

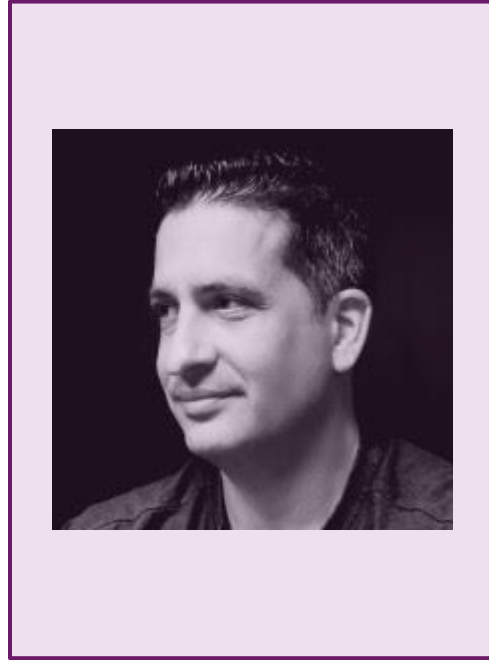


Workshop series by
Christopher Diggins



Introduction to Coding

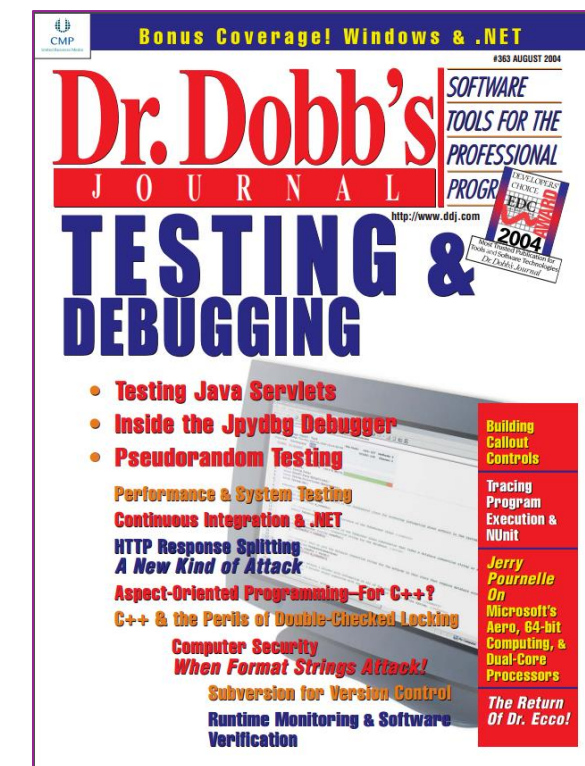
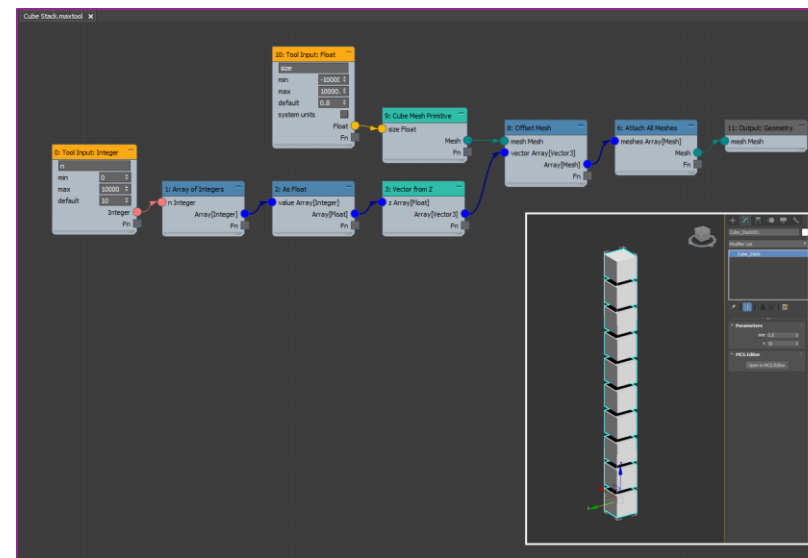
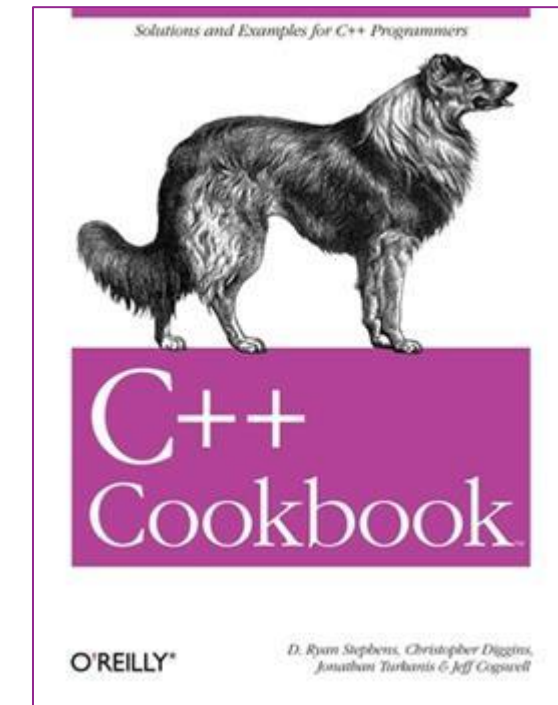
Workshop #1
Getting Started



Christopher Diggins

I've been a professional software developer for over 25 years. I've invented a couple of programming languages, and I'm passionate about making programming easier for myself and others.

I work as head of research at VIM, where we apply game engine technology to the construction industry.



- Introductions
- Personal Objectives

Coding

“Simplicity is the soul of efficiency.” – Austin Freeman

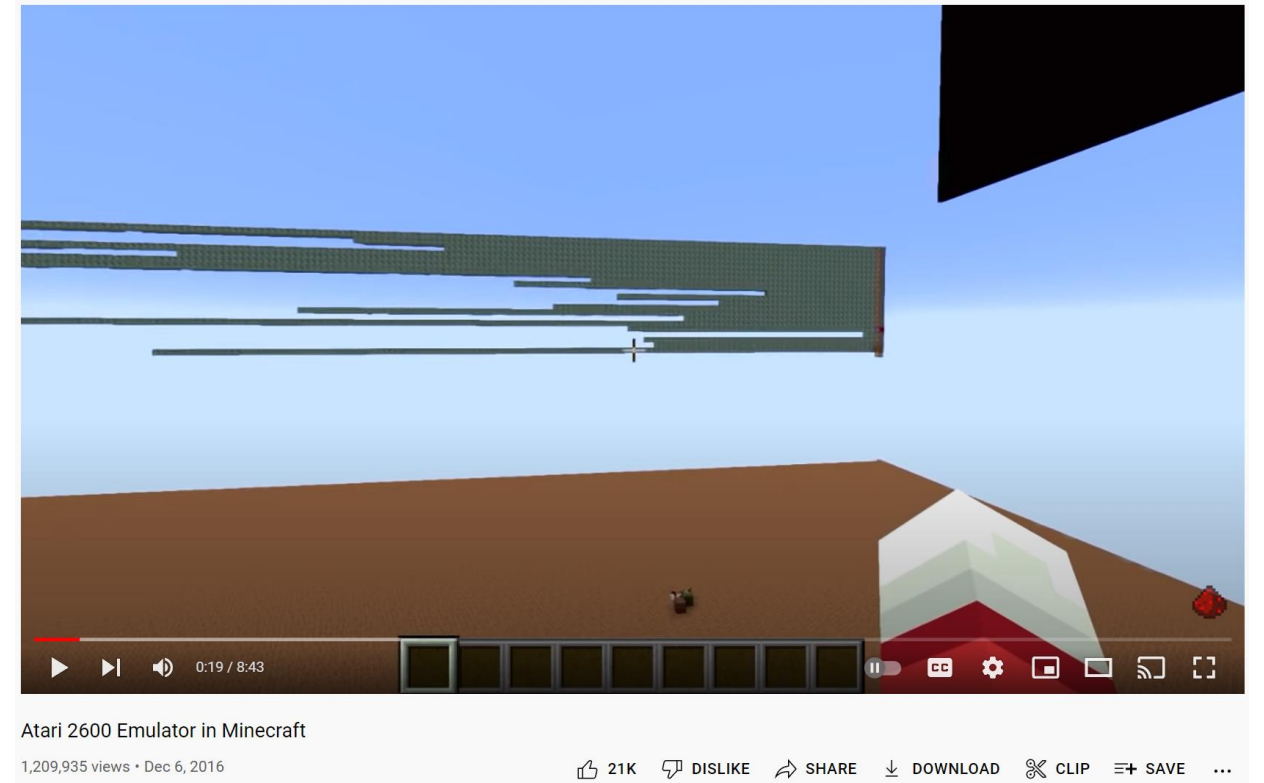
Class Objectives

- Create and run a simple Unity project
- Configure Unity and Visual Studio
- Write your first script / program
- Learn about the key components of a program
- Write functions

Coding is for Humans ... *not Computers!*

- Writing code is the craft of inventing, naming, and organizing concepts that make it easier for us (and others) to ask computers to do what we want.
- Computers can already do virtually anything, we just need to invent ways of telling them, that don't take an infinite amount of time to communicate.

Computer Science is the study of minimizing keystrokes
- Luc Boulianne

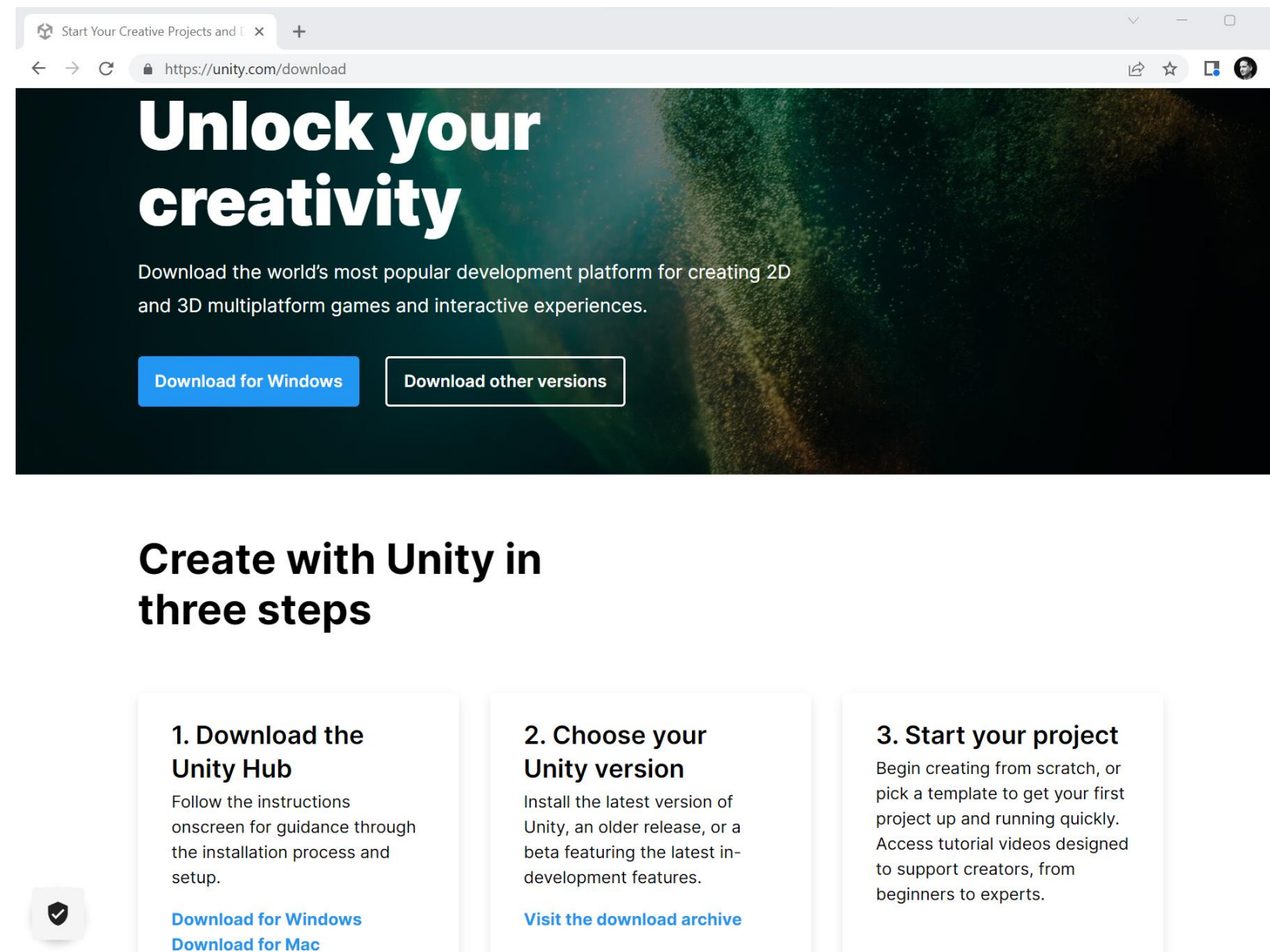


A computer made from dirt in Minecraft!

https://www.youtube.com/watch?v=mq7T5_xH24M

Download and Install Unity

<https://unity.com/download> (any version after 2021 is fine)

A screenshot of the Unity website's download page. The browser's address bar shows 'https://unity.com/download'. The main heading is 'Unlock your creativity' in large white text on a dark, abstract background. Below it, a subtitle reads: 'Download the world's most popular development platform for creating 2D and 3D multiplatform games and interactive experiences.' There are two buttons: a blue 'Download for Windows' button and a white 'Download other versions' button. Below this section, the heading 'Create with Unity in three steps' is displayed. This is followed by three numbered steps in white boxes. Step 1, 'Download the Unity Hub', includes instructions to follow onscreen guidance and provides links for 'Download for Windows' and 'Download for Mac'. Step 2, 'Choose your Unity version', instructs users to install the latest version or a beta and provides a link to 'Visit the download archive'. Step 3, 'Start your project', describes how to begin creating from scratch or use a template, and mentions access to tutorial videos for creators from beginners to experts.

Start Your Creative Projects and T x +

← → ↻ https://unity.com/download

Unlock your creativity

Download the world's most popular development platform for creating 2D and 3D multiplatform games and interactive experiences.

[Download for Windows](#) [Download other versions](#)

Create with Unity in three steps

- 1. Download the Unity Hub**

Follow the instructions onscreen for guidance through the installation process and setup.

[Download for Windows](#)
[Download for Mac](#)
- 2. Choose your Unity version**

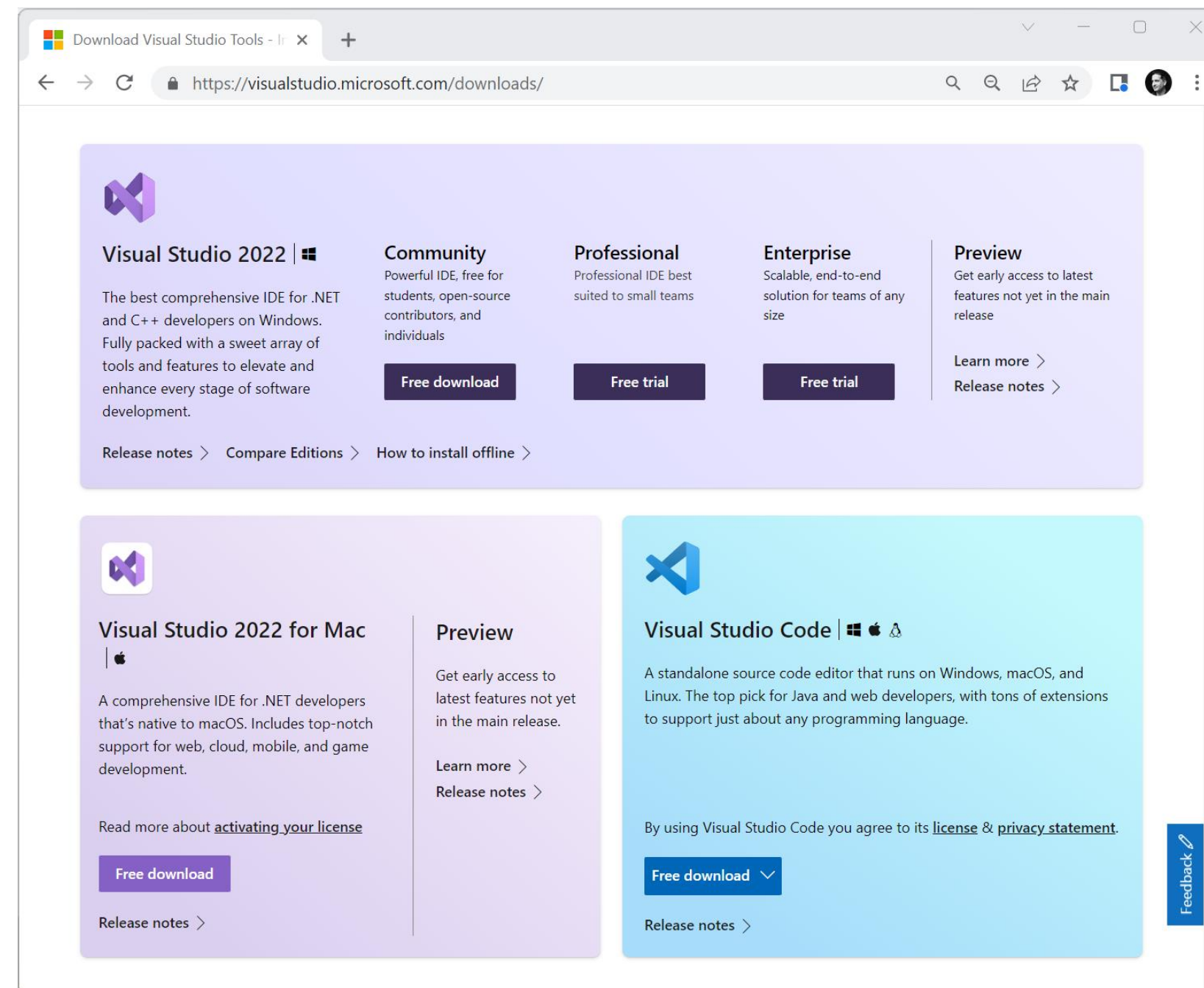
Install the latest version of Unity, an older release, or a beta featuring the latest in-development features.

[Visit the download archive](#)
- 3. Start your project**

Begin creating from scratch, or pick a template to get your first project up and running quickly. Access tutorial videos designed to support creators, from beginners to experts.

Install Visual Studio Community

<https://visualstudio.microsoft.com/downloads/>

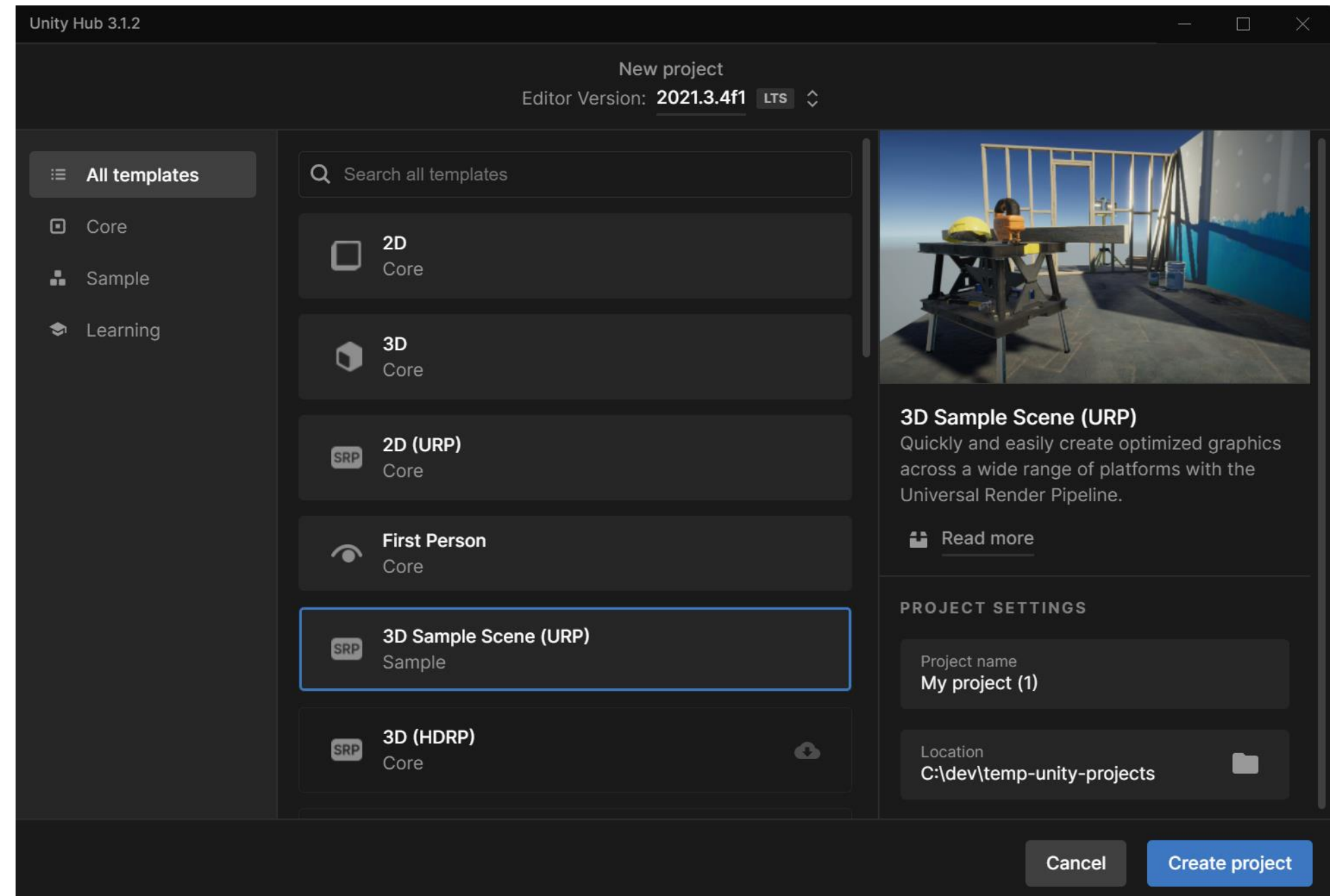


Create a New Unity Project

1. Launch Unity Hub
2. Create a New Project using the “3D Sample Scene (URP)” template

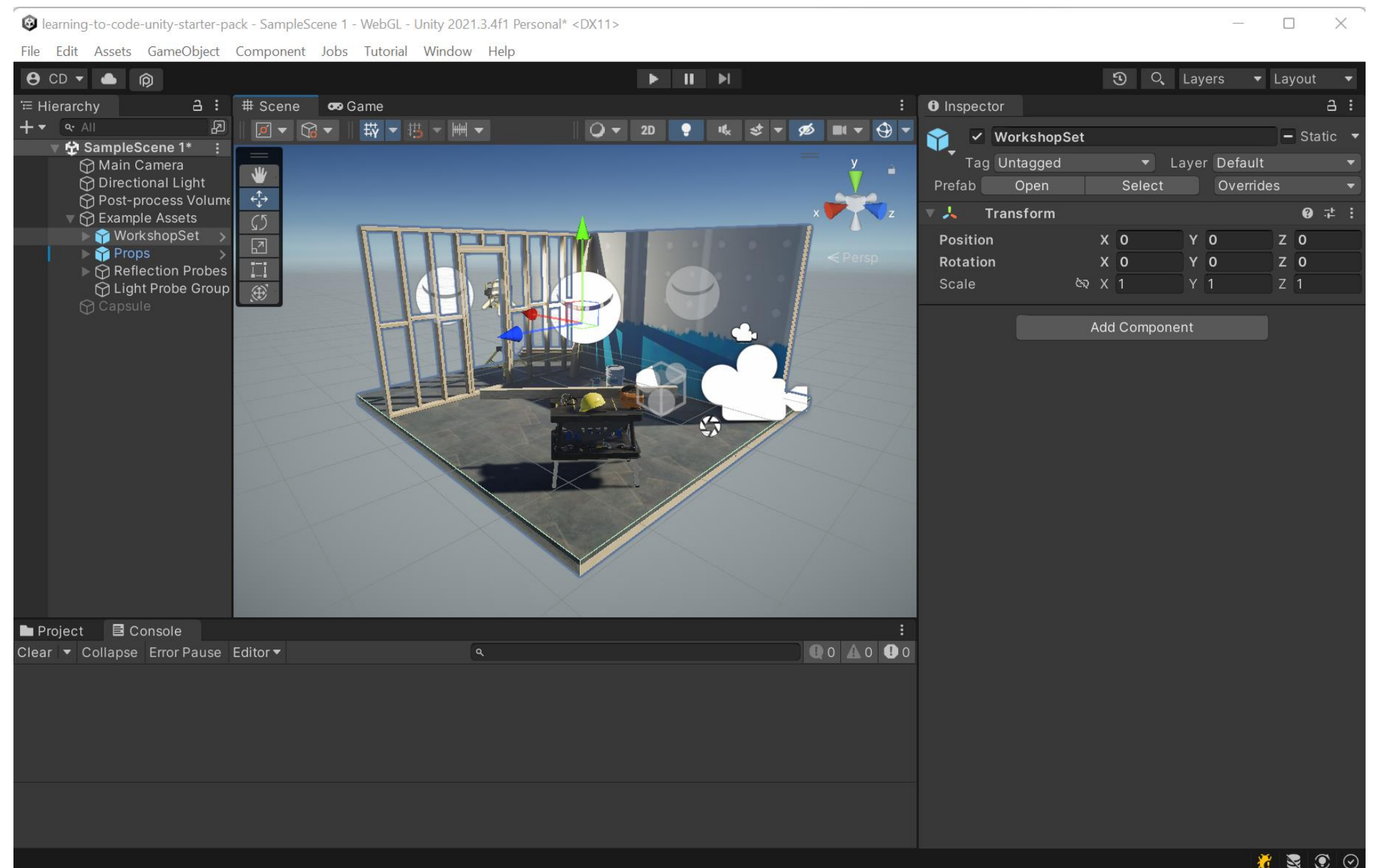
NOTE: The version of Unity or Unity Hub doesn't matter much. Anything from 2019 onwards should work great.

The project name and location are up to you. This will take a few minutes.



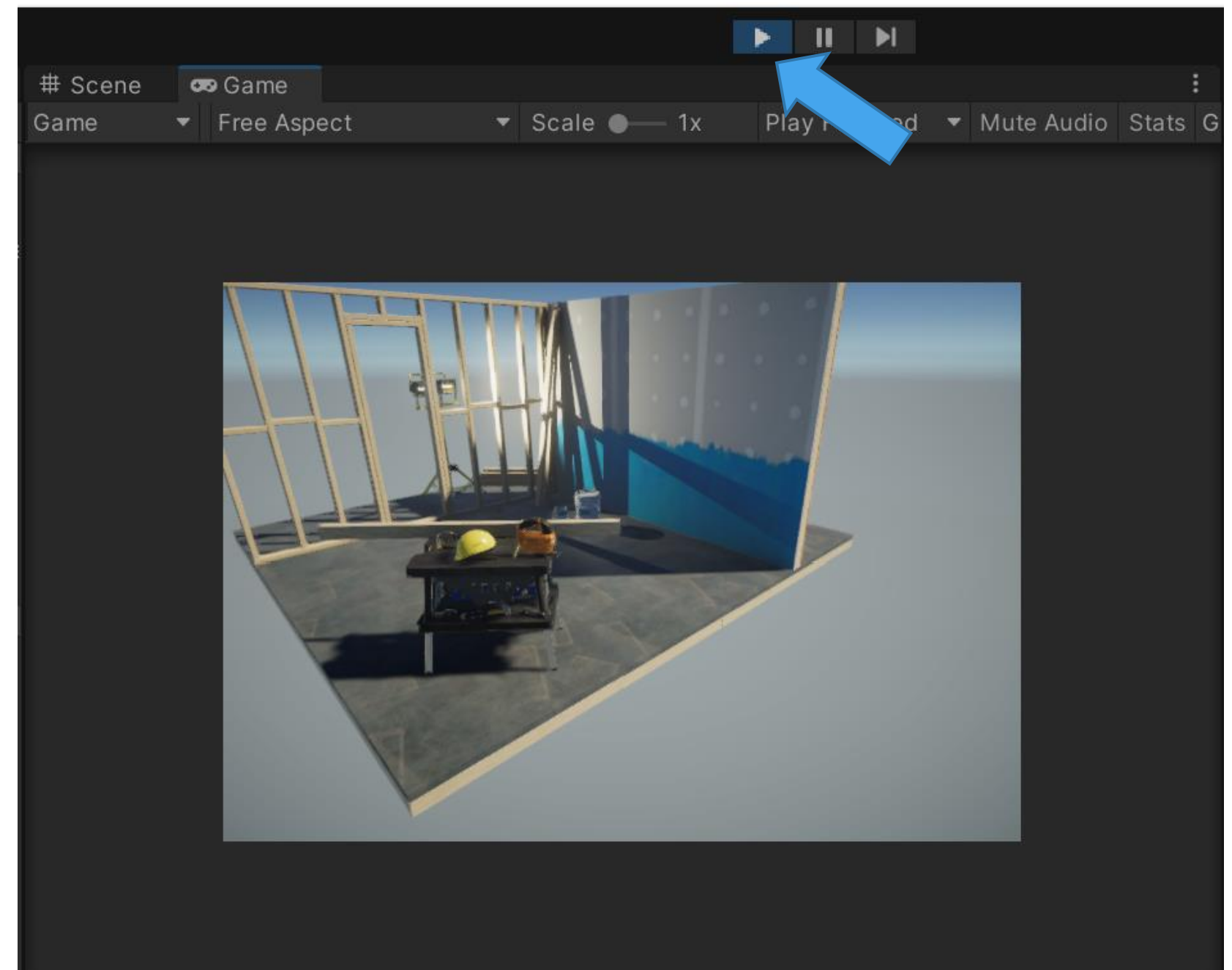
Open and Explore the Project

- Click the middle or right button and drag the mouse to pan and look.
- While holding right mouse button down press: Q,W,E,A,S,D on keyboard to move.



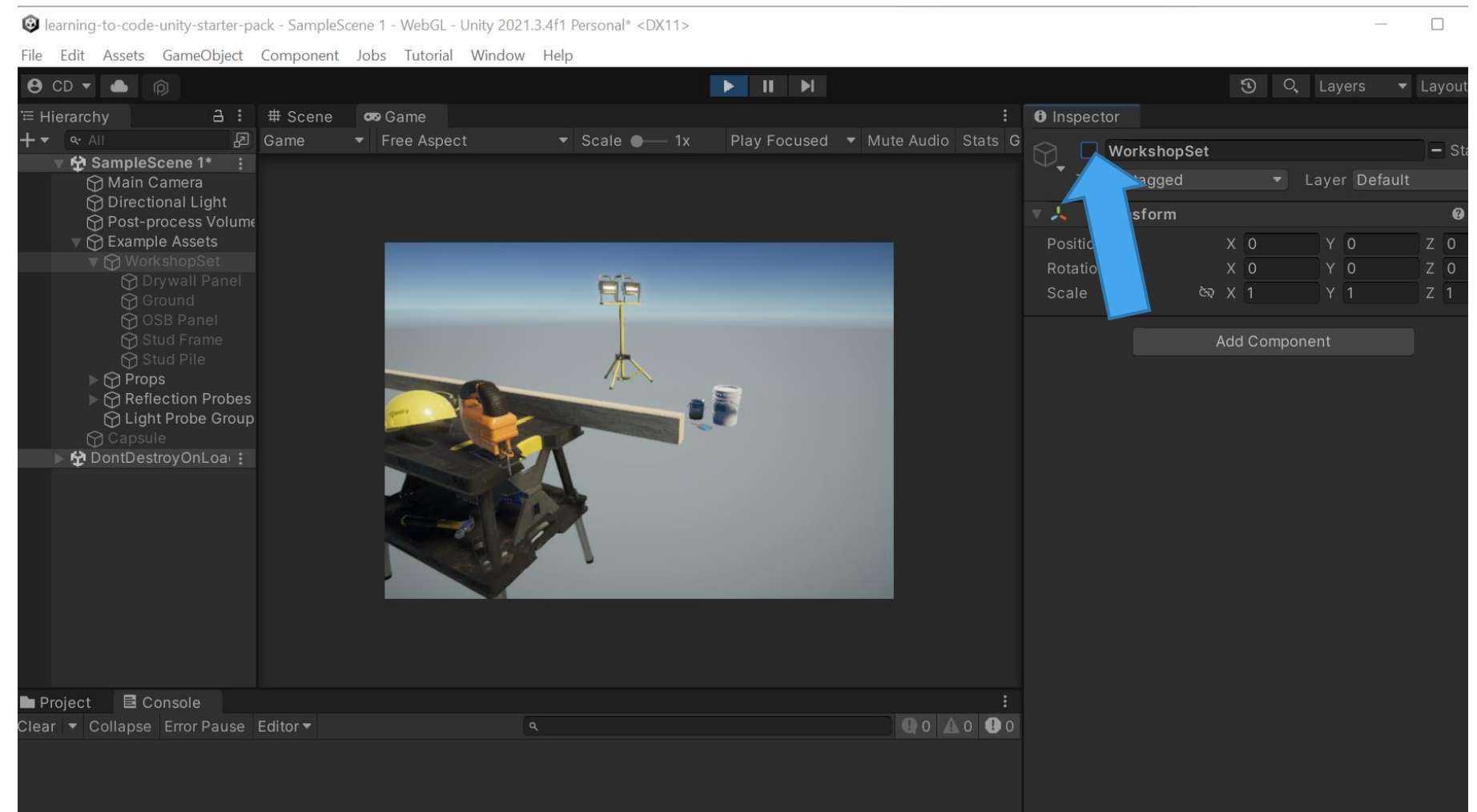
Enter Play Mode

- Press the “play” button to start.
- W,A,S,D,Q,E all work without having to hold down the right mouse button.
- Right click and drag mouse to look around.



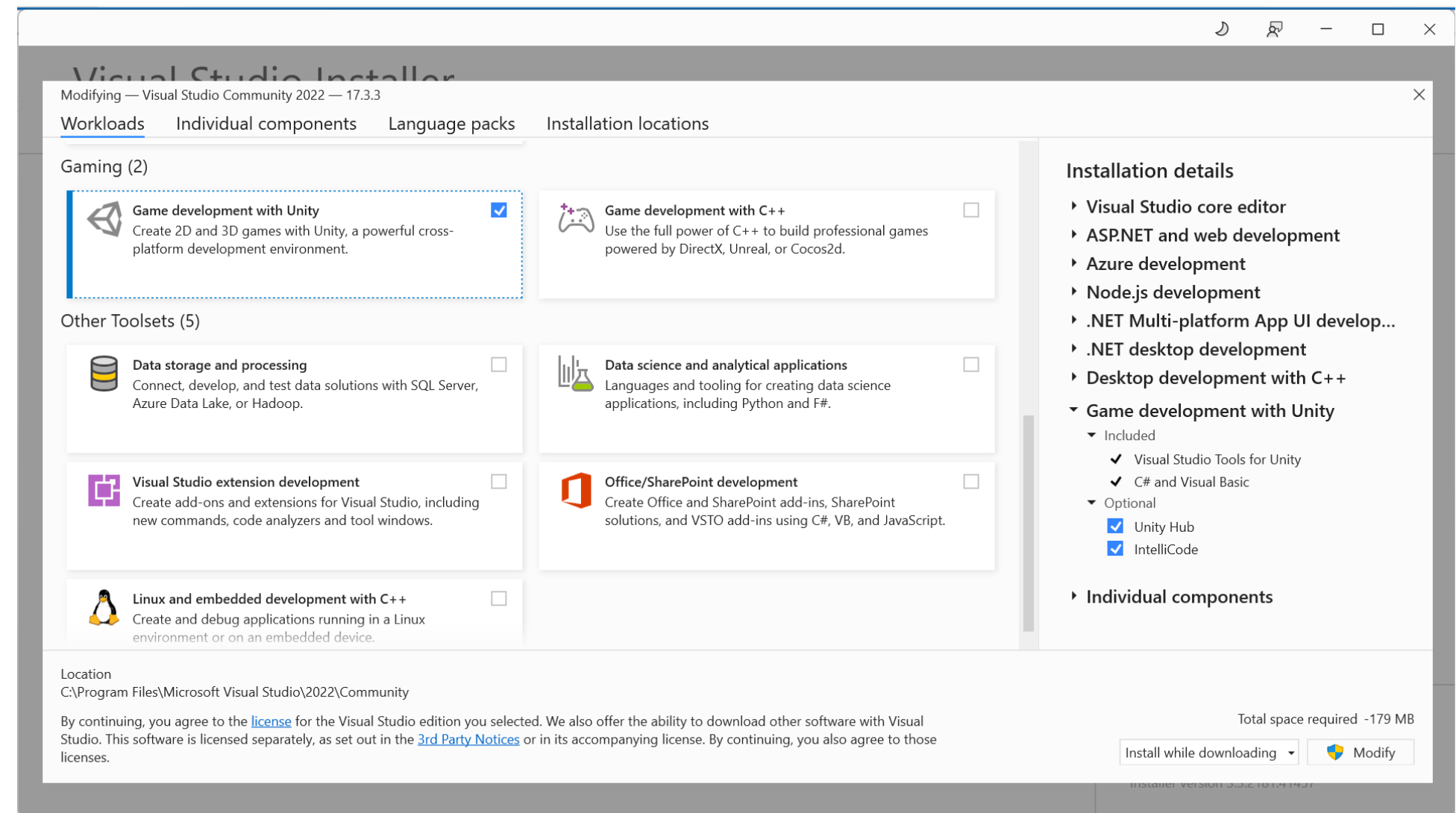
Editing the Scene while Playing

- You can edit the scene while in play mode.
- After stopping those edits are lost.
- Experiment with the scene and see what happens.

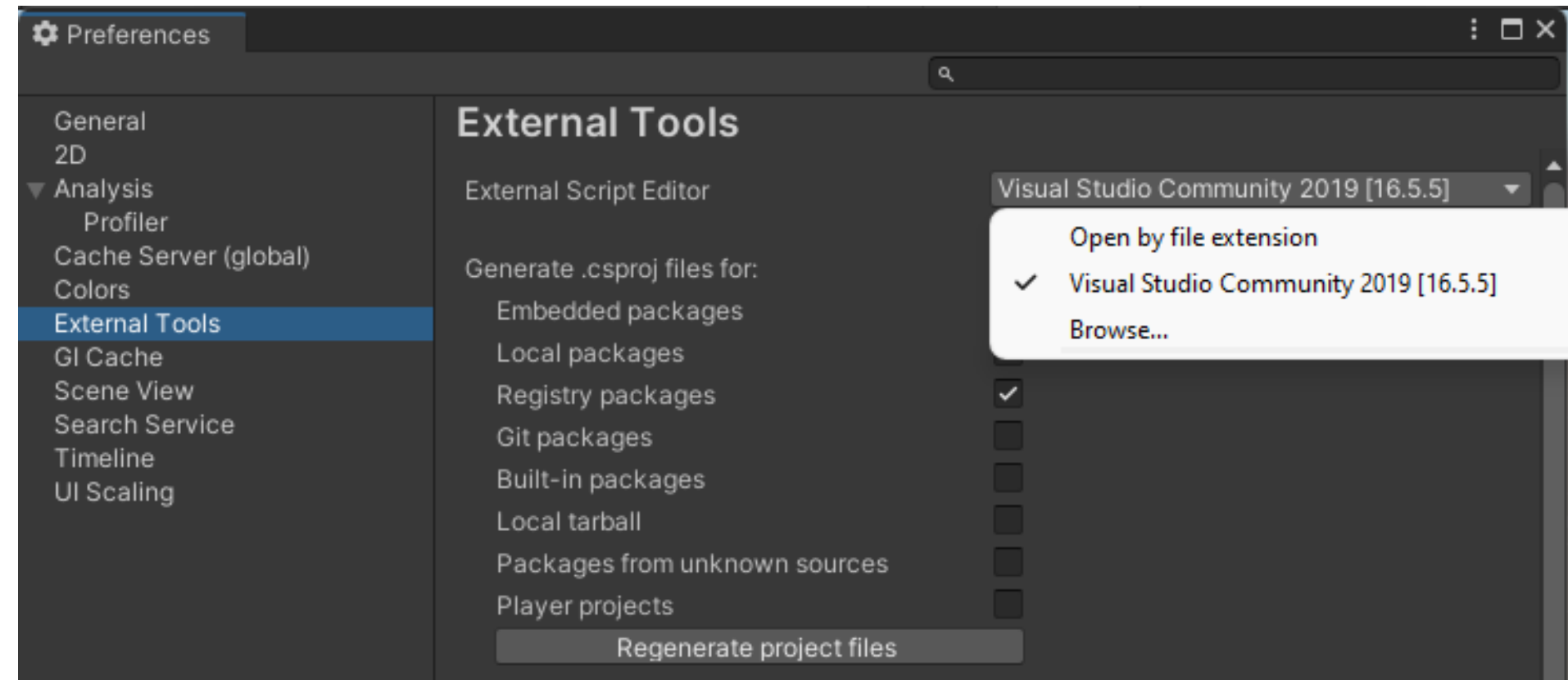
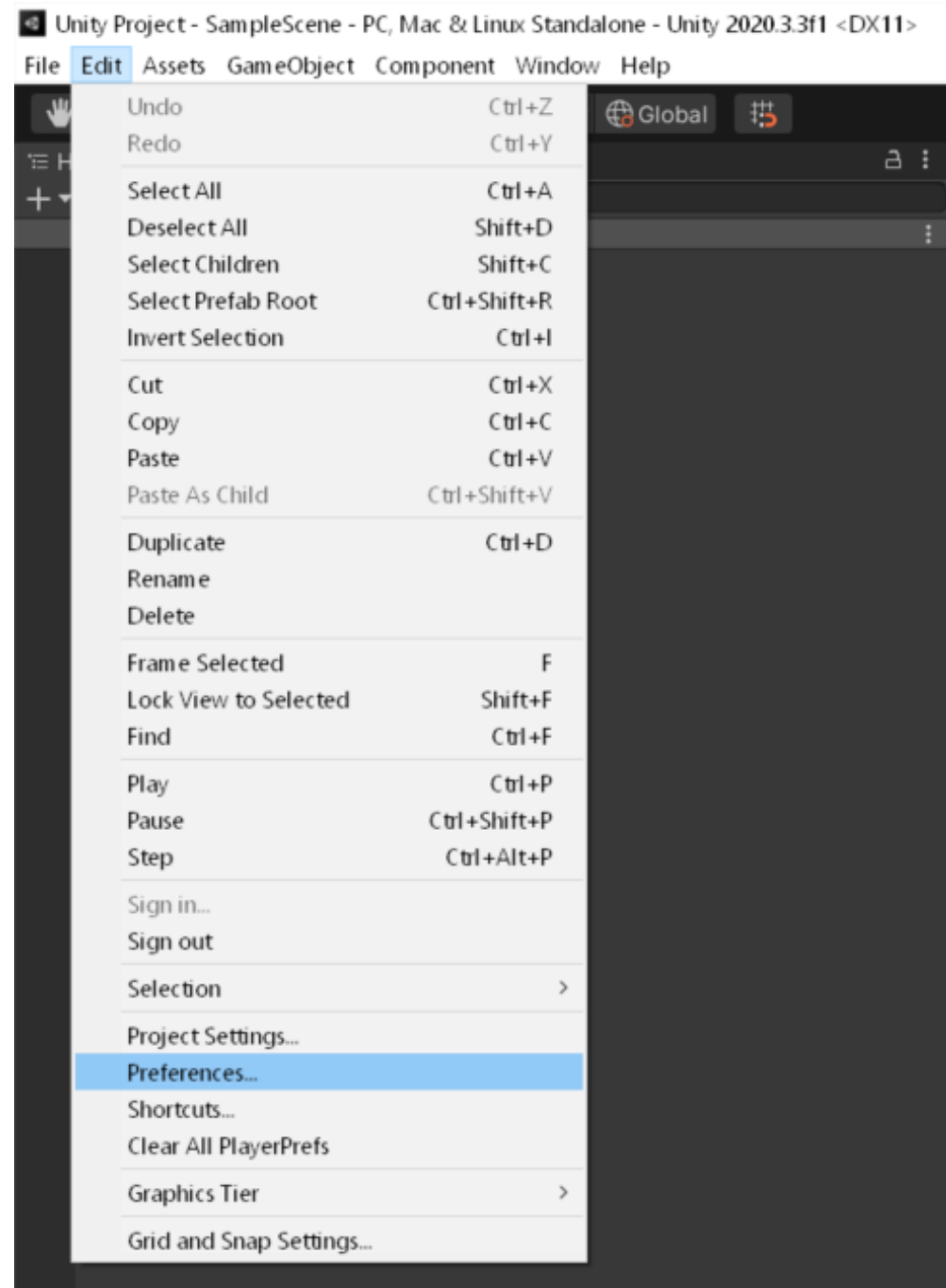


Configure Visual Studio for Unity

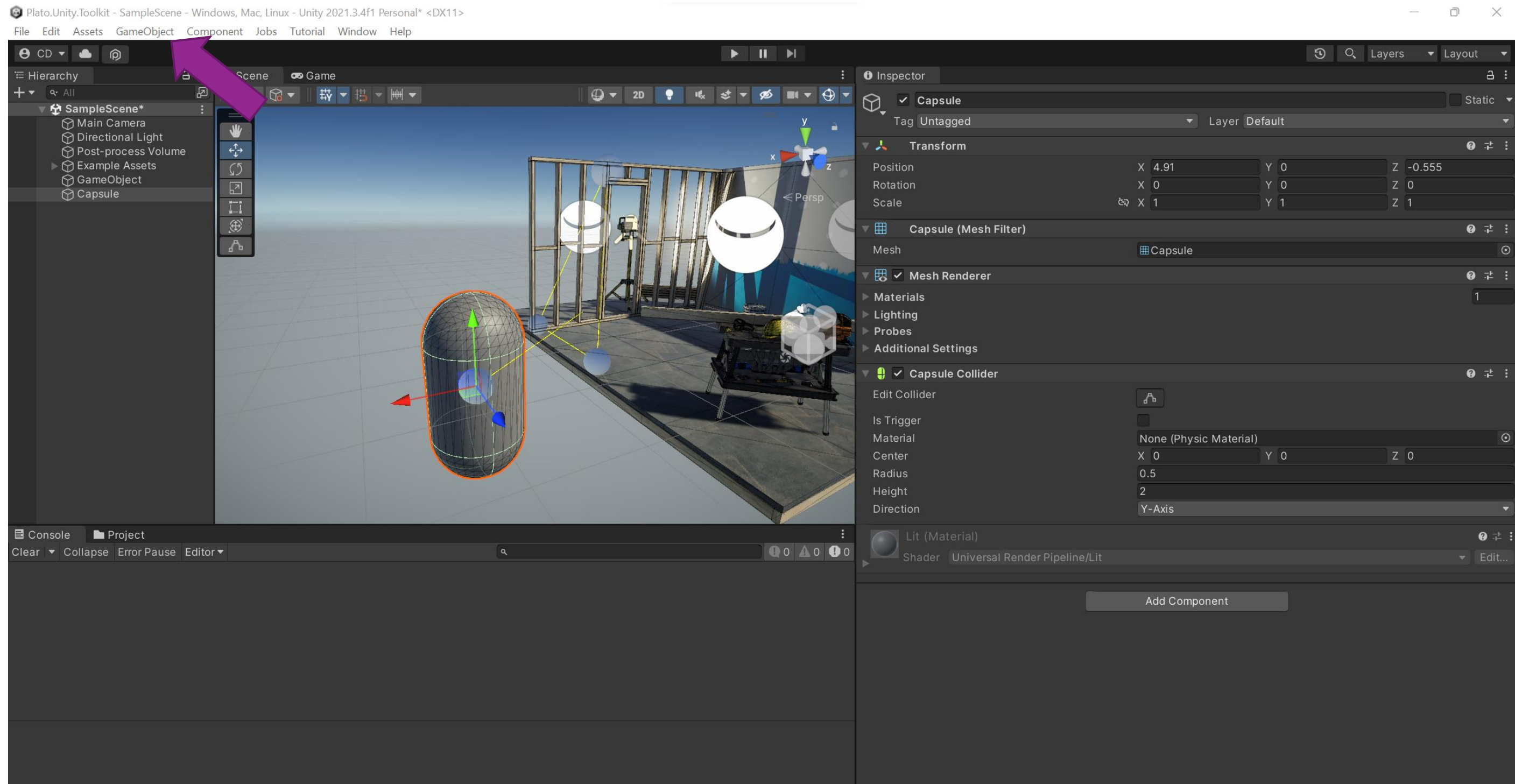
- Launch the Visual Studio Installer.
- Install the “Game development with Unity” workload.



In Unity Configure the Script Editor

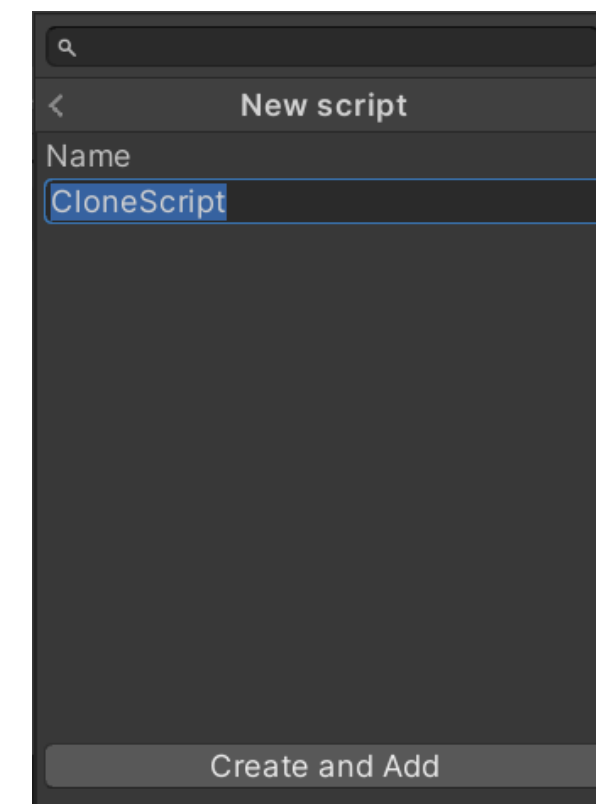
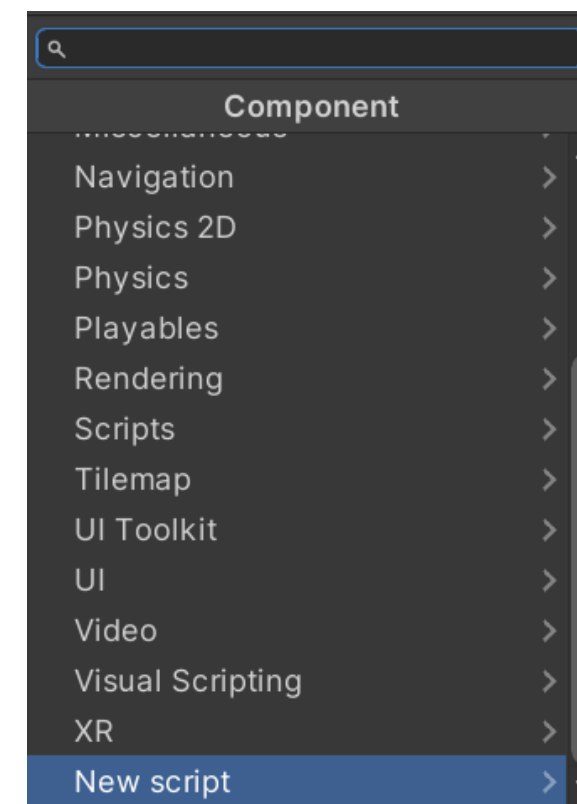
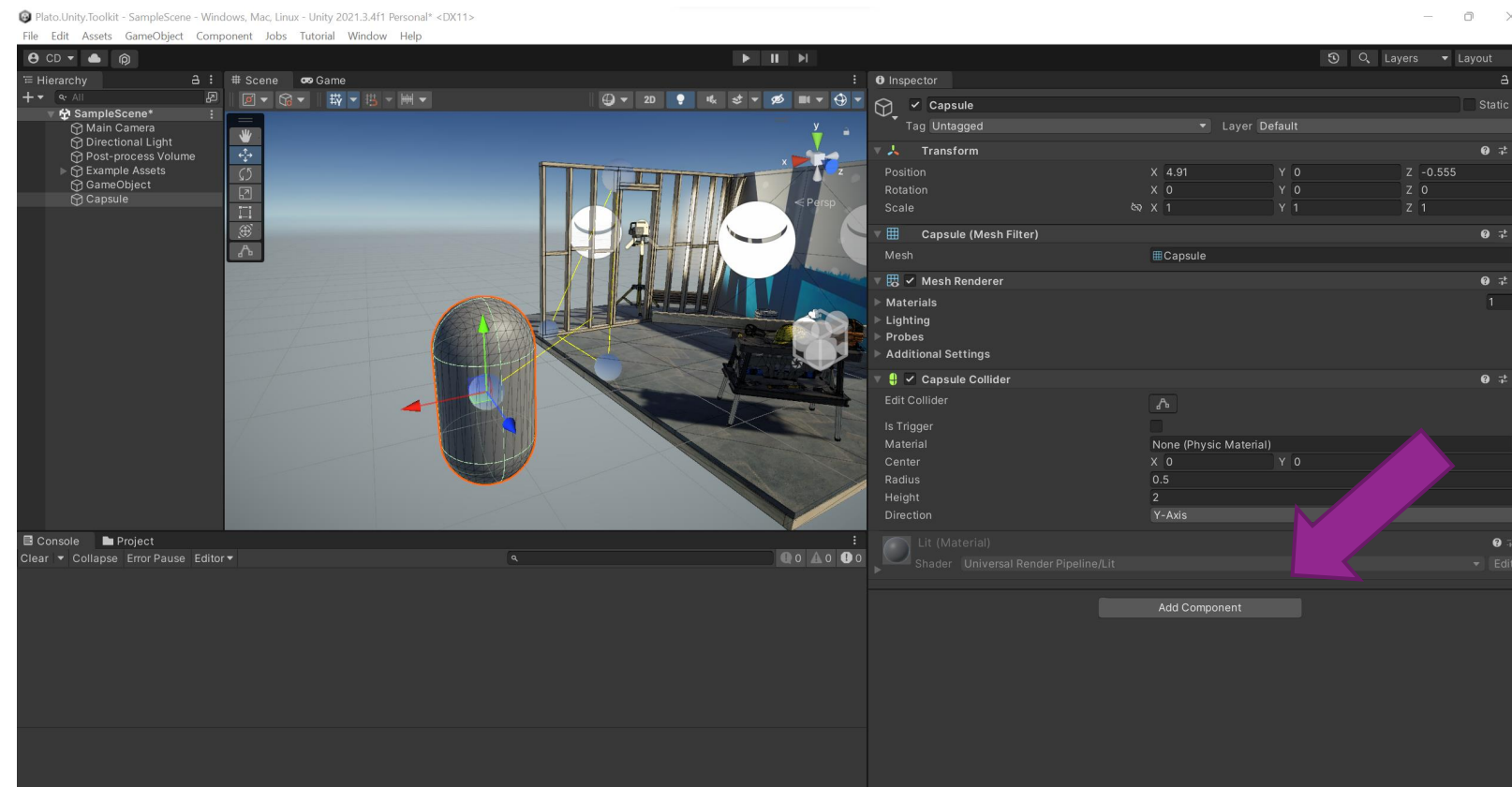


GameObject > 3D Object > Capsule



Select Object and add a New Script

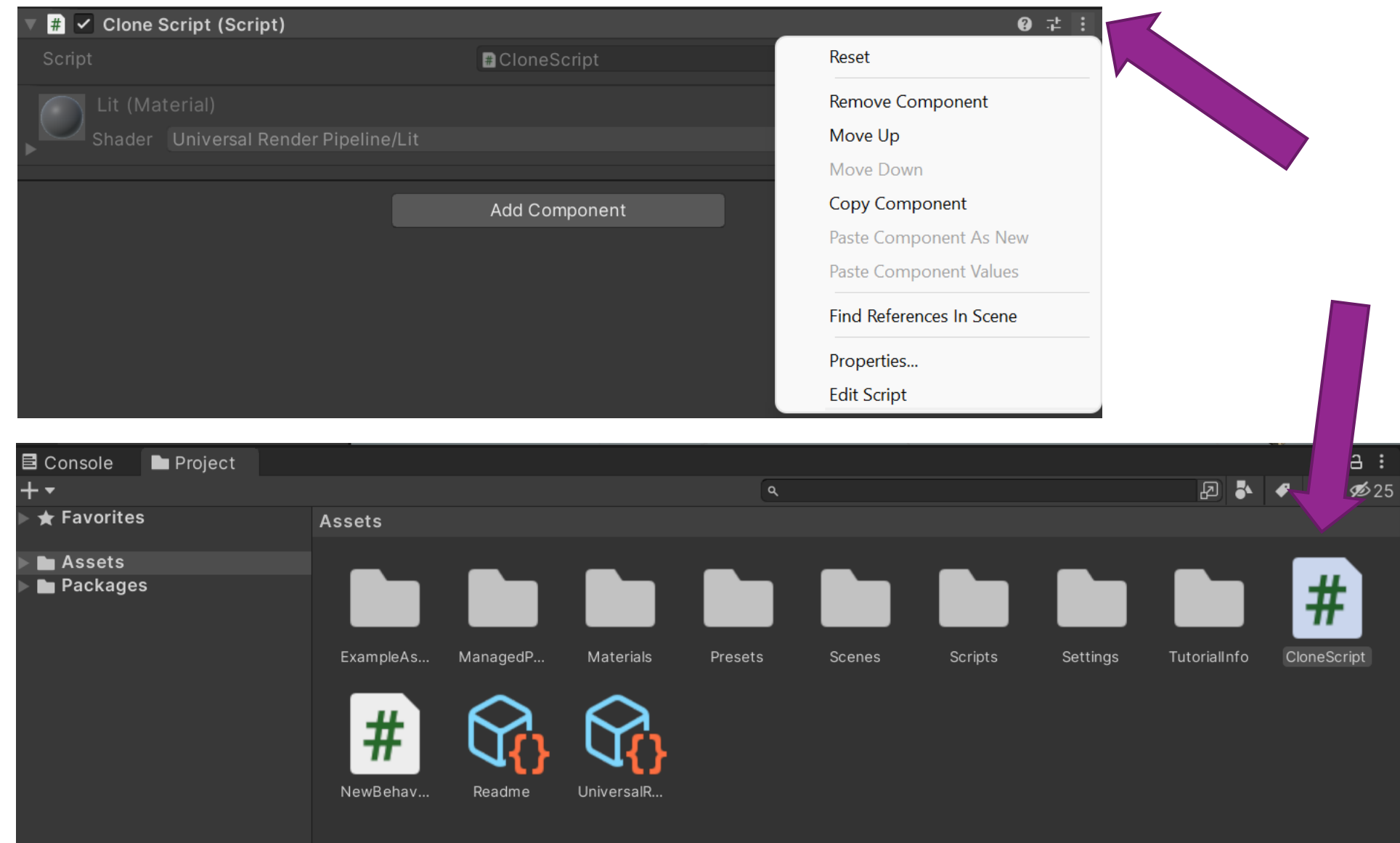
Press "Add Component" choose "New Script" and name the script "CloneScript"



Edit the Script

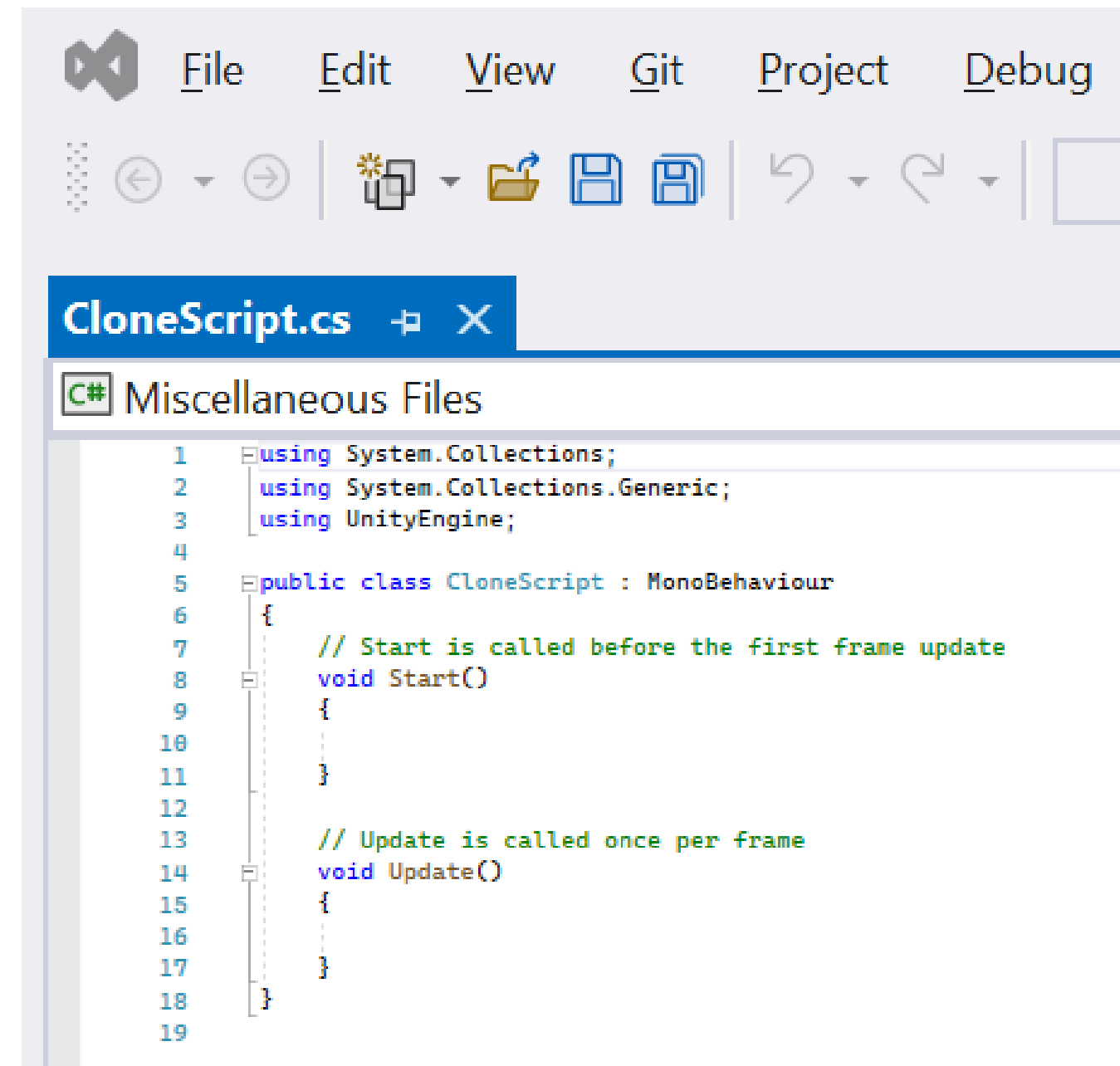
Click the upper right-hand corner of Script Component.

Or double click the script file from project windows.



Script Source Code

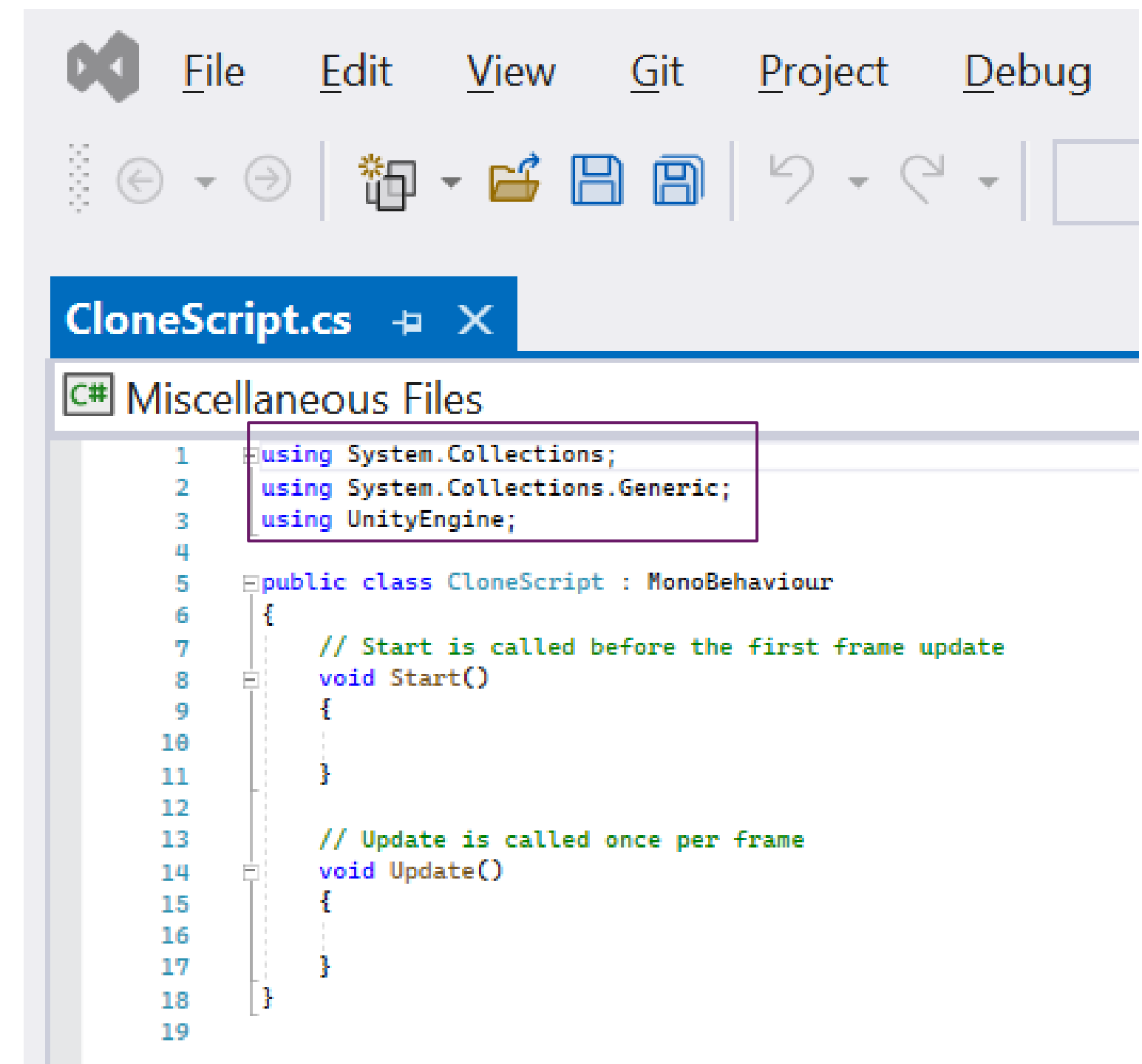
- This is the “source code” of a script.
- A script is a small program, or part of a program, that is executed by a host program.
- Scripts are often interpreted, but not always (e.g., Unity compiles them).
- Coding, programming, and scripting are all the same.



Using Declarations

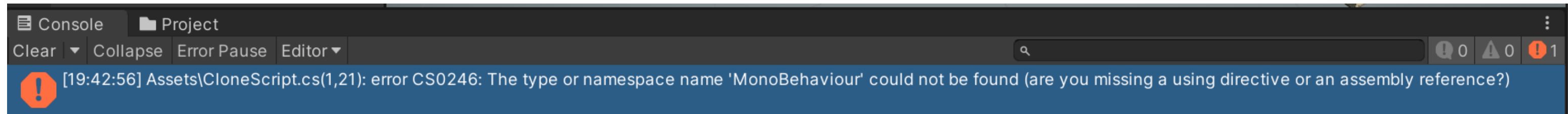
The “using” declaration says what library parts are used in the current file.

- ❑ Extra functionality is provided by libraries.
- ❑ Libraries are organized using “namespaces”.
- ❑ For example: two things called “List” might exist in different libraries.
- ❑ Namespaces disambiguate: “MyLibrary.List” and “System.Collections.List”.
- ❑ Delete them and see what happens.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CloneScript : 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 }
19
```

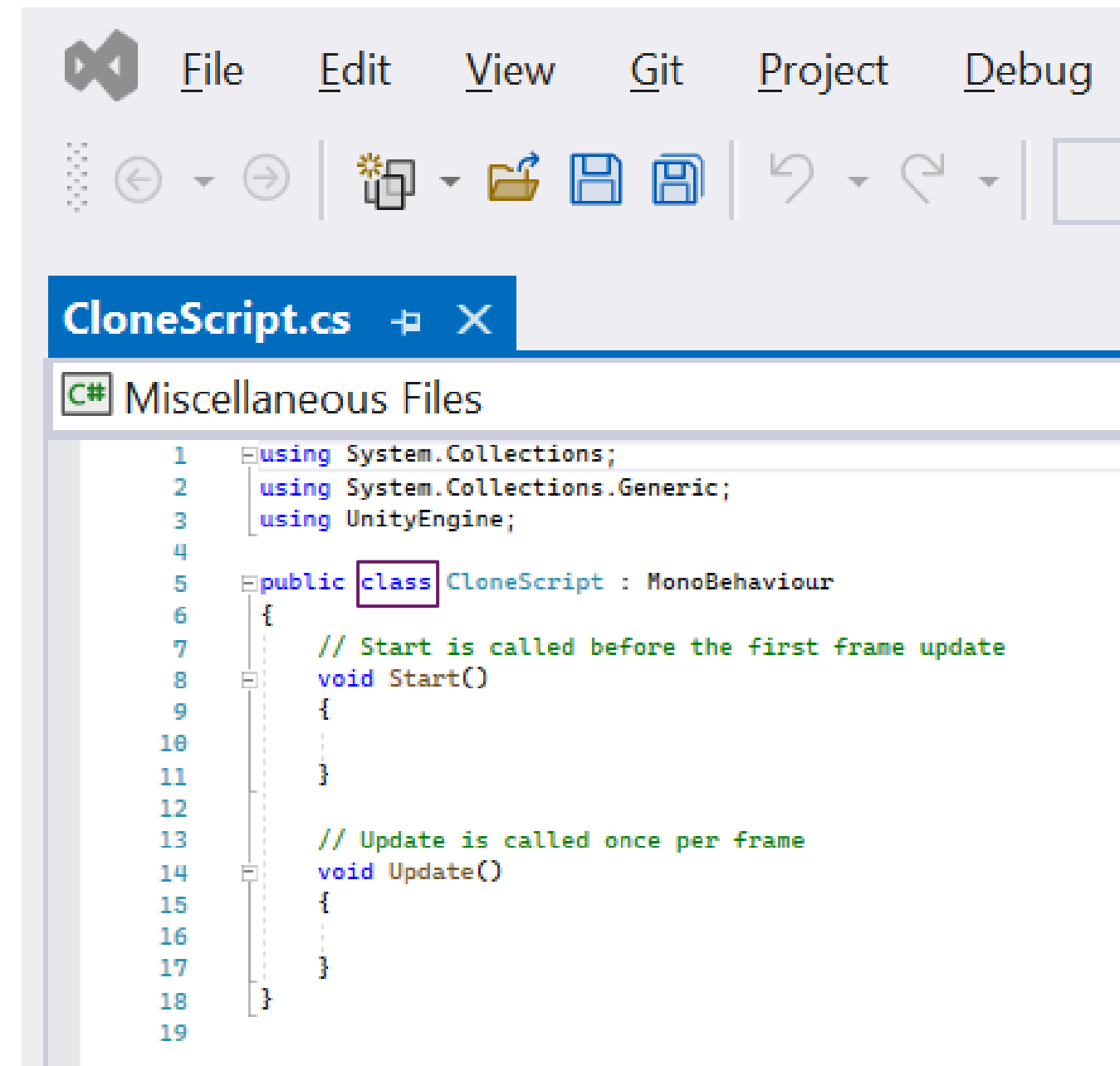

Don't Panic!



- This is a very common type of error, often because of misspelling, or a missing “using directive”.
- Try double-clicking it.
- Now fix the mistake.

Class

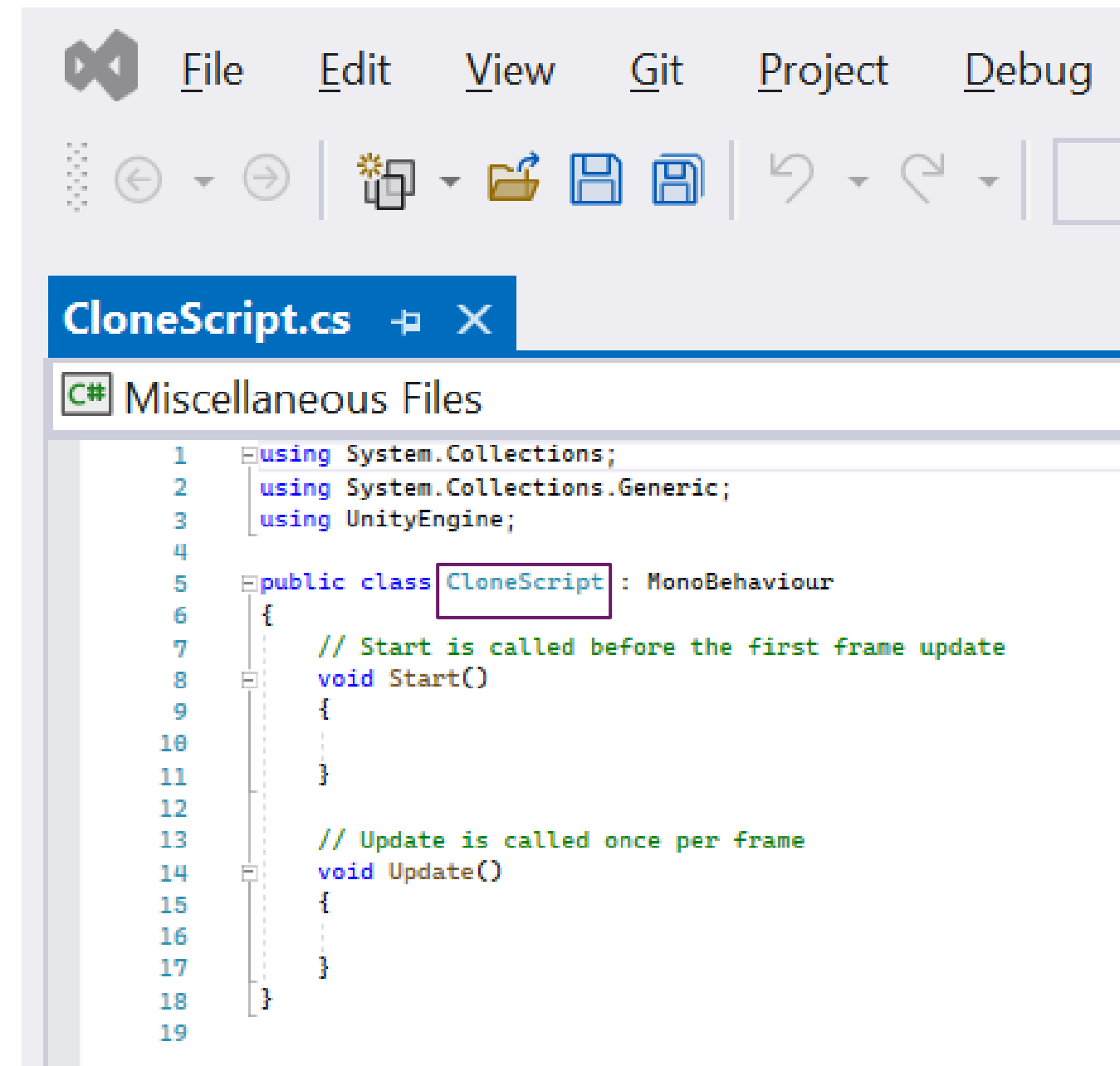
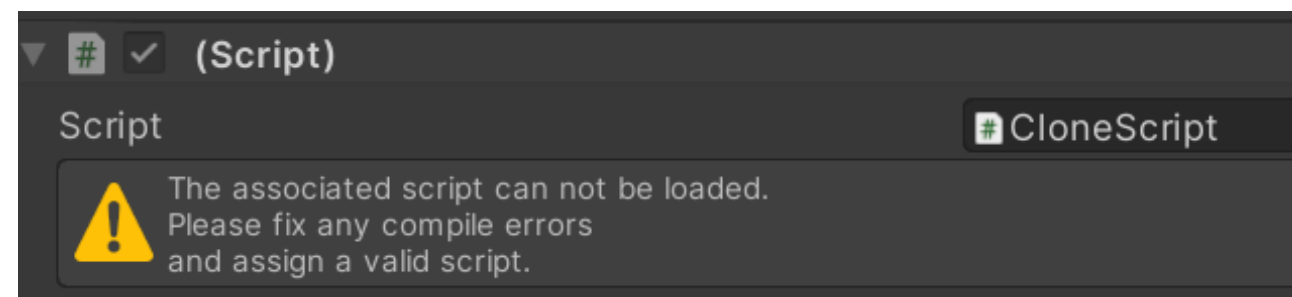
- A class describes a particular type of object.
- Objects contain data and provide operations for accessing or transforming that data.
- This is part of object-oriented programming (OOP) and will be covered in depth on later classes.
- In Unity, all scripts are classes.



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CloneScript : 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 }
19
```

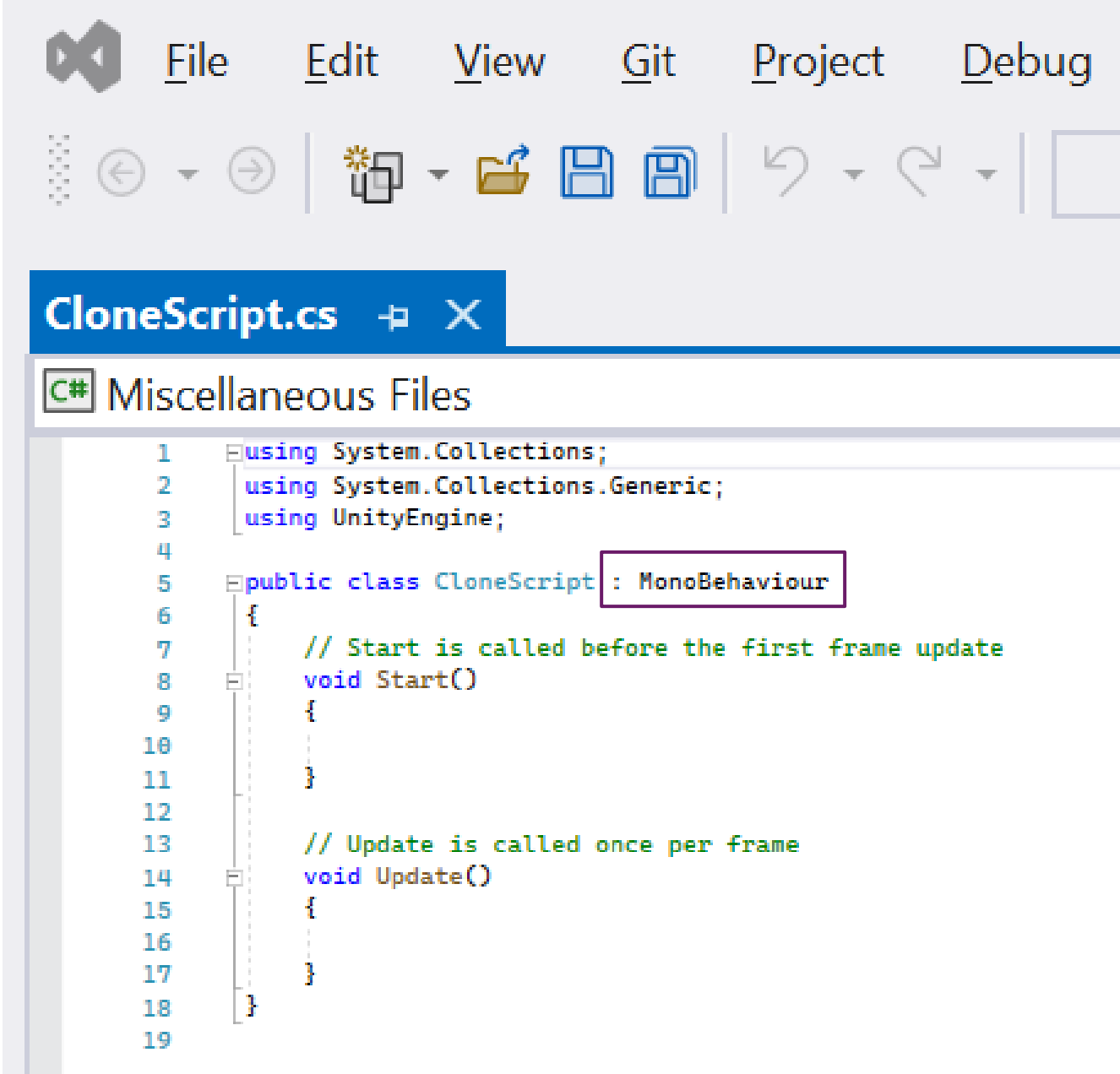
Class Name

- The name of the script class must match the file name.
- Try changing the name.
- You should see this in the inspector:



Inheritance

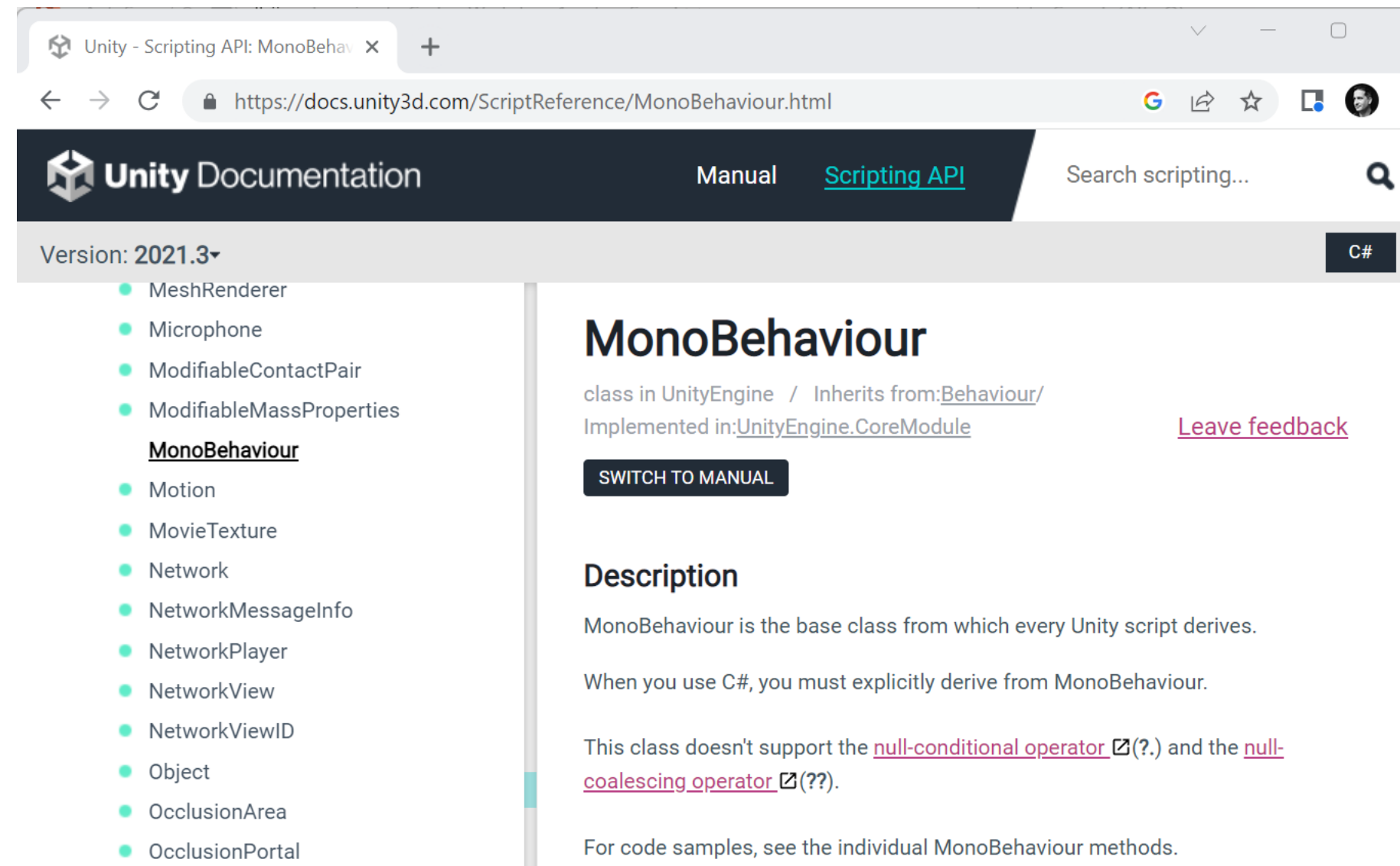
- In Unity all scripts are classes that inherit from a Unity class called "MonoBehavior".
- This means that the things that a MonoBehavior can do, and the values that it contains, are accessible to the new class.
- This is called inheritance and is a feature of object-oriented programming (OOP).



The screenshot shows the Visual Studio IDE with a C# script named `CloneScript.cs` open. The script is part of a project named "Miscellaneous Files". The code defines a public class `CloneScript` that inherits from `MonoBehaviour`. The class contains two methods: `Start()` and `Update()`. The `Start()` method is commented as being called before the first frame update, and the `Update()` method is commented as being called once per frame. The code is as follows:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CloneScript : 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 }
19
```

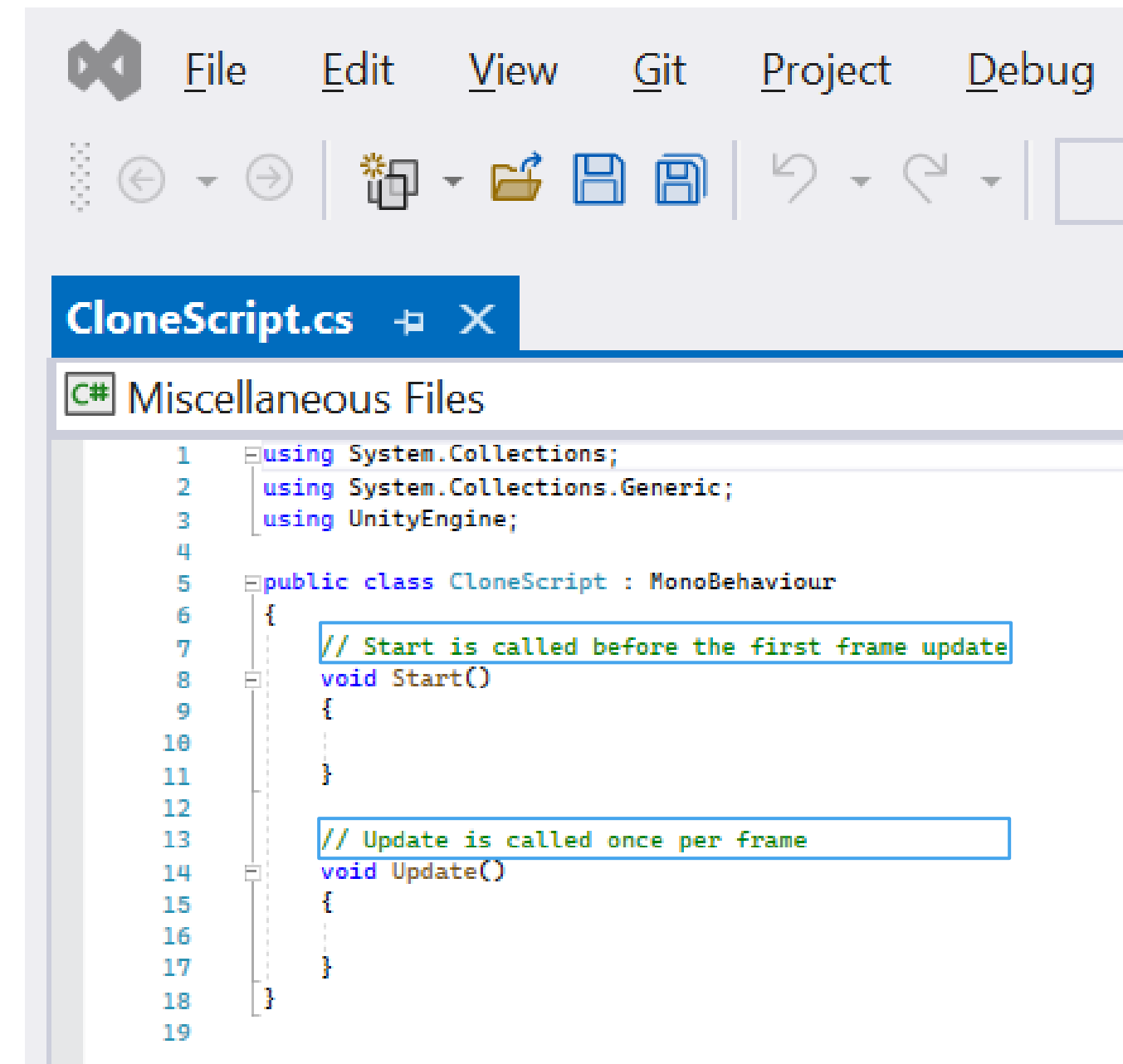
Read the Docs, Luke!



<https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

Comments

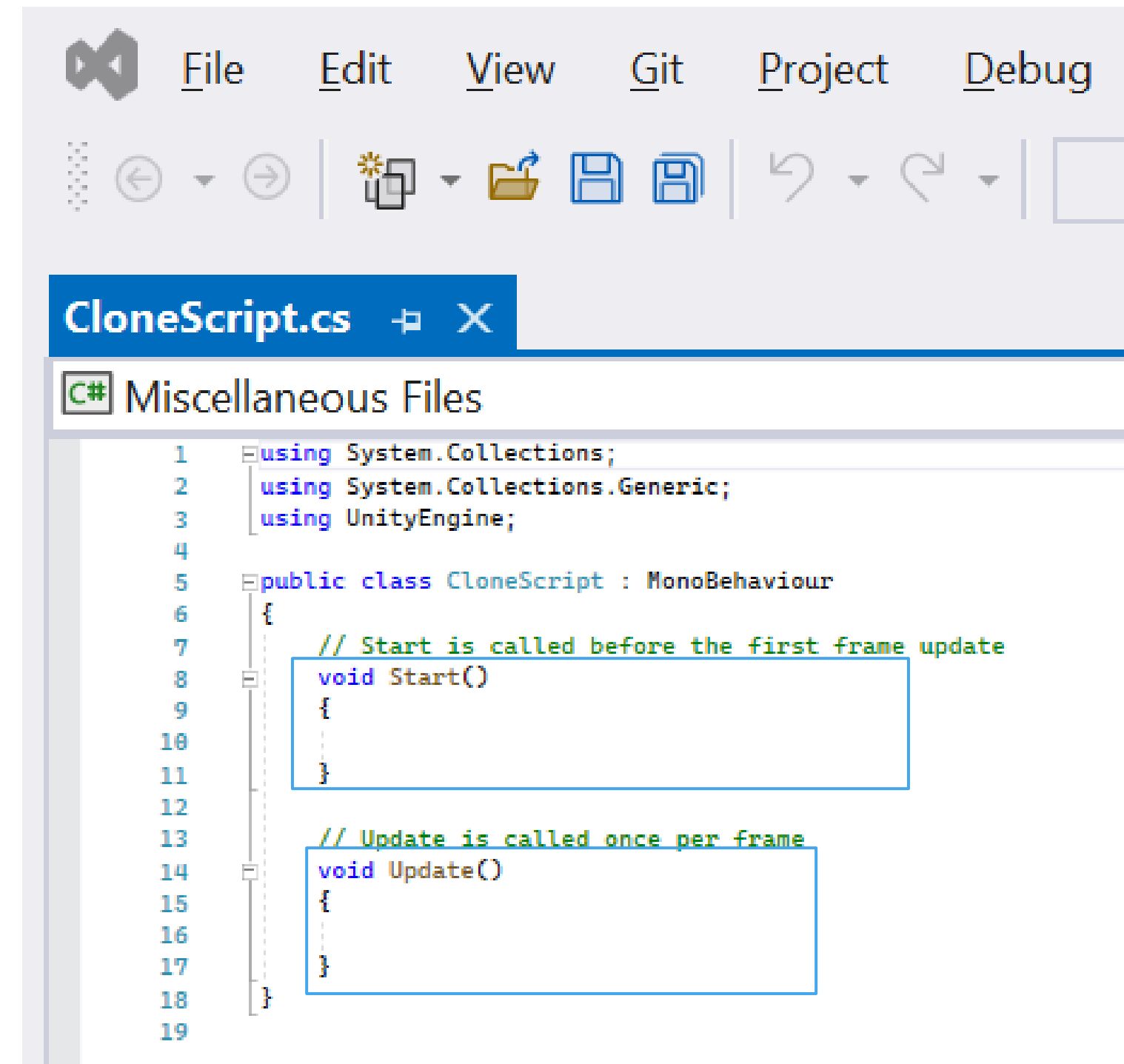
- Comments are for humans and have no impact on programs.
- The compiler ignores everything from the “//” until the end of the line.
- Try deleting them.
- Try commenting things out.



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CloneScript : 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  }
19
```

Functions

- The start function is called by Unity once when the game starts.
- The update function is called by Unity every frame of the game.
- They are not called when the game object is deactivated.



The screenshot shows the Visual Studio IDE with the file 'CloneScript.cs' open. The file is a C# script for Unity, inheriting from 'MonoBehaviour'. It contains two methods: 'Start()' and 'Update()'. The 'Start()' method is annotated with a comment indicating it is called before the first frame update. The 'Update()' method is annotated with a comment indicating it is called once per frame. The code is as follows:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CloneScript : 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 }
19
```

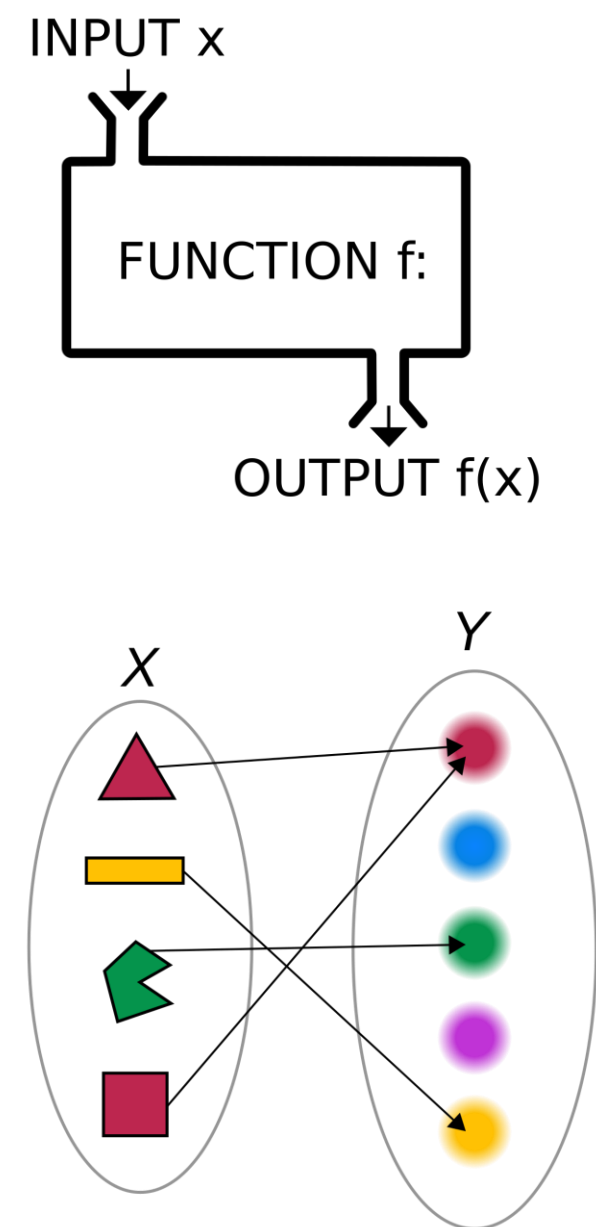
Functions in Code versus Mathematics

In Mathematics, a function from a set X to a set Y assigns to each element of X exactly one element of Y .

In computer languages what are called “functions” are actually “subroutines”.

Unlike mathematical functions:

- ❑ They might not return a value (usually labeled as “void”).
- ❑ They might have zero, one, or more inputs.
- ❑ They might (and often do) have side-effects (e.g., writing to the console).
- ❑ Calling them with the same input might yield different results.

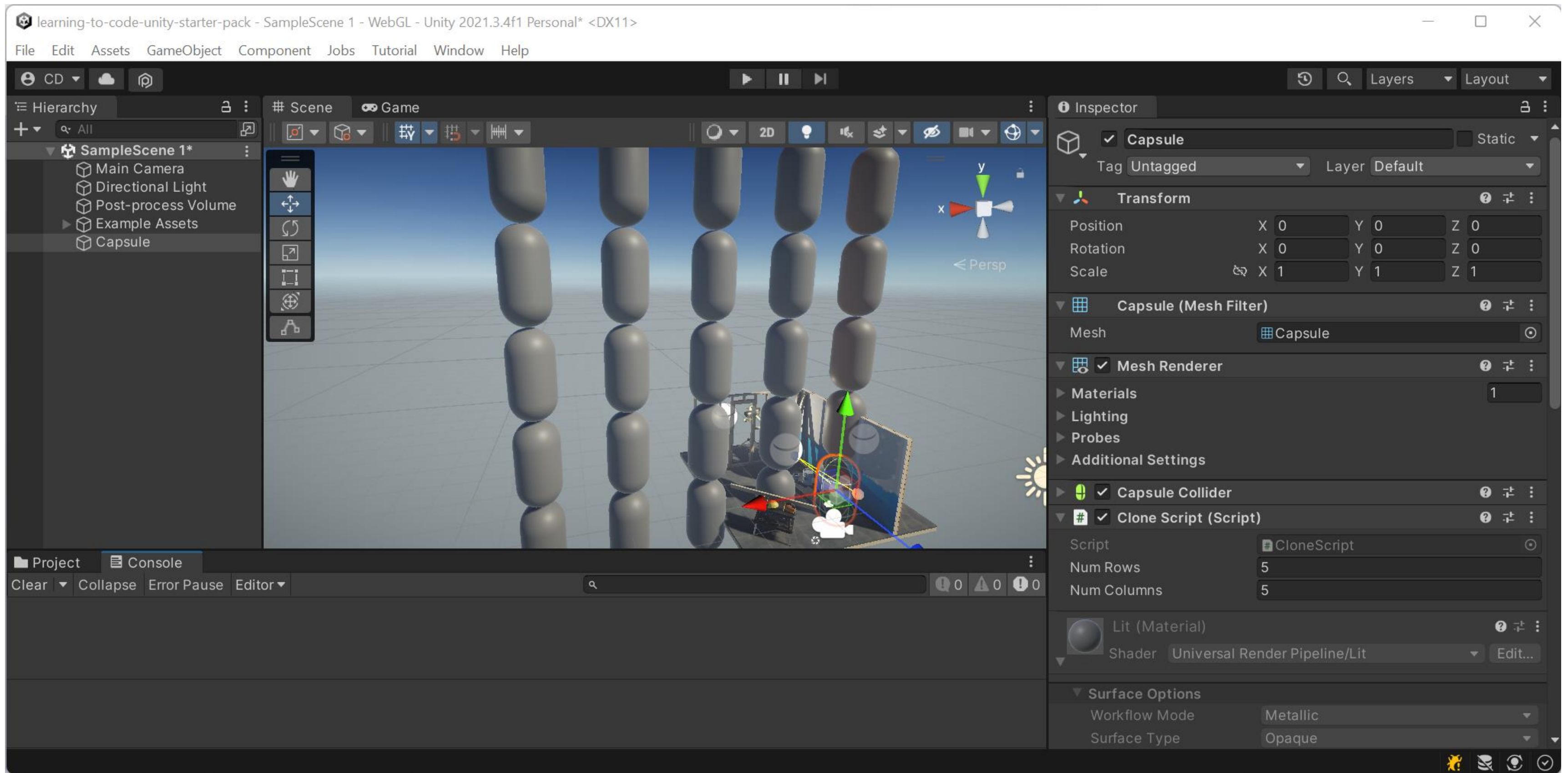


Type in your First Script

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

You really should practice typing it in!

- Making mistakes and fixing them, is a big part of learning to code.
- That said, I posted a fully documented version you can cut and paste here:
<https://gist.github.com/cdiggins/8cagf928a463145b9e186ab707dfbfb7>



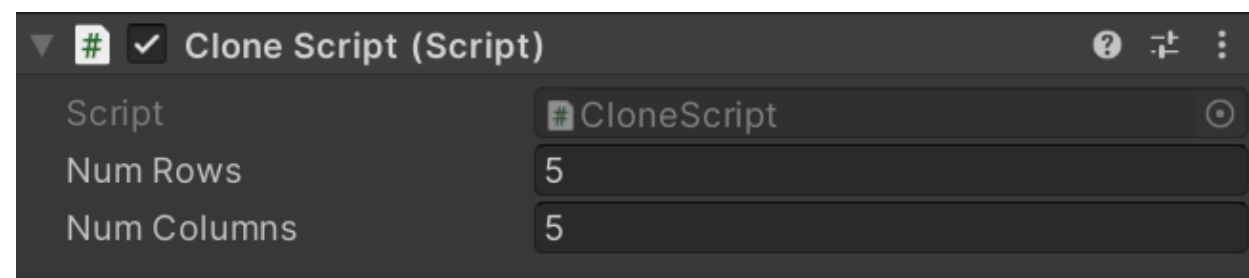
Attribute

- This is meta-information for the compiler or host environment.
- This attribute tells the script to run in editor even when the game is not playing.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Fields

- A field is named data associated with an object. Also known as a “member variable”.
- When declared as “public” exposed in the editor as a property.



```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Variable Declaration

- A variable is a name associated with a value.
- The value associated with the name can change.
- However, the type of data (number, text, vector) associated can't be changed.
- You can use the variable name instead of expression.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Variables are your friend

This is equivalent to the other script – which do you prefer?

```
using UnityEngine;

[ExecuteInEditMode]
public class CloneScriptWithVars : MonoBehaviour
{
    public int NumRows = 5;
    public int NumColumns = 5;

    public void Update()
    {
        var mesh = GetComponent<MeshFilter>().sharedMesh;
        var material = GetComponent<MaterialFilter>().sharedMaterial;

        for (var column = 0; column < NumColumns; ++column)
        {
            for (var row = 0; row < NumRows; ++row)
            {
                var position = new Vector3(column * 2, row * 2, 0);
                var rotation = Quaternion.identity;
                var layer = 0;
                Graphics.DrawMesh(mesh, position, rotation, material, layer);
            }
        }
    }
}
```

Expressions

A sequence of symbols (operators, numbers, variables) that represent computations. They are transformed into values (evaluated) when the program is executed.

Some examples include:

- Numbers: `42`, `3.15`
- Booleans: `true`, `false`
- Variables: `x`, `MyVector`, `this_is_a_variable`
- An operation with operands: `x + 1`, `y >= 12`, `-z`
- Parenthesized expression: `3 * (y - 2)`
- A function call: `Math.Sqrt(16)`
- A member variable: `this.NumRows`

Practice: Find the Expressions

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Condition

- An expression (operation / function call / constant) with a value of type Boolean (true or false).
- In the context of a for loop is called the invariant.
- Loop is executed while condition is true.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Increment Statement

- Adds one to a variable.
- Returns the value of a variable after adding one to the variable.
- The same as "x = x + 1".
- In the context of a for loop, it is called after each loop iteration.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Block Statements

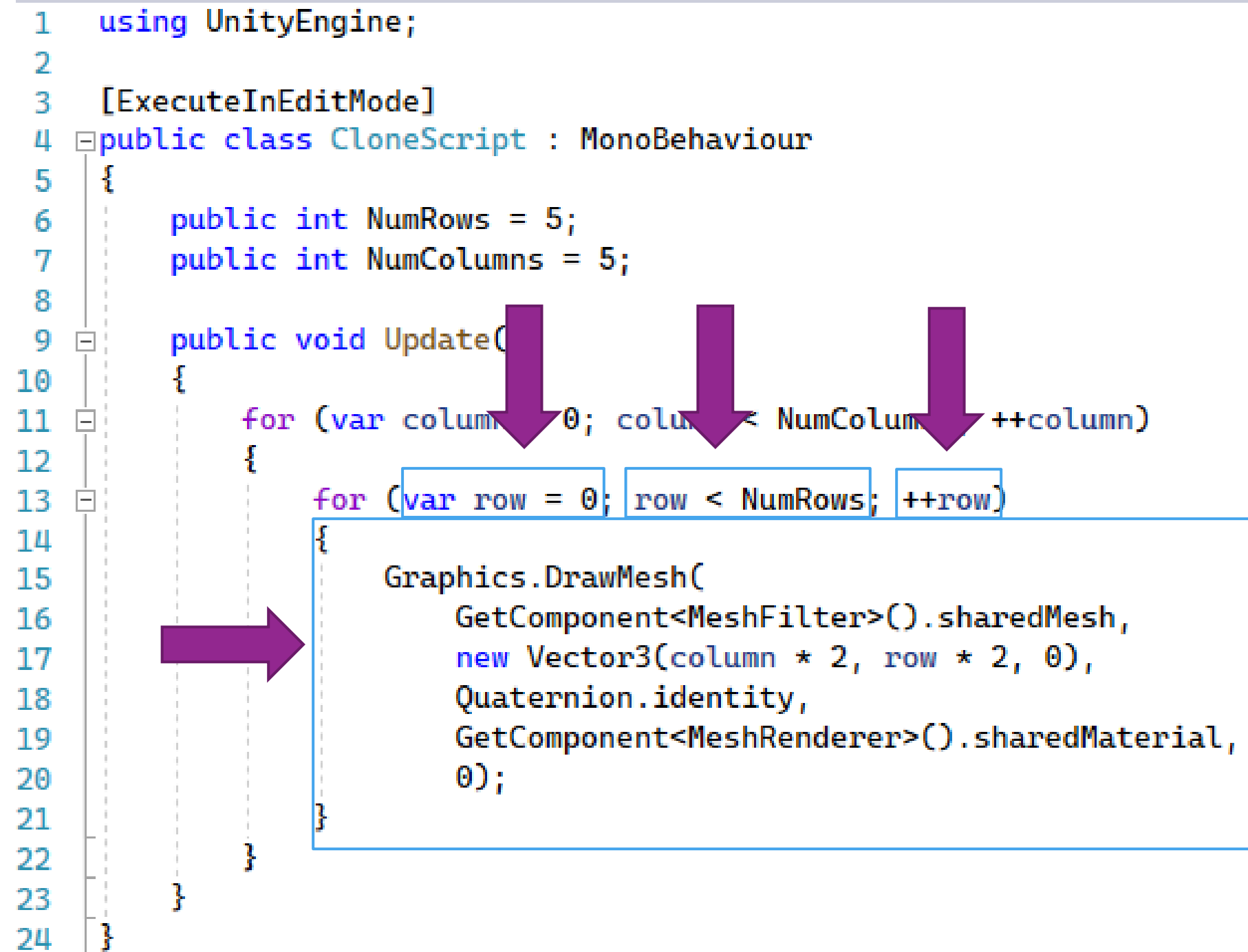
- Also called a **compound statement**.
- Allows any number of statements (0 or more) to be treated like one statement.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

For Loop Statement

- Calls the next statement (the loop body) multiple times.
- Executes an initialization statement before starting.
- Only executes while the invariant is true.
- After each loop iteration, calls an iteration statement.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```



Why are Loops Important?

The computer doesn't care ... you could write out a statement 25 times.

What could possibly go wrong with this approach?

An important thing to ask yourself frequently when programming.

```
1 using UnityEngine;
2
3 [ExecuteInEditMode]
4 public class CloneScript : MonoBehaviour
5 {
6     public void Update()
7     {
8         var mesh = GetComponent<MeshFilter>().sharedMesh;
9         var material = GetComponent<MeshRenderer>().sharedMaterial;
10
11         // Row 1
12         Graphics.DrawMesh(mesh, new Vector3(0, 0), Quaternion.identity, material, 0);
13         Graphics.DrawMesh(mesh, new Vector3(0, 2), Quaternion.identity, material, 0);
14         Graphics.DrawMesh(mesh, new Vector3(0, 4), Quaternion.identity, material, 0);
15         Graphics.DrawMesh(mesh, new Vector3(0, 6), Quaternion.identity, material, 0);
16         Graphics.DrawMesh(mesh, new Vector3(0, 8), Quaternion.identity, material, 0);
17
18         // Row 2
19         Graphics.DrawMesh(mesh, new Vector3(2, 0), Quaternion.identity, material, 0);
20         Graphics.DrawMesh(mesh, new Vector3(2, 2), Quaternion.identity, material, 0);
21         Graphics.DrawMesh(mesh, new Vector3(2, 4), Quaternion.identity, material, 0);
22         Graphics.DrawMesh(mesh, new Vector3(2, 6), Quaternion.identity, material, 0);
23         Graphics.DrawMesh(mesh, new Vector3(2, 8), Quaternion.identity, material, 0);
24
25         // Row 3
26         Graphics.DrawMesh(mesh, new Vector3(4, 0), Quaternion.identity, material, 0);
27         Graphics.DrawMesh(mesh, new Vector3(4, 2), Quaternion.identity, material, 0);
28         Graphics.DrawMesh(mesh, new Vector3(4, 4), Quaternion.identity, material, 0);
29         Graphics.DrawMesh(mesh, new Vector3(4, 6), Quaternion.identity, material, 0);
30         Graphics.DrawMesh(mesh, new Vector3(4, 8), Quaternion.identity, material, 0);
31
32         // Row 4
33         Graphics.DrawMesh(mesh, new Vector3(6, 0), Quaternion.identity, material, 0);
34         Graphics.DrawMesh(mesh, new Vector3(6, 2), Quaternion.identity, material, 0);
35         Graphics.DrawMesh(mesh, new Vector3(6, 4), Quaternion.identity, material, 0);
36         Graphics.DrawMesh(mesh, new Vector3(6, 6), Quaternion.identity, material, 0);
37         Graphics.DrawMesh(mesh, new Vector3(6, 8), Quaternion.identity, material, 0);
38
39         // Row 5
40         Graphics.DrawMesh(mesh, new Vector3(8, 0), Quaternion.identity, material, 0);
41         Graphics.DrawMesh(mesh, new Vector3(8, 2), Quaternion.identity, material, 0);
42         Graphics.DrawMesh(mesh, new Vector3(8, 4), Quaternion.identity, material, 0);
43         Graphics.DrawMesh(mesh, new Vector3(8, 6), Quaternion.identity, material, 0);
44         Graphics.DrawMesh(mesh, new Vector3(8, 8), Quaternion.identity, material, 0);
45     }
46 }
```


Function Calls

- An expression or statement that executes a function.
- If it returns a value, can be used as an expression.
- Accepts types and/or expressions as inputs.

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void Update()
10     {
11         for (var column = 0; column < NumColumns; ++column)
12         {
13             for (var row = 0; row < NumRows; ++row)
14             {
15                 Graphics.DrawMesh(
16                     GetComponent<MeshFilter>().sharedMesh,
17                     new Vector3(column * 2, row * 2, 0),
18                     Quaternion.identity,
19                     GetComponent<MeshRenderer>().sharedMaterial,
20                     0);
21             }
22         }
23     }
24 }
```

Functions

- Functions are the fundamental building blocks of a computer program.
- Also known as procedures, subroutines, or methods.
- Let's write a function that draws the mesh at a specified location
- Consider other ways we could have written it

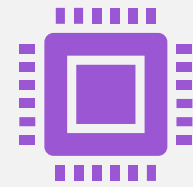
```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void DrawMesh(float x, float y)
10     {
11         var mesh = GetComponent<MeshFilter>().sharedMesh;
12         var material = GetComponent<MeshRenderer>().sharedMaterial;
13         Graphics.DrawMesh(mesh, new Vector3(x, y), Quaternion.identity, material, 0)
14     }
15
16     public void Update()
17     {
18         for (var column = 0; column < NumColumns; ++column)
19         {
20             for (var row = 0; row < NumRows; ++row)
21             {
22                 DrawMesh(column * 2, row * 2);
23             }
24         }
25     }
26 }
```

Statements

Statements are sequences of instructions. They might declare something, execute a subroutine, define a loop

```
1  using UnityEngine;
2
3  [ExecuteInEditMode]
4  public class CloneScript : MonoBehaviour
5  {
6      public int NumRows = 5;
7      public int NumColumns = 5;
8
9      public void DrawMesh(float x, float y)
10     {
11         var mesh = GetComponent<MeshFilter>().sharedMesh;
12         var material = GetComponent<MeshRenderer>().sharedMaterial;
13         Graphics.DrawMesh(mesh, new Vector3(x, y), Quaternion.identity, material, 0)
14     }
15
16     public void Update()
17     {
18         for (var column = 0; column < NumColumns; ++column)
19         {
20             for (var row = 0; row < NumRows; ++row)
21             {
22                 DrawMesh(column * 2, row * 2);
23             }
24         }
25     }
26 }
```

DRY - Don't Repeat Yourself



My favorite guiding principle for programming.



Repetition is a sign that your code is going to be hard to change or reuse.



Usually indicates an opportunity to use variables or functions.

Self-Guided Learning

Coding Exercises:

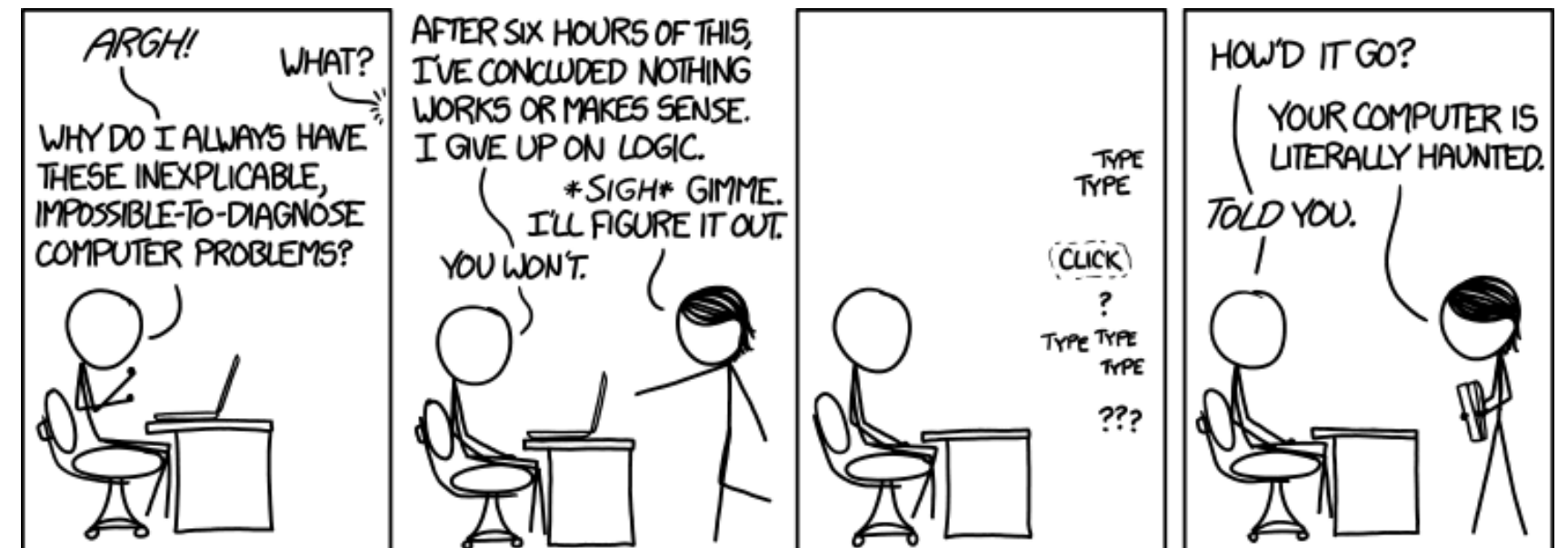
- Experiment with more variables
- Make the grid align with ground plane
- Set the colors of items random
- Arrange the objects in a circle
- Move the objects in the circle

Familiarize yourself with Unity

- Learn about components and game objects
- Learn about materials

Next Workshop

- Arrays and collections
- More functions and variables
- Debugging - breakpoints and watches
- Using libraries and packages
- Managing and organizing code
- Intellisense and code completion



<https://xkcd.com/1316/>

By Randall Munroe