**C# Programming for Unity:**

**GAMEDEVEREJECTS**

**WHAT IS SCRIPTING ?**

**Date: 19-Apr-22**

**All Copyright GAMEDEVREJECTS©2022**

**Contents**

[**2.0** **What is Scripting ?** 3](#_Toc101255415)

[**2.1** **Double Check your IDE Settings** 3](#_Toc101255416)

[**2.2** **How to Create a Script ?** 4](#_Toc101255417)

[**2.3** **The Default Code : Namespaces, Inheritance** 4](#_Toc101255418)

[**2.4** **The Class Declaration Statement, introduction to the Inheritance principle** 4](#_Toc101255419)

[**Glossary** 6](#_Toc101255420)

[**Resources** 7](#_Toc101255421)

# **What is Scripting ?**

With reference to Unity, you will often hear developers talk about scripting. What they really mean is…using a code editor like **Visual Studio or MonoDevelop** to create and write the source code for the computer program. OR in other words, giving instructions to the game objects in the scene view to do something. For example, if we create a new game object in our scene window, as follows:

(NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).3D Object => / **Cube** /

(IN).INSPECTOR – Transform – (3 dots) => / **(Select).Reset** /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / **NewBehaviourScript** /

(IN).\_SCRIPTS -- (Select).**NewBehaviourScript**.(Drag and Drop) – HIERARCHY => / (Onto).Cube /

(IN).\_SCRIPTS => / (Select).**NewBehaviourScript**.(Double Left Click) – VSCODE /

Naturally we want to be able to interact and control that object. E.g. Let's say we want to move the object (in the scene view) from a point A in the scene to another point B in the scene. To do this in unity we write a “script” (the source code) then attach it to the object in the scene window that we want to move.

Now, the “strict” technical definition of scripting means => the program does not need to be “compiled” before running it. Rather your instructions/the program are interpreted at runtime. In effect you do not have the “hassle” of creating an executable .exe file (like with C++) before experiencing it. The source code is translated directly to the CPU at run-time (i.e. the source code is processed at the same time your program is being run and experiencing the application in real time) from any format into CPU machine language instructions.

Whereas compiling means **“all your source code”** is executed directly by the computer's CPU. i.e. the source code you have typed in your IDE is converted “all-at-the same-time” to the CPU's native assembly language before it is run or executed (before runtime).

*Please be aware effective from 01 Apr 2017 Unity has deprecated JavaScript (aka UnityScript) for programming, i.e. they now only officially support C# for programming*.

# **Double Check your IDE Settings**

Now before we create the 1st script, there are several housekeeping tasks you should do first:

**1)Double check your IDE settin**gs are set up correctly. In other word’s check you have installed/install **visual studio 2015 or 2017** setup as your default IDE.

[ Navigate to TOP.LEFT.TOOLBAR] -- Edit – Preferences – External Script Editor =>/(Drop Down Arrow).Visual Studio Code /

**2)Create Asset folders** **(\_MATERIALS, \_PREFABS, \_SCENES, \_SCRIPTS**) have been created.

(NavigateTo).PROJECT – (Select.Left Click).Assets – (Right Click).Create => / (Select).Folder – (Rename)**\_MATERIALS** /

(NavigateTo).PROJECT – (Select.Left Click).Assets – (Right Click).Create => / (Select).Folder – (Rename)**\_SCRIPTS** /

(NavigateTo).PROJECT – (Select.Left Click).Assets – (Right Click).Create => / (Select).Folder – (Rename)**\_SCENES** /

(NavigateTo).PROJECT – (Select.Left Click).Assets – (Right Click).Create => / (Select).Folder – (Rename)**\_PREFABS** /

Graphical user interface, application, PowerPoint

Description automatically generated

# **How to Create a Script ?**

Ok In unity there are **2 keyways** to create a script 1)From inside the **Project panel** via the asset folder, or 2)Adding a script as a **component** to the object inside of the Inspector.

(NavigateTo).PROJECTA picture containing text, screenshot, computer, computer

Description automatically generated – (Select).Assets – \_SCRIPTS – (Right Click).Create => / C# Script – (Rename).XXX/

# **The Default Code : Namespaces, Inheritance**

It is very important to understand what the default code is and does, because that will help you narrow down what code you need. Also, whenever you create a new script in unity. It automatically creates a new class with the same name as the script

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

After the using statements, at the top of the code base unity automatically creates a namespace. Anytime you see a statement with the command **"using"** --> it means you can access the **commands, classes, methods & objects** from those libraries for use in your code. **Namespaces are used to avoid duplicate naming conflicts**. For example, when you start to add libraries from other programmers it is highly likely you are going to use classes, methods & objects with the same names. Therefore, we create a **namespace** as a unique group name identifier for the collection of classes, methods & objects that it contains.

# **The Class Declaration Statement, introduction to the Inheritance principle**

Moving down to the main body of the script. In **OOP** there is a principle called **inheritance**. Which means that the class NewBehaviourScript "inherits" all the **commands, classes, methods and data** from the Monobehaviour class. The **class declaration statement** is typed as follows:

public class NewBehaviourScript : MonoBehaviour

{

//Ditto

}

The **MonoBehaviour class** = **is aka the Parent (Base) class - used to derive your C# script** -> It contains all the **commands, methods, classes** and components from the unity game engine to use in your C# script. MonoBehaviour is an **API.** For example, using JavaScript in Unity every script **automatically or implicitly** derives its code from the MonoBehaviour class.

However, when using C#, you must **explicitly** derive (or inherit) your code from the MonoBehaviour class. MonoBehaviour is also part of the UnityEngine namespace (i.e. using UnityEngine). So, if we delete the UnityEngine namespace => you will get an error. When you create a new script; by default, Unity creates 2 methods or functions (between the two curly braces inside the body of the class) => **void Start()** and **void Update()** aka **The Default Execution Order** **functions.** If you do not inherit from MonoBehaviour you “WILL NOT” be able to use these two functions as they are derived from the Monobehaviour class. The **void start() function is called only once for the duration of your script, at the start of your application. Whereas Void update() is called repeatedly at each frame.** OK so how do we use the code. Let’s start by typing **Debug.Log** which echo outputs or prints the string contents as a message to the Unity Console Window, which is kind of like the Console Window or Powershell used in other languages like C++, Java etc.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NewBehaviourScript : MonoBehaviour

{

    // Start is called before the first frame update

    void Start()

    {

    }

    // Update is called once per frame

    void Update()

    {

    }

}

# **Glossary**

Namespace Namespaces are used to avoid naming conflicts. When you start to add libraries from other programmers its highly likely you could use the same names. Therefore, we create a namespace as a unique group name identifier for the collection of classes that it contains.

Inheritance Inheritance is when one class inherits the behaviour/s: methods & properties of another class. The child inherits from the parent, but the Parent does not inherit from the Child. This allows code re-usability, saving time on code development.

# **Resources**

Scripting API: MonoBehaviour.StartCoroutine - Unity – Manual

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

/End