Sammy Newhide / [sammynewhide@gmail.com](mailto:sammynewhide@gmail.com) / (909)-573-4476

This is my major personal project that I learn my C++ coding practices from. I tune into the livestream Handmade Hero (<https://handmadehero.org/>) to learn and follow along with an experienced game engine developer, Casey Muratori. The goal of the project is to create a quality game that has industry-level code, all without the use of libraries and pre-written/pre-compiled code. In its current state, I am currently on day 84 of the project. With each coding session taking me roughly 2.5 hours, I have spent 210 hours on this project so far. Below, I will provide examples of code I wrote and explain certain systems of the architecture so far.

Walkthrough of the Code:

**\handmade\misc – shell.bat**

I use a shortcut to cmd that runs this batch file first to set up Visual Studio’s compiler in the cmd.

**\build** **– build.bat**

This is how the build system for the game. ‘win32\_handmade.cpp’ is compiled into an executable, and ‘handmade.cpp’, the main game code, is compiled as a DLL. It contains presets that ignore unnecessary errors, and even sets constants that tells the code what type of build we are running (debug only for now though). It is simple and very effective, VS runs in command prompt with less problems than using the builder in Visual Studio Community.

**\build – win32\_handmade.exe, handmade.dll**

A compiled version of the build so far! Open the .exe and the debug version will hopefully load properly. The program is incredibly inefficient at rendering so far, hopefully it is not too slow on your computer. You will be able to see a couple of features at work:

* Bitmap Loading: (the hero and the trees are stored in bitmaps, and the patch of ground on the left side is pseudo-randomly generated and then cached as one bitmap)
* Staircases and Concept of Z: Even though the game is planned to be from a top-down angle, I am already experimenting with creating the concept that the Z-axis exists in this world. If you travel from the bottom-up on the orange rectangle (the staircase), you can see the hero move up in Z and eventually reach the next level. I am very happy with how it feels like a real staircase without having to teleport you to different stories on touch.
* Procedural Generation: The basic layout of the world is pseudo-randomly generated (the random system is in the early stages so, though you won’t get different results unless you play with the random functions inside).
* Entities & Position:The player, staircases and walls are each individually stored as entities. The entities are place in the world with floating point precision. A less obvious fact is that I am currently implementing a system to allow the game to have thousands upon thousands of entities stored in floating point and moving outside of the camera without running into memory problems with floating point. Please see sim. regions in pages below.

If anyone wants to build it themselves, there is a .sln file for Visual Studio in the build folder, as well as build.bat and shell.bat.

**\handmade\code\win32\_handmade.cpp**

This is the code for the Windows platform layer. It is separated from the game code since another goal for the project to ship the game on multiple platforms.

* XInput Library Loading (Lines 35-84): This is where I learned how to load DLLs on Windows. The function includes advanced function stubbing so that the game doesn’t have to require XInput for computers that do not have the DLLs ready.
* Custom String Handling (Lines 88-142): Functions that handle strings using for loops and pointers instead of using C++ libraries. It is interesting how short the implementation is, and saves a lot of heading from using str. The EXEPathFileName and EXEFileName functions provide the code data for the games location in Windows with ease.
* Custom File Input and Output (Lines 158-206, 269-296): For debug purposes these functions read and write raw bits by calling Windows functions and their file handle system. The parameters for the functions are elsewhere in the code (because these are stubbed for non-debug builds). The reason this is debug because I will be learning how to do asset streaming at a later date.
* Live Code Editing and DLL Hot-loading (Lines 1202-1221):One of my most favorite parts of the code so far. Since the game code is exported to a separate DLL (handmade.dll), I have the power, even in this early stage, to compile new versions of the DLL and see the game change without having to close it. The lower half of the section is inside the main Windows loop, where I have variables that keep track of times the game DLL was written. If there is a newer DLL available (line 1214), the game will reload the DLL by calling Win32LoadGameCode and Win32UnloadGameCode and Windows. The feature is incredibly useful when tuning the game code; I could change the player’s speed in the code, recompile, and see it in game without having to reload Visual Studio’s debugger. The code is relatively simple for such a complex idea, and it amazes me each time I look back at it.
* Game State Recording and Looping (Lines 499-582): Another great part of the code, these functions allow for recording the game’s state and input and replaying them in a loop. Toying with the memory in this way was tough Please try it out with the ‘L’ button, although I cannot be certain that it will work.