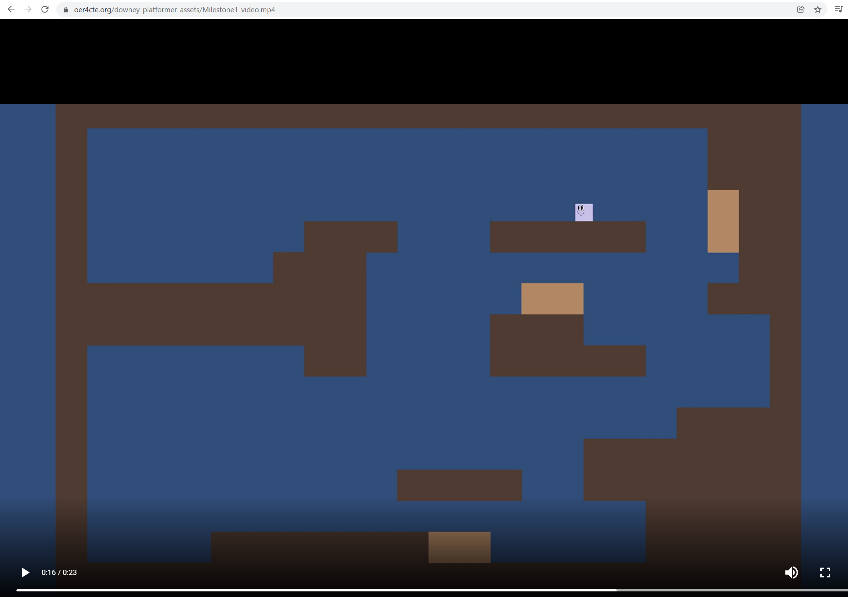
# Milestone 1 – Deliverable 1 Detailed Tutorial

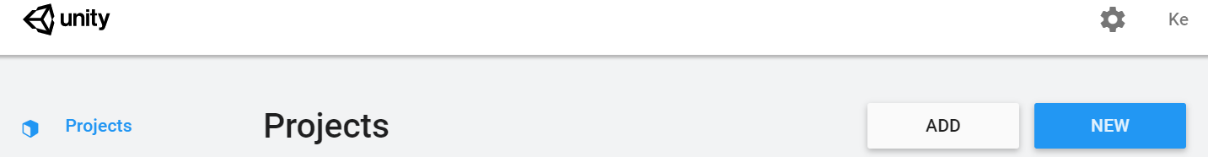
This link leads to a short video that demonstrates what you will create in Milestone 1 Deliverables 1-4. Please go to <https://www.oer4cte.org/downey_platformer_assets/Milestone1_video.mp4>



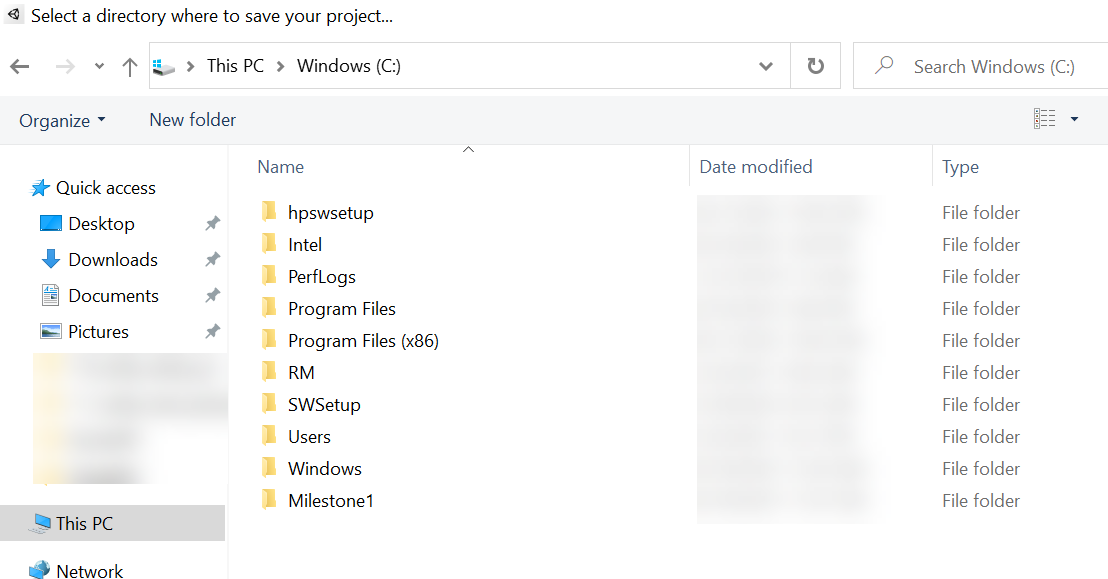
## Create a Unity Project

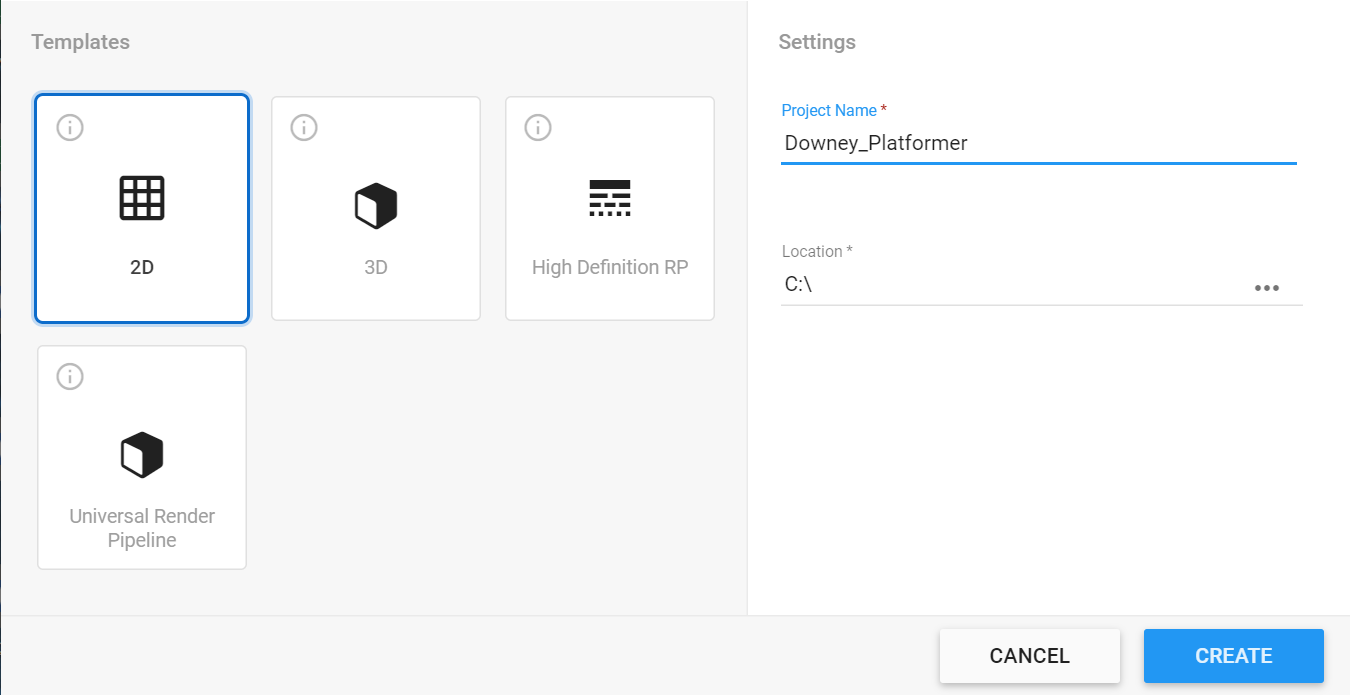
Create a folder. I created a new folder on drive C: and named it Downey\_Platformer. You may have an assigned or a preferred location on your computer.

Launch the Unity Hub. Click the New icon in the upper-right corner of the Projects window.



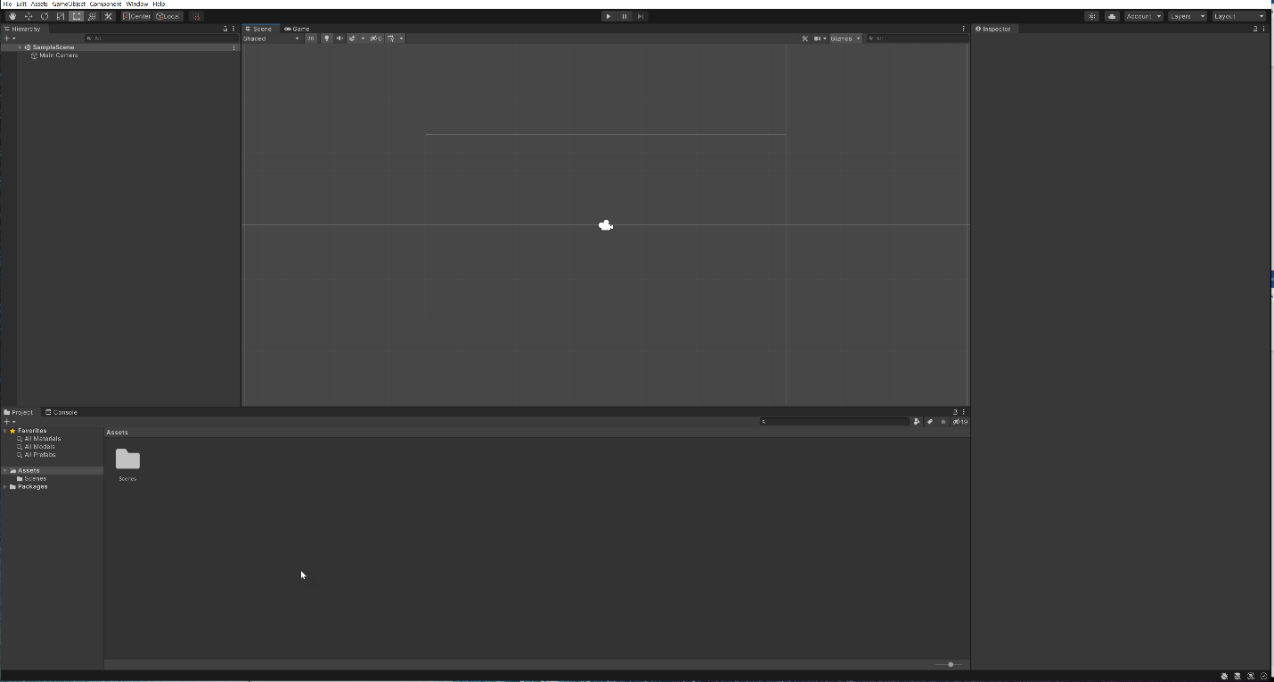
Select NEW to create a Project. Give your Project a name and set the Project's location (the folder you created above). I named my Project Milestone1



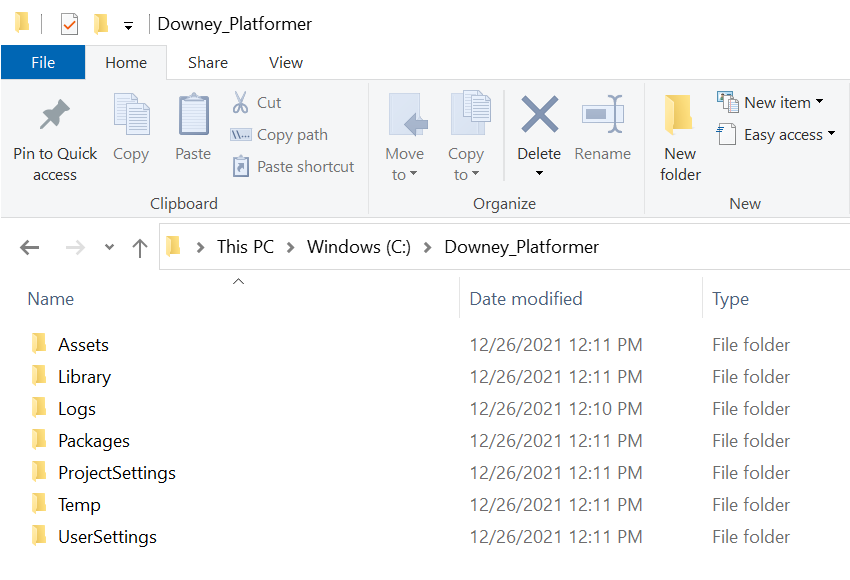


Select 2D for this project. Name the New Project Downey\_Platformer and set the Project’s location to your C: or wherever your teacher or team determine.

Select the CREATE Button. You will see a bar that indicates the New Project in progress. When complete, Unity's main screen will launch.



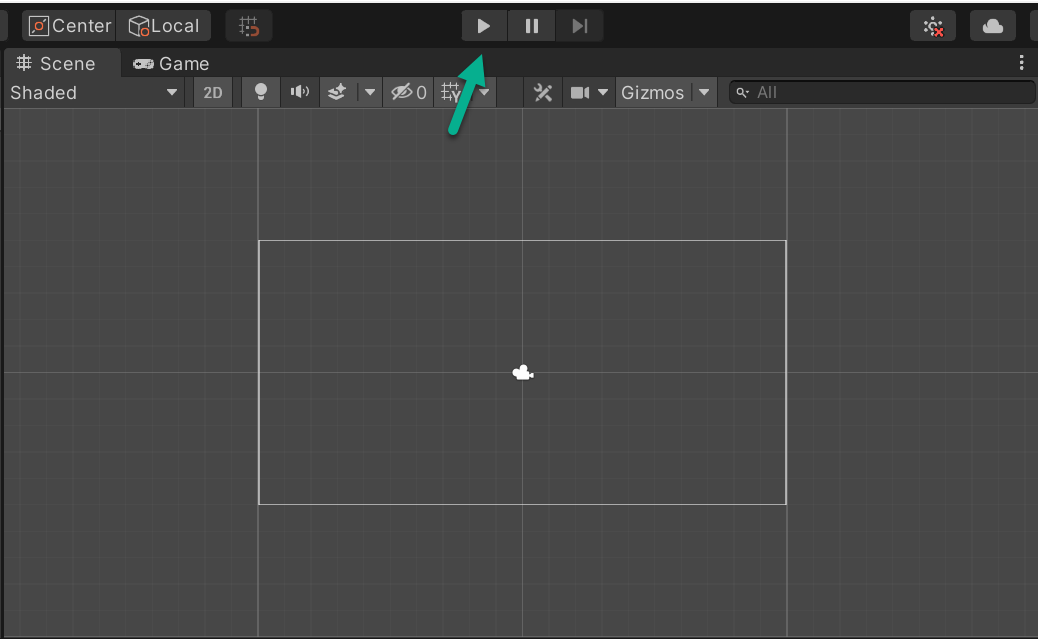
Open your Downey Platformer folder to view folders Unity created when you created your project.



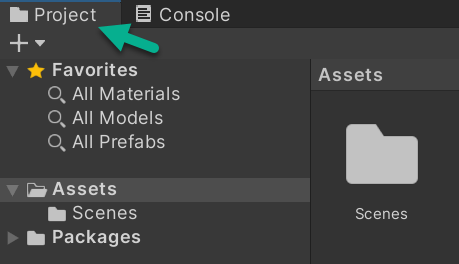
The most important advice as you start is to pause and focus whenever you name a file. Naming a file in Unity creates code in several areas. What seems a simple error such as an added space or a change from lowercase to uppercase completely changes how Unity and Visual Studio sync because a file named BreakableBlock is not the same as Breakable Block, breakableBlock or any other combination. In some cases, you're safe to Rename a file; however, not all.

There are four areas inside Unity that you should know what they do before you get started:

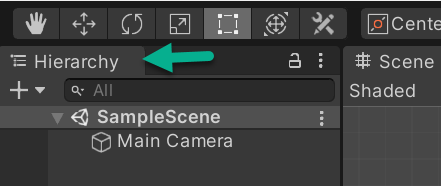
1. The **Scene** and **Game** tabs are the world space where your game exists. Scene is where you can edit your scene and clicking on the play button at the top switches you over to the Game tab, where you can see the game when it is running. The game tab by default is showing the perspective of the Main Camera in your world space.



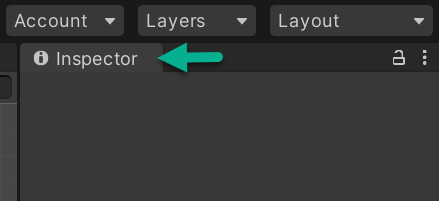
1. The **Project** tab is where the asset files for your game are. All the files you create, and import will be shown here in the Assets folder, so you can bring them into the scene.



1. **Hierarchy** is where you access all objects that are in the open scene. Objects you edit or create here will only exist in this open scene unless you make them a prefab file. (How to do this will be shown in Section 3 of this tutorial.)

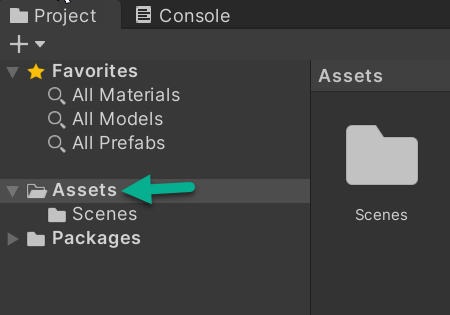


1. The **Inspector** tab is where you can edit the properties of objects you select in the Project or Hierarchy tabs.

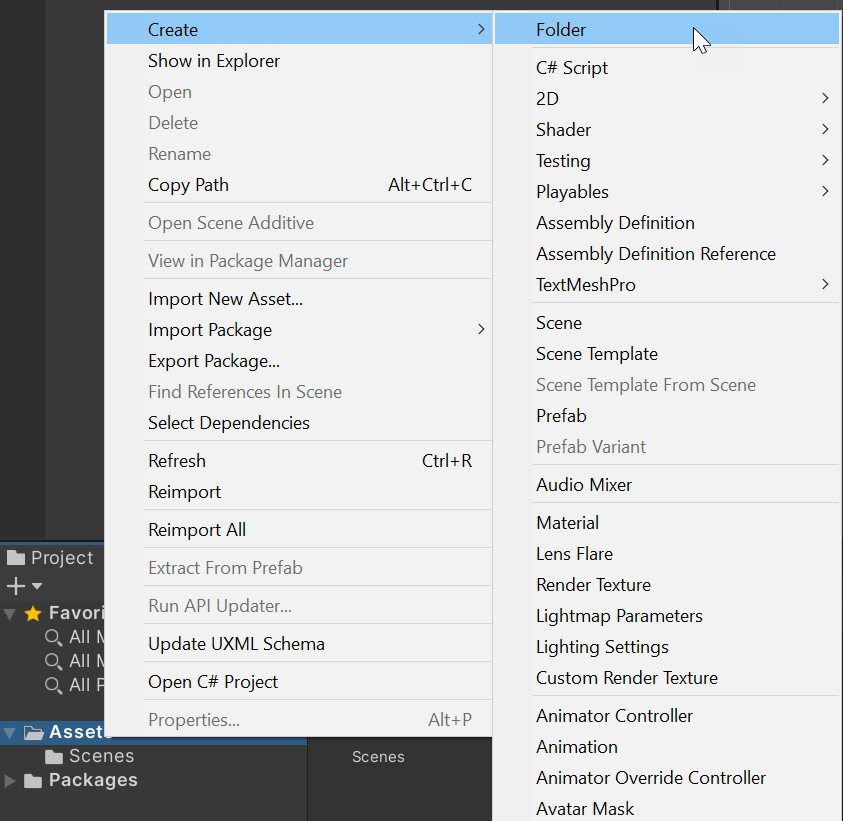


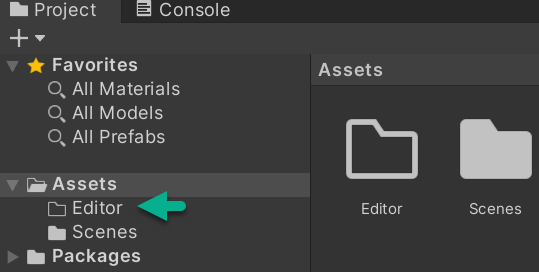
## Create a New Scene

Navigate to the Project tab and **right-click** on the Assets folder.



Select Create. Select Folder. Name the folder Editor. Double click on your Editor folder to open it.





A SnapToGridEditor script aligns all objects in a scene on a grid. **There will be errors in your console after adding this script because it reads that some things are missing.** You will add them later in the project; for now, they can be ignored and will resolve themselves as you progress.

Since you are new to Unity, we created this script file rather than have you code it.

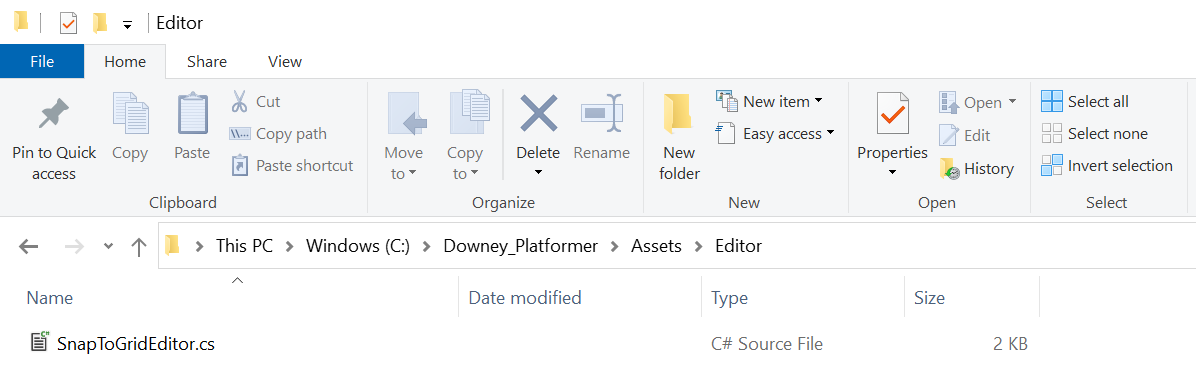
Please go to:

<https://www.oer4cte.org/downey_platformer_assets/SnapToGridEditor.cs>

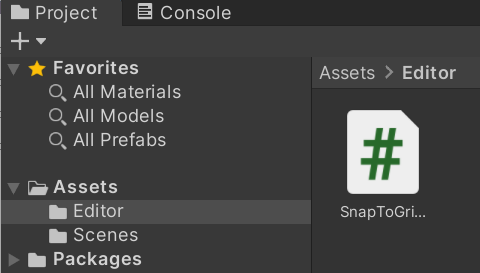
Right-click to Save as...

SnapToGridEditor.cs

to your Unity Project in Downey\_Platformer/Assets/Editor



This saves SnapToGridEditor.cs into Downey\_Platformer > Assets > Editor. Click into Unity to see the script import into your Project tab > Assets > Editor. Save your Project.

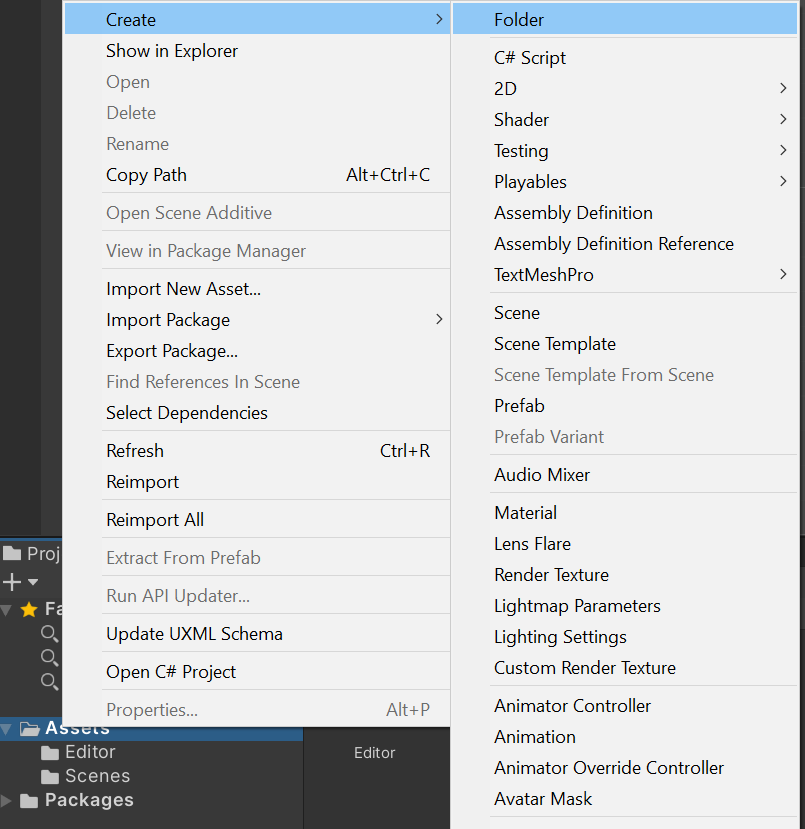


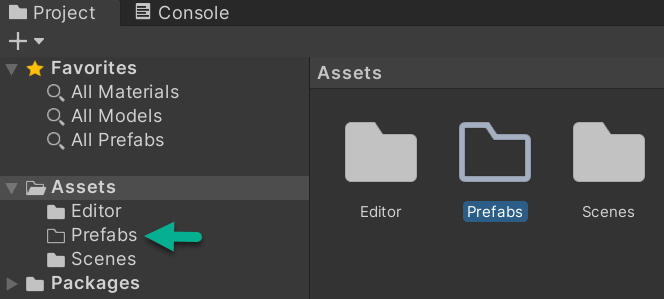
## Create Solid Block and Breakable Block prefabs

### Term Definitions:

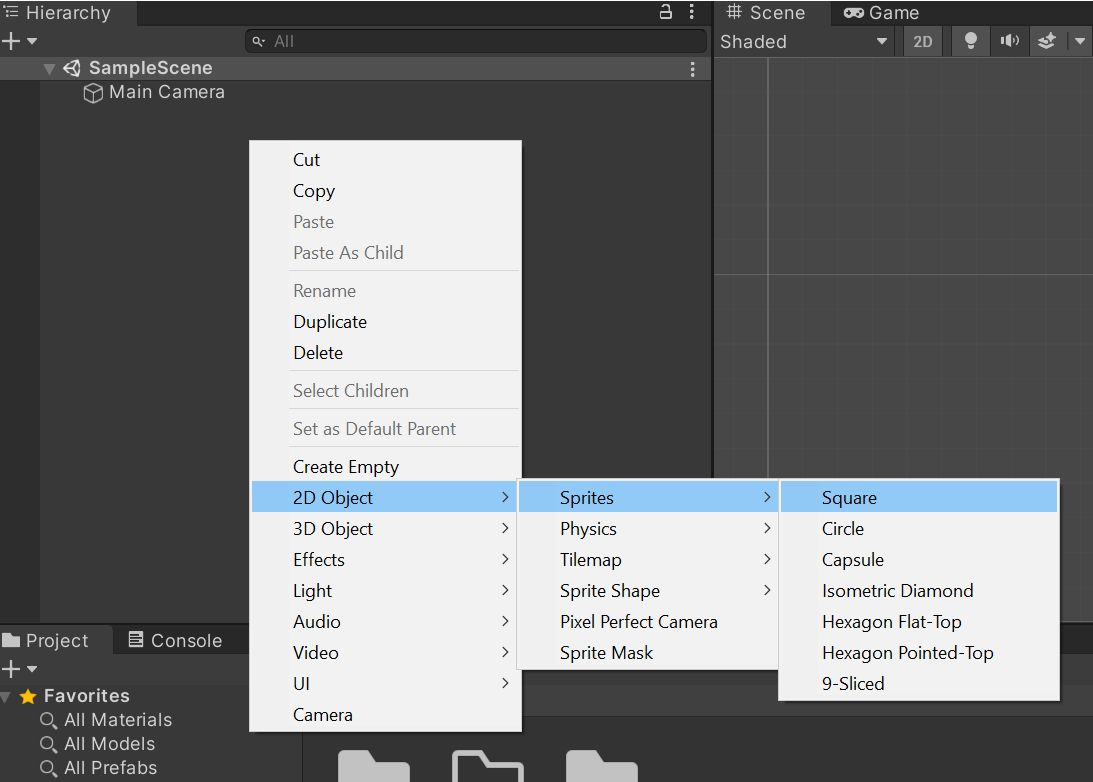
* **Prefab** is short for Pre-fabricated Object. To reuse objects, save as a Prefab to create a template for future use. The Prefab contains all required components and settings for reuse. When a Prefab is changed, all instances of that prefab in use change.
* **Main Camera** renders what is viewed in the Game window.
* **Player** is the character we control and that moves around the environment.
* **Game borders** are walls on four sides.
* **Solid Blocks** are unbreakable blocks.
* **Breakable Blocks** are breakable.

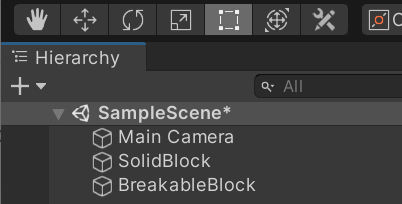
Select the Assets folder, right-click to create a new folder, and name it Prefabs



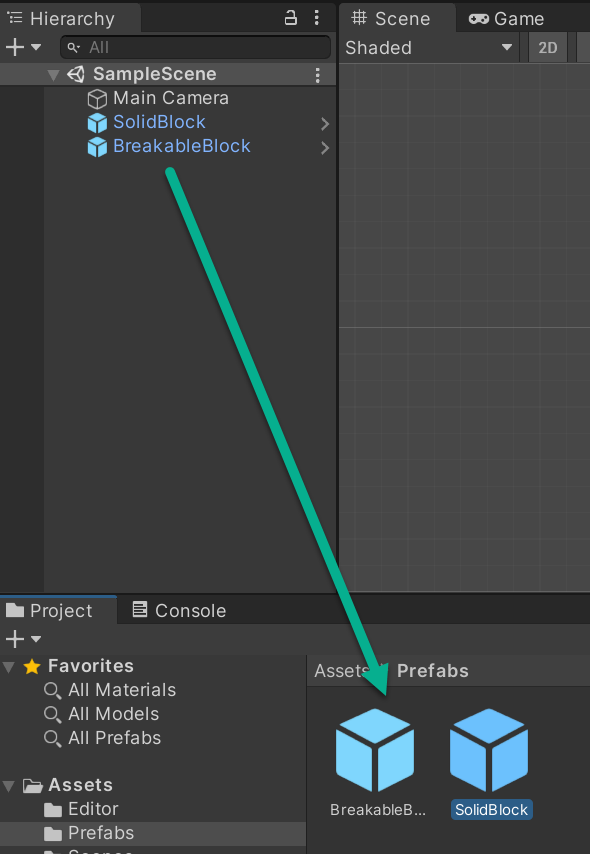


Right click in the Hierarchy tab space and create a 2D Object > Sprites > Square. Name it SolidBlock instead of Square. If you miss on naming, right click the name and Rename. Repeat to create a second 2D > Object > Sprite > Square and name it BreakableBlock. Note\* there is no space in SolidBlock or BreakableBlock. Watch for your hand adding spaces in code due to habit.





Double click on your Prefabs folder to open the Prefabs folder. One at a time, drag and drop SolidBlock and BreakableBlock into your Prefabs folder. Save your Project.

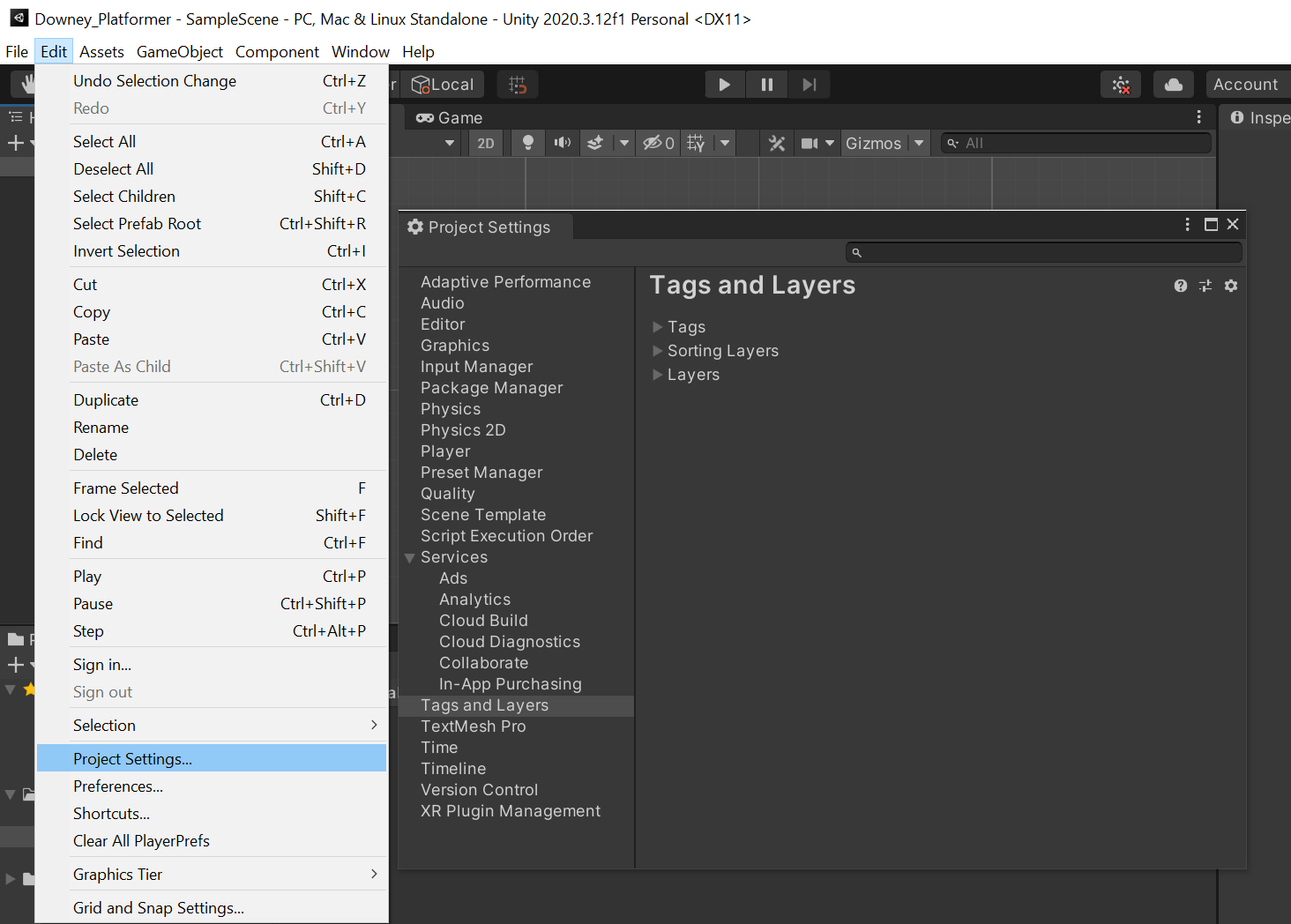


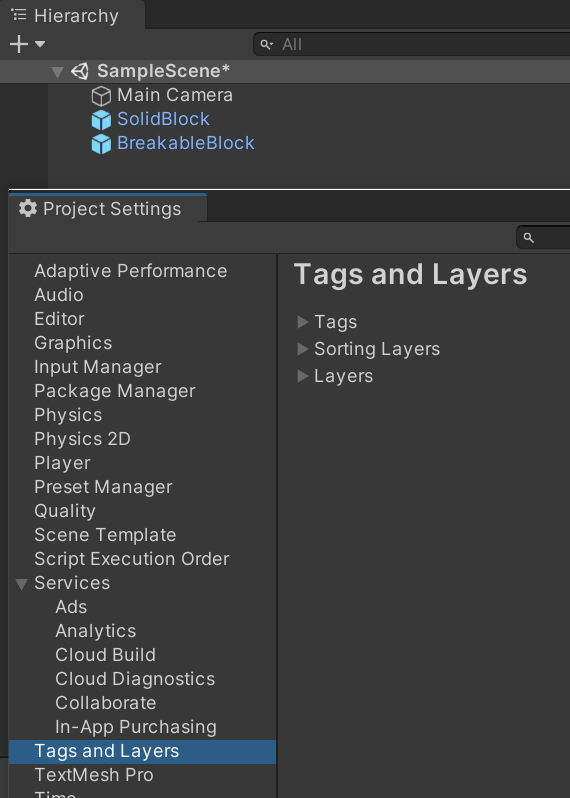
## Tags and Layers

### Term Definitions:

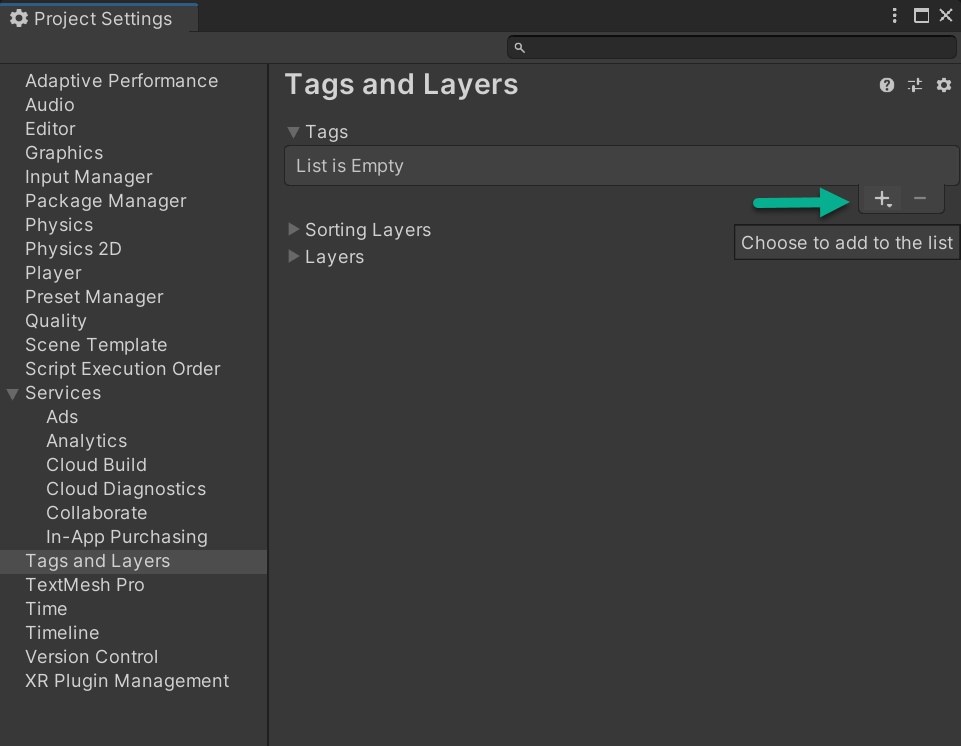
* **Tags** and Layers help you identify Game Objects for scripting purposes. They save time when you use the same script code with multiple Game Objects. For example, if you have 100 enemy objects in a scene and want to check if any enemy collided with a player, you can set the tag of 100 enemies as Enemy and check in your script to see if the collided object has the Enemy tag.
* **Layers** are like Tags and differ in their use case. Layers are used when you want to cast rays on a specific layer of objects. Layers are also used in Camera to manage lighting and show on some layered objects in the camera.

Go to the Unity primary (top) menu. Select Edit > Project Settings and select the Tags and Layers Category.

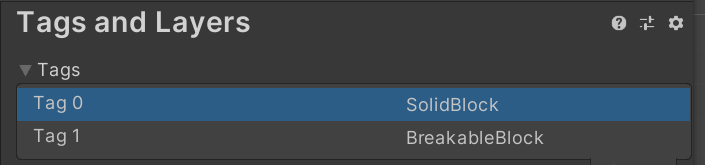




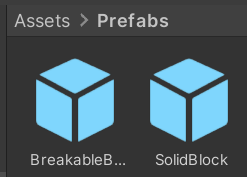
To add a new tag, select Tags (first menu item below Tags and Layers) and click the plus button (+), bottom-right below List is Empty.

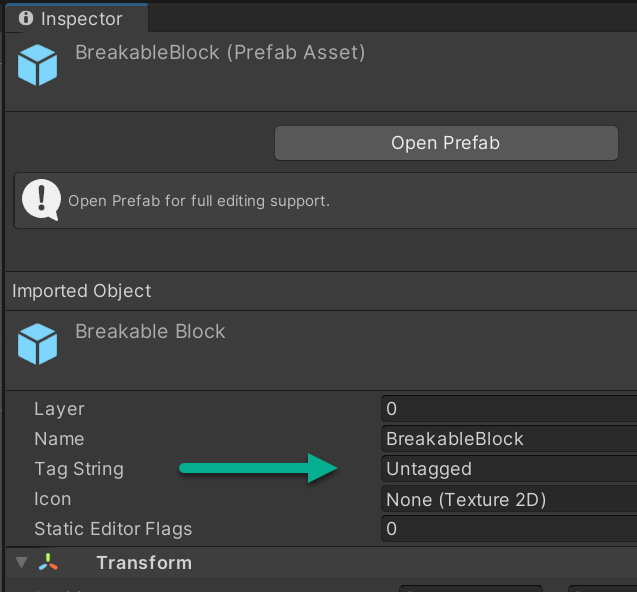


To create SolidBlock and BreakableBlock tags, one at a time, select the +, enter tag name SolidBlock and save. Select the +, enter tag name BreakableBlockand save. A tag number will automatically be assigned to each. SolidBlock is assigned Tag 0. BreakableBlock is assigned Tag 1. Be sure to Save afterward (Ctrl + S or in the Unity top menu).

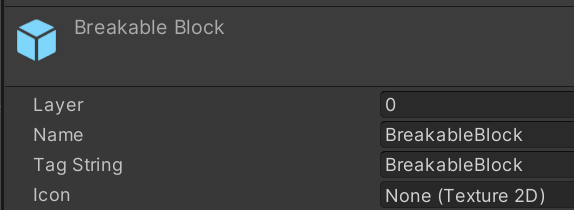


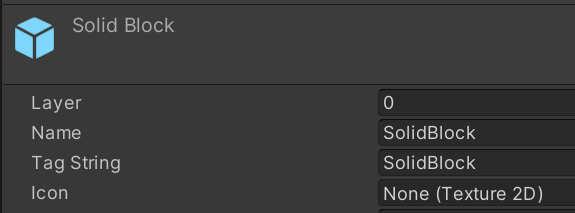
Exit Project Settings by clicking the X in the upper right corner of the Project Settings pane. To assign the tags to their respective prefabs, click on each Prefab Sprite, one at a time, and a window opens in Inspector.





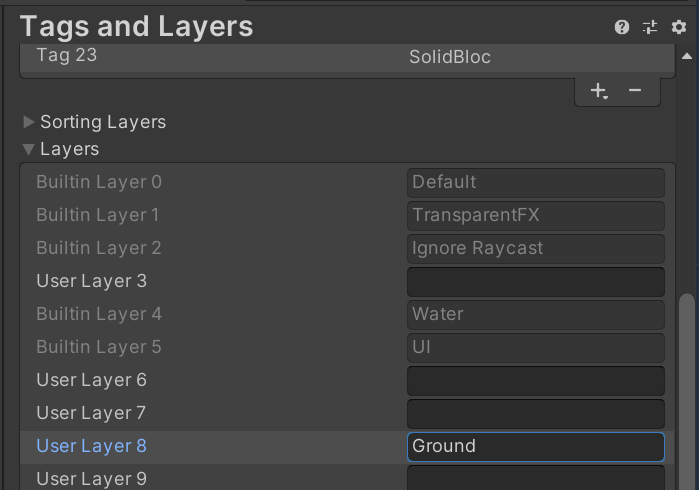
Change the Tag String for each to their respective Prefabs to BreakableBlock and SolidBlock. Save your Project.





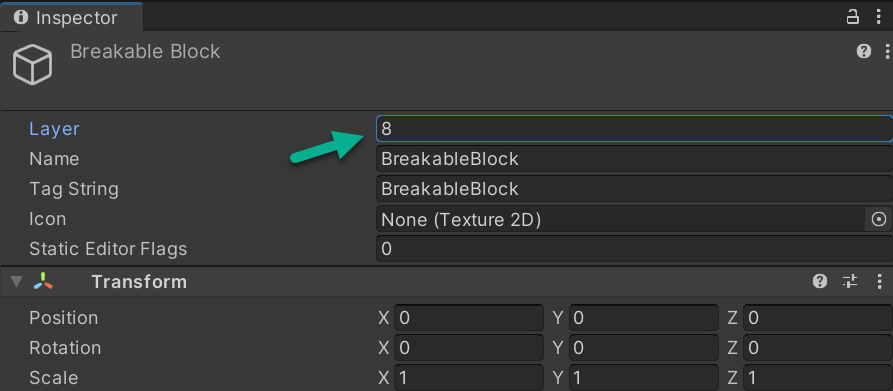
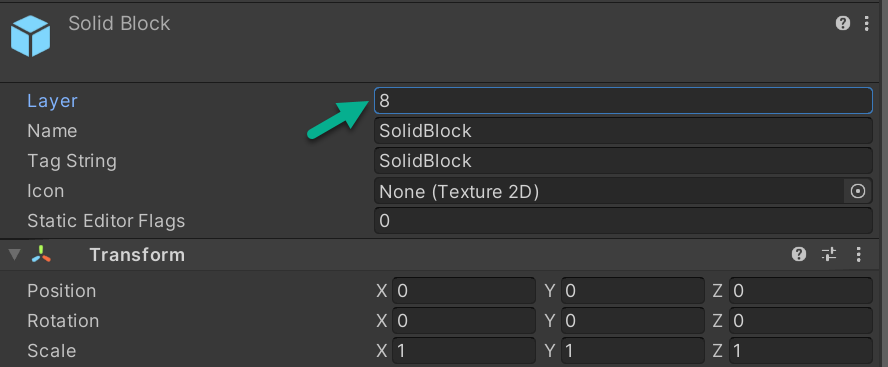
## Create a Ground Layer

Go to Edit > Project Settings > Tags and Layers. You may need to minimize Tags to view and open Layers (not Sorting Layers). Create a Ground Layer at User Layer 8.



There is no window Save button, close Tags and Layers and Save your Project.

Next, assign Ground to User Layer 8 for both block prefabs (BreakableBlock and SolidBlock). Select Breakable Block. In Inspector, assign Layer as 8. Repeat for SolidBlock. Save your Project.

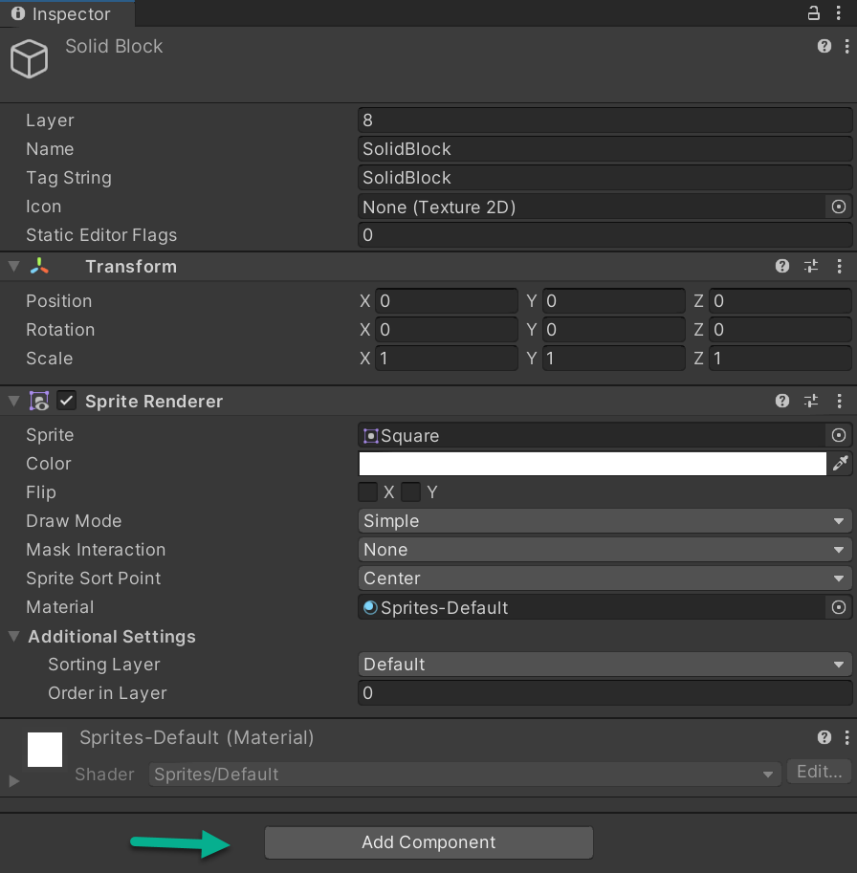
  
  


## BoxCollider2D

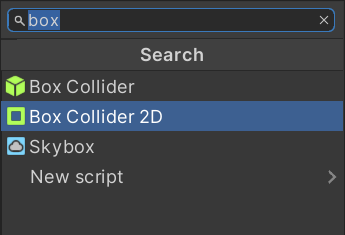
### Term Definitions:

* **Rigidbodies** enable Game Objects to act as though under the control of physics. A Rigidbody can receive forces and torque to make objects move in a realistic way. Unity has two versions of Rigidbodies, one for 3D games and one for 2D. Make sure to use a Rigidbody2D component when adding to them your game objects.
* **Colliders** are components added alongside Rigidbodies to receive collisions and forces. Collider tells the physics engine the shape of an object so when objects collide, they do not pass through each other. Like Rigidbodies, there are 3D and 2D versions of them, so make sure to add Colliders that have 2D at the end. For example: BoxCollider2D
* **Ray casts** are ways to be aware of nearby objects in the environment. In shooting games, for example, when you fire weapons, a Ray is fired in the direction of the weapon and returns information on the Game Objects interacted with. This triggers actions such as show splash for water or show blood for enemy weapons.

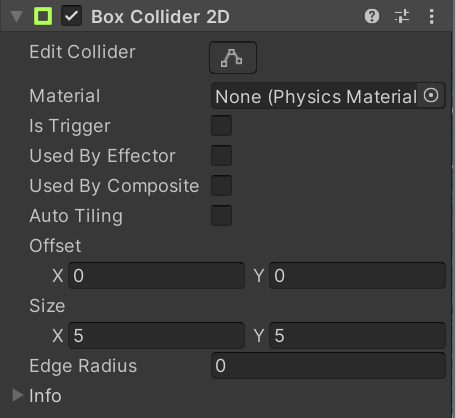
Add a BoxCollider2D component to both block prefabs. Select the the SolidBlock prefab, go to the Inspector tab, and click Add Component.



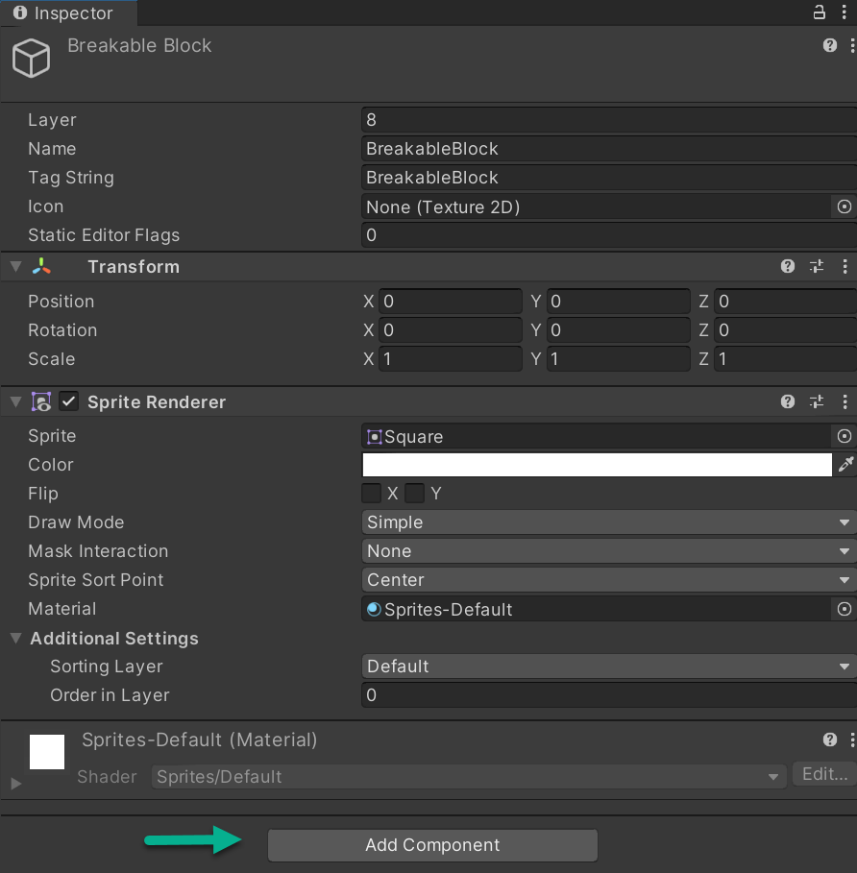
The Add Component takes you to a search menu. Key in box to select BoxCollider2D



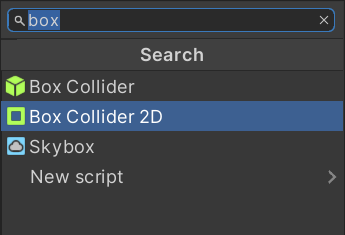
Enter box in the search bar and select Box Collider 2D. Change the Size of Box Collider 2D to X = 5 and Y = 5



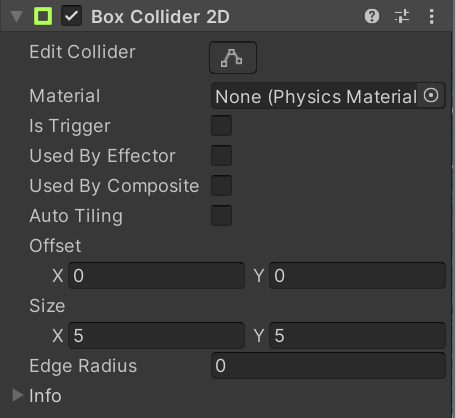
Select the BreakableBlock prefab, go to the Inspector tab, and click Add Component.



The Add Component again takes you to a search menu. Key in box to select BoxCollider2D

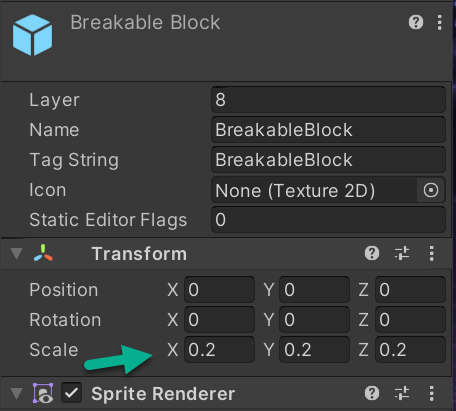
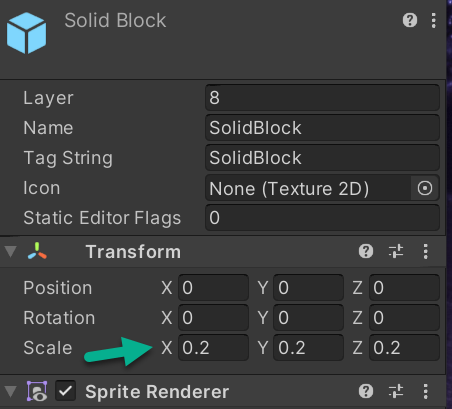


Enter box in the search bar and select Box Collider 2D. Change the Size of Box Collider 2D to X = 5 and Y = 5



In Inspector, Transform, set Scale for BreakableBlock and SolidBlock to X = 0.2, Y = 0.2, and Z = 0.2

Save your Project.

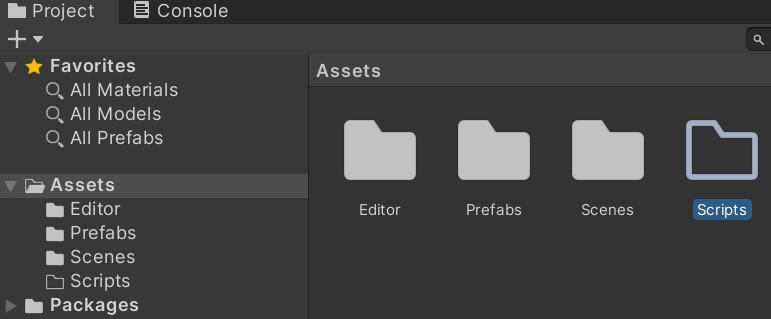
 

## Adding Scripts

### Term Definitions:

* **Scripts** are the files that contain the code for our game objects. In Unity we attach scripts as a component to game objects we want to add custom behaviors to.
* **Classes** are the blueprints for your script. This is the syntax structure that lets Unity read the actions to take inside the script.
* **Variables** are lines of code inside a class that contain a type of data that the class can access and change if wanted.
* **Methods** are modules inside the class that wrap code in easy to repeat instruction. For instance, all Unity scripts have a Start()method that is called by Unity every time the game object the script is attached to is enabled. Methods can have **parameters** that pass variables and data to the method.
* **Comments** are lines of code that are descriptions and notes on what the code is doing. Comments are not needed to make a script work, but they are expected programming practice. We add them to communicate what the code does for other programmers who work on the team. We will use three types of comments:
  1. Single-line comments start with two forward slashes //
     + Any text between // and the end of the line is ignored by C# (will not be executed).
  2. Multi-line comments start with /\* and end with \*/
     + Any text between /\* and \*/ will be ignored by C#.
  3. Three forward slash, ///
* A special type of comment in C# used to create the documentation of C# code by adding XML elements in the source code.

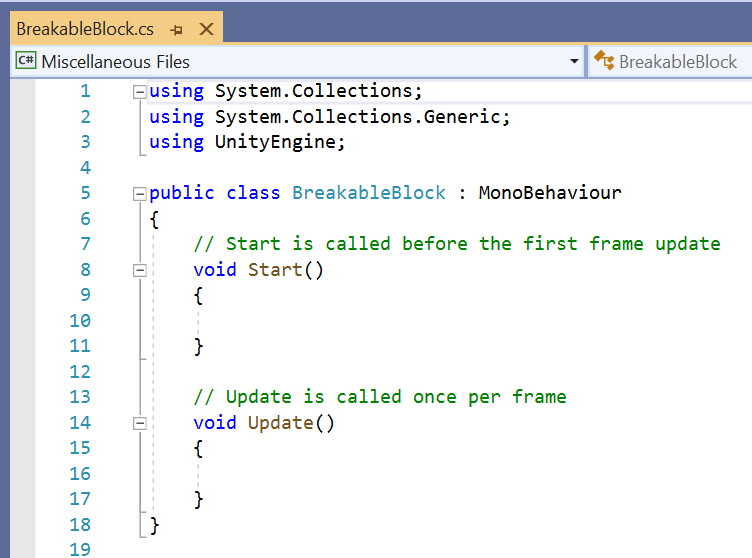
In the Assets folder, create a new folder and name it Scripts. Double click on the Scripts folder to open it.



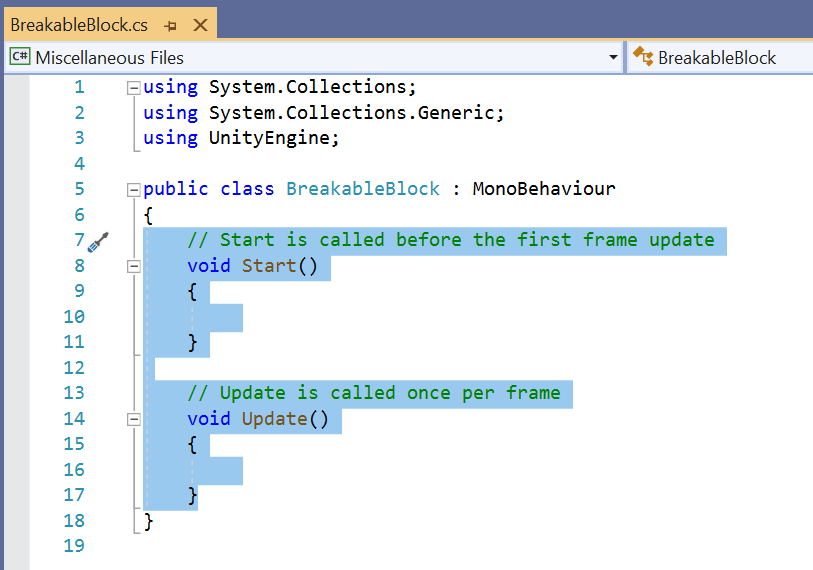
In the Scripts folder, right click to Create > C# Script and name it BreakableBlock

Screenshot of: BreakableBlock C# script in Assets > Scripts.

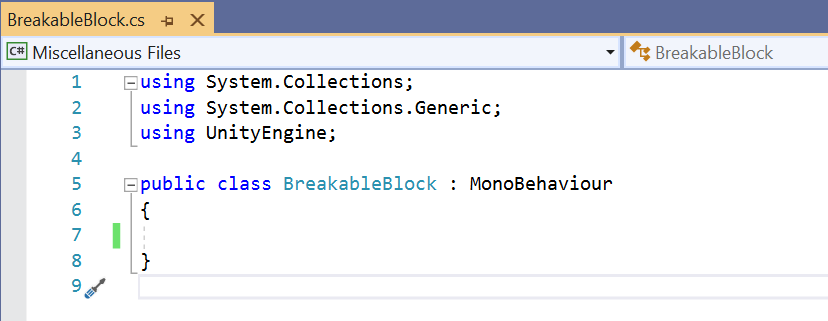

Double click on the BreakableBlock to open Visual Studio once Unity finishes loading the new script.



Inside Visual Studio, Unity will have generated a blank script with two methods: Start() and Update(). Neither of these methods will be used in this script, so delete them and their comments. Do not delete the open and closing braces, please delete the code you see highlighted in blue below.



Your script will be:



The green bar on the left means I saved my change to the file. A yellow bar indicates the edit has been made and needs to be saved. Notice the filename is BreakableBlock.cs. If yours says NewBehaviourScript you forgot to name your C# file when you created it in Unity. It's best to go back to Unity, delete NewBehaviourScript and create a new BreakableBlock.cs script. Keep in mind, you're coding and with multiple files, if they don't match, the code is not going to run as planned.

Class names needs to be identical to script names or you will have errors in the project. Usually, when you create a script and name it right away, the class name and script name are synchronized. Make sure your naming is correct, and with no space in the name.

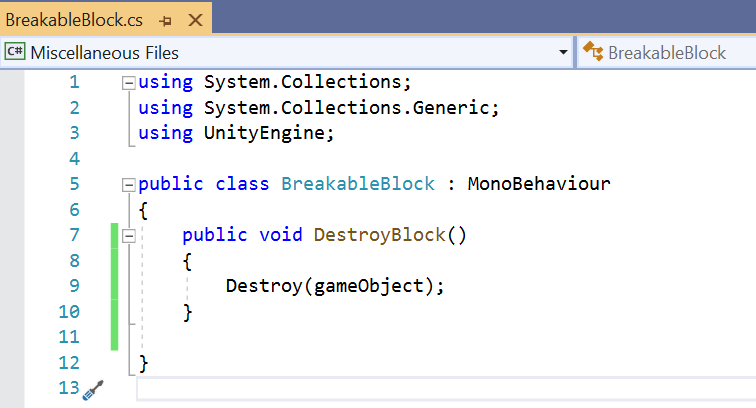
Within the class BreakableBlock, create a new method DestroyBlock() using the code below. Save the file.

public void DestroyBlock()

{

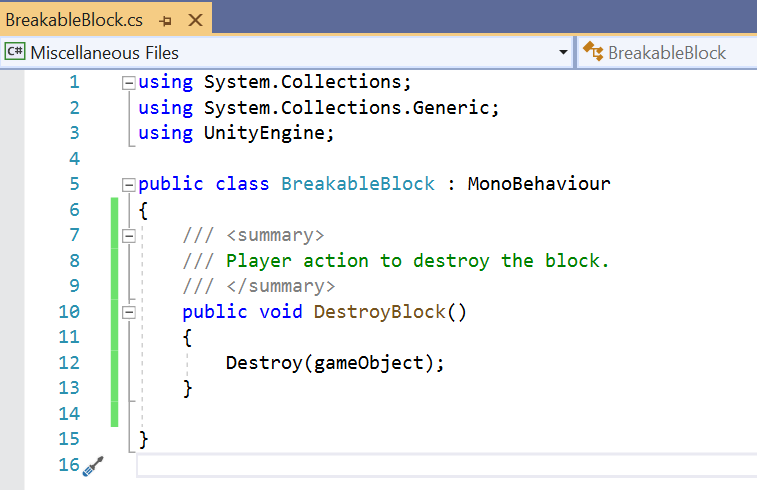
Destroy(gameObject);

}



This method is called when the player does an action that would destroy the block. The Destroy() method is a default method in Unity that will delete whatever object is passed into it from its parameters. In this case, the reference is to the game object this script is attached to. (gameObject is a default variable on Unity scripts that reference this.)

Add a comment template by typing three forward slashes above the method line. The /// of a comment template is an XML comment used for documentation. Fill it out with a description of the method.



Save All and exit Visual Studio. Go to Unity where you will see the update. Save your Project. This ends Milestone 1 Deliverable 1 of Downey Platformer. Proceed to: milestone1\_deliverable2.pdf for the next steps in developing Milestone 1 Deliverable 2 of Downey Platformer. Please link to:

<https://www.oer4cte.org/downey_platformer_tutorials/milestone1_deliverable2.pdf>

and right click to

Save as…

and download the tutorial to your computer.