Consyl Engine Documentation

Easy to understand!

**Contents:**

(Shows the pages of each section of the documentation)

Contents 01

Requirements 02

Game Code Basics 03

Graphics 08

Textures 09

Audio 10

Example Game Code 11

Engine’s Code 12

Extra Functions 13

**Requirements:**

**Recommended Hardware Requirements:**

CPU: Processor with 2 GHz dual core

Memory: 2 Gb of Ram DDR3

GPU: Not needed (pc at least needs graphics display on screen)

Storage: Consyl takes up around 3 Mb of storage, so you don’t need to worry about it

**Recommended Software Requirements:**

1. Windows 10 Operating System

2. Microsoft Visual Studio that can run C# .NET Core 3.1

**Recommended User Knowledge:**

1. Basic C# code knowledge

2. Basic Math Knowledge

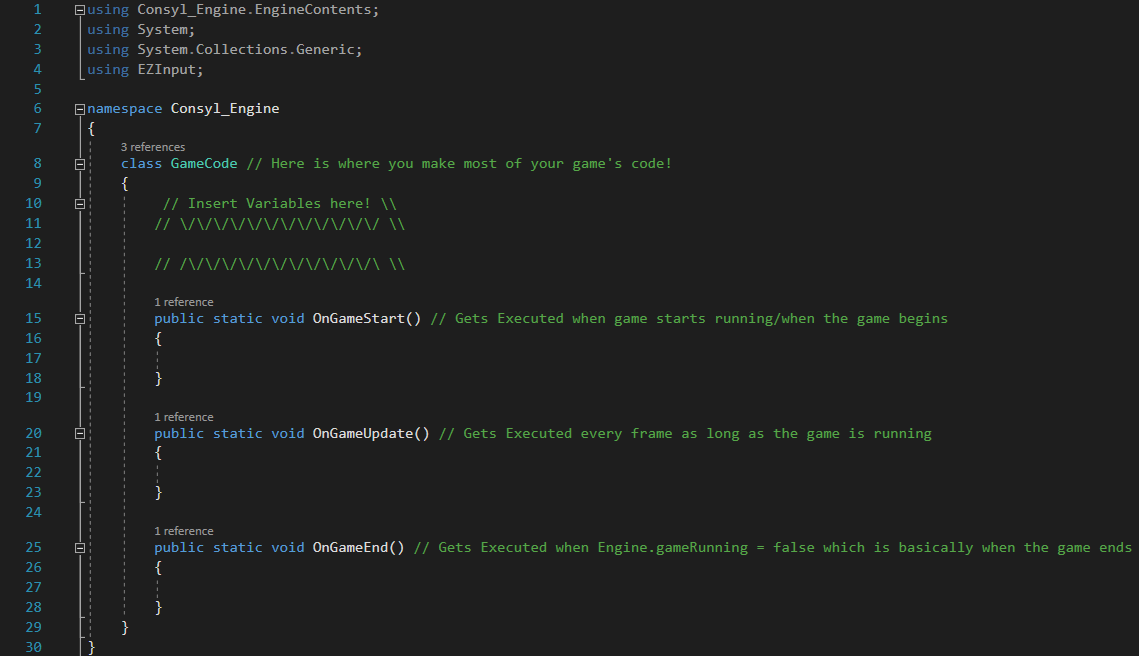
3. Experience with playing Games (not very necessary)

**Note:** Don’t worry too much on requirements because most of the requirements like hardware and software you already fulfilled.

**Game Code Basics:**

**Navigating to the Main Scripting File:**

To code your first game, you need to open “Consyl Engine.sln” and from the Solution Explorer, choose “GameCode.cs” which is the part where you put all of your code in for your game. (“Image A” shows how the code is supposed to look like)



**Image A:** This image shows the place where you code your game.

**A Look Around Inside the File:**

“GameCode.cs” contains the basics of what you need in making your games. It has two important sections, one where you put your variables and data, and the other is where your code gets executed.

The Section where your code is executed has three main parts, “OnGameStart”, “OnGameUpdate”, and “OnGameEnd”.

**OnGameStart:** Is a place where the code you inserted gets executed when the game starts.

**OnGameUpdate:** Is a place where the code you inserted gets executed repeatedly only when the game is running.

**OnGameEnd:** Is a place where the code you inserted there gets executed when the game closes through code.

**What are Variables?**

Variables are pieces of information gets created when the program runs, it is stored inside your computer until you decide to close your program/game. Variables can be manipulated and read by the code to do many fancy instructions.

**The Main Relevant Types of Variables**

1. **Boolean:** A variable coded in as “bool” that can store either a true or false value.

2. **Integer:** A variable coded in as “int” that can store non-decimal numbers.

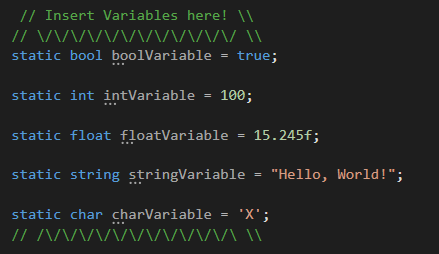
3. **Floating-point number:** A variable coded in as “float” that store numbers with decimals.

4. **String:** A variable coded as “string” that can store text.

5. **Character:** A variable coded as “char” which stores a single letter/symbol

**How to Code in Variables?**

To add in your variable, you need to type in “static” then you type in the variable type, then the variable name and equal sign then the value/data. Here are some examples of variables in the image shown below:



**Image B:** Shows examples of different types of variables.

**Important notes about variables**

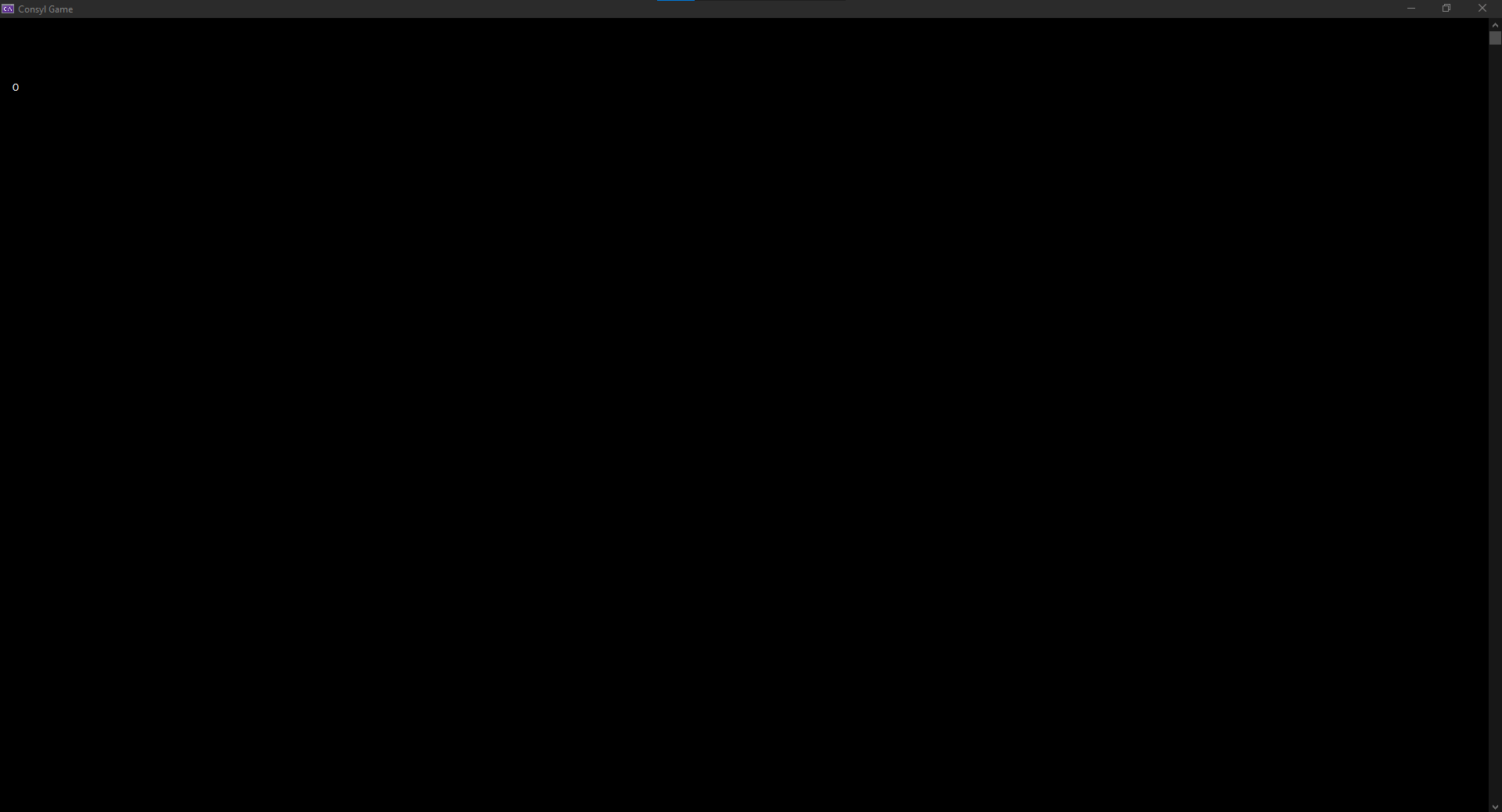
It is important to make your variable names easy to read and understand, so you format them like by making the first letter to be lower case and if the name has multiple words, the beginning of every word other than the first one will be upper case. Do not add space between the variable name as it would cause errors. Also, you don’t need to add a value/data inside the variable when you make it, you can just make it without values.

**An ASCII Pixel**

When you press “F5” key you’ll start to load the game, and it’ll open a window on screen with nothing but black screen in it. We’ll start from the very basic level of drawing an image on screen by drawing a single “ASCII” pixel. “ASCII” is a format of displaying letters/symbols on a computer, but in Consyl Engine we don’t have true pixels, but we do have something named an “ASCII Pixel” which is a pixel made of an ASCII Symbol, which is not a real pixel but will resemble one.

**Drawing an ASCII Pixel on screen**

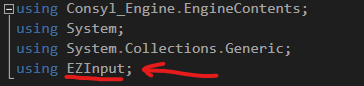
To Draw an ASCII Pixel, you’ll need to write down in between the curly brackets below “OnGameUpdate” function “gfx.DrawPixel();”. Right now, it will not draw anything, and it will display an error, because you need to tell it where on the screen you want the pixel to appear and how it looks like. For example, you want your pixel to be placed at a coordinate of x = 1 and y = 5 and the ASCII pixel you want to draw looks like this “O”. So, you want instead to type “gfx.DrawPixel(1, 5, ‘O’);”, and when you run your code, you’ll see the output as shown in Image C. You can add more pixels by creating a new line with the same code with different coordinates and ASCII symbol. (Make sure when running your game, you maximize your window to show your result properly)



**Image C:** This Image shows what happens if you code in “gfx.DrawPixel(1, 5, ‘O’);”

**Before adding player Input**

All the way at the top of the “GameCode.cs” script, you’ll see “using” codes which gives access to more code for the developer to use. “Image D” shows how section displaying “using EZInput;” looks like. If you don’t have that for whatever reason, type it down since you will need this to allow for user input.



**Image D:** Shows EZInput.

**How to Add Player Input?**

You need to type in, inside the OnGameUpdate function an if statement, which is used to check for bools and functions. To make an if statement, you need to type “if()” and open and close curly brackets like this:

If ()

{

// Code here

}

Right now, there is not logic. So, you add between the parenthesis next to “if” the logic, if the logic is true, then execute the code between the curly brackets. So, we do this:

if (Keyboard.IsKeyPressed(Key.A))

{

// Code here

}

This will check if you have pressed the A key. if you did the code will execute. If you add the DrawPixel() code from before inside the curly brackets, this will result when you run it to make the pixel only appear when you press A. Note that you can change Key.A to any key on the keyboard, like Key.B, Key.Shift, etc.

**Creating a controllable player**