

# SFML Installation Guide

## Step 1: Downloading SFML

Let's begin the installation by heading over to the SFML website and downloading the correct version of SFML. [This link](#) will open up the download page for SFML 2.5.0. The reason we are not installing the latest version is that the version of MinGW that is installed on your PCs is old, and so the latest version of SFML will not be compatible with your version. Now click on the button to download the SFML library:

Home » Download » SFML 2.5.0

Download SFML 2.5.0

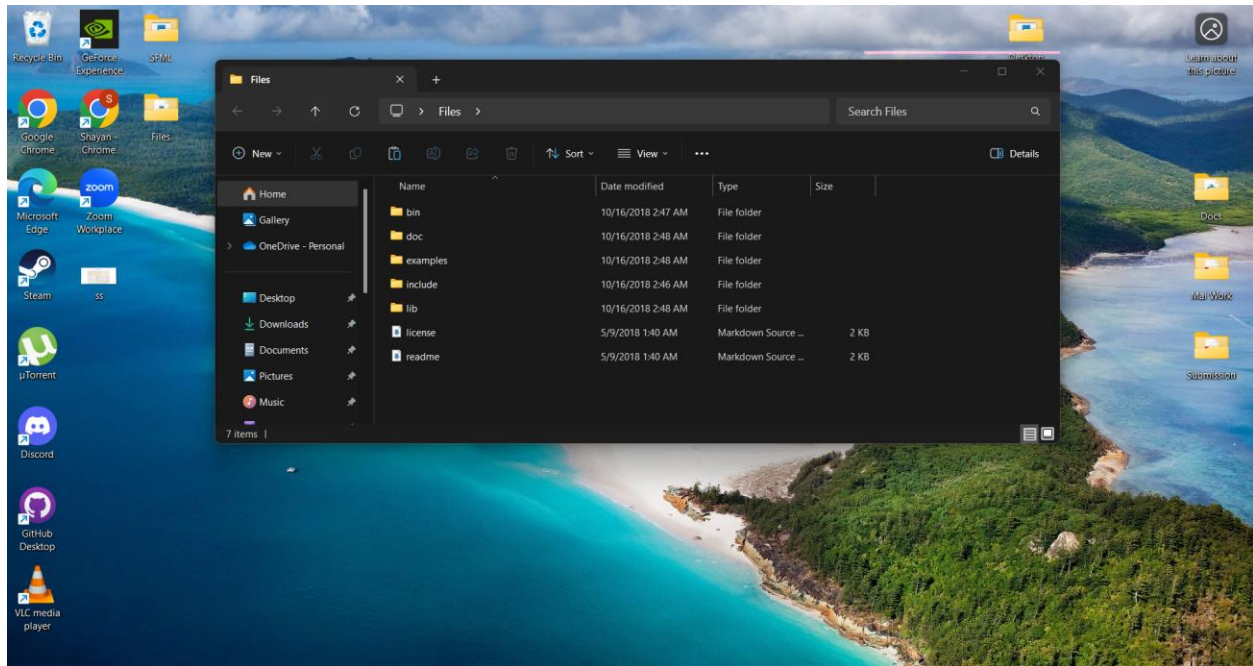
On Windows, choosing 32 or 64-bit libraries should be based on which platform you want to compile for, not which OS you have. Indeed, you can perfectly compile and run a 32-bit program on a 64-bit Windows. So you'll most likely want to target 32-bit platforms, to have the largest possible audience. Choose 64-bit packages only if you have good reasons.

**Unless you are using a newer version of Visual Studio, the compiler versions have to match 100%!**  
In case you are using Visual Studio 2015 or newer, you can go ahead and use the SFML version compiled for Visual C++ 15 (VS 2017), which is still compatible with newer versions of the toolchain.  
Here are links to the specific MinGW compiler versions used to build the provided packages:  
TDM 5.1.0 (32-bit), MinGW Builds 7.3.0 (32-bit), MinGW Builds 7.3.0 (64-bit)

Visual C++ 15 (2017) - 32-bit	Download   17.8 MB	Visual C++ 15 (2017) - 64-bit	Download   19.4 MB
Visual C++ 14 (2015) - 32-bit	Download   17.6 MB	Visual C++ 14 (2015) - 64-bit	Download   19.4 MB
Visual C++ 12 (2013) - 32-bit	Download   17.8 MB	Visual C++ 12 (2013) - 64-bit	Download   19.6 MB
GCC 5.1.0 TDM (SJLJ) - Code::Blocks - 32-bit	Download   15.6 MB		
GCC 7.3.0 MinGW (DW2) - 32-bit	Download   17.0 MB	GCC 7.3.0 MinGW (SEH) - 64-bit	Download   17.9 MB

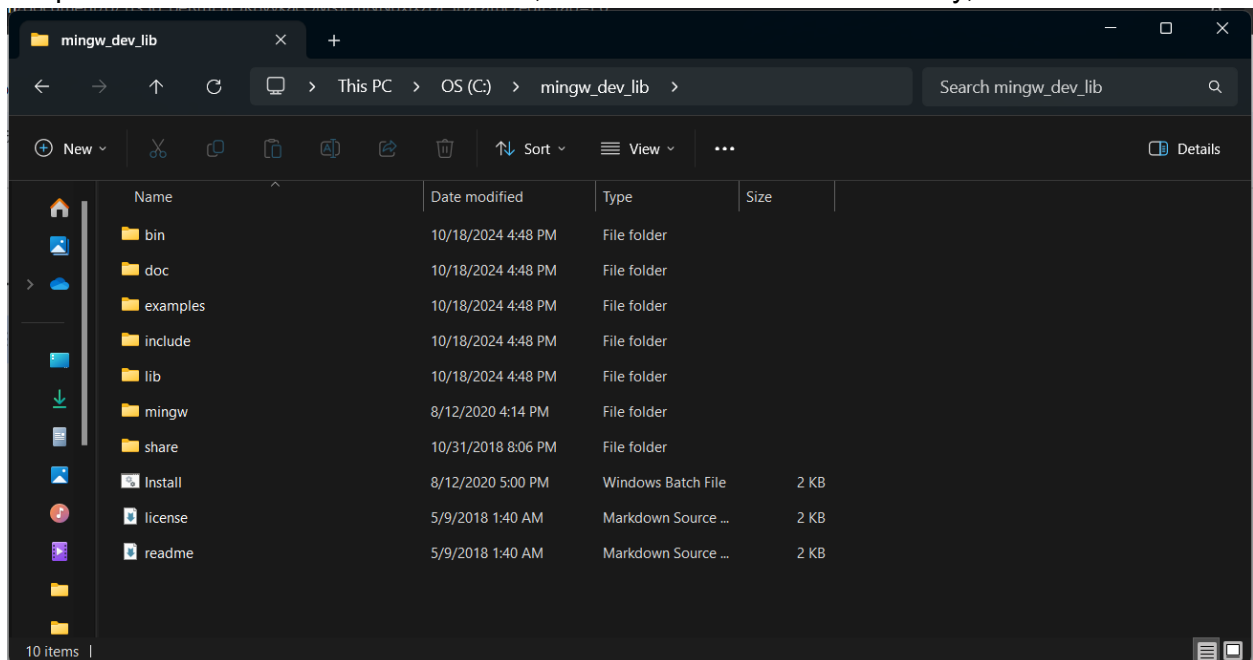
On Linux, if you have a 64-bit OS then you have the 64-bit toolchain installed by default. Compiling for 32-bit is possible, but you have to install specific packages and/or use specific compiler options to do so. So downloading the 64-bit libraries is the easiest solution if you're on a 64-bit Linux.

Once your SFML library has been downloaded extract it to a folder. Here, for example, I have extracted the library inside a **Files** folder on the Desktop:



## Step 2: Installing SFML

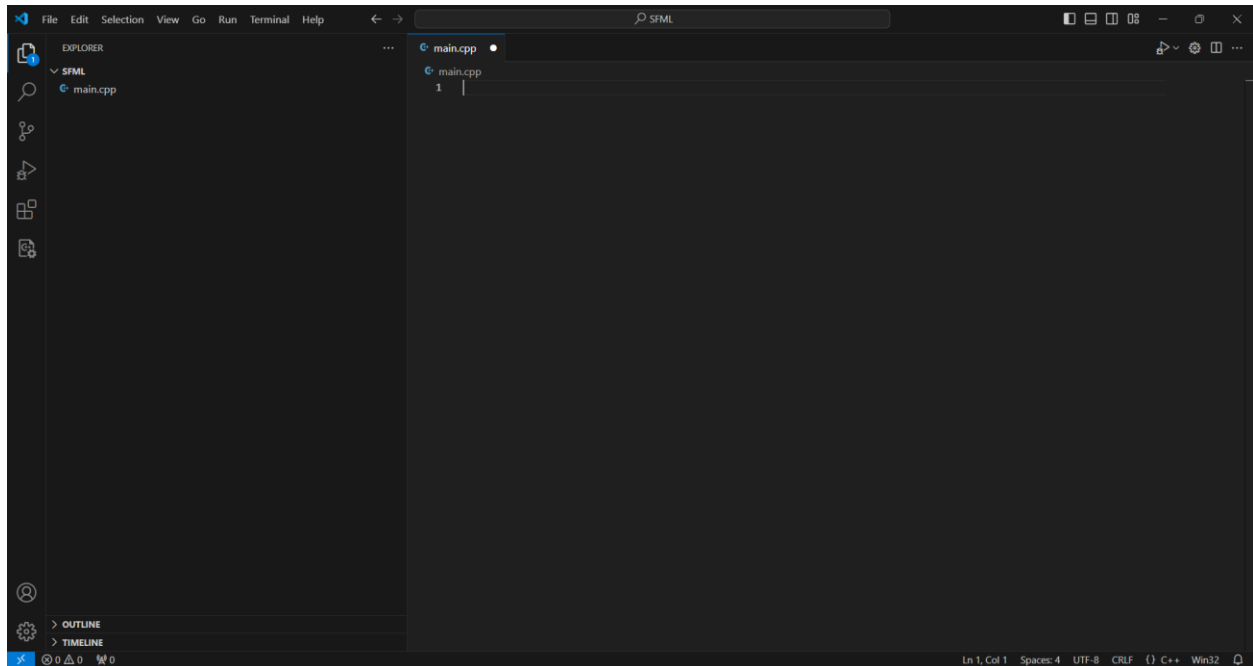
Now copy the files in the directory, and head over to your MinGW installation folder. Here you will paste the contents of the **Files** folder, which contains the SFML library, and :



Once, you have copied the contents, SFML will be ready to use in your projects.

### Step3: Testing our Installation

Now that SFML is completely installed, we will open up a sample project folder and create a simple C++ file. Here I've created a folder titled **SFML** on the Desktop:



Next, copy and paste the following code inside the **main.cpp** file:

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

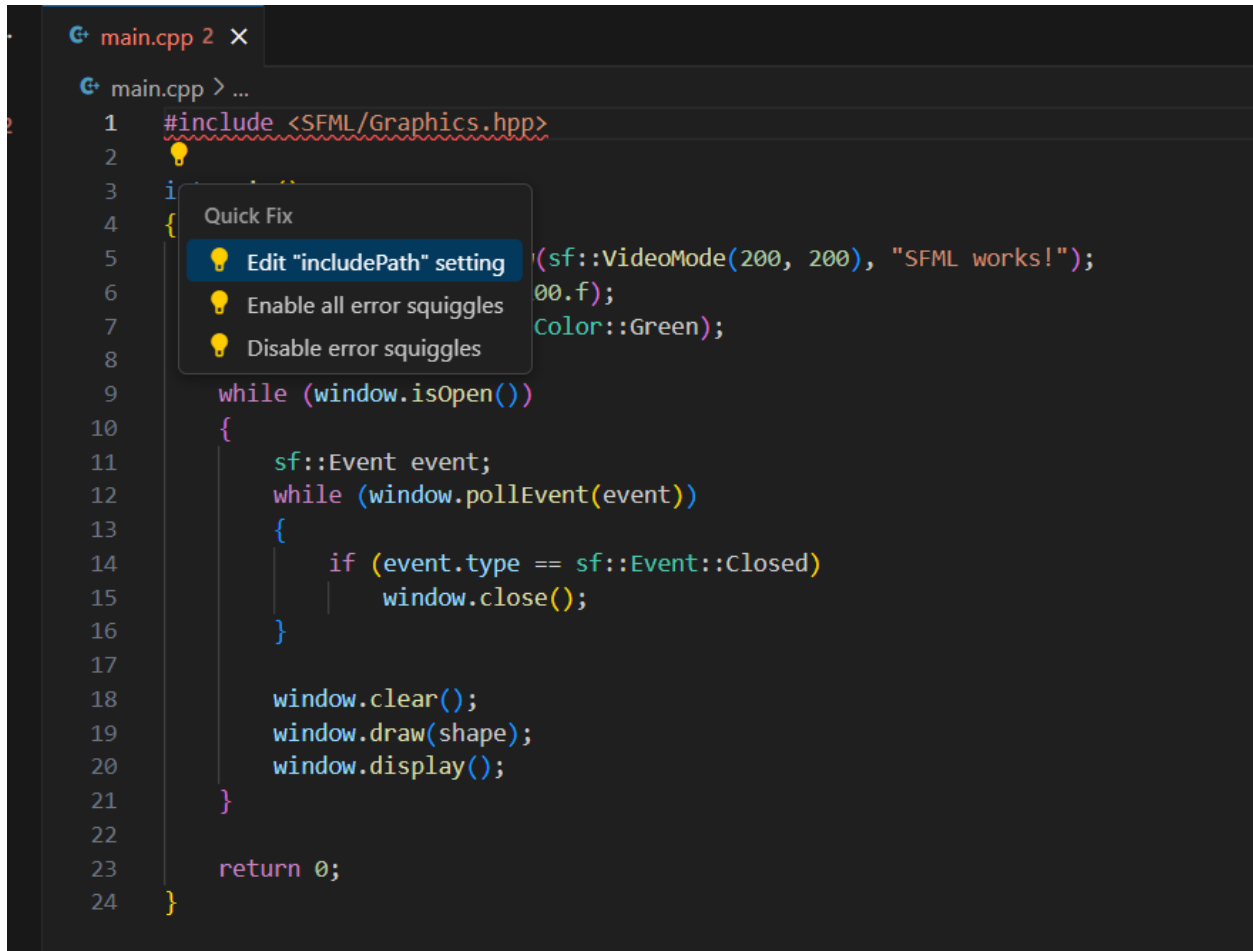
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        window.draw(shape);
        window.display();
    }
}
```

```
}

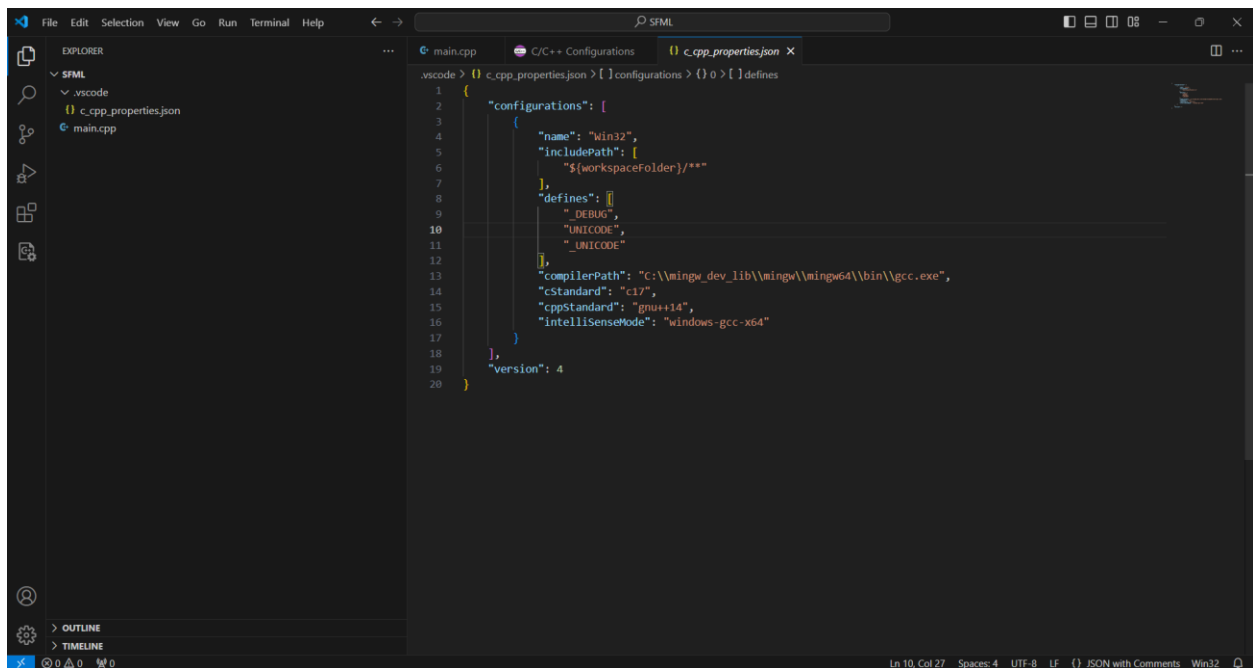
return 0;
}
```

As you can see Visual Studio will give you an include error that the SFML graphics library is not detected. Though this error is irrelevant to the installation, you can fix this by simply clicking on the **light bulb icon** to open the **Quick Fix** bar. Here, select the **Edit "includePath" setting** button:



```
main.cpp 2 x
main.cpp > ...
1  #include <SFML/Graphics.hpp>
2  //
3  int main()
4  {
5      sf::VideoMode mode(sf::VideoMode(200, 200), "SFML works!");
6      sf::Window window(mode, "SFML window");
7      sf::Color color(sf::Color::Green);
8
9      while (window.isOpen())
10     {
11         sf::Event event;
12         while (window.pollEvent(event))
13         {
14             if (event.type == sf::Event::Closed)
15                 window.close();
16         }
17
18         window.clear();
19         window.draw(shape);
20         window.display();
21     }
22
23     return 0;
24 }
```

This will create a `.vscode` folder in your project directory containing a `c_cpp_properties.json` file. Next, you will open up the JSON file:



The screenshot shows the Visual Studio Code interface with the `c_cpp_properties.json` file open. The Explorer sidebar on the left shows the project structure with `.vscode`, `c_cpp_properties.json`, and `main.cpp`. The main editor displays the JSON configuration for the `Win32` configuration. The `includePath` currently only contains the workspace folder path.

```
1 {
2   "configurations": [
3     {
4       "name": "Win32",
5       "includePath": [
6         "${workspaceFolder}/*"
7       ],
8       "defines": [
9         "_DEBUG",
10        "UNICODE",
11        "_UNICODE"
12      ],
13      "compilerPath": "C:\\mingw_dev_lib\\mingw\\mingw64\\bin\\gcc.exe",
14      "cStandard": "c17",
15      "cppStandard": "gnu++14",
16      "intelliSenseMode": "windows-gcc-x64"
17    }
18  ],
19   "version": 4
20 }
```

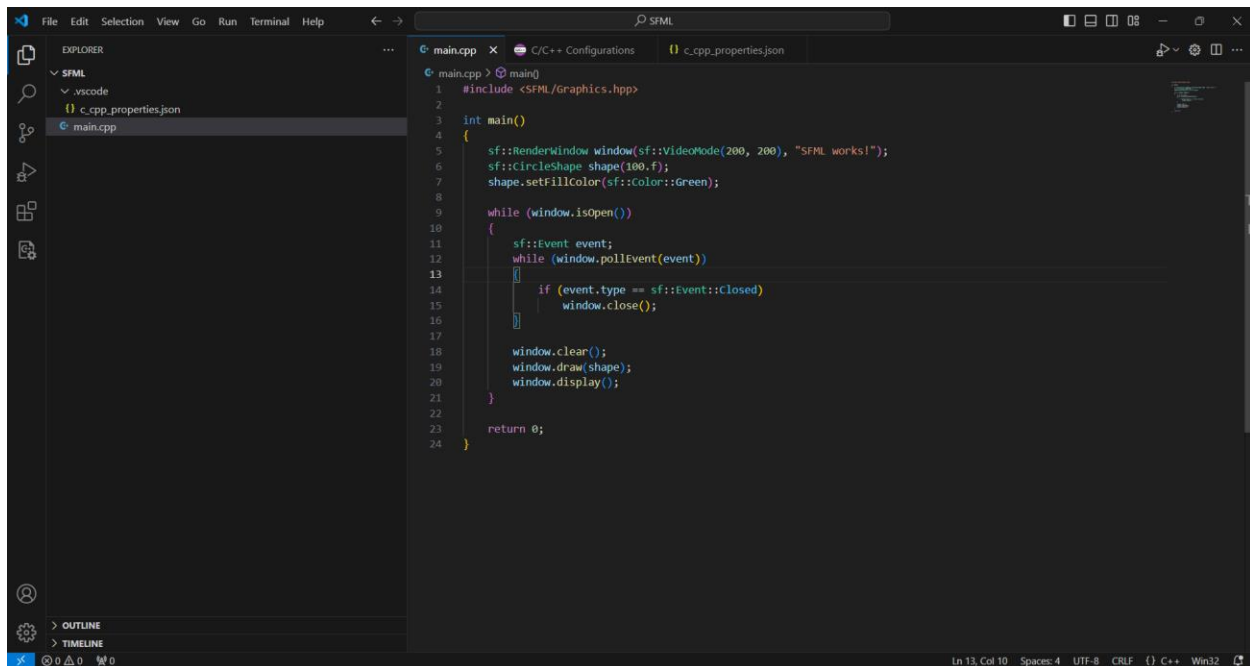
Here inside the `includePath` list, you will add a path to your MinGW **include** folder as follows:



This screenshot shows the same `c_cpp_properties.json` file, but with the `includePath` list updated to include the MinGW include folder path. The `includePath` now contains both the workspace folder path and the MinGW include path.

```
1 {
2   "configurations": [
3     {
4       "name": "Win32",
5       "includePath": [
6         "${workspaceFolder}/*",
7         "C:\\mingw_dev_lib\\include"
8       ],
9       "defines": [
10        "_DEBUG",
11        "UNICODE",
12        "_UNICODE"
13      ],
14      "compilerPath": "C:\\mingw_dev_lib\\mingw\\mingw64\\bin\\gcc.exe",
15      "cStandard": "c17",
16      "cppStandard": "gnu++14",
17      "intelliSenseMode": "windows-gcc-x64"
18    }
19  ],
20   "version": 4
21 }
```

This should eliminate the **include** error:



Now all we need to do is to compile our `main.cpp` file. To do this, we will open up the terminal and run the following command:

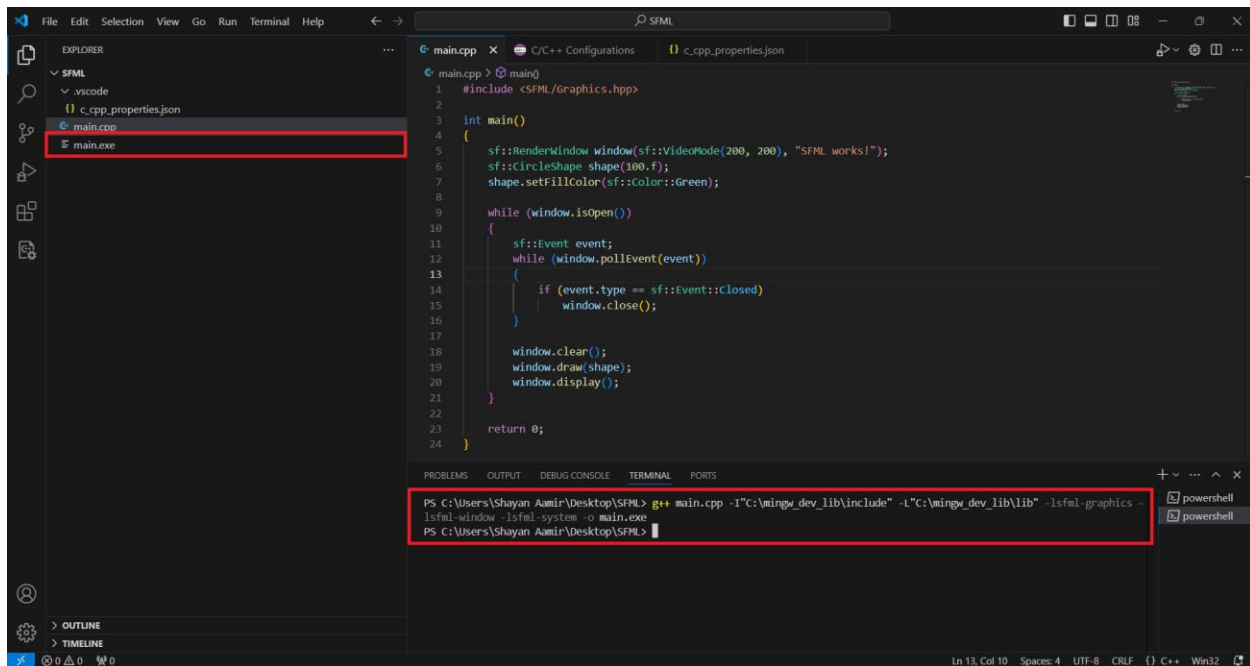
```
g++ main.cpp -I"<Path-To-Your-MinGW-Include-Folder>" -L"<Path-To-Your-MinGW-Lib-Folder>" -lsfml-graphics -lsfml-window -lsfml-system -o main.exe
```

Make sure to replace the tags with your actual path. For example, replacing the tags for my laptops will result in the following command:

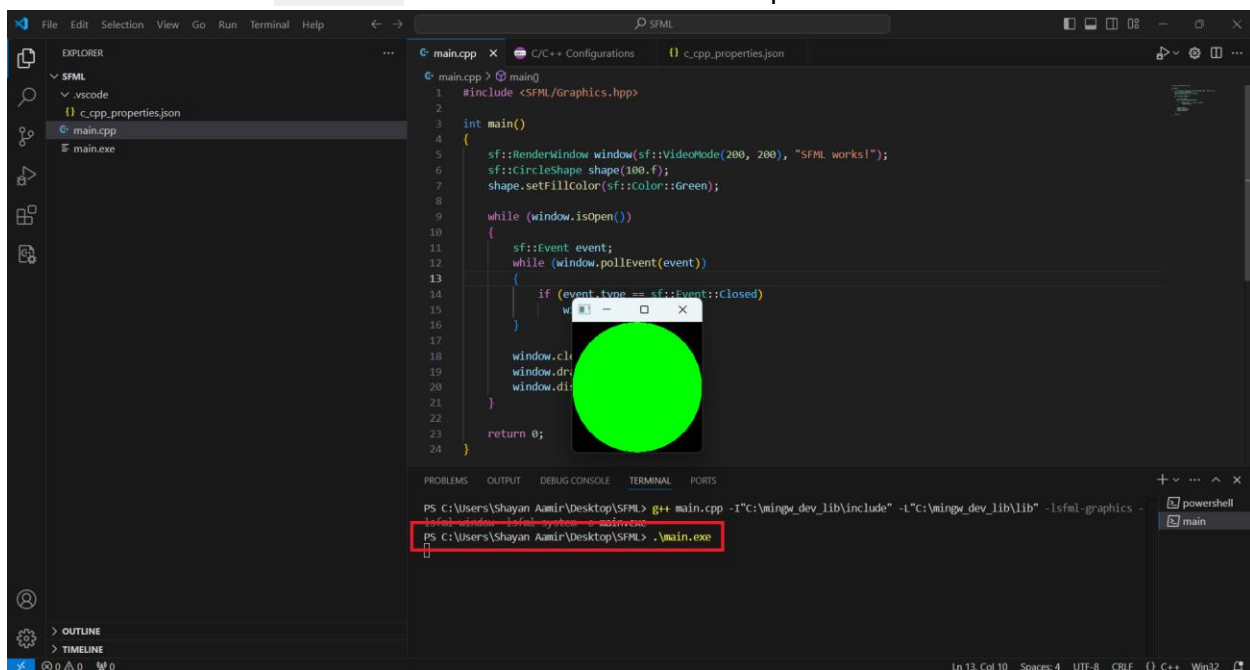
```
g++ main.cpp -I"C:\mingw_dev_lib\include" -L"C:\mingw_dev_lib\lib" -lsfml-graphics -lsfml-window -lsfml-system -o main.exe
```

Optionally, you can modify the `-o` tag to change the name of your output file.

Now open up the terminal, and run the command. This will create a `main.exe` file in your project directory as shown below:



Now we can run the `main.exe` file and here is what the output looks like:

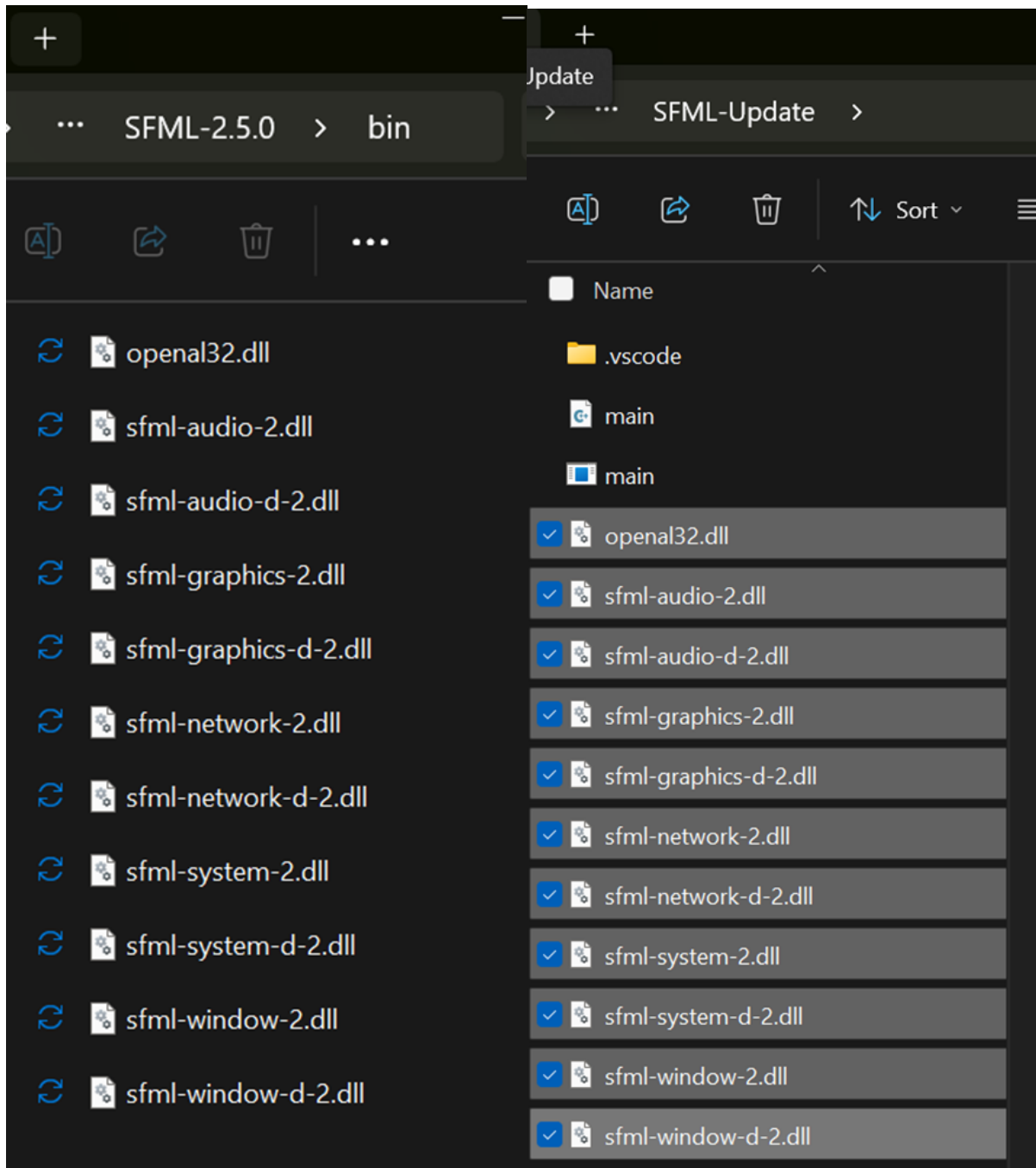


## Alternative Method:

In case your program does not run, (or if you are using Lab PC's) go to the **Files** folder and copy all of the files in the SFML **bin** folder to the folder where your `main.cpp` is located (And other lab files will reside) like as follows:

So copy everything from here

-> to here



Then, all we need to do is to compile our `main.cpp` file. To do this, we will open up the terminal



and run the following command:

```
g++ main.cpp -I"<Path-To-Your-SMFL-Include-Folder>" -L"<Path-To-Your-SMFL-Lib-Folder>" -lsfml-graphics -lsfml-window -lsfml-system -o main.exe
```

Note that this SMFL Include Folder and SMFL Lib Folder is the .zip that you downloaded at the start of the lab!