CREATING A GAME USING C#

Jacob Green

May 4, 2018

Abstract

This document aims to assist you with learning how to create a C# application in a popular game engine. The skills you are expected to learn is how to use development tools to help you program and create applications. Understand the critical logical thinking involved when creating a program. Learn how to use C# in an object oriented way so you can create things.

Tools and Environment

I expect you to learn by actively experimenting and using the below tools, but you can also research information online on using these tools. I have included some bare bones information which I will expand on later.

0.1 Unity Game Engine

Unity is a popular game engine with a large amount of community resources. It has been used to make small game and large triple AAA games, it also comes with a simple build tool so you can repackage your application on other platforms like mobile.

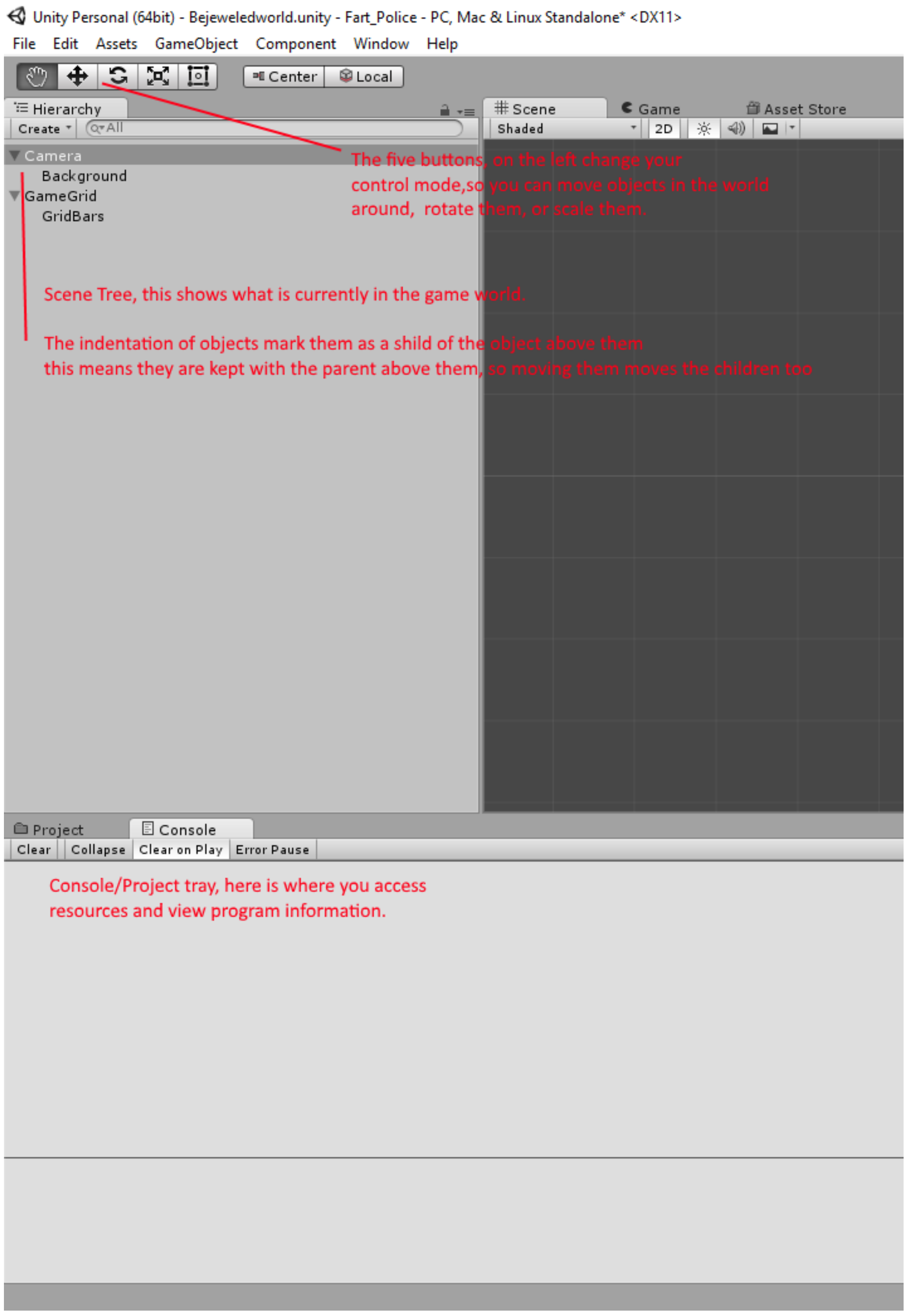
If there is something you want to find on Unity you can easily find those resources online. The only issue with the online resources is they are a mixed bag in usability. A lot of tutorials are video tutorials which can be an annoyance when you just want to read some information quickly on a subject, while other things might not have much coverage at all, so you either have to search through the written documentation of find a forum discussion/answer on the issue. For now you are unlikely to have those issues until you start trying to tackle certain problems.

The Editor

You can find a full tutorial here https://unity3d.com/learn/tutorials/topics/interface-essentials/interface-overview , but I will try to cover things with images so you do not need to watch through an entire series on how to use the editor.

You will pick up how to use the editor as you begin to use it more. There is information only but the core break down is that you only need to know a few things to get started.

These are the main area's of the editor


Unity Personal (64bit) - Bejeweledworld.unity - Fart_Police - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Center
Local
Hierarchy
Create
All
Camera
Background
GameGrid
GridBars
Scene
Game
Asset Store
Shaded
2D
Project
Console
Clear
Collapse
Clear on Play
Error Pause

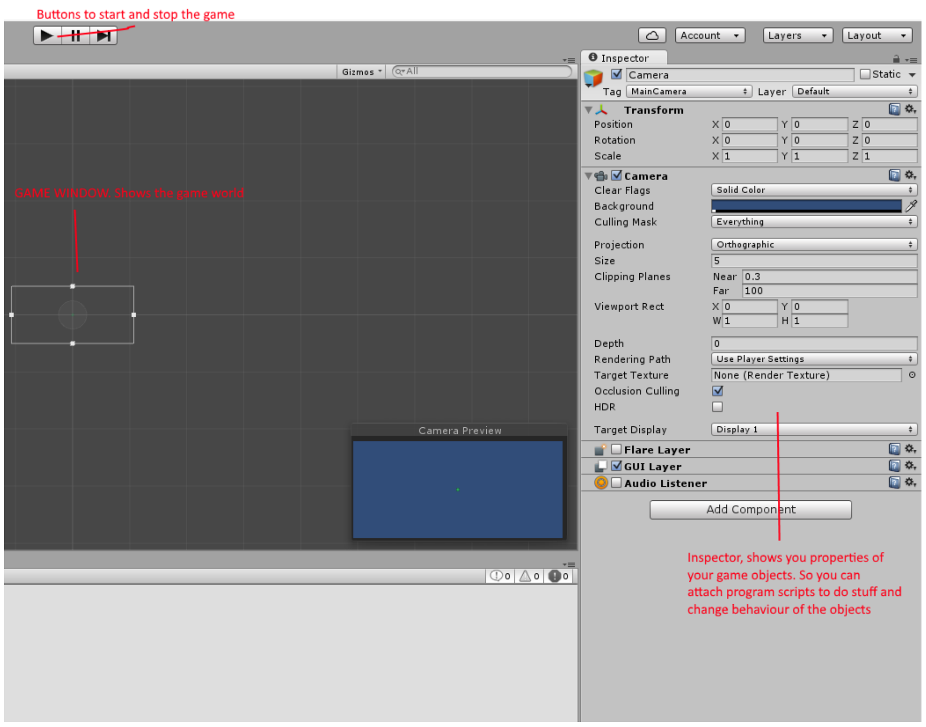
The five buttons, on the left change your control mode,so you can move objects in the world around, rotate them, or scale them.

Scene Tree, this shows what is currently in the game world.

The indentation of objects mark them as a shild of the object above them this means they are kept with the parent above them, so moving them moves the children too

Console/Project tray, here is where you access resources and view program information.

Figure 1: We have the scene hierarchy view, the main control settings for objects and our asset and output trays. The console output tray prints information on the game and the assets show us our code and textures that can be brought into the game world.



Buttons to start and stop the game

GAME WINDOW: Shows the game world

Inspector, shows you properties of your game objects. So you can attach program scripts to do stuff and change behaviour of the objects

Camera ☑ | Static ☐
Tag MainCamera | Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Camera ☑

Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Orthographic
Size	5
Clipping Planes	Near 0.3
	Far 100
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	0
Rendering Path	Use Player Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	☐
Target Display	Display 1

☐ **Flare Layer**
☑ **GUI Layer**
☐ **Audio Listener**

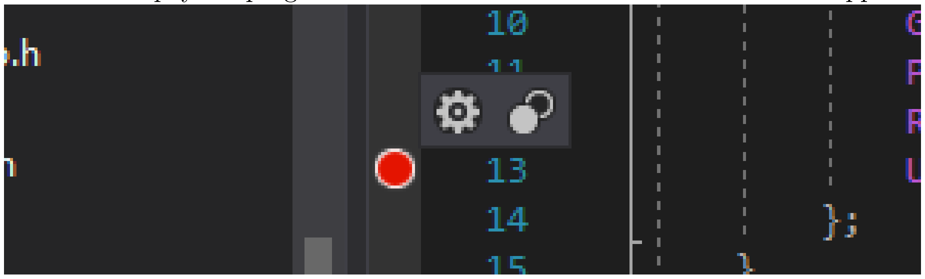
Figure 2: We have the game view and the actions to start and stop the game. And most importantly we have an inspector view which covers the properties and what actually makes an object in our game.

0.2 Visual Studio 2017

Visual studio 2017 will be the programming editor we will use for creating code and debugging code. For now all you need to know is that opening a script in unity should open visual studio for you.

At the top will be a green button called attach to unity, this is how you attach the debugger to the game. A debugger allows you to stop the game mid way through code to inspect the value of variables and see how your code is behaving. Basically it lets you inspect and see what is wrong with your program.

Clicking in the space next to a valid line will create a red dot this is known as a break point and when your program reaches that line the program will stop and you can hover over variables to see what value they hold and you can even step your program forward a line at a time to see what happens.



0.3 Source Control Git (Using GitKraken Tool)

Get Rufus to do this with you, until you get the hang of it. I'll add to this when I have some time.

C# Basics

I will be explaining simple concepts and trying to impart a basic understanding of using C#. This does mean some of the things I try to teach might be a bit technically inaccurate just to teach you how to use them rather than how they work.

0.4 Sourcefile

A source file is the name of the file you write into to author your programming code. These files would end in a .cs extension.

Another important thing to let you know about is that in a C# program you must define a entry point to your program somewhere. For our Unity project we do not need to as the project defines one.

But so you know how to do it. You would simply have to declare a function like this

static void Main(string[] args);

0.5 What is Object Oriented Programming

There are many more formal explanations on what Object oriented programming (OOP) is, however I'll try to make it simple to understand.

It is basically categorising data and treating things in a logical hierarchy. A program is made up of many functioning parts which would be separated into unique components of the system. These can be isolated classes that themselves own and manage classes to handle data and logical actions.

0.6 simple usage

In the next section I am going to cover a lot of complex stuff. Here I will describe the very basics of using the language. Ask for more info if there are things you feel you need help with.

Every statement must end in a semi-colon. A statement would be an expression to execute. It will become obvious over time why you need to do this.

Things must exist in scope. That is the { } brackets. So if I created a class, namespace or function, to be apart of one of those things you must be inside the scope { } block.

Creating a variable. Quite simple unless you want something really special (using other keywords together). You declare the type on the left and then the name on the right. E.G SpriteRender variableName; This creates a variable called variableName which is of type SpriteRender.

To create a new value depends on the data. Usually it can be as simple as a number such as.

$$intmagicNumber = 5; \tag{1}$$

or it could be a complex class, which you need to create with a special word called new.

$$newSpriteRender(); \tag{2}$$

Assigning a value to a variable. You use the ōperator to do this. So always have the declaration on the left and the value to set it to on the right. For example(this creates a variable and assigns it a brand new value):

$$SpriteRendervariableName = newSpriteRender(); \tag{3}$$

When declaring a variable as the type and then the name. It means you create a brand new variable. If you want to reuse the variable you just need to assign a different variable with out the type declaration.

E.G

$$intvariableName = 5; \tag{4}$$

$$variableName = 8; \tag{5}$$

0.7 langauges keywords and datatypes

Language keywords are not programming keywords, but special keywords that have meaning in C#.

Now some of this information will feel a little incomplete as it lacks true examples. I'm hoping you can learn them through demonstrations or from online research. If you get stumped or want more information do ask.

0.7.1 namespaces

Namespaces are important for providing context to data and programming. Everything has a namespace even if you do not declare one (that would be the global namespace). It is important for maintaining code and separating your program into more isolated contexts.

0.7.2 using directive

We have already seen these. Those '#using System;' seen at the top of our class source file. By having those using statements we let the compiler know that we will be using code and functions defined in those namespaces within our source file, this means we do not need to explicitly write out the namespaces to access functions we intend to use.

0.7.3 classes

Is a type of data structure. You can create functions for the data structure. Classes can also hold data.

Classes in C# can be passed around the program by reference which means if you created an instance of a class, and then passed the instance into a function as an argument it will have the original object.(it is referred to as passing by reference when it references the original object when passed around the code)

0.7.4 structs

Exactly the same as classes except when passing it into functions it copies the data inside to make a new instance of a struct object, basically it copies it. This is important to know because there will be times you mean to copy and times you want to change the original (it is referred to as passing by value when it copies the object when passed around the code).

0.7.5 functions

The functions are composed of 3 keys things.

The return type, a function when called can return something. This could be a boolean (true or false), or it could be something like an object, or it could be nothing, void.

The function parameters. These are the things the function can take in and do something with. When using the function in code, you refer to the things passed in as the parameters as arguments. There are a bunch of other things you can do with these parameters including special words to mean the argument does not change, or force the argument to be passed by reference instead of copy, even for structs (see what was said above in class and structs where we talk about this).

And finally protection level. You will see in functions that they sometimes have things like public, protected or private written next to functions. This means what level of access the function is,

If it is public anything can call the function on the object it lives on. Private means it is used internally and only the owner of the function can use it, such as the class.

And lastly a special one called protected. If it is protected it means the owning class and sub-objects of that class have permission to call those functions. You won't have to use protected much.

0.7.6 types

There are many different data types in programming, not just in C#. These are known as the primitive types, here are some common ones.

1. void (special type, it means nothing.)
2. char (a single character, it is represented as a byte. Which is 8 bits.)
3. bool (a single bytes, returns true or false)
4. int (an integer number, meaning no decimal places. Usually 4 bytes)
5. unsigned int (same as int except you can't have negative numbers so it can support bigger numbers)
6. float (also 4 bytes. It is a decimal number, letting you have non whole numbers)
7. double (a much bigger and more precise float, it used 8 bytes)
8. string (a type that is a consecutive grouping of chars to create a word or sentence)

There are also arrays which is a data type, that allows contiguous collection of data.

You can also create your own types as they are just data structures! Without knowing it, you have already been doing that. Creating a class or struct is a data type with functions and operations that can be performed on it.

0.7.7 const

Const is a keyword that can be used to mean the object can not have its original value altered after it is set. A const value needs to be set when it is created, it cannot be set a value after being created.

0.7.8 static

This can be a pitfall for many programmers. You must be careful with this, as it could have behaviour you don't intend. A static variable is initialised before the program does anything else. The object always exists, and if the static variable is inside something like a class. It is shared between all classes, meaning you could have multiple different objects sharing the exact same object.