Tutorial: Create 2D Game Engine using C++

URL: <https://www.youtube.com/playlist?list=PL-K0viiuJ2RctP5nlJlqmHGeh66-GOZR_>

# Purpose

These notes summarize the you tube video playlist “How to Make a 2D Game Engine with C++ & SDL”. We will try to follow the videos in order but will not hesitate to re-arrange the material to make it easier to read and follow and understand the concepts presented.

I was going to use the latest version of SDL – SDL3 but found too many differences between SDL2 and SDL3 so decided to stick to the older version.

# 1. Setup SDL, SDL\_image, SDL\_ttf in Code::Blocks

The first video walks you through obtaining and installing the application Code::Blocks. Code::Blocks will be used to create our C++ source files and compile and run our application.

In addition this video creates the initial project to be used throughout the series, which of course starts as a “Hello, World” project. Lastly, we install the two main SDL libraries – SDL2 and SDL\_image.

## Why use Code::Blocks?

Code::Blocks is a free open-source, cross-platform Integrated Development Environment (IDE) primarily designed for C, C++ and Fortran programming. The IDE was originally released in 2005, and its development is driven by a team of contributors using wxWidgets library for its graphical user interface (GUI). The first official stable version was 8.02 released in 2008. The release marks the year and month of the Code::Blocks version. Like most modern IDEs it provides for extensions via a plugin system.

|  |
| --- |
| What is wxWidgets?  wxWidgets is a free and open-source C++ library used for creating cross-platform graphical user interfaces (GUIs). It allows developers to write applications that can run on multiple operating systems, such as Windows, macOS, Linux, and more, without needing significant code changes. This is achieved by using the native API of each platform, ensuring that applications have a native look and feel.  Originally created in 1992 by Julian Smart, wxWidgets was initially called wxWindows but was renamed in 2004 due to a trademark issue. It supports a wide range of features, including GUI components, event handling, multithreading, file handling, and more. Additionally, it has bindings for other programming languages like Python, Perl, and C#.  Developers often use wxWidgets for its portability and ability to create sophisticated applications with minimal effort. Let me know if you'd like to explore its features further! |

## Install Code::Blocks

The official website to obtain Code::Blocks is <https://www.codeblocks.org/downloads/> . You have the option of:

* Downloading the setup file for Code::Blocks and executing it.
* Download the source code and build from scratch

The Binary release has a Windows, Linux and Mac OS X version. The fact that there is a version of Code::Blocks that runs on all three platforms is the primary reason it has been selected for this and other tutorials. At the time of this writing the latest version is 25.03, which means it was released in March of 2025.

My choices for Windows are:

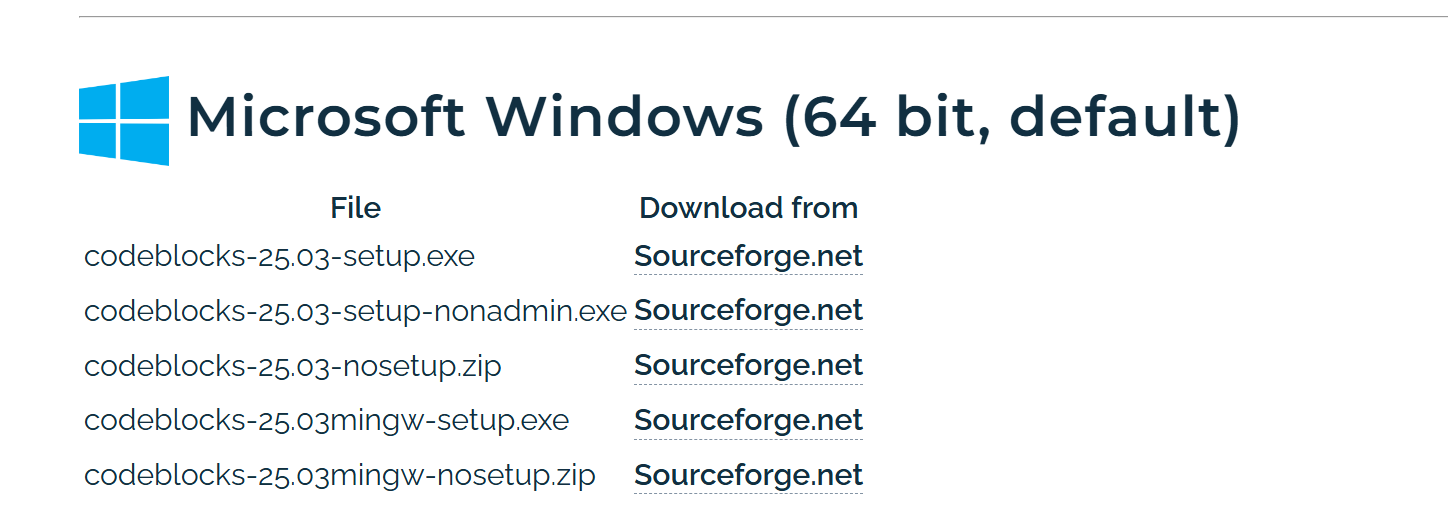


Figure - Selecting the right setup.exe to download

I decided to download codeblock-25.03mingw-setup.exe for 64-bit.

Side note: I am surprised that the project has not moved to GitHub.



Figure - Downloading file from Sourceforge

I then run the setup.exe file:



Figure - Locating and executing the setup.exe file

### Running the Setup

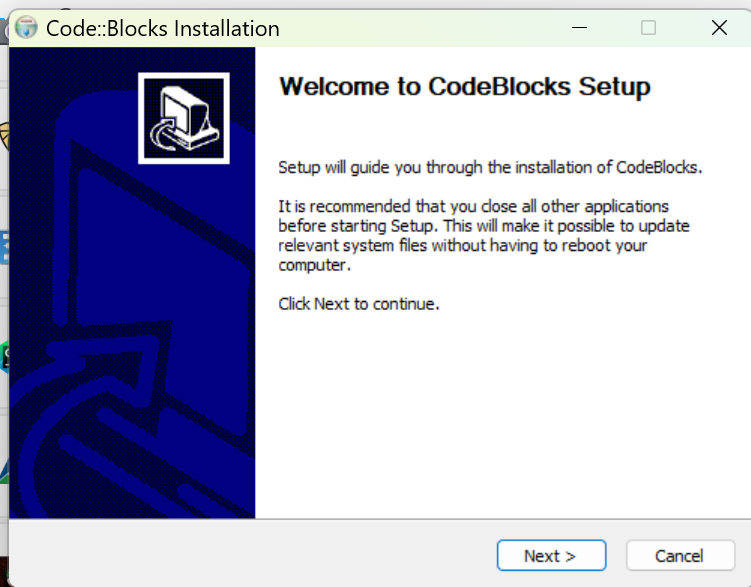


Figure - Initial Code::Blocks Installation dialog

I did not find it necessary to close any other applications. I just pressed “Next >”.

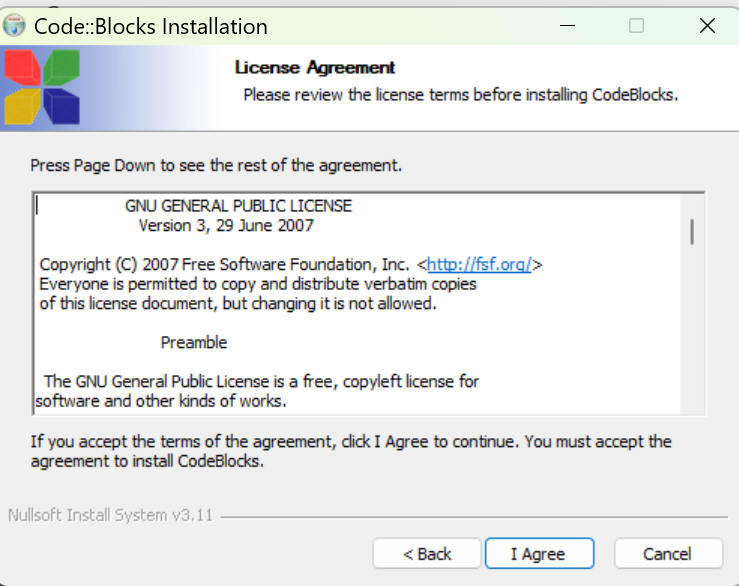


Figure - The License Agreement screen

OK, I did not read the license agreement but knowing it is the GNU license assures me that it is open source and free. I just clicked on “I Agree”.

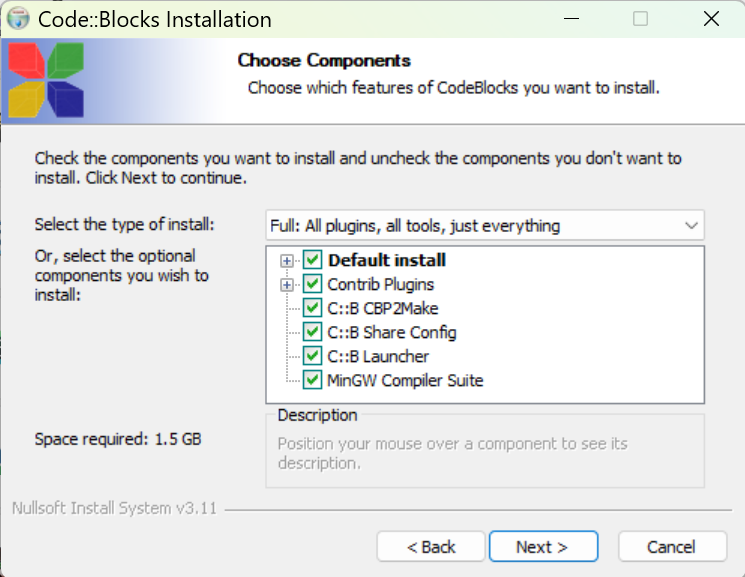


Figure - The "Choose Components" screen

It appears by default all the components are selected. A quick description of each component:

* **Contrib Plugins** – these are additional plugins developed by the community to extend the functionality of the Code::Blocks IDE. These plugins are not part of the core set but have proven to be valuable enough to be included in the official Code::Blocks repository.
  + **Code Snippets Plugin**: Helps to manage and insert reusable code snippets.
  + **DoxyBlocks Plugin**: Integrates Doxygen for generating documentation from your code
  + **CppCheck Plugin**: Provides static code analysis to identify potential bugs or issues
  + **SpellChecker Plugin**: Checks spelling in comments and string literals
  + **Valgrind Plugin**: Integrates Valgrind for memory debugging and profiling
* **C::B CBP2Make** – is a tool designed to generate Makefiles from Code::Blocks project files (\*.cbp) or workspace files. Essentially, it allows you to convert your Code::Blocks projects into Makefiles that can be used with GNU Make or other build systems.
* **C::B Share Config** – this is a tool that allows you to import and export parts of your Code::Blocks configuration. It’s particularly useful when you want to transfer settings between different computers or configurations.
* **C::B Launcher** – is a utility that helps manage the launching of the Code::Blocks IDE. This tool is useful for advanced users.
* **MinGW Compiler Suite** – The MinGW (Minimalist GNU for Windows) is a development environment that provides a native Windows port of the GNU Compiler Collection (GCC). It allows you to build native Window applications without relying on third-party runtime libraries.

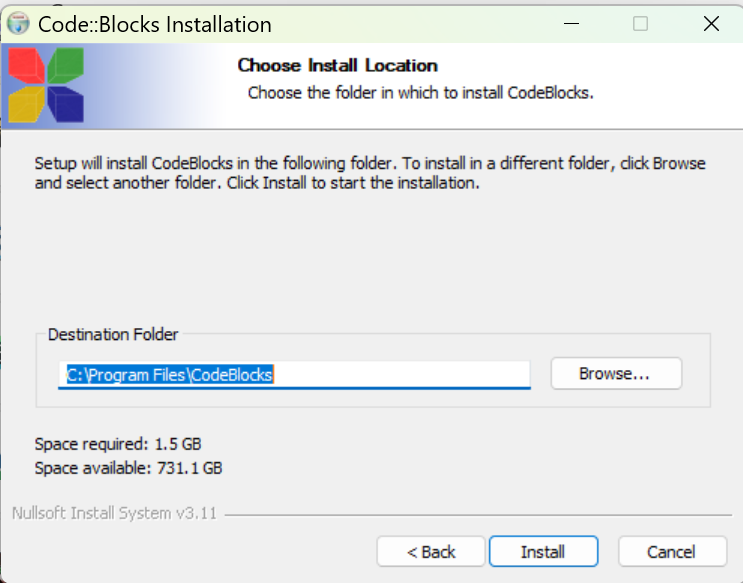


Figure - Installation location on your PC

I usually take the default location.

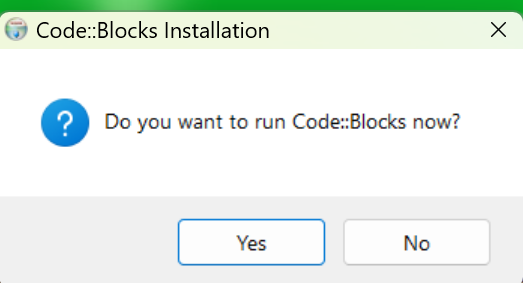


Figure - Prompt to run Code::Blocks now?

I clicked on “Yes” so I can enter a simple “hello world” program to make sure everything works.

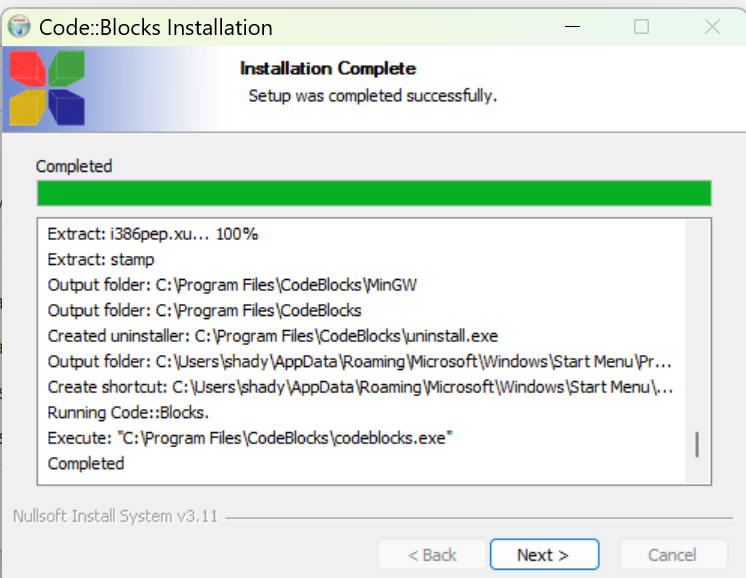


Figure - Code::Blocks installation completed dialog

I clicked on “Next >”.

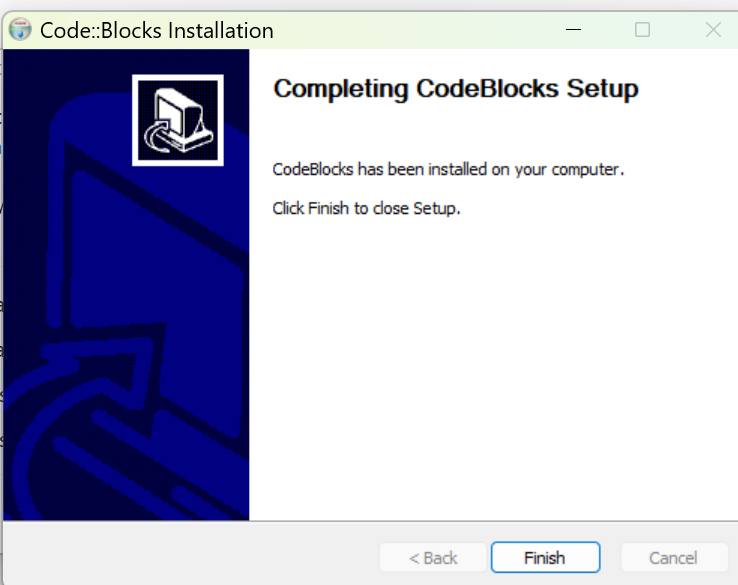


Figure - The final Code::Blocks setup screen

Click on “Finish” and you will see Code::Blocks starting if you clicked “Yes” to start Code::Blocks now.

### Starting up Code::Blocks

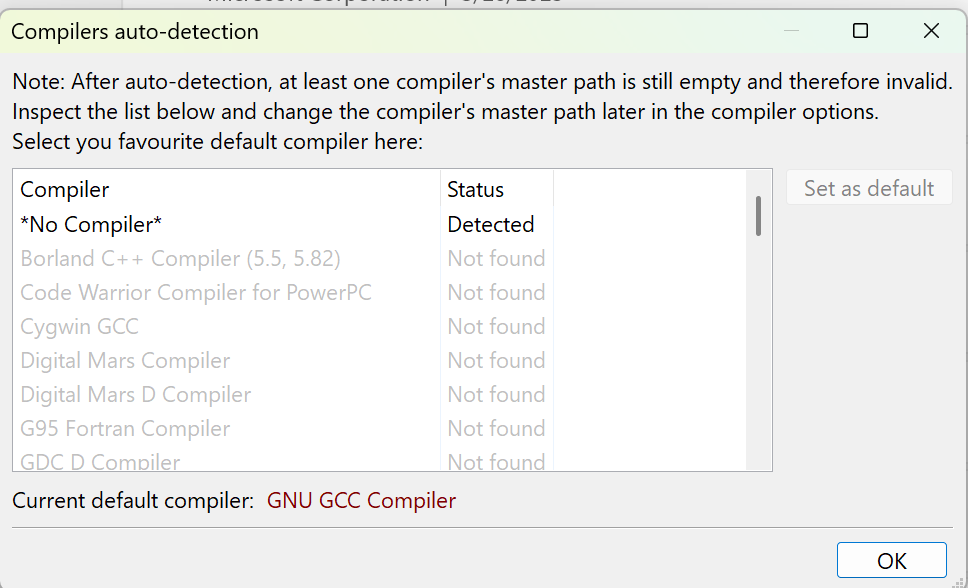


Figure - Code::Blocks lists all the compilers it detected

As you can see from the list you have the option of using many different compilers in Code::Blocks. We do expect Code::Blocks to find MinGW compiler because that is the version we downloaded.

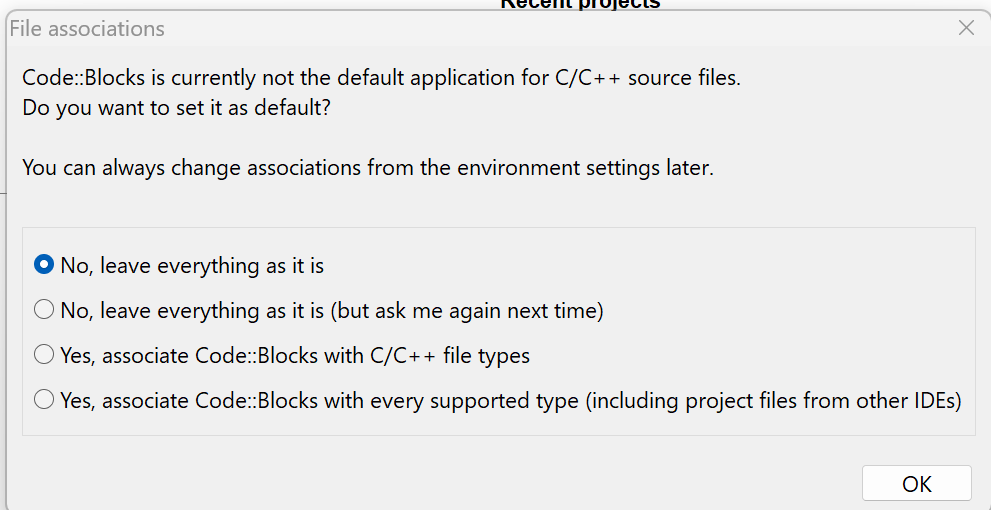


Figure - Option to update file association

I opted for the default of “No, leave everything as it is” since I do use Visual Studio 2022 for other projects.

## Starting a Project with Code::Blocks

A screenshot of a computer

AI-generated content may be incorrect.

Figure - The starting Code::Blocks screen

Note: I downloaded the Code::Blocks manual but the screenshots of the application appear to be dated!

### Making Adjustments to the screen

Another note: The screen icons and fonts appear to be small to me (it is probably due to my screen resolution). I made things larger by doing the following:

* Go to Settings 🡺 Environment 🡺 View
* Increase Message logs’ font size the Toolbar icon size

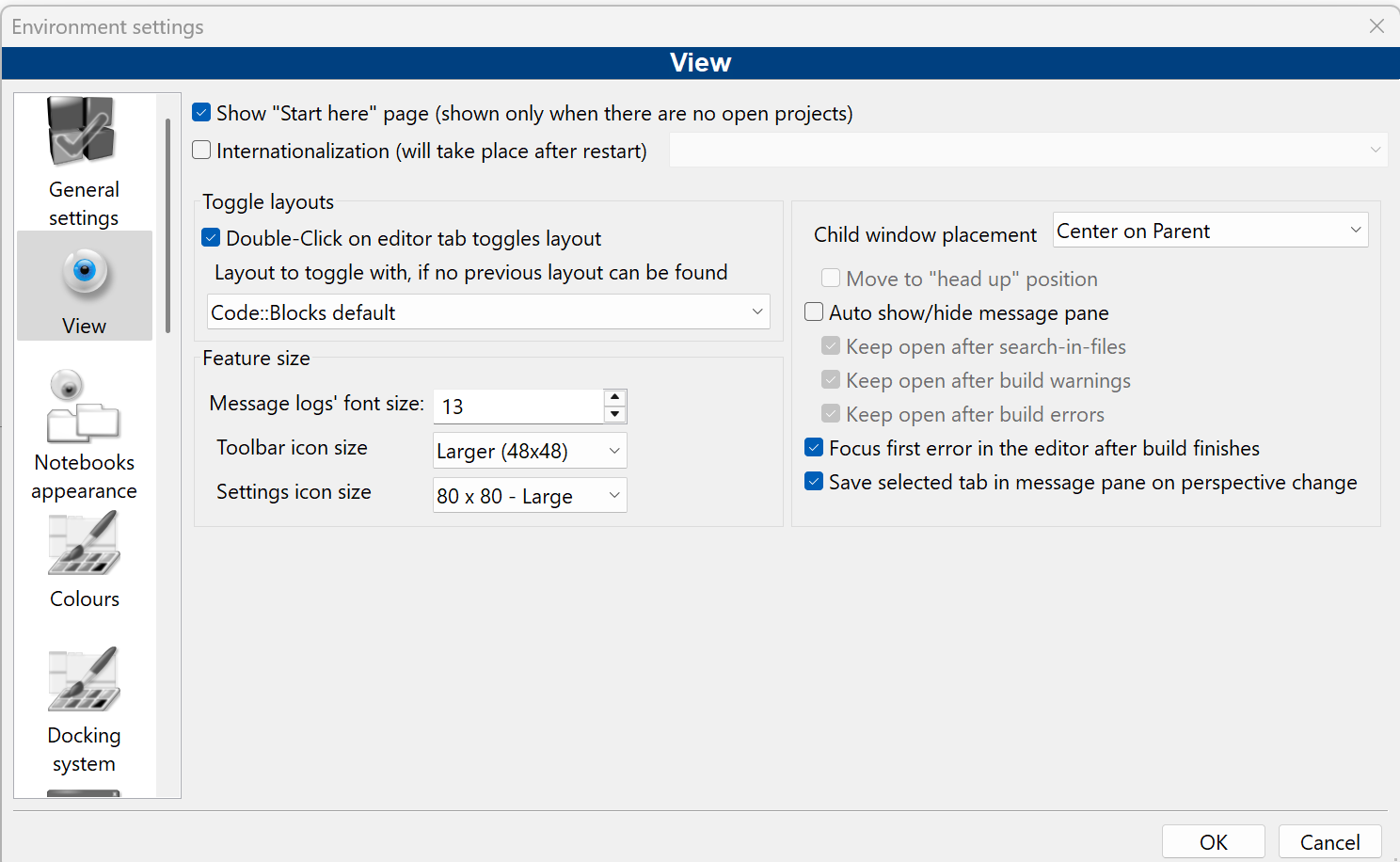
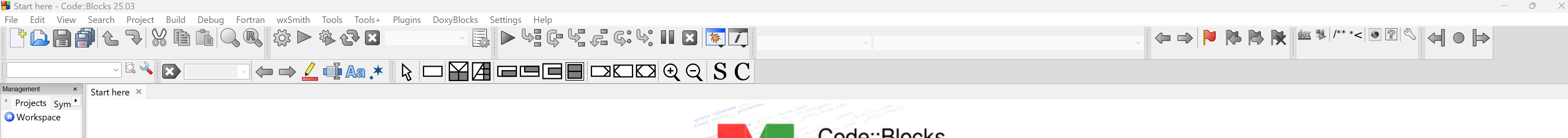


Figure - Increasing the toolbar icon sizes

* Click “OK”
* Select File 🡺 Save Everything
* Restart Code::Blocks



I also increased the font size used by:

* Select Settings 🡺 Editor

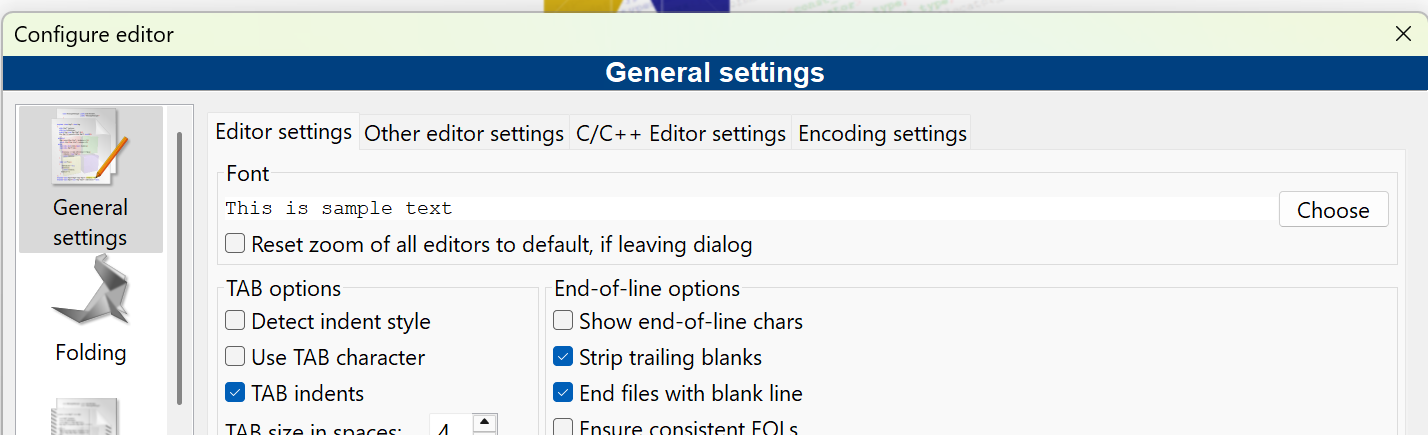


Figure - Updating font-size

* Click on “Choose” button

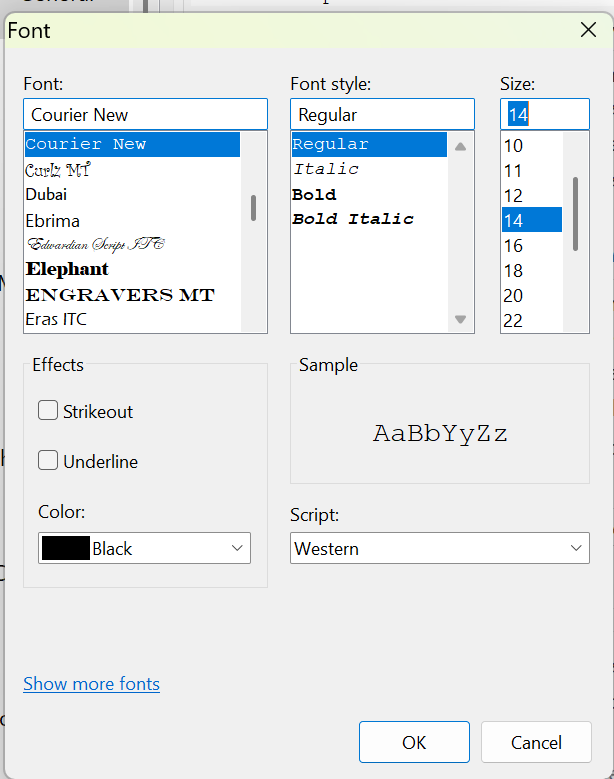


Figure - Select desired font size

* I selected 14 and clicked on “OK”

### Creating the Project

* Click on “Create a new project” link



Figure - Creating a new Project

* Select “Console Application”

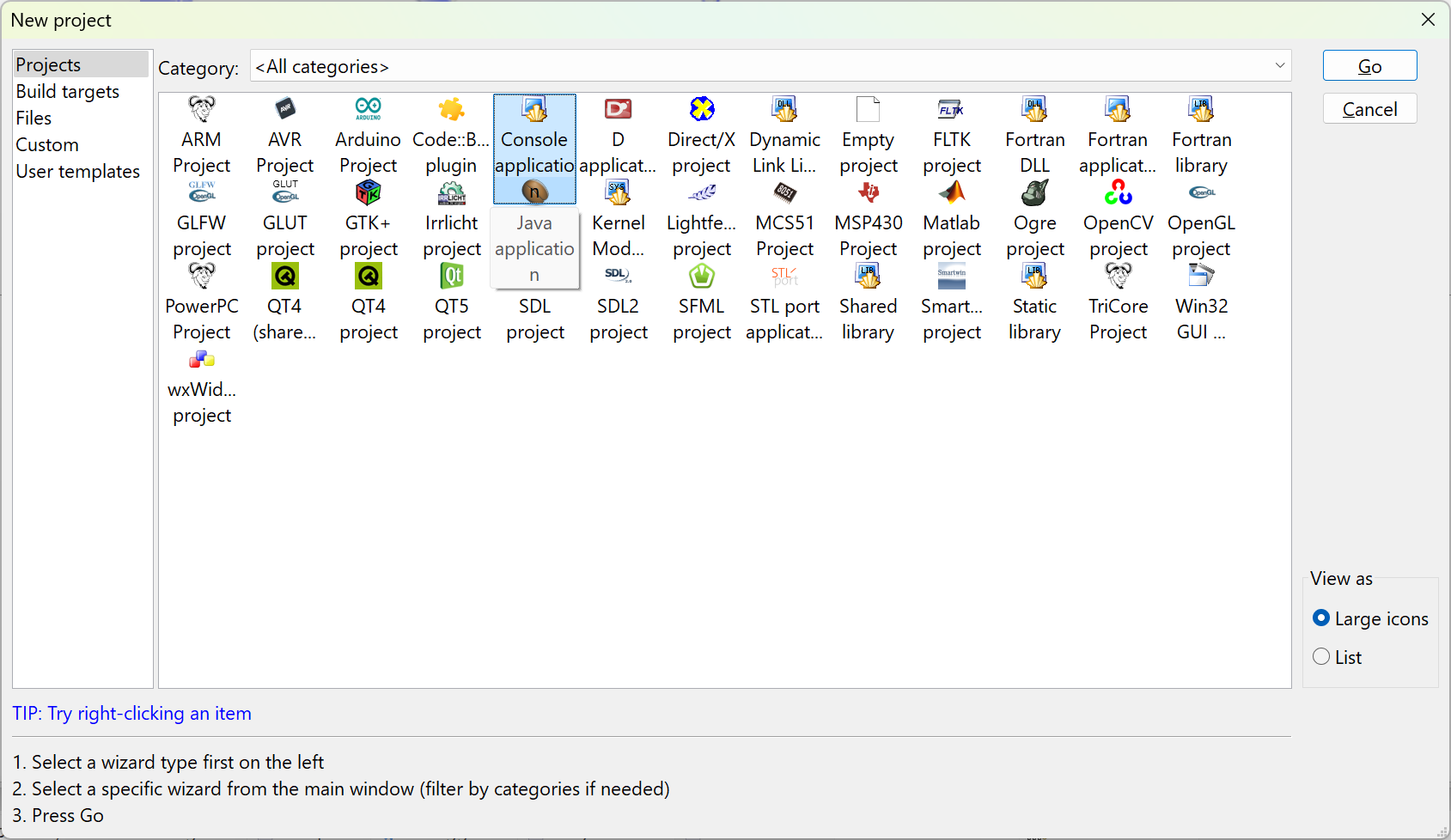


Figure - Creating our first console application

* Click on “Go”, if this is your first time you will see the following dialog box appear:

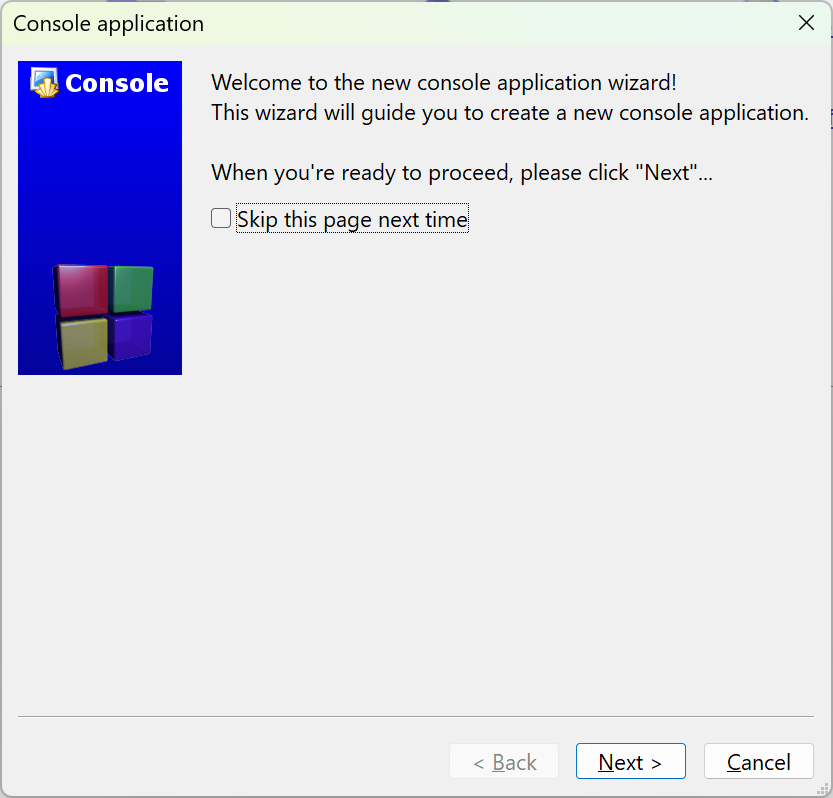
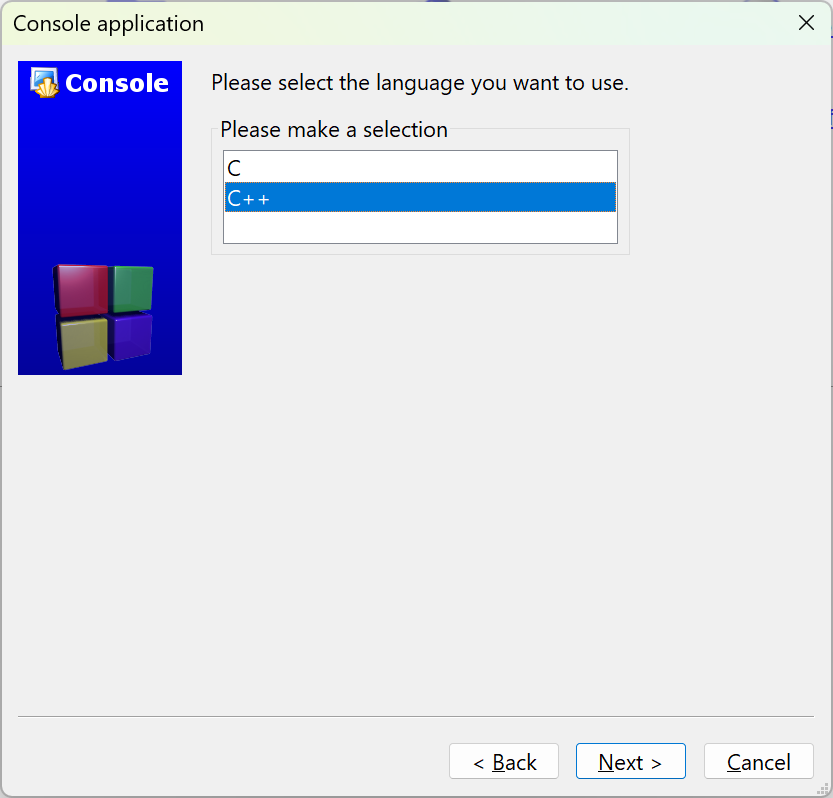


Figure - The Console application wizard starting...

* Click on “Skip this page next time” and press on “Next >”



* Take the default C++ and click on “Next >”
* Fill in the Project information:

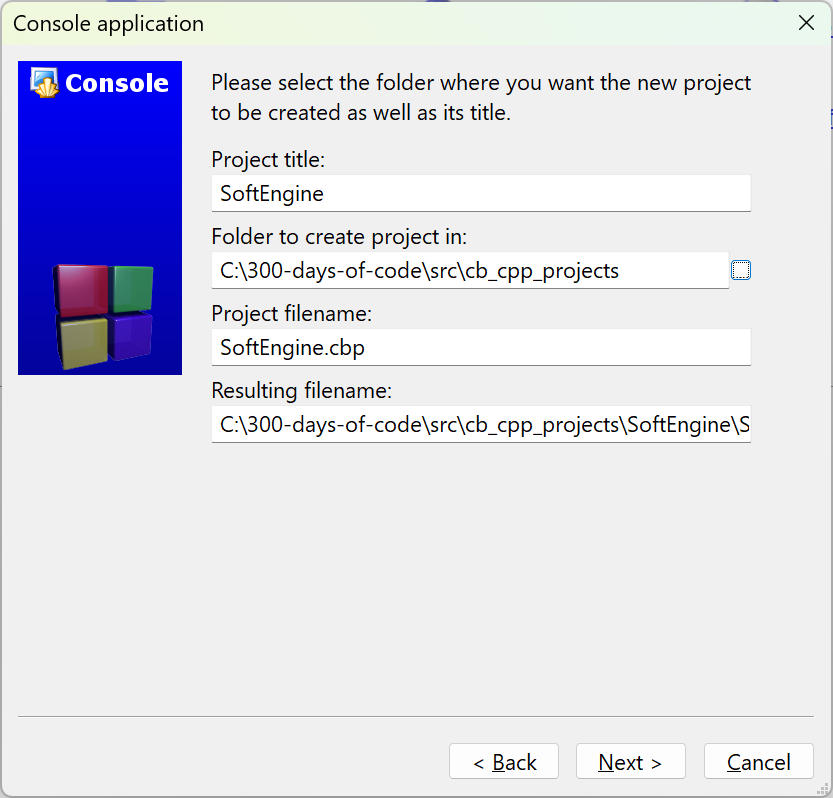


Figure - Project information

The Project name will be SoftEngine. Since this is part of my 300-days-of-code effort I select a folder where I plan on placing all my Code::Blocks C++ projects. Enter a location that makes sense for your setup.

* Click on “Next >”

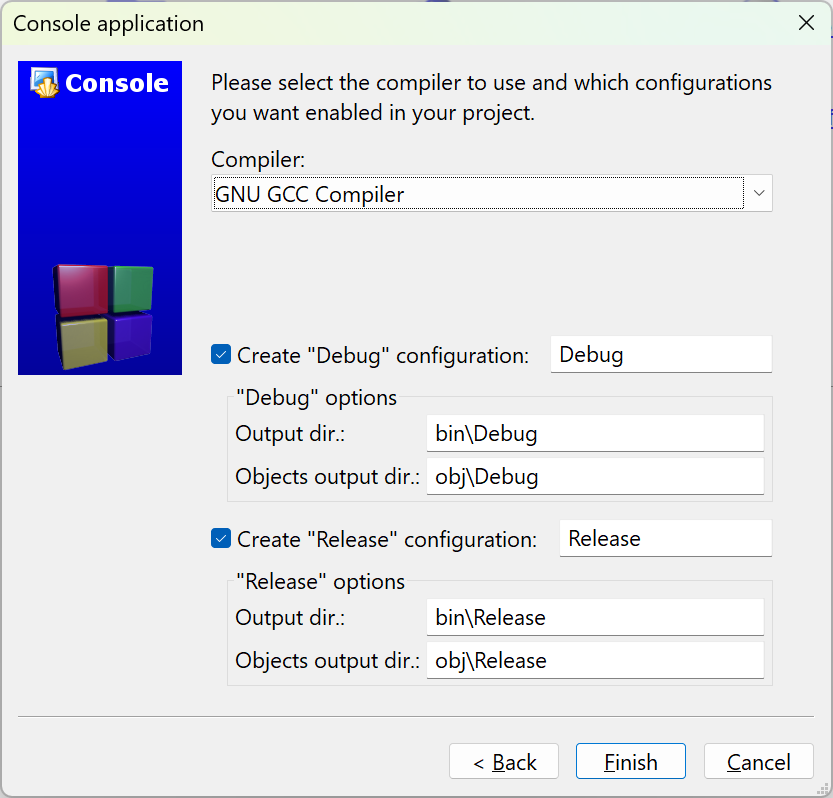


Figure - Final step in creating our game engine project

* Make sure the “Debug” and “Release” configuration are selected and click on “Finish”

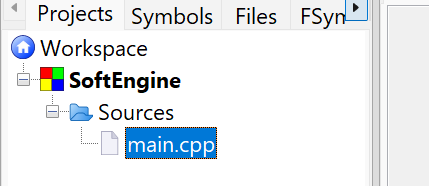


Figure - Default main.cpp is created

* Following the video author’s convention, let’s rename the file main.cpp to Main.cpp
  + Right-click on the filename and select Rename file:

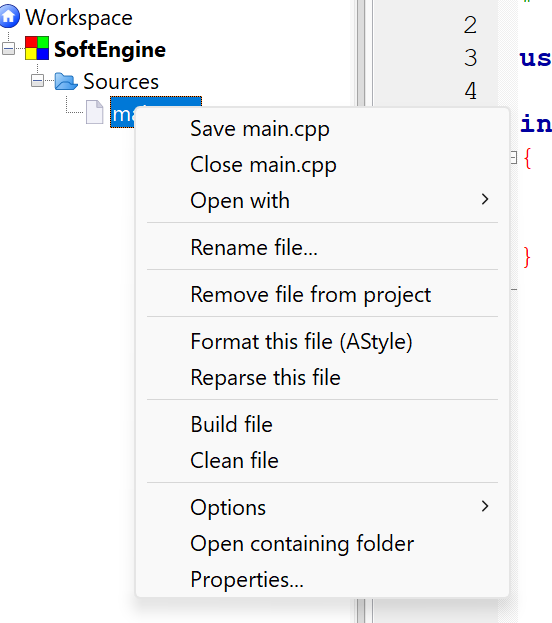


Figure - Context menu for file

* Enter the name Main.cpp:

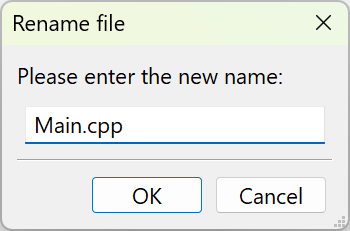


Figure - Rename dialog box

* Click “OK”

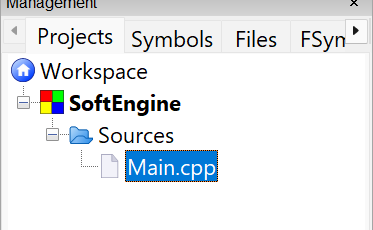


Figure - Update name in Workspace

### Build and Run the Program

The ***Workspace*** contains one or more projects, in our case it shall only contain one project – SoftEngine. The workspace is the most top-level container. A ***project*** contains one or more build targets and the project’s files.

You should familiarize yourself with the following icons/operations:

A diagram of a program

AI-generated content may be incorrect.

Figure - Common operations

* **Build** – this function compiles your source code into an executable program. It processes the code files in your project, checks for errors, and links them to create a standalone application. This is where you written code is transformed into something that can run on your machine.
* **Run** – this function executes the program that you’ve built. If the program is a console application (which is true in our case), it will open a terminal or console window to display the output.
* **Build and Run** – it combines the two steps above
* **Rebuild** – This is a more thorough version of Build. While “Build” compiles *only* the files that have been modified since the last build, “Rebuild” forces the *entire* project to be recompiled from scratch, regardless of whether files have changed.
* **Abort** – this command is used to stop an ongoing build or compilation process. It is specifically for interrupting the build process.

#### Let’s Build and run!

Our Main.cpp code is:

1. #include <iostream>

2.

3. using namespace std;

4.

5. int main()

6. {

7. cout << "Hello world!" << endl;

8. return 0;

9. }

10.

The code above is our simple “Hello, world!” program.

* Click on the “Build and Run” icon

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Result of building and running our Main.cpp

* Press any key on the keyboard to dismiss

## Install and Setup SDL

### What is SDL?

SDL, or ***Simple DirectMedia Layer***, is a cross-platform software development library designed to provide a hardware abstraction layer for multimedia hardware components. It was originally created by Sam Lantinga in 1998. SDL is widely used for developing high-performance computer games and multimedia applications across various operating systems, including Android, iOS, Linux, macOS, and Windows.

The library is written in C and provides an application programming interface (API) in C, with bindings available for other programming languages. Over the years, SDL has evolved significantly, with major updates like SDL 2.0 in 2013, which introduced better support for 3D hardware acceleration. SDL 3.0, released in January 2025, brought further enhancements and new features.

SDL is free and open-source software, licensed under the zlib License since version 2.0, allowing developers to use it in both open-source and closed-source projects. It has been extensively used in the industry, with hundreds of games, applications, and demos built using the library.

The website to obtain more information is: <https://www.libsdl.org/>

### Installing SDL2 and SDL2\_image

* Create a folder that will hold both SDL2 and SDL2\_image download files. I will create the folder D:\SDL2\_dev folder.
* Download the latest SDL2
  + Go to: <https://github.com/libsdl-org/SDL>

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Versions of sdl on Github

The sdl2 link displays a list of projects that uses sdl2.

* Go to <https://github.com/libsdl-org/SDL/releases/tag/release-2.32.4> to get the latest SDL2 version.

Note: We will be using the latest version of SDL – SDL2.32.4. The video series uses SDL2-2.0.10.

Another note: I prefer to learn and use SDL3 but there are too many differences between SDL2 and SDL3 that will introduce another set of issues that will get in the way of the video series!

* Navigate to the release github url (for me it is: <https://github.com/libsdl-org/SDL/releases/tag/release-3.2.10>)

#### What version of SDL2 should I use?

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Versions of SDL3 to choose from

Since I am using Code::Blocks with mingw I will utilize SDL2-devel-2.32.4-mingw.zip version. If you are using Visual Studio you should download and install SDL2-devel-2.32.4-VC.zip

Note: If you want to utilize SDL2 you can find the latest SDL2 release on the same github website: <https://github.com/libsdl-org/SDL/releases>

* Unzip the file[[1]](#footnote-1) to D:\SDL2\_dev:

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Unzipping the SDL2 version

You will see:

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Unzipped SDL2 file

* Open the INSTALL.md file to determine which of the folders you will use:

A number with numbers and lines

AI-generated content may be incorrect.

Figure - Contents of INSTALL.txt file

I will be using the 64-bit architecture of the files in x86\_64-w64-mingw32. Select the folder that makes sense for your machine and setup.

* Go to [https://github.com/libsdl-org/SDL\_image/releases to obtain the 2.8.9](https://github.com/libsdl-org/SDL_image/releases%20to%20obtain%20the%202.8.9) release of SDL2\_image

A screenshot of a computer

AI-generated content may be incorrect.

Figure - SDL2\_image choices

* I downloaded SDL2\_image-devel-2.8.8-mingw.zip to be consistent with the version of SDL2 I downloaded.
* Unzip in the D:\SDL2\_dev folder

You will see the following two top-level folders in D:\SDL2\_dev:

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Top-level folders

Note: Unlike the video – I will leave the files in the location I unzipped them.

### Setting up Code::Blocks to access SDL folders

* Open the project (if not open) we created SoftEngine. To re-open the project
  + Open Code::Blocks
  + Click on “Open an existing project”
  + Navigate to the folder you placed your project and select SoftEngine.cbp and click “Open”

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Opening an existing project

* Go to Settings 🡺 Compiler…

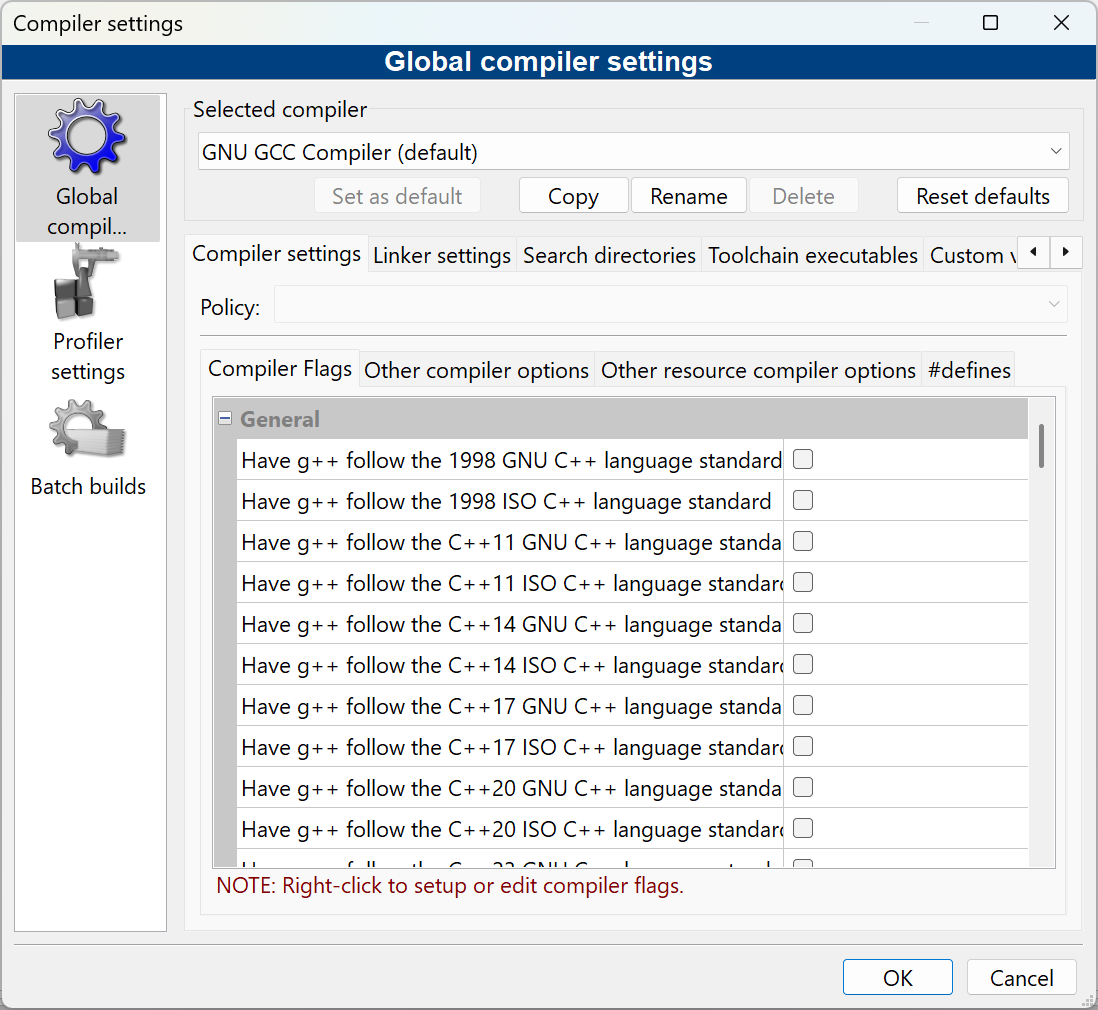


Figure - Global compiler settings

* Click on the “Linker settings” tab

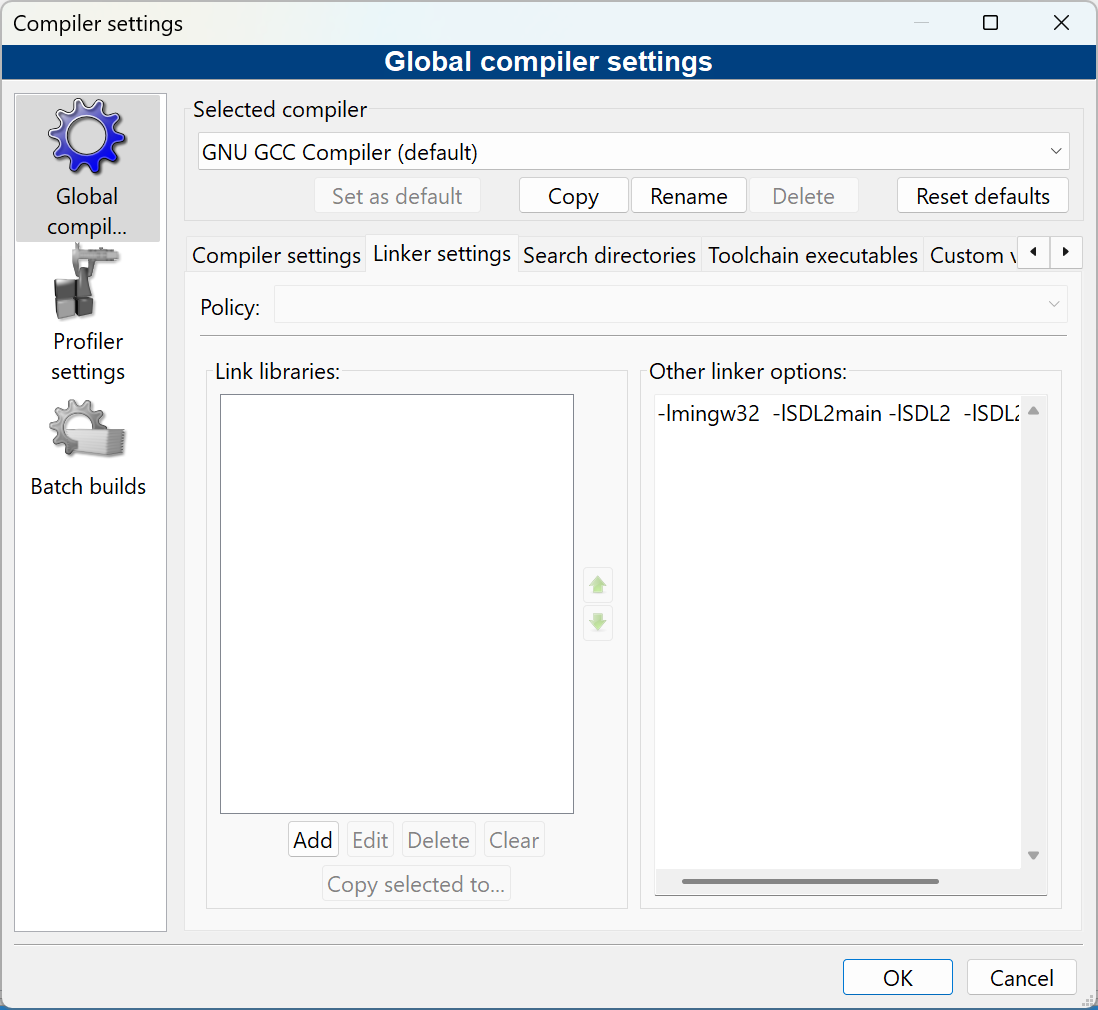
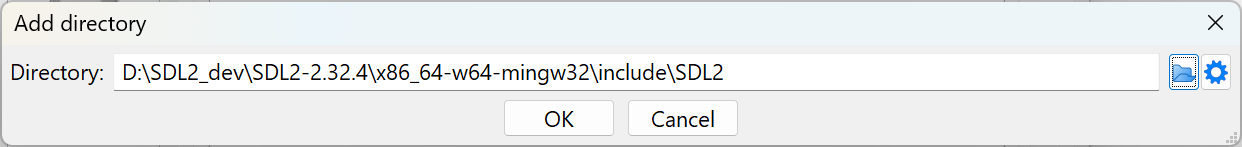


Figure - Adding linker options

* In the “Other linker options:” input box enter:

-lmingw32 -lSDL2main -lSDL2 -lSDL2\_image

* Click on the “Search directories” tab
* Make sure the “Compiler” tab is selected
* Click on “Add” and navigate to the include folder for SDL



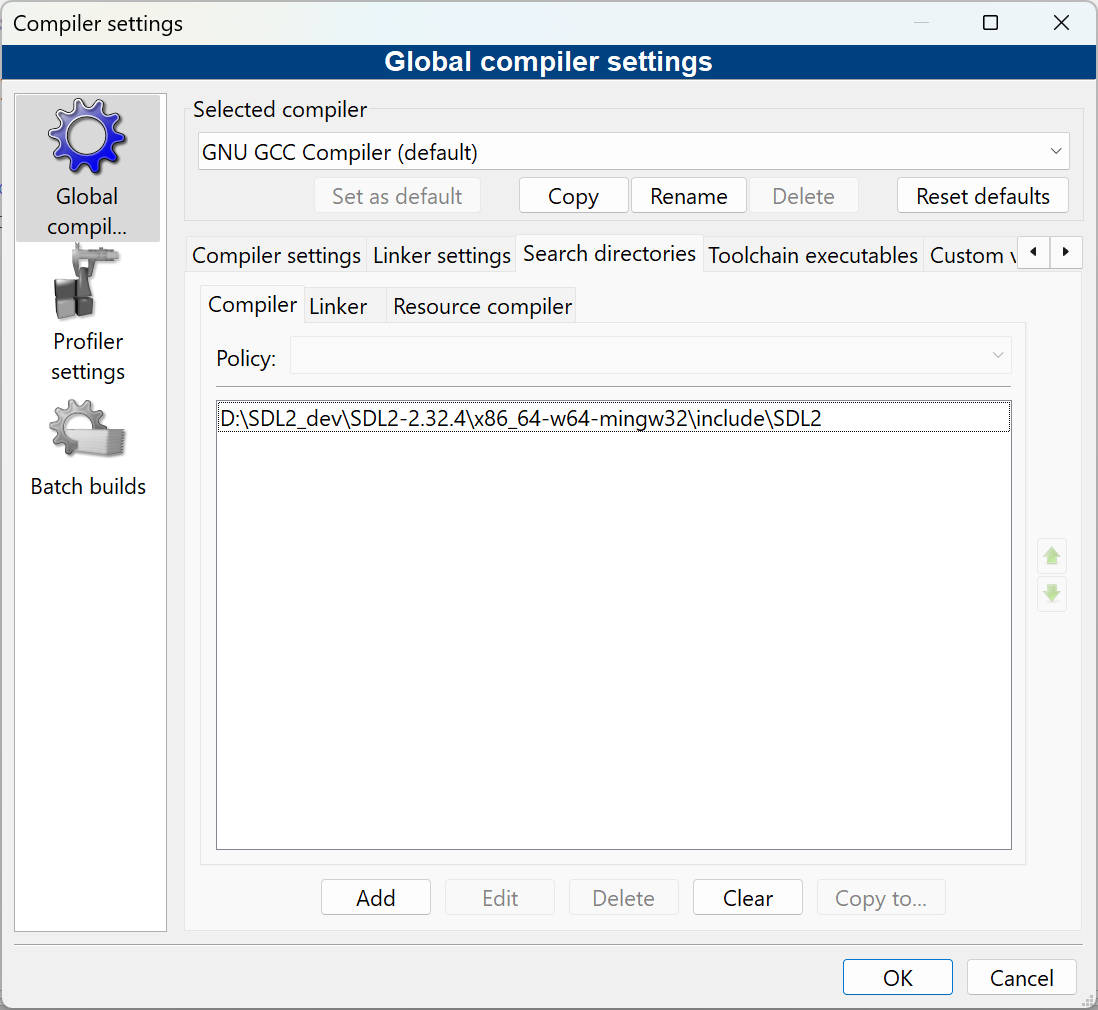


Figure - Adding SDL3 include folder for the compiler

* Add the SDL2\_image include as well

A screenshot of a computer

AI-generated content may be incorrect.

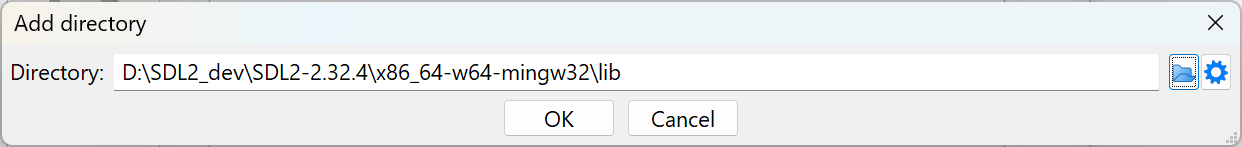
* Select the “Linker” sub-tab and add the location of the lib folder:

A screenshot of a search box

AI-generated content may be incorrect.

Figure - Linker sub-tab

* Navigate to the SDL2 lib folder



* Click on OK
* Add lib folder under SDL\_Image directory

A screenshot of a computer

AI-generated content may be incorrect.

Figure - SDL2 and SDL2\_image lib folders for the linker

### Testing the setup

* Update the code as follows:

1. #include <iostream>

2. #include "SDL.h"

3.

4. using namespace std;

5.

6. int main(int argc, char\*\* argv)

7. {

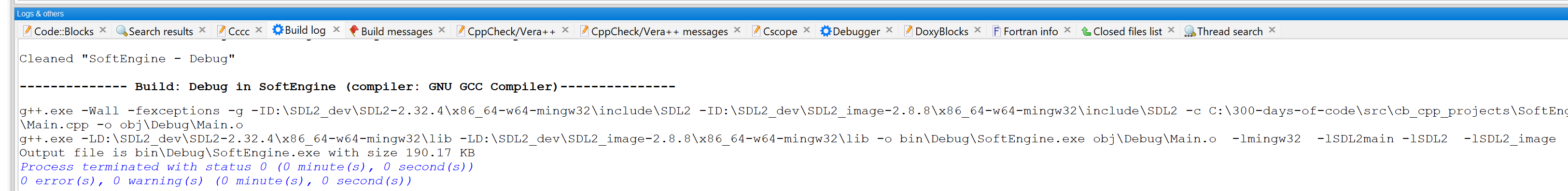
8. cout << "Hello world!" << endl;

9. return 0;

10. }

11.

* Click on the “Build” or “Rebuild”



You should see no errors.

The video uses SDL\_INIT\_EVERYTHING to initialize all the SDL subsystems at the same time. This is considered a bad practice and the flag SDL\_INIT\_EVERYTHING no longer exists in SDL3. We will follow the video but note that it does not exist in SDL3.

* Let’s initialize SDL to make sure everything runs correctly. Update the Main.cpp:

1. #include <iostream>

2. #include "SDL.h"

3.

4. using namespace std;

5.

6. int main(int argc, char\*\* argv)

7. {

8. if (SDL\_Init(SDL\_INIT\_EVERYTHING) == 0) {

9. cout << "SDL\_Init worked!" << endl;

10. }

11. SDL\_Quit(); // clean up resources

12. return 0;

13. }

* Try to build it again. It should build.
* Now, try to run the application. It fails:

A screen shot of a computer error

AI-generated content may be incorrect.

Figure - Application failed to find SDL3.dll file

The problem is that the application could not find the SDL2.dll file that is located in the SDL2 \bin folder:

A screenshot of a computer

AI-generated content may be incorrect.

Figure - The location of the SDL2.dll file

* The video presenter suggests putting the dll files for SDL2 and SDL2\_image in the project debug directory:

A screenshot of a computer

AI-generated content may be incorrect.

I usually have a D:\bin or a C:\bin directory where I put in all my generic tools and common dlls that will be utilized across many projects:

A screenshot of a computer

AI-generated content may be incorrect.I recommend that a more general location be created similar to the above and the folder be placed in the environment path.

Which ever you choose, once the \*.dll files of SDL2.dll and SDL2\_image.dll are made accessible the program should now execute successfully:

A screen shot of a computer

AI-generated content may be incorrect.

Figure - Program with SDL component running ran successfully

# 2. Game Loop for SDL Game

If you are opening the project anew for the next video a fast way is to select File 🡺 Recent projects and select the project you are working on .

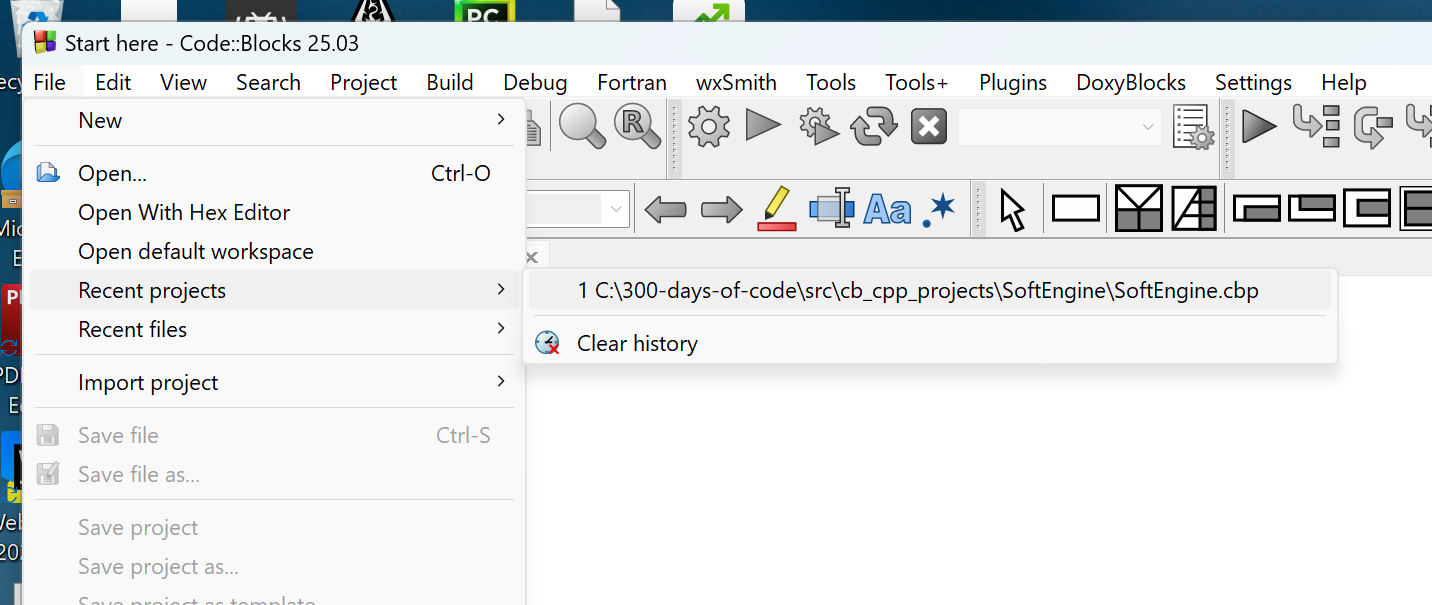


Figure - Selecting a recent project

The goal of these set of videos is to create a 2D game engine that appears as follows:



Figure - The final look of our 2D Game Engine

## Create an Engine class file

* Select File 🡺 New 🡺 Class…

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Adding a new class to the project

* Fill in the “Create new class” dialog as shown:

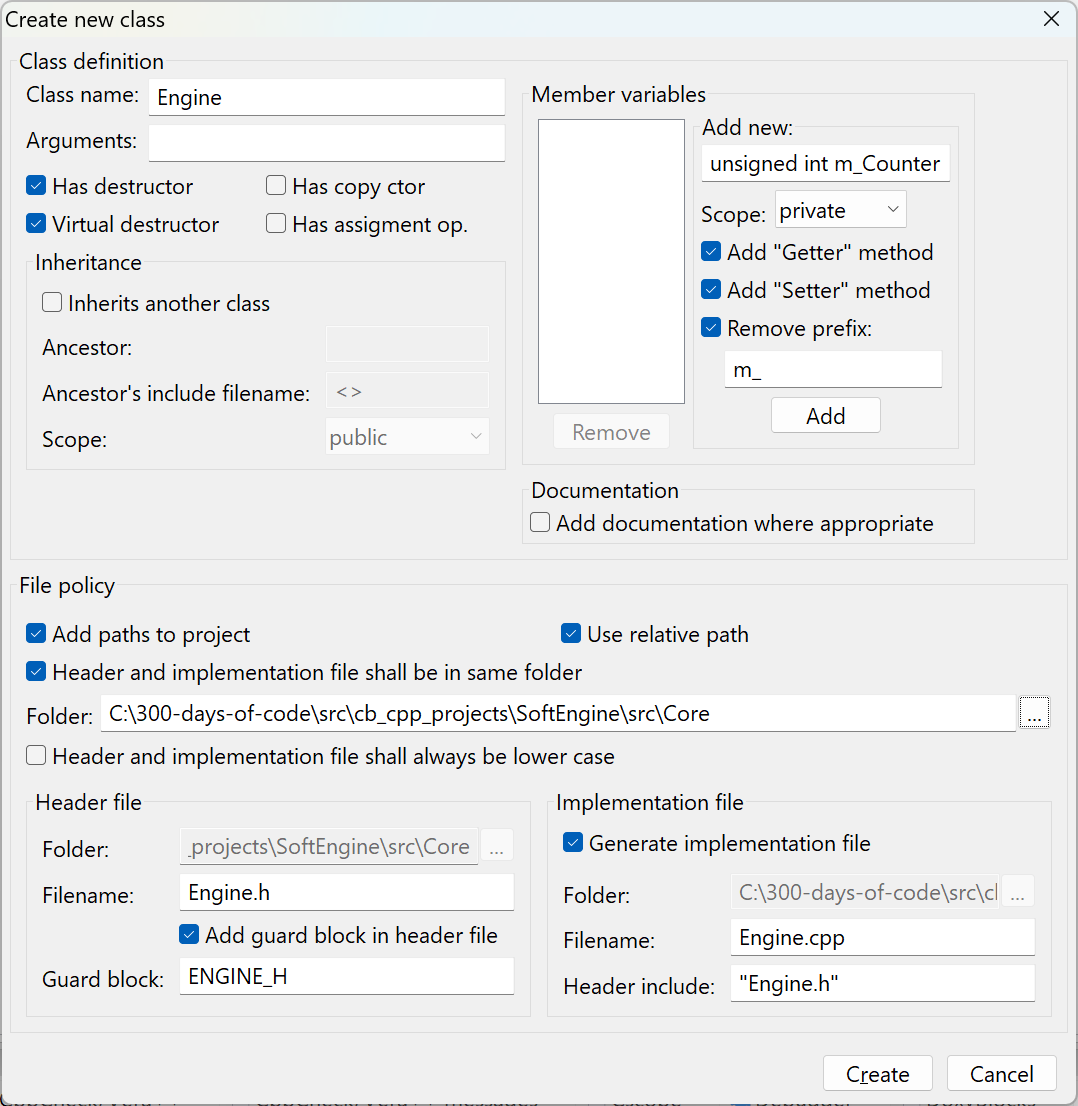


Figure - Creating our Engine.cpp and Engine.h files

Note, the Folder for our new class is \src\Core.

* Click on “Create”

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Add to project prompt?

* Click “Yes” to the above prompt.

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Add to Debug and Release prompt

* Click “OK” to the above prompt.

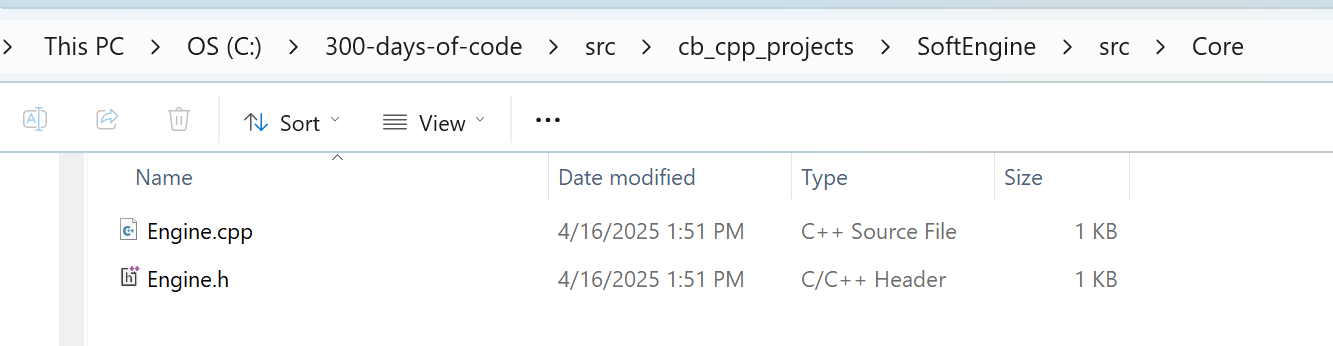
The project will appear as:

A screenshot of a computer

AI-generated content may be incorrect.

Figure - Project view of the files

In fact if you examine your folders:



You see that the Engine.cpp and Engine.h are actually in the same folder.

1. I use 7-zip to manage my zip files [↑](#footnote-ref-1)