
5. Click “Add Component” and search for “Sprite Renderer” and add it as a component.

*Further explanation will be added to this section of the article in the future.*

## 7. Scripting

This section will cover the basics of programming within Unity. To demonstrate how we can incorporate scripts into our game, this section will also include an example script that spawns instances of the prefab we created earlier in this tutorial.

To create a new script, **right click in the Assets window > Create > C# Script**

In most cases, your script should take the following format:

wikiExample.cs

Miscellaneous Files — wikiExample

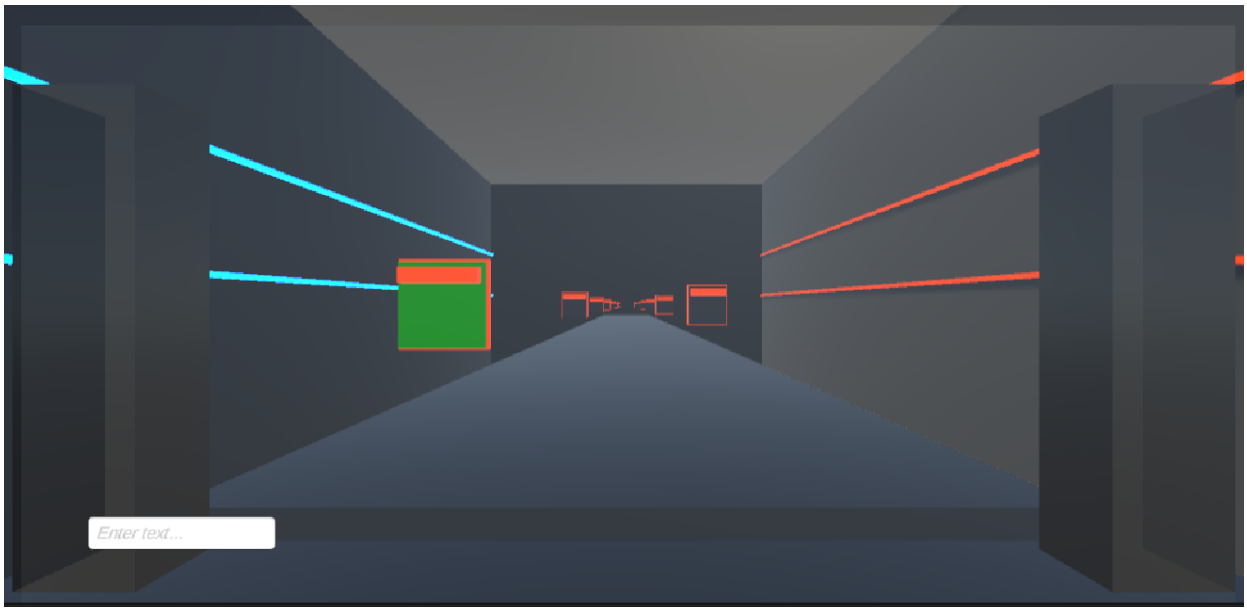
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class wikiExample : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

To apply your script to an object, drag it from the Assets window into the Inspector window after selecting the appropriate GameObject from the Hierarchy window. As soon as the code

successfully compiles (automatically), any assets referred to in the script will appear in the Inspector window.

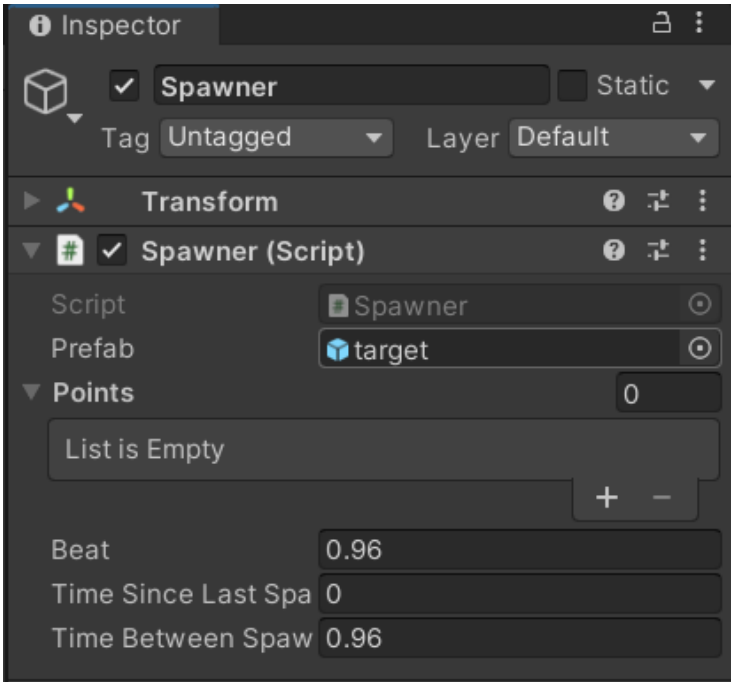
Click here for an example of a script that spawns targets and moves them towards the player camera. The script's filename is "Spawner.cs". As you can see in the image below, the prefab objects we created earlier spawn at one of two starting positions and move towards the player. The reason the closest target is highlighted green is to indicate to the player that they should destroy the object with their controller. This change in color was accomplished using a collision trigger (more on that later).

Target Spawning Script




To implement this script follow the following steps:

1. Add an empty GameObject in the Hierarchy window
2. Add the script asset to the GameObject by dragging it into the Inspector window
3. Drag the prefab from the Assets window into the Spawner section of the Inspector window as shown:




## 8. Rendering

This section will discuss the basics of rendering your final game by building a single executable file that can run on various operating systems (OS). To create an executable file follow these steps:

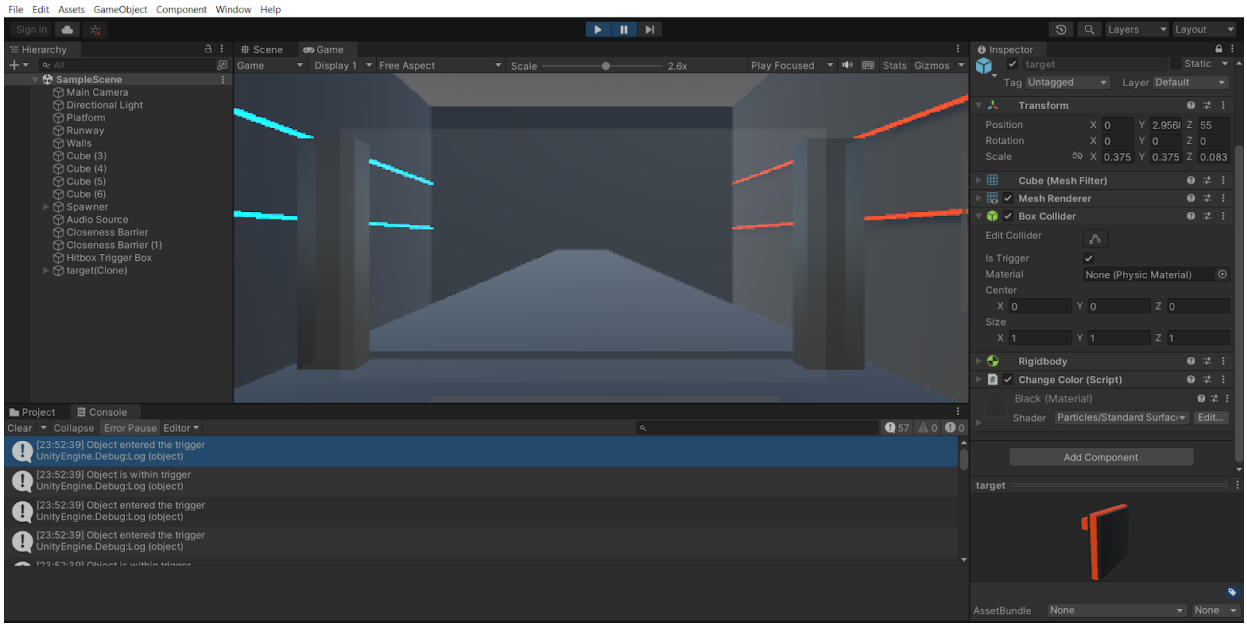
1. Navigate to File > Build Settings
2. Select the platform (OS) you want to build for by clicking on it in the platform list.
3. Verify that your scenes have been added to the build list.
4. Configure any build setting requirements for your specific platform.
5. Click the “Build” button to start the build process.

Unity will then create a standalone executable file for your game along with any additional assets that are required for it to run.

## 9. Collision Triggers

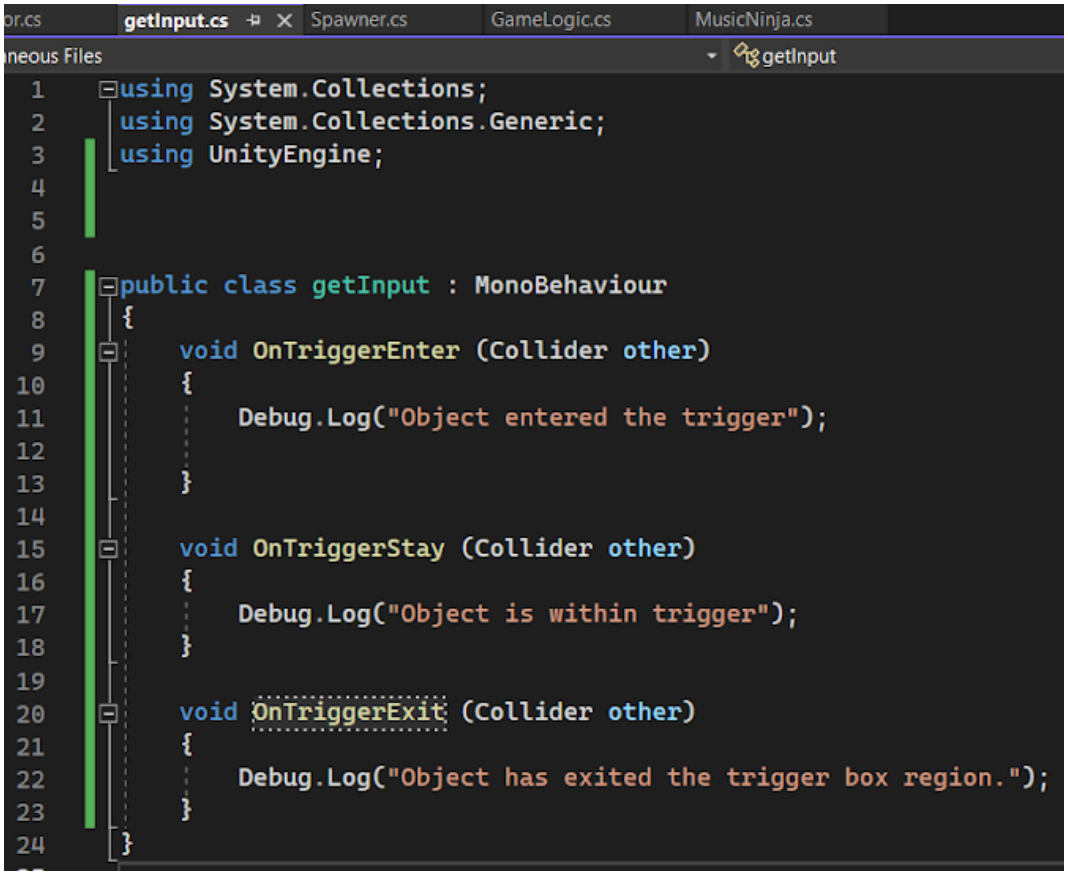
A trigger is basically just a way of making one event trigger another. In our example we will use the passing of a moving target through a region to trigger the object to change color. This will be useful for signaling to the player when they should hit the target.

A common way to use a trigger is to **add a box collider component to an object and clicking the “Is Trigger” box as shown** in the picture below. We will also want to **add a script component to this trigger object** to see if other objects pass through it. For another object to trigger when passing through the region, there are three more requirements:
**(i) At least one of the two objects must have a Rigidbody component.**
**(ii) Both objects must have a box collider component.**
**(iii) Only the trigger box should have Is Trigger checked.**




Is it standard practice to make our box collider static in the inspector window so that it will not interact physically with other objects. This can be done by clicking the Static box in the upper-right-hand corner of the Inspector window after selecting our trigger object from the hierarchy menu on the left-side of the Unity editor.

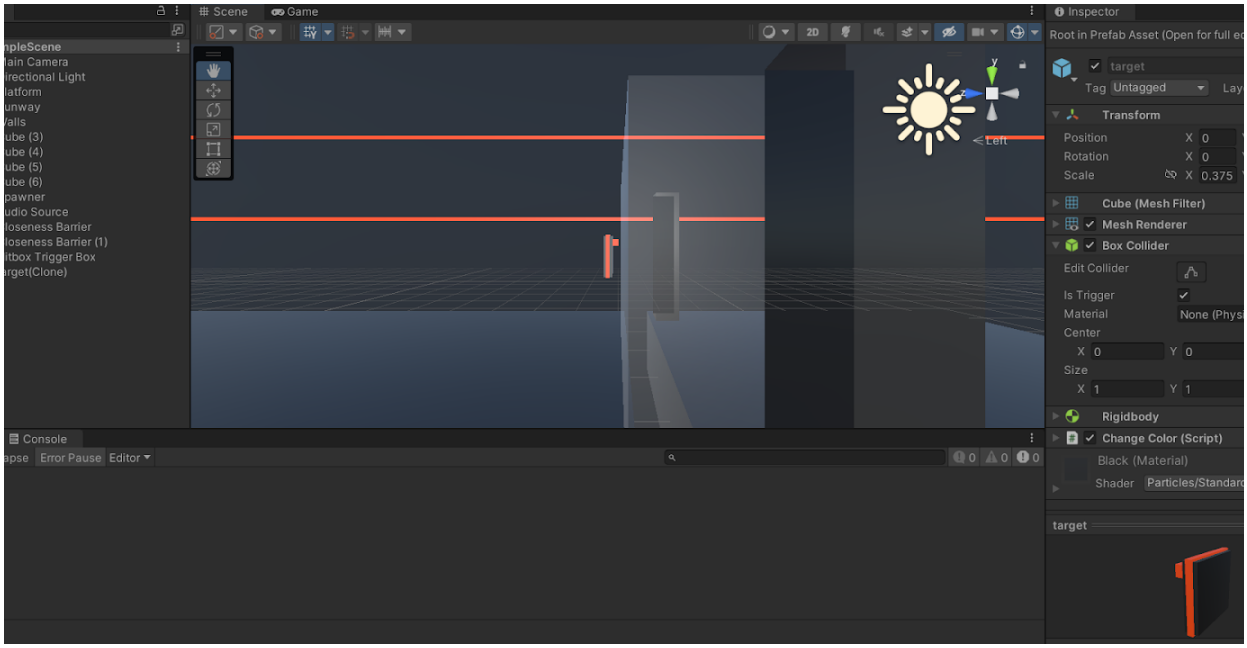
To use this trigger we use the following three built-in “OnTrigger” functions:

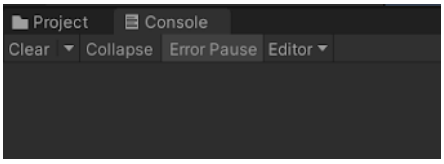



Note: Other collider’s such as Mesh collider work as well but the Box Collider is the least computationally demanding.

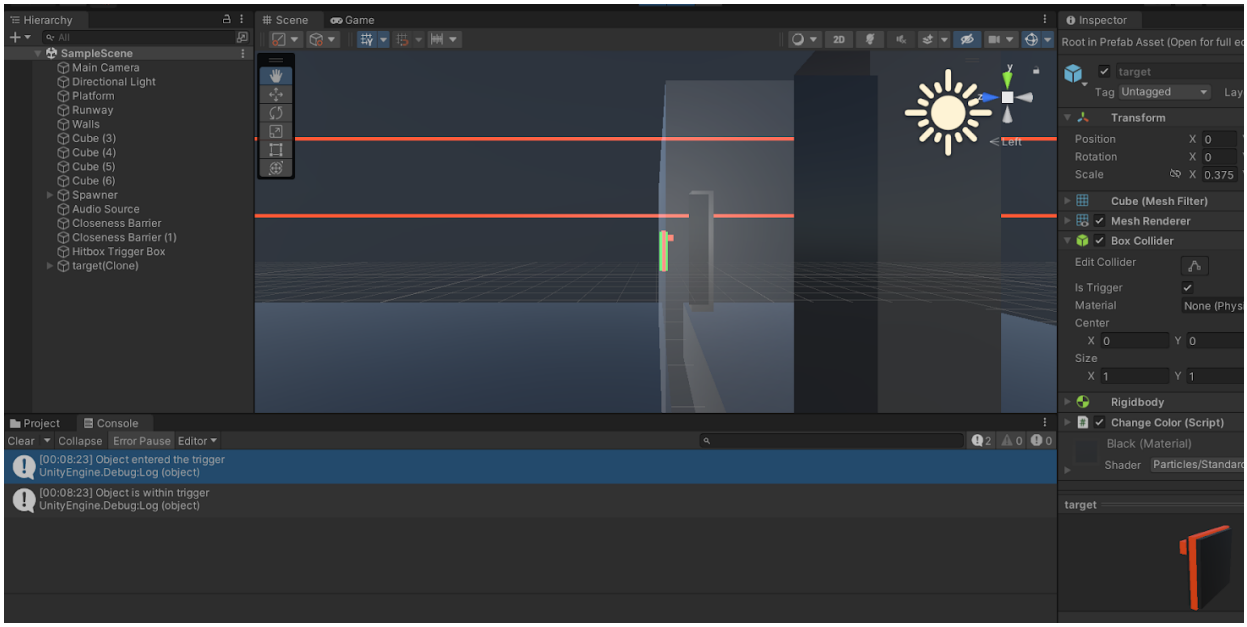
**Example of collision-event triggering:**

In the picture below you can see a target object moving towards the trigger box in gray. Notice the Console output at the bottom is currently empty.

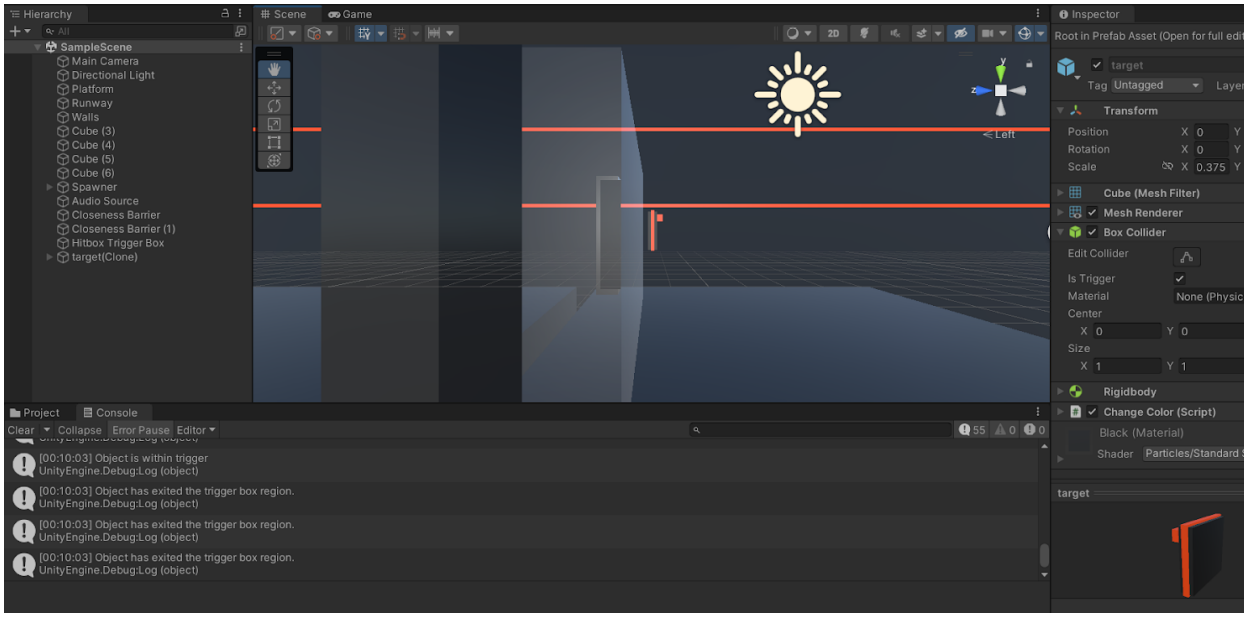





Now I will step forward a couple of frames using the play button to the right of the pause button at the top of the editor.  You can see that the target has turned green, due to a trigger script component attached to the target prefab.  Also notice that the console has recognized that the object has entered the triggering region due to a script attached to the trigger object itself.




When I step forward further until the target exits the triggering region, notice that the color is set back to black and the console recognizes that the target has exited.




To achieve the color change I added this script to the target prefab:

```cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class changeColor : MonoBehaviour
{
    private Renderer rend;

    void Start()
    {
        rend = GetComponent<Renderer>();
    }

    void OnTriggerEnter(Collider other)
    {
        rend.material.color = Color.green;
    }

    void OnTriggerExit(Collider other)
    {
        rend.material.color = Color.black;
    }
}
```

Triggers are a useful way to introduce dynamic events within your game to help create a more interactive experience for the players, keeping them engaged and immersed in the game experience.

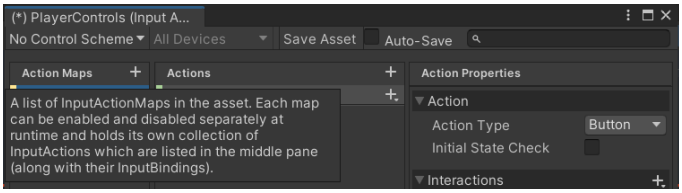
## 10. User-Input

This section will show some basic ways in which we can get user-generated data at runtime to allow players to interact with our game. Some methods include:

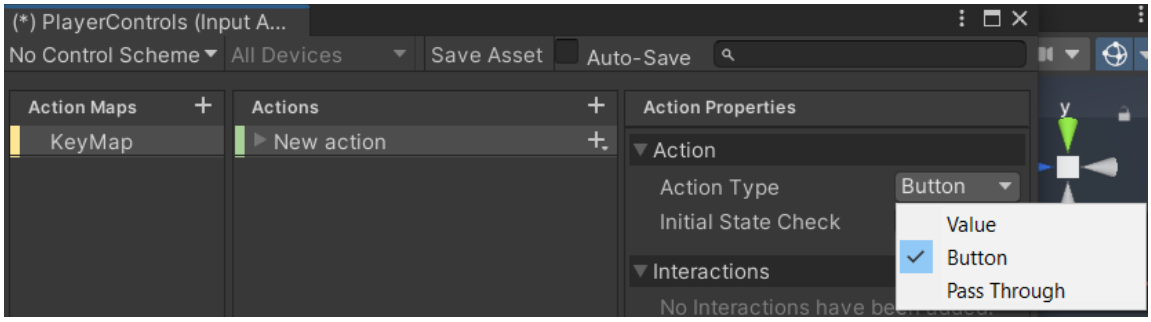
- Keystrokes
- Mouse Clicks
- Speech Recognition

Before we can implement any of these methods we must first install the new Unity input system and then do some setup.  Follow the steps outlined below.

1. Navigate to: Window > Package Manager > Packages: In Project > Unity Registry
2. Scroll down until you see “Input System.”
3. Click Input System and then click Install in the bottom right.
4. Save your project and restart the Unity editor.
5. Right click in the Assets window and select Create > Input Actions
6. Give the asset a name like “Player Controls.”
7. Double-click on Player Controls in the Assets window and add a new Action Map in the upper-left corner of the PlayerControls window by clicking the “+” icon beside Action Maps and giving it a name like “KeyMap” or “PlayerBindings” as shown.




8. Add an action by clicking the “+” icon beside Actions and give it a name like TapAction. An Action Map is a set of controls whereas Actions are the controls themselves.  Note: An Action can have multiple key bindings.
9. Select the appropriate Action Type from the drop-down menu shown below.




*Further explanation will be added to this article in the future.*

## 6.1 | Reading Keystrokes

Once you have completed the setup part of this section you can now create a script that will read user-inputs.

*Further explanation will be added to this article in the future.*

## 6.2 | Mouse Clicks

In this section we will give an example of how user clicks can be used to delete objects within the game.

1. Create an empty GameObject in your scene. You can do this by right-clicking in the Hierarchy panel and selecting "Create Empty".
2. Rename the empty GameObject to something descriptive, such as "DeleteTarget".

3. Attach a Collider component to the "DeleteTarget" GameObject. This will allow it to detect mouse clicks on it. To do this, select the "DeleteTarget" GameObject in the Hierarchy panel and go to the Inspector panel. Under "Add Component", search for "Box Collider" and add it to the object.
4. Create a new C# script and attach it to the "DeleteTarget" GameObject. To do this, right-click in the Assets panel, select "Create" > "C# Script", name it "DeleteOnClick", and drag it onto the "DeleteTarget" GameObject in the Hierarchy panel.
5. Open the "DeleteOnClick" script in Visual Studio or your preferred code editor and paste the code I provided in the previous answer.
6. Save the script and return to Unity. The "DeleteTarget" GameObject should now be deletable when clicked with the mouse.
7. If you want to delete multiple objects, you can attach the script to each one or use a loop in the script to check for mouse clicks on all objects with a certain tag.

*Further explanation will be added to this section of the article in the future.*

**6.3 | Speech Recognition**

Third party api's are recommended.

*Further explanation will be added to this section of the article in the future.*

## 11. Resources


C# Basic Input and Output
Learn Unity Tutorials
Unity Scripting Documentation
How to play audio in Unity (with examples)
Speech recognition Tutorial for Unity3D 5
Mini Unity Tutorial - How To Use TRIGGERS To Do Things
Colliders as Triggers - Unity Official Tutorials
MAKING BEAT SABER IN 10 MIN - Unity Challenge
Make Object Transparent in Unity 2022! | 1 Minute Tutorial
How to Add Voice Recognition to Your Game - Unity Tutorial
How To Use Input.GetKey in Unity's New Input System | Unity Tutorial
Input System Interactions Explained | Press, Hold, Tap, SlowTap, MultiTap - Unity
Input System Action Types Explained | Value, Passthrough, Button - Unity