

Basic Game Development in Unity

Introduction:

Unity is a free, industry proven cross-platform game engine whose powerful 3d-modeling software makes game development simple. With its user-friendly editor, you can create professional looking games easily, regardless of your level of experience or genre of interest. This article will familiarize you with the Unity editor and teach you the basic skills you need to start making your own game. For more information on each section, refer to the appendix or references.

Table of Contents:

1. [Downloads](#)
2. [Unity Editor: UI](#)
3. [GameObjects and Components:](#)
4. [Lighting](#)
5. [Music](#)
6. [User-Input](#)
7. [Collisions](#)
8. [Rendering](#)
9. [References](#)
10. [Appendix](#)

1. Downloads

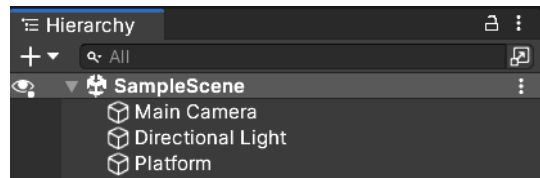
Download [Unity](#), [Microsoft Visual Studio](#) or [Atom](#), [libraries](#) for C# and [Winrar](#) to unzip files.

2. Unity Editor UI

This section covers the basic modules within the editor including: Hierarchy Menu, Inspector, Assets, Navigating the Scene and Game Window, Pause/Play, Console and more.

A. Hierarchy Menu

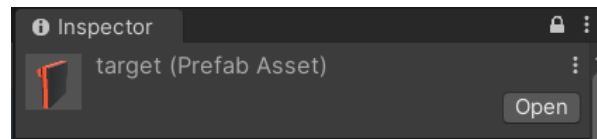
The [Hierarchy window](#) can be found on the left side of the Unity editor and contains all of the scenes and GameObjects that make up your project. GameObjects are organized into a tree structure, with parent GameObjects listed above their children. This feature makes it easy to create, delete, and rename GameObjects, or change their child-parent relationships.



Right click in the empty space to add a new scene or GameObject such as a light source, camera, or 3d shape. You can also create an Empty object and attach a script to it if you want to program some logic within the scene.

B. Inspector

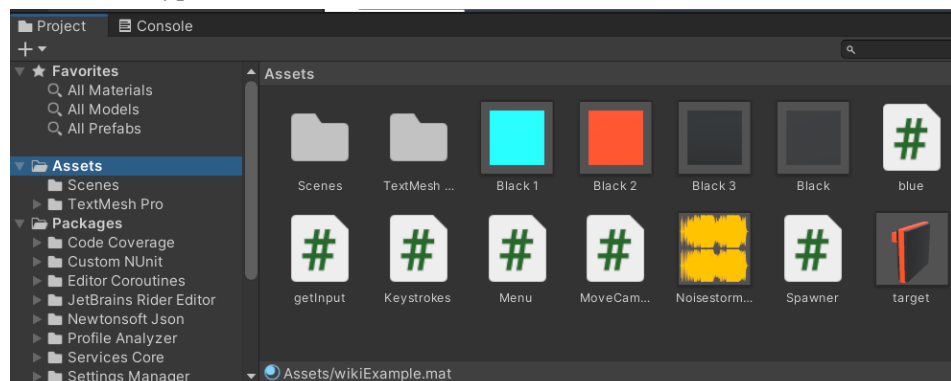
The Inspector window can be found on the right side of the Unity editor whenever you select a GameObject from the Hierarchy window or an Asset from your assets folder.



This is where you can manipulate various properties of - and add components to - your GameObject or asset. Components such as scripts and renderers can be used to manipulate the behavior or appearance of your GameObject.

C. Assets

Go to **Project > Assets** as shown to see all of the assets in your project including textures, audio files, models, scripts, and prefabs. To create a new material or other asset, right click in the empty space and select your desired asset type.



D. Scenes

Scenes are like different levels in a game. Each has its own set of GameObjects and assets. Use multiple scenes to add depth to your game and expedite its development.

To add a new scene to your project **click File > New Scene**. Rename the scene and save it to the Assets folder of your project by clicking **File > Save Scene**.

To transition from one scene to another you can use the **SceneManager** like so:

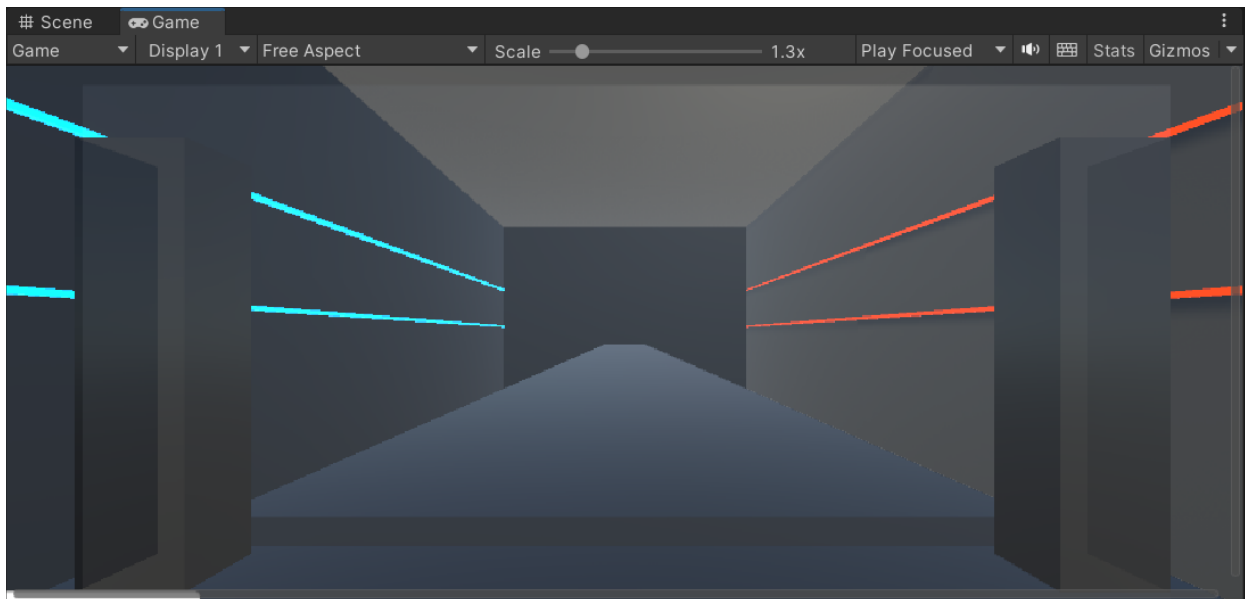
1. Navigate to **File > Build Settings**.
2. Click **“Add Open Scenes”** to add all of the scenes in your project to the build list.
3. Specify the starting scene by clicking on it in the build list and then clicking **“Set as Start Scene.”**
4. **Transition from one scene to the scene named “EndScreen” as shown:**

```
using UnityEngine.SceneManagement;
SceneManager.LoadScene(“EndScreen”);
```

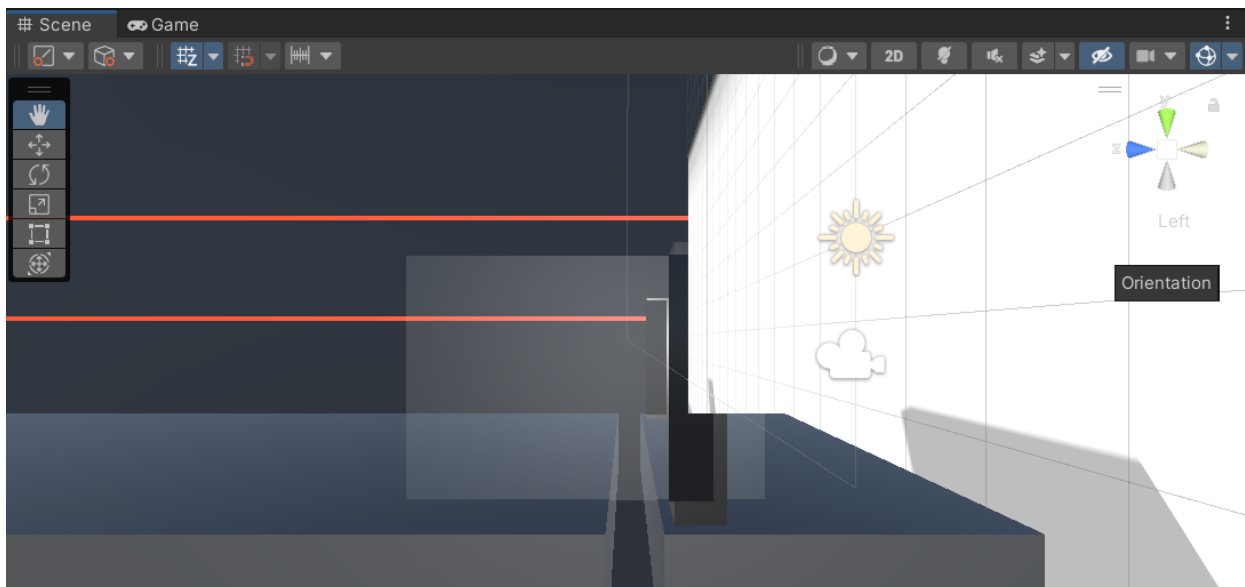
Scene and Game Window

The Scene and Game displays are used to develop and playtest your game. You can toggle between the two views in the top left of the window as shown below.

Game View



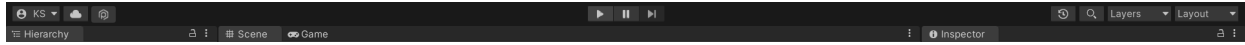
Scene View



The scene view is used to navigate your player environment and is where you can design prefabs and transform objects within your scene. Use the tools from the menu on the left to transform objects. To change your point of view use the orientation buttons in the upper-right or left click+drag with the *hand* tool.

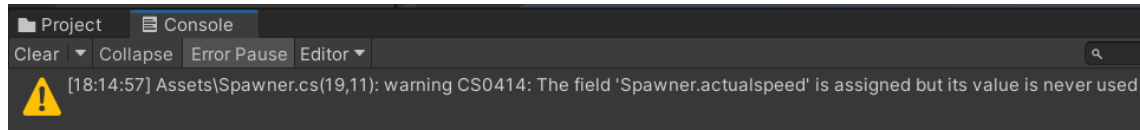
E. Pause and Play

To start your game, click the play button at the top-center part of the screen. 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).



F. Console

Select “Console” at the bottom of the Unity editor to see warnings, compiler errors and debug logs related to your scripts.



This window is highly useful for debugging purposes. 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. GameObjects and Components

This section will cover the basics of creating objects, materials, and prefabs with a demonstration.

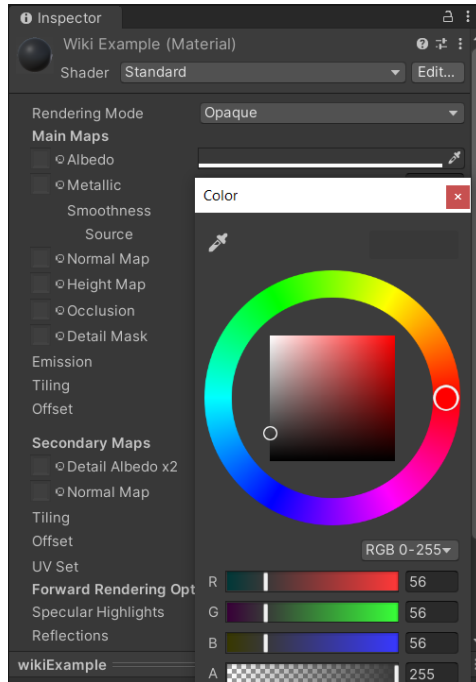
A. Prefabs

Prefabs are like copies of GameObjects that you can create in your asset folder. Below you can see an example of a square target prefab with an indicator at the top.

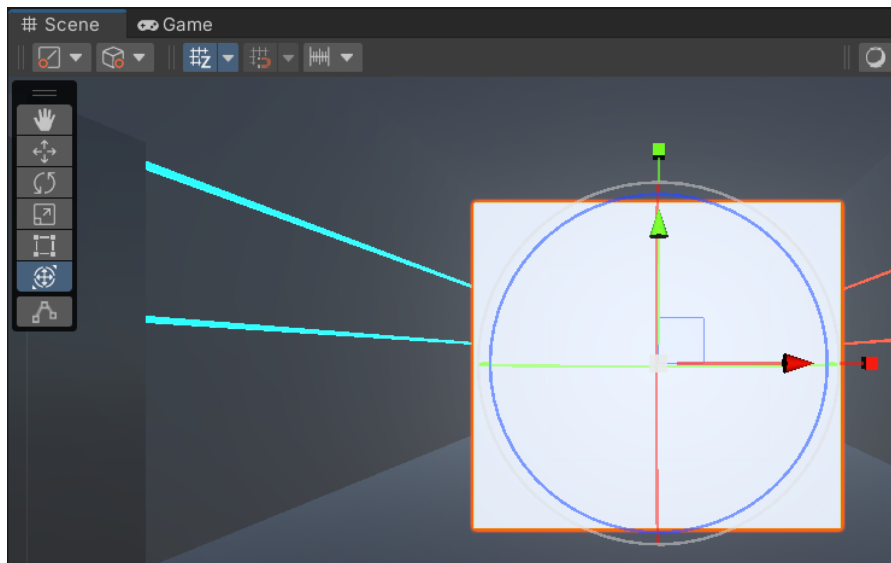


To create this:

1. Right click in the Hierarchy window and select **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 and select **Create > Material**
4. Select the Material in the Assets window and find Main Maps in the Inspector.
5. Find Albedo and click on the white color to change the Material’s color as shown:



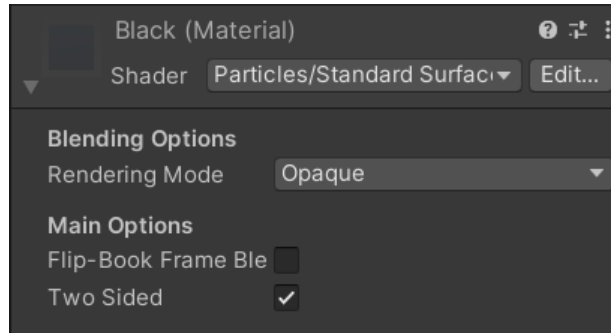
6. Create a second material but with the color orange instead of gray.
7. Right click on the GameObject > 3D Object > Cube.
This will create a child GameObject of the original cube. Repeat so that you have three total cubes (two of which are child objects of the original).
8. Select each child cube and drag the orange material into the Inspector window.
9. Resize the two new objects using the Transform component in the Inspector or the Transform Tool in the Scene view to recreate the above prefab. Click and drag on one of the (red or green) **arrows to translate the position** of the object or click and drag on one of the (red or green) **squares to resize the object**.



10. Click and drag the original GameObject from the Hierarchy window into the Assets window to create a prefab automatically.

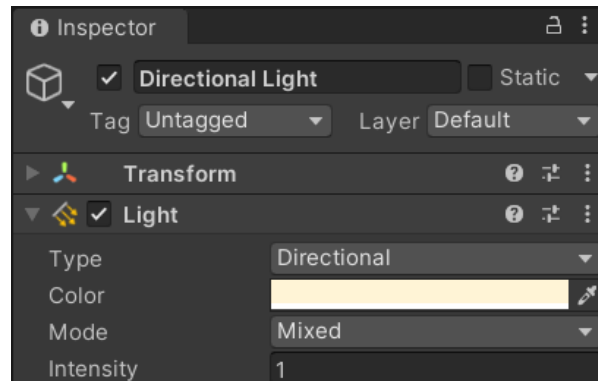
B. Player Environment

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. 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 to serve as your player space. The Inspector window can be seen below.



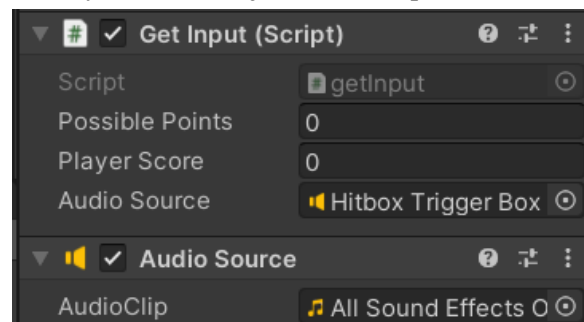
4. Lighting

To set up your light right click in the Hierarchy window and select “Light”. Select Point Light or other source type. You can alter intensity and color among other parameters in the Inspector window as shown:



5. Music

1. Add your audio sample to the Assets folder via drag and drop.
2. Right click and select **Audio > Audio Source** in the Hierarchy window.
3. Drag the Audio Source to your GameObject in the Inspector window.



To play the audio source use:

```
using System;
public AudioSource audioSource; // in your MonoBehaviour derived class
audioSource = GameObject.Find("AudioObject").GetComponent<AudioSource>(); // in void
Start() {}
audioSource.Play();
```

6. Reading 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

Install the new Unity input system like so:

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.**

Now you can check for a user’s keyboard inputs as shown in the script below.

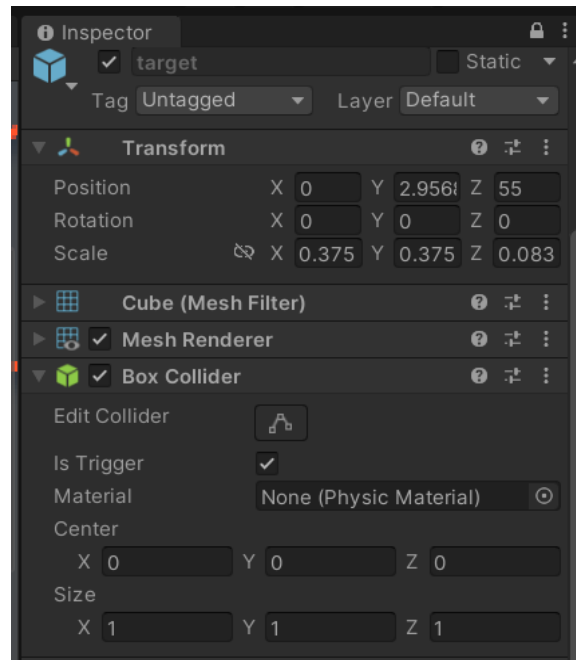
```
81
82   if (Input.GetKey(KeyCode.LeftArrow))
83   {
84       Debug.Log("You swiped left!");
85       swipeDirection = "left";
86   }
87   else if (Input.GetKey(KeyCode.RightArrow))
88   {
89       Debug.Log("You swiped right!");
90       swipeDirection = "right";
91   }
92   else if (Input.GetKey(KeyCode.UpArrow))
93   {
94       Debug.Log("You swiped up!");
95       swipeDirection = "up";
96   }
97   else if (Input.GetKey(KeyCode.DownArrow))
98   {
99       Debug.Log("You swiped down!");
100      swipeDirection = "down";
101   }
102
```

For more information on user-inputs refer to the Appendix or References.

7. Collisions

Triggers are a useful way to create an interactive experience for players. They allow us to make one event trigger another. In the following example I used the passing of a moving target through a collision region to change the object's color.

First, **add a box collider component to your GameObject and click the “Is Trigger” box. Add a script component to this trigger object** which will check if other objects pass through it.



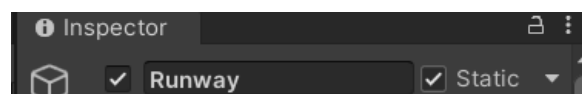
Note:

(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.

Make the box collider static in the inspector window so that it will not interact physically with other objects.



Use the following three built-in “OnTrigger” functions:

```
Void OnTriggerEnter (Collider other) { // you code goes here } [Runs once when the target enters]
```

```
Void OnTriggerStay (Collider other) { // you code goes here } [Runs once per frame while inside]
```

```
Void OnTriggerExit( Collider other) { // you code goes here } [Runs once upon target exiting]
```

See the Collisions section in the Appendix for an example.

8. Rendering

To create an executable file follow these steps:

1. Select the platform you want to build from the platform list under *File > Build Settings*.
2. Validate the build list.

3. Configure any build setting requirements for your specific platform and click Build.

9. References

[Unity Documentation](https://docs.unity3d.com/Manual/index.html): <https://docs.unity3d.com/Manual/index.html>

[Learn Unity Tutorials](https://learn.unity.com/tutorials): <https://learn.unity.com/tutorials>

[Unity Scripting Documentation](https://docs.unity3d.com/ScriptReference/): <https://docs.unity3d.com/ScriptReference/>

[How to play audio in Unity \(with examples\)](https://gamedevbeginner.com/how-to-play-audio-in-unity-with-examples/):

<https://gamedevbeginner.com/how-to-play-audio-in-unity-with-examples/>

[Splash](https://docs.unity3d.com/560/Documentation/Manual/class-PlayerSettingsSplashScreen.html): <https://docs.unity3d.com/560/Documentation/Manual/class-PlayerSettingsSplashScreen.html>

[C# Basic Input and Output](https://www.programiz.com/csharp-programming/basic-input-output#:~:text=In%20C%23%2C%20the%20simplest%20method,also%20included%20in%20Console%20class):

<https://www.programiz.com/csharp-programming/basic-input-output#:~:text=In%20C%23%2C%20the%20simplest%20method,also%20included%20in%20Console%20class>

[Speech recognition Tutorial for Unity3D 5](https://www.youtube.com/watch?v=HwT6QyOA80E): <https://www.youtube.com/watch?v=HwT6QyOA80E>

[Mini Unity Tutorial - How To Use TRIGGERS To Do Things](https://www.youtube.com/watch?v=CSn0hENOIT0):

<https://www.youtube.com/watch?v=CSn0hENOIT0>

[Colliders as Triggers - Unity Official Tutorials](https://www.youtube.com/watch?v=m0fjrQkaES4): <https://www.youtube.com/watch?v=m0fjrQkaES4>

[MAKING BEAT SABER IN 10 MIN - Unity Challenge](https://www.youtube.com/watch?v=gh4k0Q1P17E&list=PLT7FCiOIP0K0MvP9qpW-1tz3yqkOJFx2r&index=6):

<https://www.youtube.com/watch?v=gh4k0Q1P17E&list=PLT7FCiOIP0K0MvP9qpW-1tz3yqkOJFx2r&index=6>

[Make Object Transparent in Unity 2022! | 1 Minute Tutorial](https://www.youtube.com/watch?v=KIWPedIvwuw):

<https://www.youtube.com/watch?v=KIWPedIvwuw>

[How to Add Voice Recognition to Your Game - Unity Tutorial](https://www.youtube.com/watch?v=29vyEOgsW8s):

<https://www.youtube.com/watch?v=29vyEOgsW8s>

[How To Use Input.GetKey in Unity's New Input System | Unity Tutorial](https://www.youtube.com/watch?v=0z-6lxL_sS4):

https://www.youtube.com/watch?v=0z-6lxL_sS4

[Input System Interactions Explained | Press, Hold, Tap, SlowTap, MultiTap - Unity](https://www.youtube.com/watch?v=rMlcwtoui4I):

<https://www.youtube.com/watch?v=rMlcwtoui4I>

[Input System Action Types Explained | Value, Passthrough, Button - Unity](https://www.youtube.com/watch?v=DMUZfVSYJfs):

<https://www.youtube.com/watch?v=DMUZfVSYJfs>

10. Appendix

Downloads

Unity: <https://unity.com/download>

Microsoft Visual Studio: <https://visualstudio.microsoft.com/downloads/>

Atom: <https://sourceforge.net/projects/atom.mirror/>

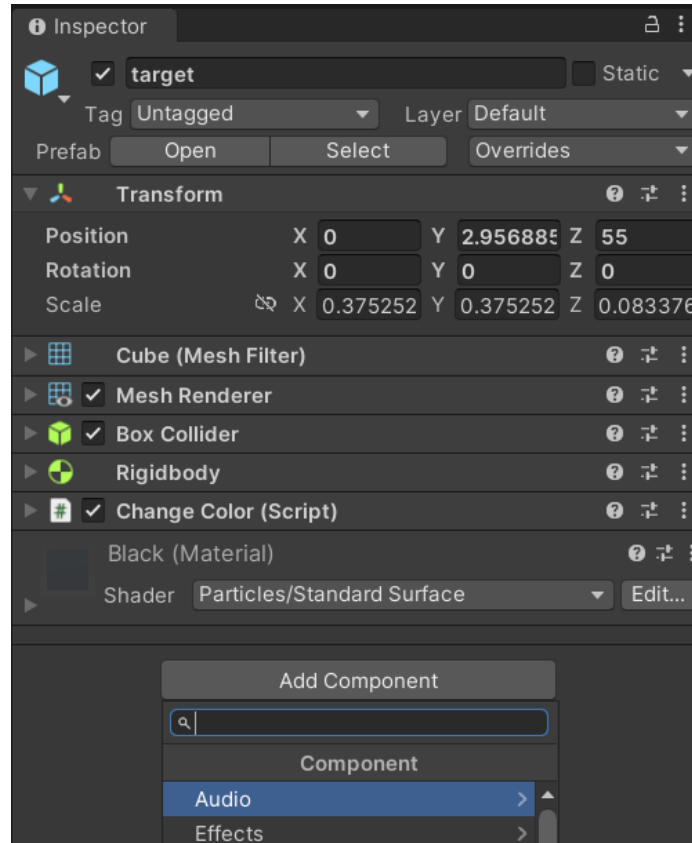
Libraries: <https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>

Winrar: <https://www.win-rar.com/download.html?&L=0>

Unity Editor UI

Inspector

To drag assets into the inspector window for a GameObject in the Hierarchy menu, select the GameObject and toggle the lock icon in the top right of the Inspector window. You can also click Add Component to search for a specific one.



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).

Scene View

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). 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.

You can access all of the scenes in your project under **Project > Scenes** near the Assets folder at the bottom-left of the editor.

Pause and Play

The game is running if the play button is highlighted in blue. Pressing it again will stop the game.

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.

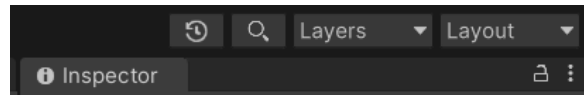
GameObjects

Duplicate GameObjects from the Hierarchy window to create variations, or delete them after you are satisfied with your prefab (you can always drag your prefab from the assets folder into the Hierarchy window to edit it later).

Layers

You can group GameObjects to manipulate the behaviors of different groups of objects. GameObjects in the scene can be assigned a specific layer using the following steps:

1. Select the object you want to assign a layer to.
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.

Note: This can also be done within a script like [this](#). To add a new layer to your game within the Unity editor find *Edit > Project Settings > Tags and Layers* and add a custom layer in the drop-down list.

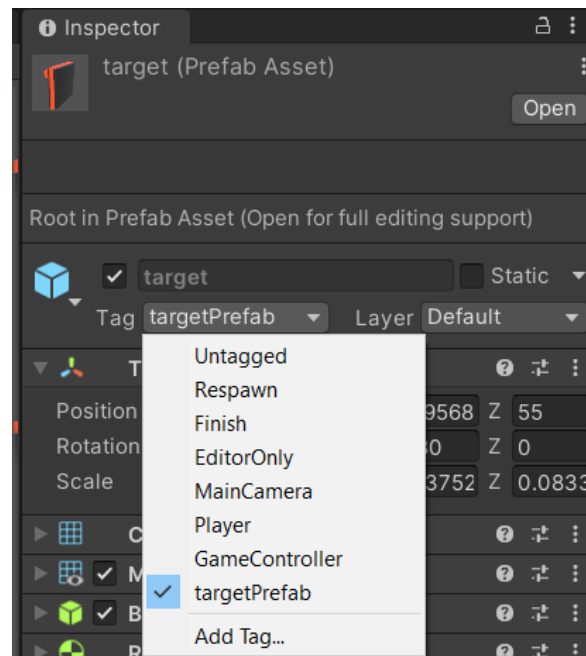
Let's say you want your player to be able to pass through certain objects but not others. Create a custom layer for all unique GameObjects that have hard boundaries ctrl+click. Lastly add a script to the player object and make sure that collision is only set for objects in this new layer. In Unity 3d, 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.

```
1 using UnityEngine;
2
3 public class WeaponController : MonoBehaviour
4 {
5     public LayerMask targetLayer;
6
7     void Update()
8     {
9         if (Input.GetMouseButtonDown(0))
10        {
11            // Fire a ray from the camera's position in the direction of the mouse
12            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
13            RaycastHit hit;
14
15            // Check if the ray hit an object in the "Target" layer
16            if (Physics.Raycast(ray, out hit, 100f, targetLayer))
17            {
18                // Do something when the object is hit
19                Debug.Log("Hit object in target layer");
20            }
21        }
22    }
23 }
```

Tags

Similar to layers, Tags can be assigned to GameObjects to categorize objects within a scene for various scripting applications. Here is an example of how we can use Tags to delete targets in our game.

Assign the Tag “targetPrefab” to the Prefab object as shown. Select an existing Tag from the drop down menu at the top, or add a custom Tag as shown below.



Now that we have created the tag for our GameObject we can attach the following script to a box-collider to delete targets when they have passed through the collision region. The “CompareTag” checks that the “other” object has the “targetPrefab” tag assigned to it and returns true.

```
21 void OnTriggerExit (Collider other)
22 {
23     Debug.Log("Object has exited the trigger box region.");
24     if (other.CompareTag("targetPrefab"))
25     {
26         Destroy(other.gameObject);
27         //targets.RemoveAt(targets.size() - 1);
```

(Atom code editor shown above)

Lighting

There are several lighting types:

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

Sprites

To display 2d images on the screen:

1. Drag your image into the Assets window.
2. Create an empty GameObject in the Hierarchy window..
3. Add the Sprite Renderer component to your GameObject from the Inspector.
4. Drag your image from the assets folder into the correct field under the script component attached to your GameObject after locking the inspector window.

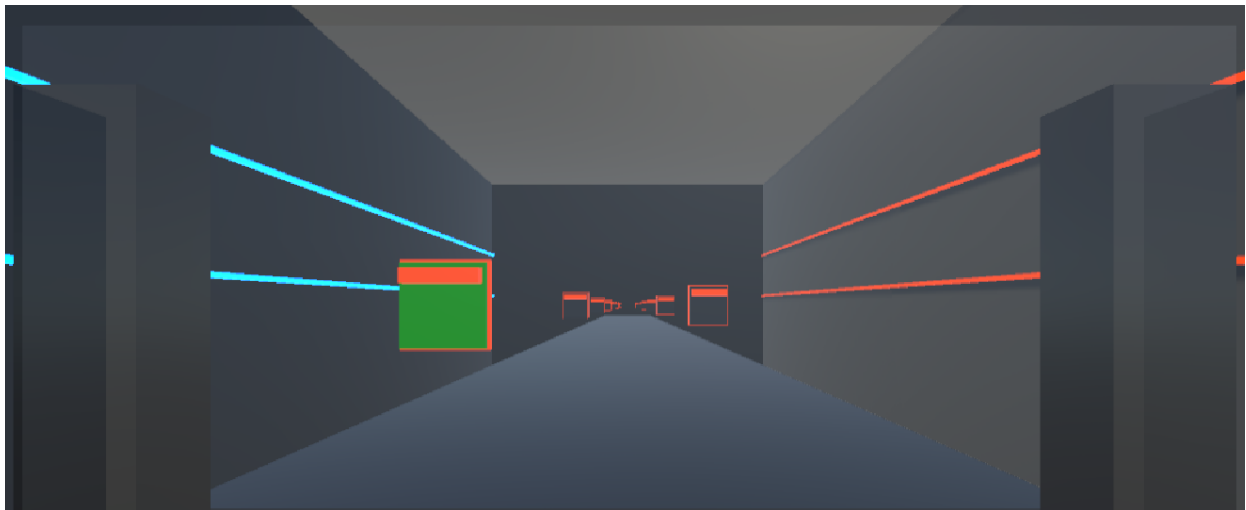
Scripting

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

Select the appropriate GameObject from the Hierarchy window and drag your script into the Inspector window.

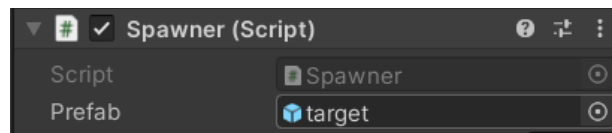
Find Spawner.cs [here](#) for an example of a script that spawns targets and moves them towards the player camera.

“Spawner.cs”



To implement this script follow the following steps:

1. Create an empty GameObject in the Hierarchy window
2. Drag the script to GameObject's Inspector window
3. Drag the prefab from the Assets window into the Spawner section of the Inspector window as shown:



To assign a GameObject to a specific layer use:

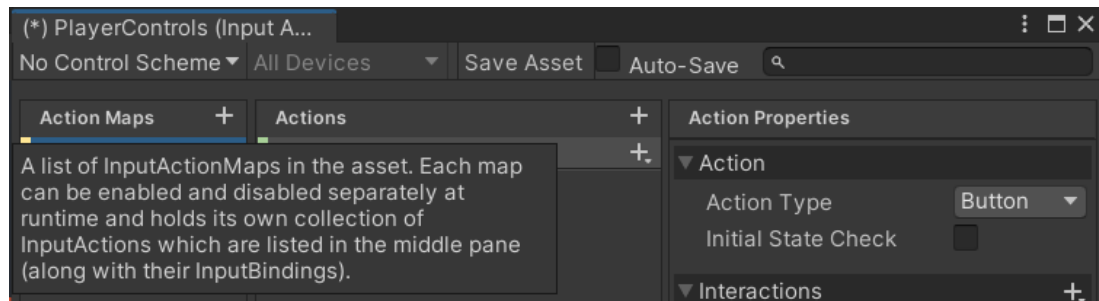
```
gameObject.layer = LayerMask.NameToLayer("Name of Layer");
```

Include the script asset as a component of its GameObject and include the above line in the void Start() function. Ensure *using UnityEngine;* is at the top of your script.

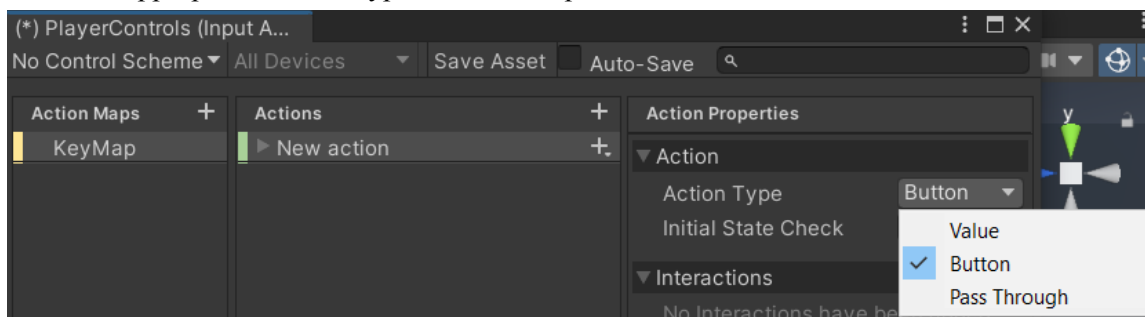
User Input

Custom Key Mappings

1. Right click in the Assets window and select Create > Input Actions
2. Give the asset a name like "Player Controls."
3. 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.



4. 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.
5. Select the appropriate Action Type from the drop-down menu shown below.



6. Specify keybinds inside the KeyMap.

Using Mouse Clicks

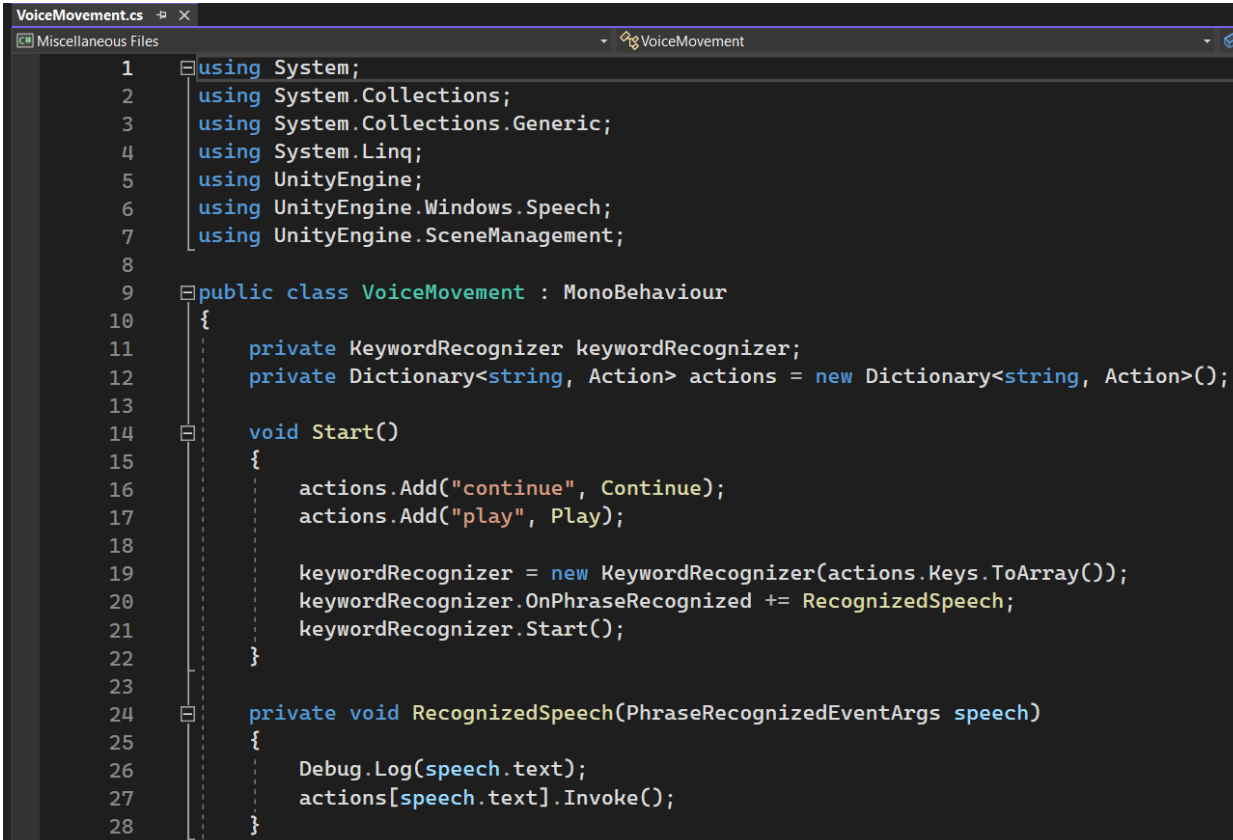
To delete objects within the game.

1. Create an empty GameObject in your scene 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.

Speech Recognition

Create a dictionary for the KeywordRecognizer as shown.

A screenshot of a Visual Studio code editor window titled "VoiceMovement.cs". The code is written in C# and defines a class named "VoiceMovement" that inherits from "MonoBehaviour". The code includes several using statements for System, System.Collections, System.Collections.Generic, System.Linq, UnityEngine, UnityEngine.Windows.Speech, and UnityEngine.SceneManagement. The class contains a private field for "keywordRecognizer" and a private dictionary "actions" of type Dictionary<string, Action>. The Start() method initializes the actions dictionary with "continue" and "play", sets up the keyword recognizer with the actions keys, and starts the recognizer. The RecognizedSpeech method logs the recognized text and invokes the corresponding action from the dictionary.

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.Linq;
5 using UnityEngine;
6 using UnityEngine.Windows.Speech;
7 using UnityEngine.SceneManagement;
8
9 public class VoiceMovement : MonoBehaviour
10 {
11     private KeywordRecognizer keywordRecognizer;
12     private Dictionary<string, Action> actions = new Dictionary<string, Action>();
13
14     void Start()
15     {
16         actions.Add("continue", Continue);
17         actions.Add("play", Play);
18
19         keywordRecognizer = new KeywordRecognizer(actions.Keys.ToArray());
20         keywordRecognizer.OnPhraseRecognized += RecognizedSpeech;
21         keywordRecognizer.Start();
22     }
23
24     private void RecognizedSpeech(PhraseRecognizedEventArgs speech)
25     {
26         Debug.Log(speech.text);
27         actions[speech.text].Invoke();
28     }
29 }
```

Make sure to implement your actions. Below you can see I utilized the Continue and Play phrase commands to switch between scenes.

```

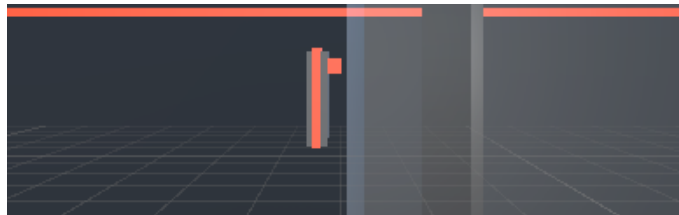
30 private void Continue()
31 {
32     SceneManager.LoadScene("MainMenu");
33 }
34
35 private void Play()
36 {
37     SceneManager.LoadScene("SampleScene");
38 }

```

Collisions

Example of collision-event triggering:

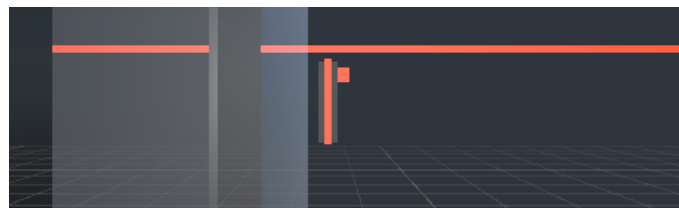
In the picture below you can see a target object (on the left) moving towards the trigger box in gray.



A few frames later the target turns green.



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



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


```
or.cs  X getInput.cs  Spawner.cs  GameLogic.cs  MusicNinja.cs
aneous Files  changed
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class changeColor : MonoBehaviour
6  {
7      private Renderer rend;
8
9      void Start()
10     {
11         rend = GetComponent<Renderer>();
12     }
13
14     void OnTriggerEnter(Collider other)
15     {
16         rend.material.color = Color.green;
17     }
18
19     void OnTriggerExit(Collider other)
20     {
21         rend.material.color = Color.black;
22     }
23 }
```

Remember to always provide your scripts with all necessary assets whenever they refer to GameObjects or prefabs by dragging them from the assets folder into the corresponding field in the inspector window like so:

