Game Programming Using Unity 3D

Lesson 3: Introduction To Scripting

Ferdinand Joseph Fernandez

Chief Technological Officer, Dreamlords Digital Inc.

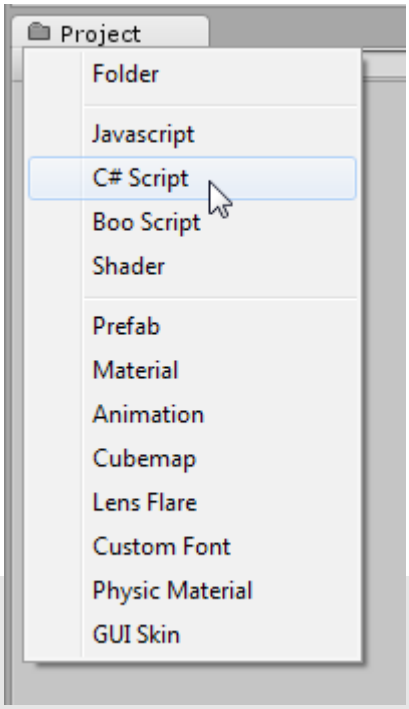
Admin and Co-founder, Unity Philippines Users Group

September 2011

Overview

In this lesson you'll be introduced to how you can program in Unity. Programming in Unity is a bit different than most game engines because of its component-based design.

Altering Behavior With Custom Components: Scripts



In Unity, your scripts exist as components, just like Rigidbody components or Light components. You have to attach them to game objects for them to run.

Let's start with a new empty project. Name it “Lesson3”.

Now we'll create a script file. Go to your Project View and you'll see a small “Create” button. Click it and choose “C# Script” from the menu that appears.

You'll be given an option to rename the file. Name it “Rotate”.

Double click on your Rotate C# script. MonoDevelop should open so you can edit the code. You'll see it has some template code written for you.

Let's start adding some code. Add the highlighted code (in line 15) in the Update function:

```
01 using UnityEngine;
02 using System.Collections;
03
04 public class  Rotate : MonoBehaviour
05 {
06       // Use this for initialization
07       void  Start()
08       {
09
10       }
11
12       // Update is called once per frame
13       void  Update()
14       {
15            transform.Rotate(2, 0, 0);
16       }
17 }
```

There are a few things you should notice in our template code.

At the very first line we import the “UnityEngine” namespace. This makes it easier for us to refer to Unity-specific code.

You'll see that our file has one class in it with the same name as our filename. This is important. The class name should match the filename or Unity will have trouble making use of it.

You'll also see our class inherits from MonoBehaviour. MonoBehaviour is a UnityEngine class meant as a base class for script components to inherit from. It has a few member variables/properties and “overrideable” member functions for you to use.

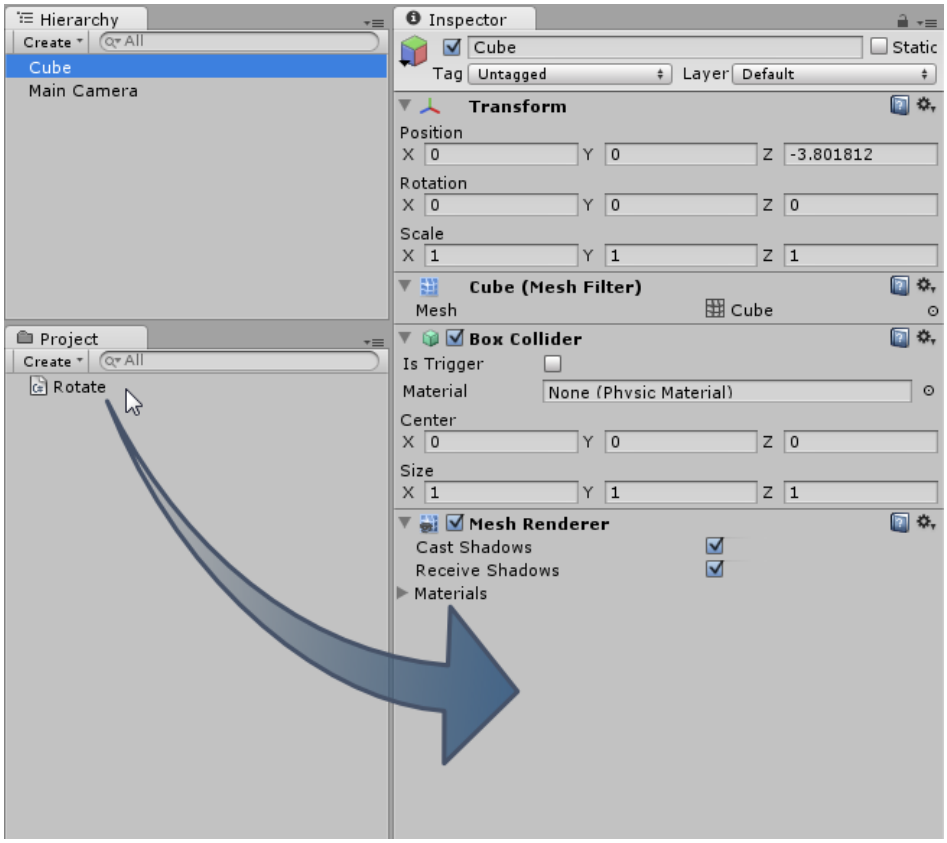
The “transform” that we access in the Update function is one such property. Recall that Transform is a property that defines our game object's position, rotation, and scale. The transform property accessible from scripts then, lets us do various things to the position, rotation, and scale. But this time, we're doing it from a script.

Now, the line of code that we added is pretty self-explanatory: it rotates our object by (2,0,0), or 2 degrees in the x-axis, every time Update is called.

Update is called automatically for us every frame. You do not need to make Update public for it to be called by Unity.

Create a cube with GameObject > Create Other > Cube. Select the cube. Position it somewhere where the camera can see it.

Now with the cube still selected, drag the Rotate script from the Project View into the Inspector View.



Now start the game. You should see the cube continually rotate. You can put a light if you want to see the cube better (GameObject > Create Other > Point Light).

Your Rotate script can be applied to any object you want. Create a few more objects. Try creating a capsule (GameObject > Create Other > Capsule). Drag your Rotate script to that too. Start the game. The capsule should rotate too.

This is how scripts in Unity work. You assign them to game objects that you want to be affected by those scripts. In fact, scripts won't run unless you assign them to game objects. There are some exceptions (static classes), but that is the general rule.

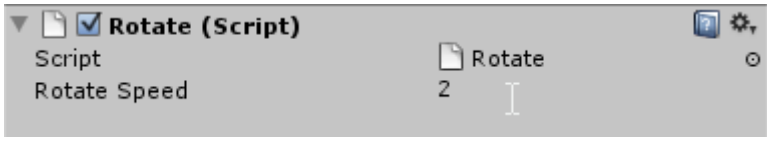
Using Variables

We all know using constant values is not good. So replace the code to this. You'll see the changes highlighted in yellow:

```
01 using  UnityEngine;
02 using  System.Collections;
03
04 public class  Rotate : MonoBehaviour
05 {
06       public float _rotateSpeed = 2;
07
08       // Use this for initialization
09       void  Start()
10       {
11
12       }
13
14       // Update is called once per frame
15       void  Update()
16       {
17             transform.Rotate(_rotateSpeed, 0, 0);
18       }
19 }
```

The convention for this course is to prefix underscores to our member variables.

Save the file. Go back to the Unity Editor window. You'll see the Rotate Speed variable show up in the Inspector.



The nice thing is, you can change the value for the Rotate Speed without needing to edit the C# file.

The value in the Inspector now has higher priority over the default value that you give in the C# file. If you edit the initial value in the C# file, it won't affect the current value in the Inspector.

Try changing the value. Then run the game. You should see the speed change accordingly. You can even change the speed while the game is running!

Making Variables Private

Most of you won't agree to using public member variables. The problem is, if we turn the variables private, it won't be editable in the Unity Inspector anymore. How do we fix that?

Fortunately, there is a way for Unity to force a variable to show up in the Inspector even if its private. Change the variable declaration to this:

```
01    [SerializeField]
02    float _rotateSpeed = 2;
```

SerializeField is an attribute that forces Unity to expose your variable to the Inspector. Removing “public” changes our variable to private by default.

Adapting To The Frame Rate

Its actually not good that we are just giving a constant value per frame to our rotate code. If the game runs on a slow computer, the Update function would be called a lesser amount of times. This is because a slow computer can't cope up. That computer is said to run the game in a low frame rate.

The rotation then, would be applied a lesser amount of times. This means the game may make the cube rotate slower in other computers, even if the speed you give in all computers is a constant value of 2.

What we need to do is to adapt the speed to the frame rate. So instead of saying “rotate by 2 degrees every frame”, we say: “rotate 2 degrees every second”. See the changes in the code below:

```
01      void Update()
02      {
03          transform.Rotate(_rotateSpeed * Time.deltaTime, 0, 0);
04      }
```

We now multiply our rotate speed with Time.deltaTime. Time is a static class available from UnityEngine. **The variable deltaTime is a float whose value is the time in seconds it took to render the last frame**.

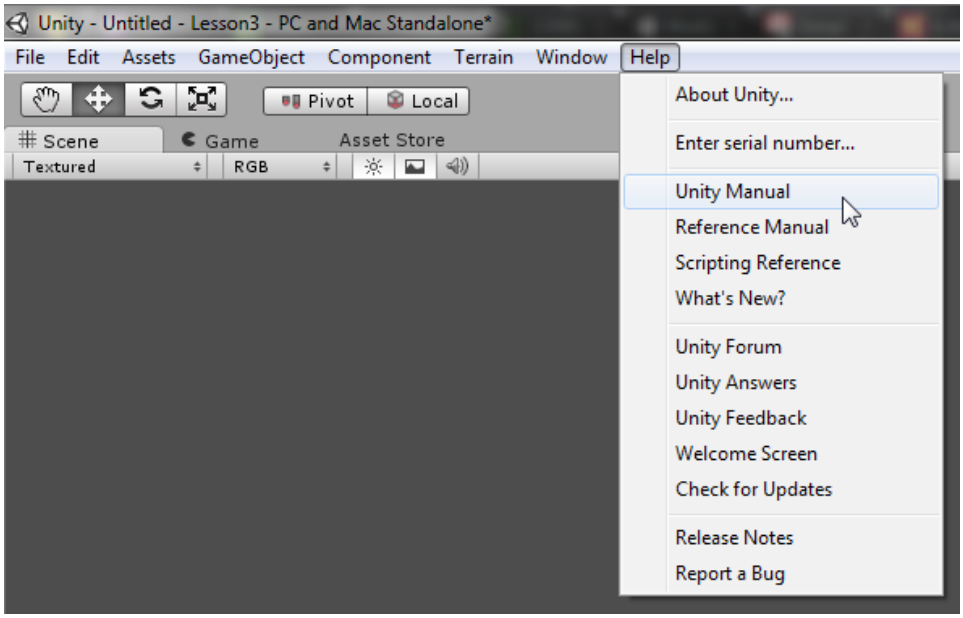
You might need to increase your rotate speed value now that it is expressed in per seconds, not per frames.

Where To Get Online Help

Before we delve in to making our simple game, here is a list of websites and resources that you can go to for help.

Official Documentation

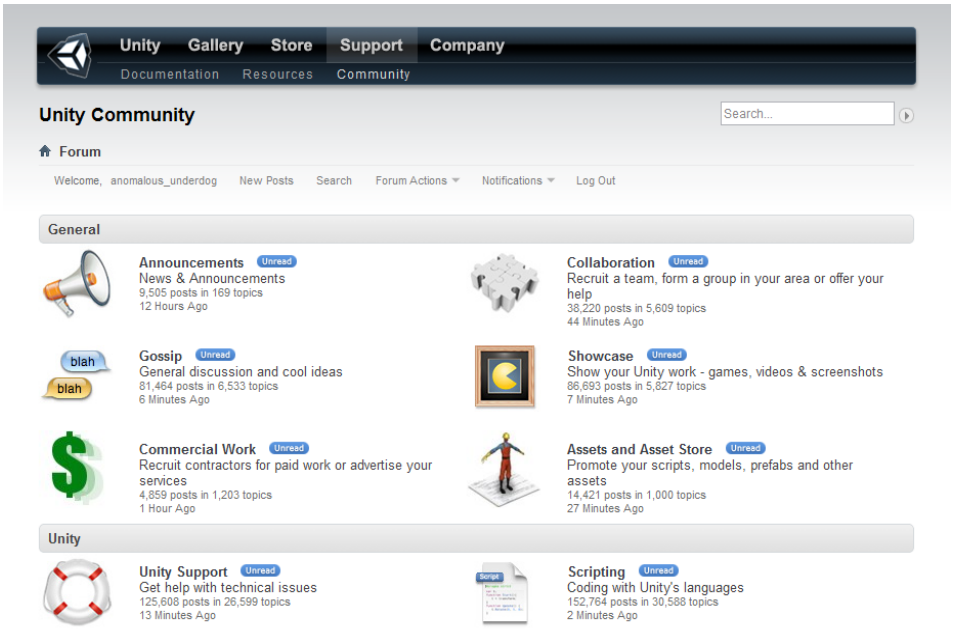
First of all, your installation of Unity comes with an offline copy of the user manual and scripting reference.



You can access it from the Unity Editor. Go to Help > Unity Manual. Beside there you'll also see the Reference Manual, and the Scripting Reference. The Scripting Reference even comes with a search function to help you get more information on any command that you want.

Community Forums

You can find the international community forums at http://forum.unity3d.com/.

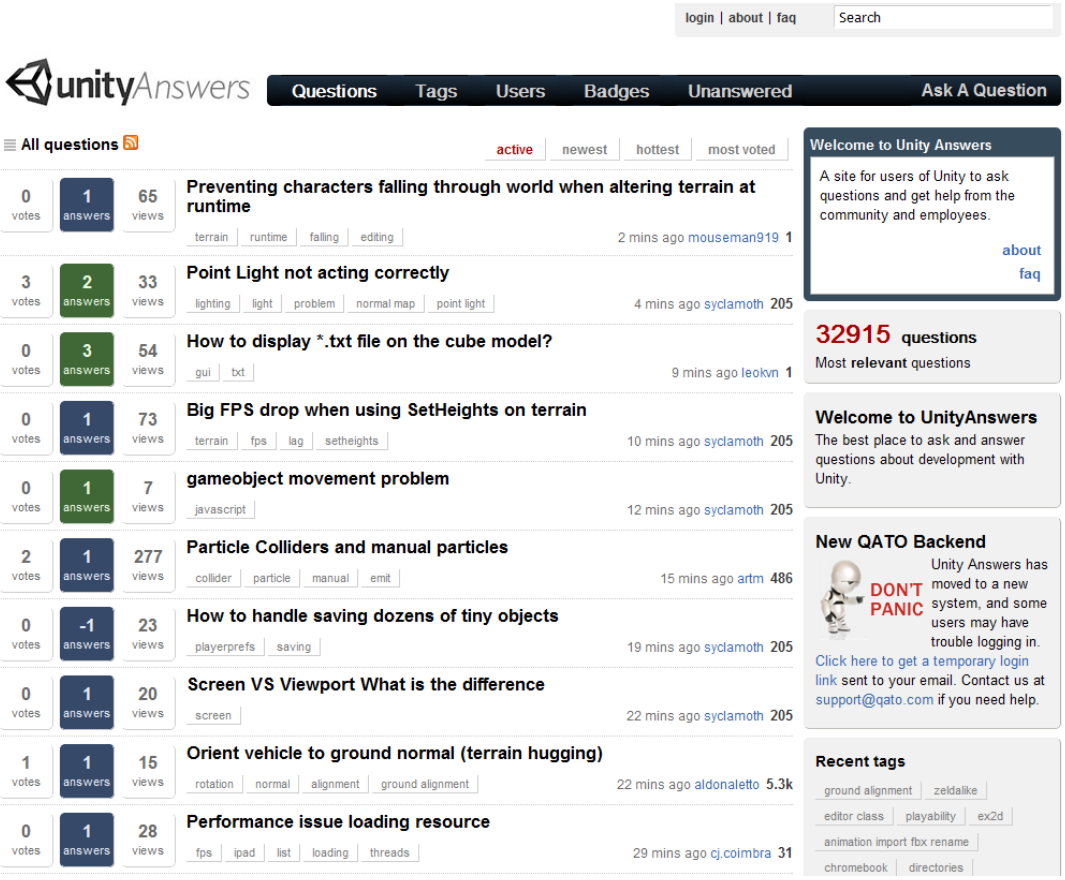


Community Wiki

A wikipedia-style site for sharing resources, scripts, and plugins. You can find it at http://www.unifycommunity.com/wiki/. Unless otherwise stated, the resources you can find there are free for personal or commercial use. It is highly appreciated to contribute code back to the community freely.

Question And Answer Site

Unity has a Q&A site similar to Stack Exchange. You can find it at http://answers.unity3d.com/.

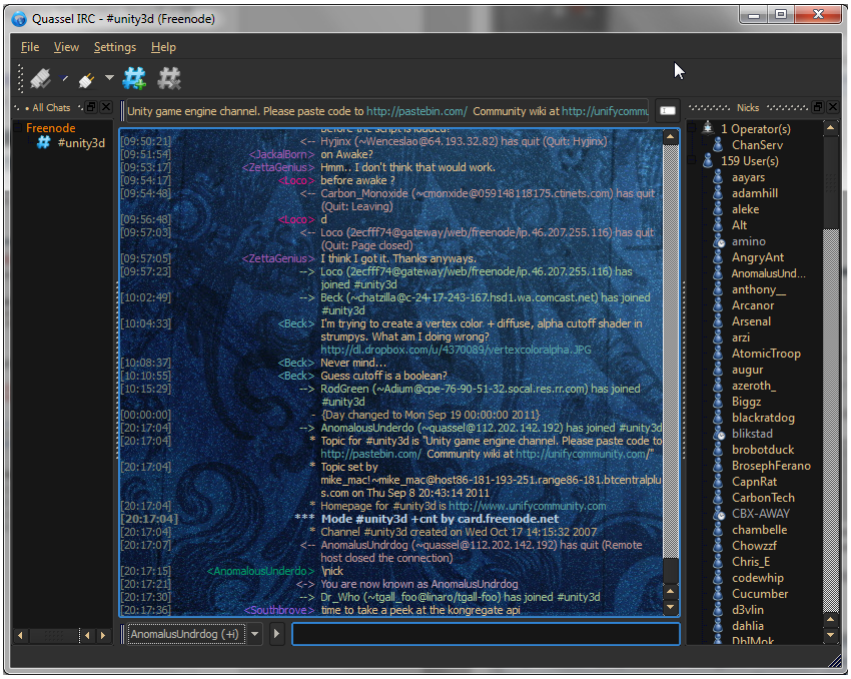


IRC Chat Channel

For immediate help, you can go to IRC chat.

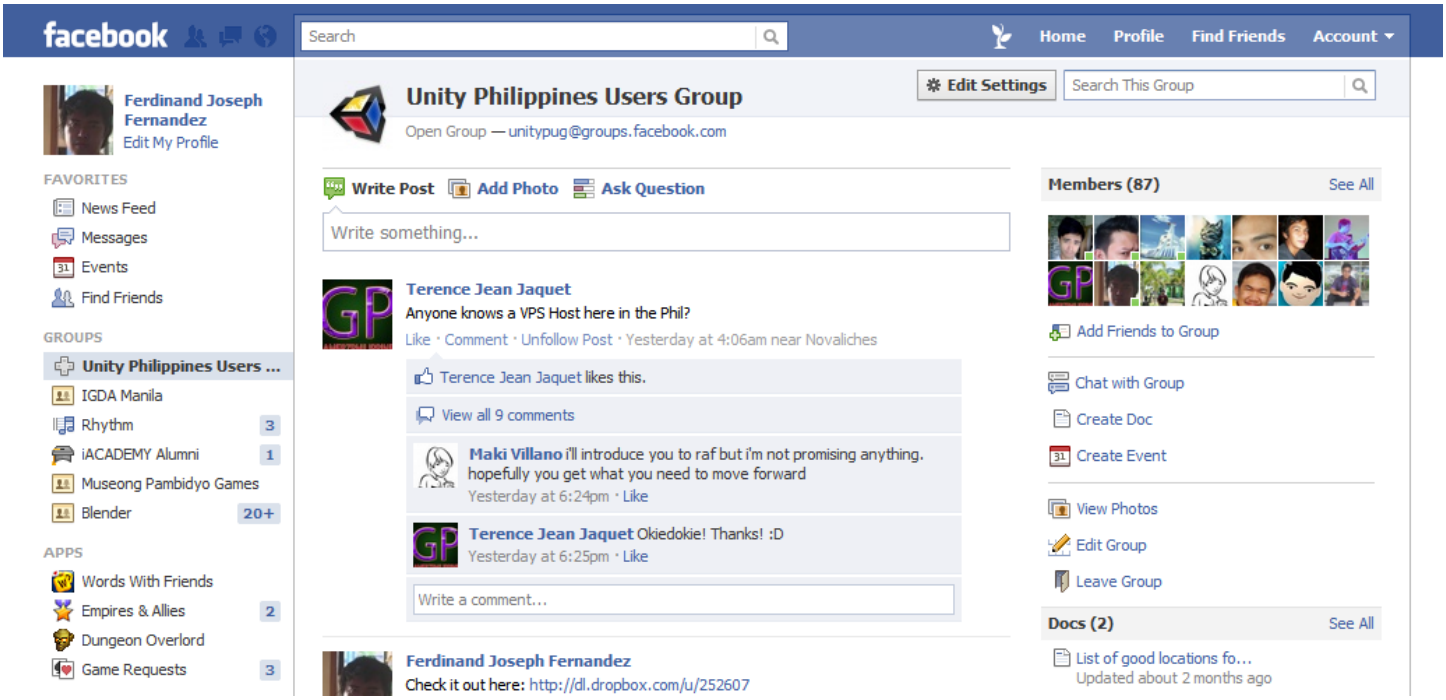
Server: irc.freenode.net

Channel: #unity3d



Unity Philippines Users Group

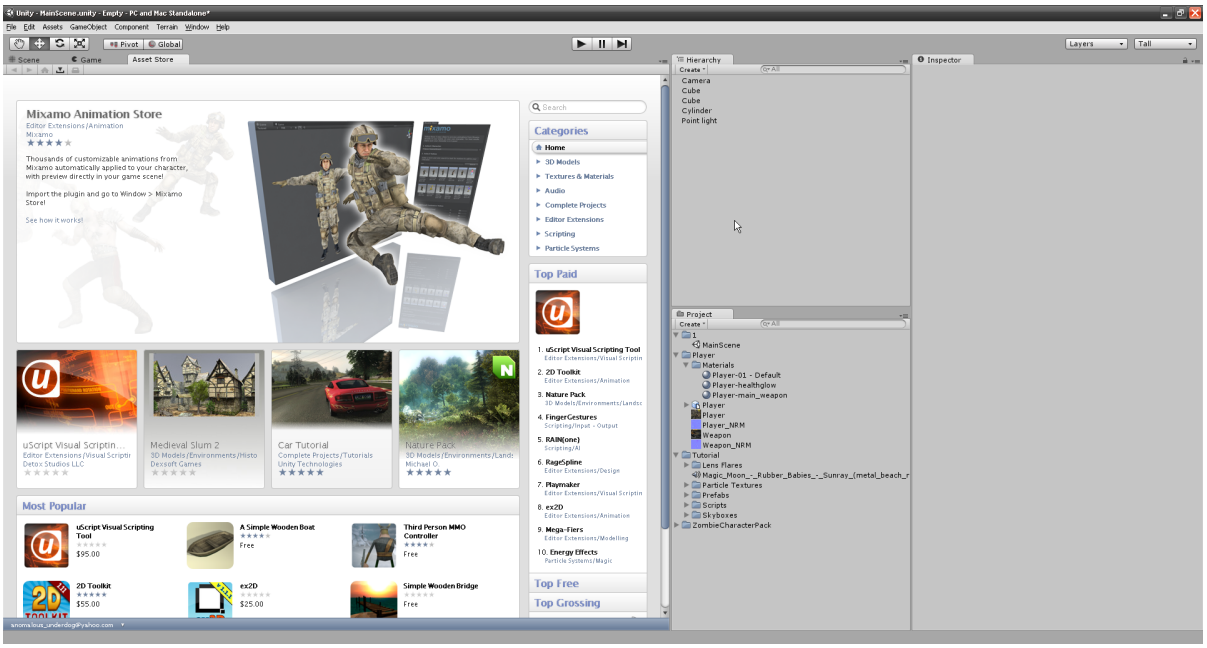
We also have a local user group here in the Philippines. For now, our temporary home is at a Facebook Group, found at http://www.facebook.com/groups/unitypug/.



Unity Asset Store

The Asset Store is a central place to buy and sell assets for Unity: 3d models, scripts, plug-ins, even music and sounds. Some are even available free.

You can find the Asset Store inside the Unity Editor. Go to Window > Asset Store, or Ctrl + 9.



In Conclusion...

You've learned the basics in programming with Unity. You also learned how to expose class variables into Unity's Inspector as if they were part of the engine.

You've also seen a few list of websites that can help you in case you get stuck.

Our next lesson starts our actual game. We'll be making a simple zombie shooter in a third-person, chase camera style view.