**Lab 1 – SFML Basic Concepts**

**Journal Due: Friday 17th January**

Learning Outcomes:

* Load a texture from an image file and create a sprite.
* Perform basic sprite operations (translate, rotate, set origin).
* Reorganise a project's directory structure to separate source and header files and update the relevant compiler settings.
* Explain the role of a filter in Visual Studio.
* Add an external library (thor) to an SFML project.
* Use the thor Resources module to manage textures.

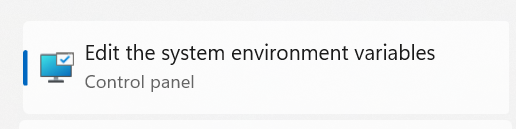
**Please sign up for your github repo first:**

[**click this link**](https://classroom.github.com/a/pg-ysWRg)

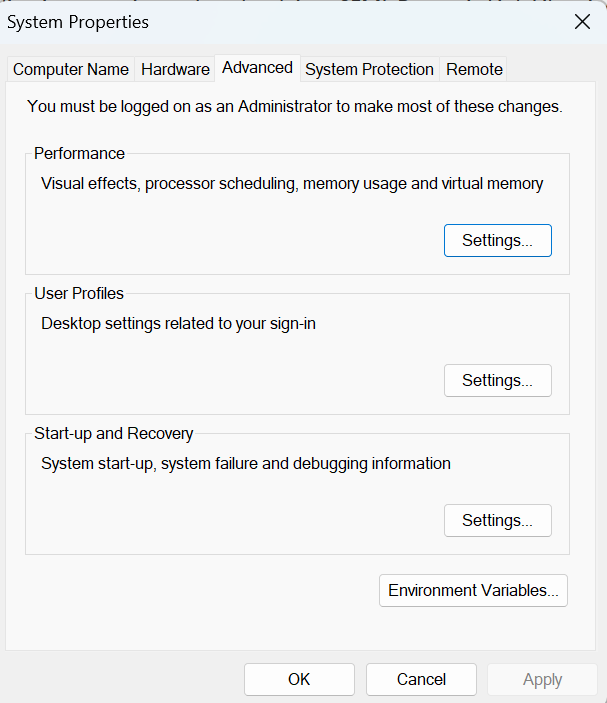
**SFML installation and configuration**

SFML 2.6 is already installed on the system image in the various computing labs so you can skip ahead to **Project Setup** on page 5. If you are installing SFML on a laptop or home PC, you can read the guide below.

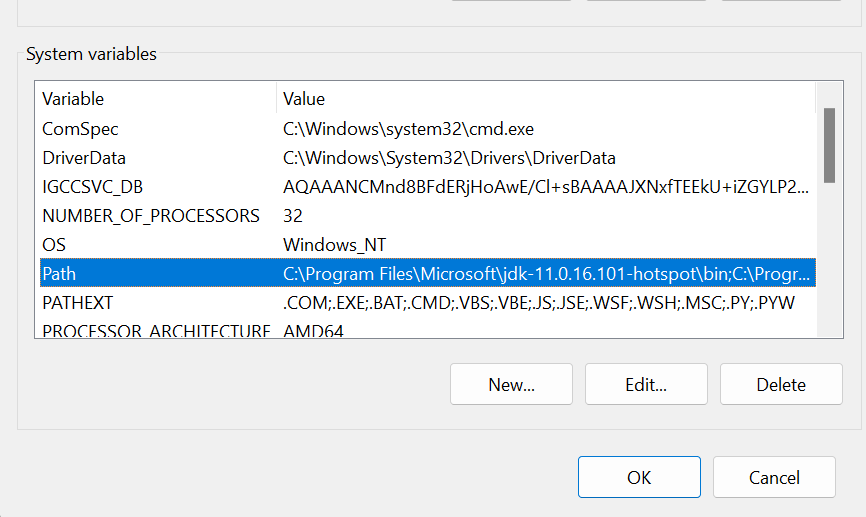
* SFML 2.6 is already installed on the image in the various computing labs. If you are setting SFML up elsewhere, download the Visual C++ 17 (2022) - 32-bit version from [here](https://www.sfml-dev.org/download/sfml/2.6.1/).
* Extract the zip file to the root of your C: drive (C:\). I suggest renaming it to **SFML-2.6.1**
* When your SFML application runs, it needs to hook into SFML Dynamic Link Libraries (DLL's). A DLL is a library that contains code and data that can be used by more than one program at the same time. The SFML DLLs can be found in the **C:\SFML-2.6.1\bin** folder. When our SFML program runs, it won't find these DLLs because our program is located in a different directory. Windows has a special environmental variable called PATH. This contains a list of directories that are searched by Windows whenever you run a program or command. A simple solution is to add the C:\SFML-2.6.1\bin directory to the PATH so that our program will be able to locate the necessary DLLs.
* In the Windows search box, type PATH and then click on the link 'Edit the System Environment Variables'.



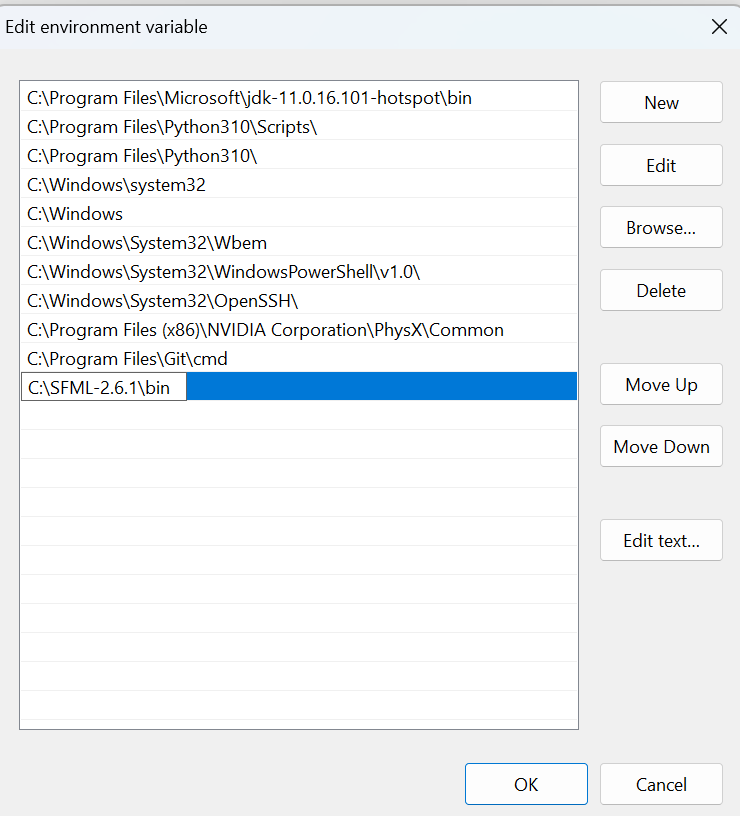
* In the System Properties dialog box, Click the **Environment Variables** button near the bottom right.



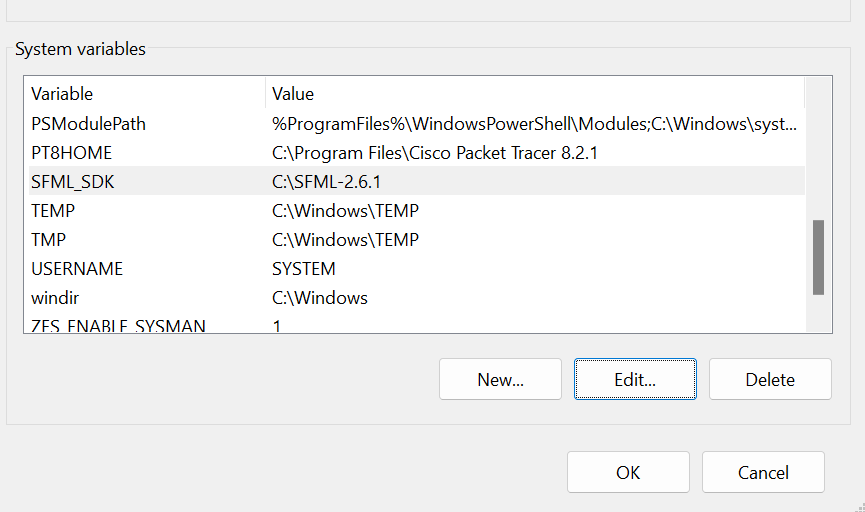
* In the System Variables section, choose Path, then click the Edit button.



* In the next dialog box, click New, then add the path C:\SFML-2.6.1\bin then click OK to return to the Environmental Variables dialog box.



* Finally, we will add a new environmental variable called SFML\_SDK. Under the System Variables section, click New, then add SFML\_SDK as the variable name and C:\SFML-2.6.1 as the variable value. We will use this environmental variable inside Visual Studio so when we build our SFML application, it will be able to find the necessary SFML header and library files.



**Project Setup**

**Download this as a zip file:**

[**https://github.com/ross-palmer/SFML\_Playground**](https://github.com/ross-palmer/SFML_Playground)

**1. Reorganising the project structure.**

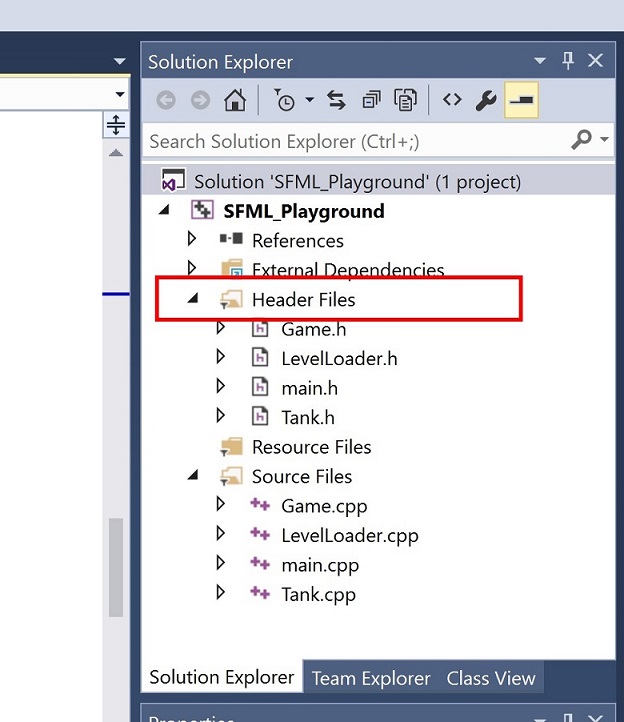
The first step with any project is to create a proper structure to organise header files, source files and resources files.

**1.1** After opening the project in Visual Studio, remove all the header and source files (right click on each, then Remove but **don't** delete). Open Windows Explorer, make an include folder and a src folder.

**1.2** In Windows explorer, move **all** the header files to the include folder and **all** the source (.cpp) files to the src folder.

**1.3** Make another folder called resources. Inside resources create two further folders called levelData and images. We will use these folders later.

**1.4** Returning to Visual Studio, note that VS provides filters that allow us manage all the source files in our project. By default, VS provides filters called Header files and Source files. Here's an example of the Header files filter:



Right click on the Header files filter and choose 'Add->Existing Item'. Browse to your include folder and the header files.

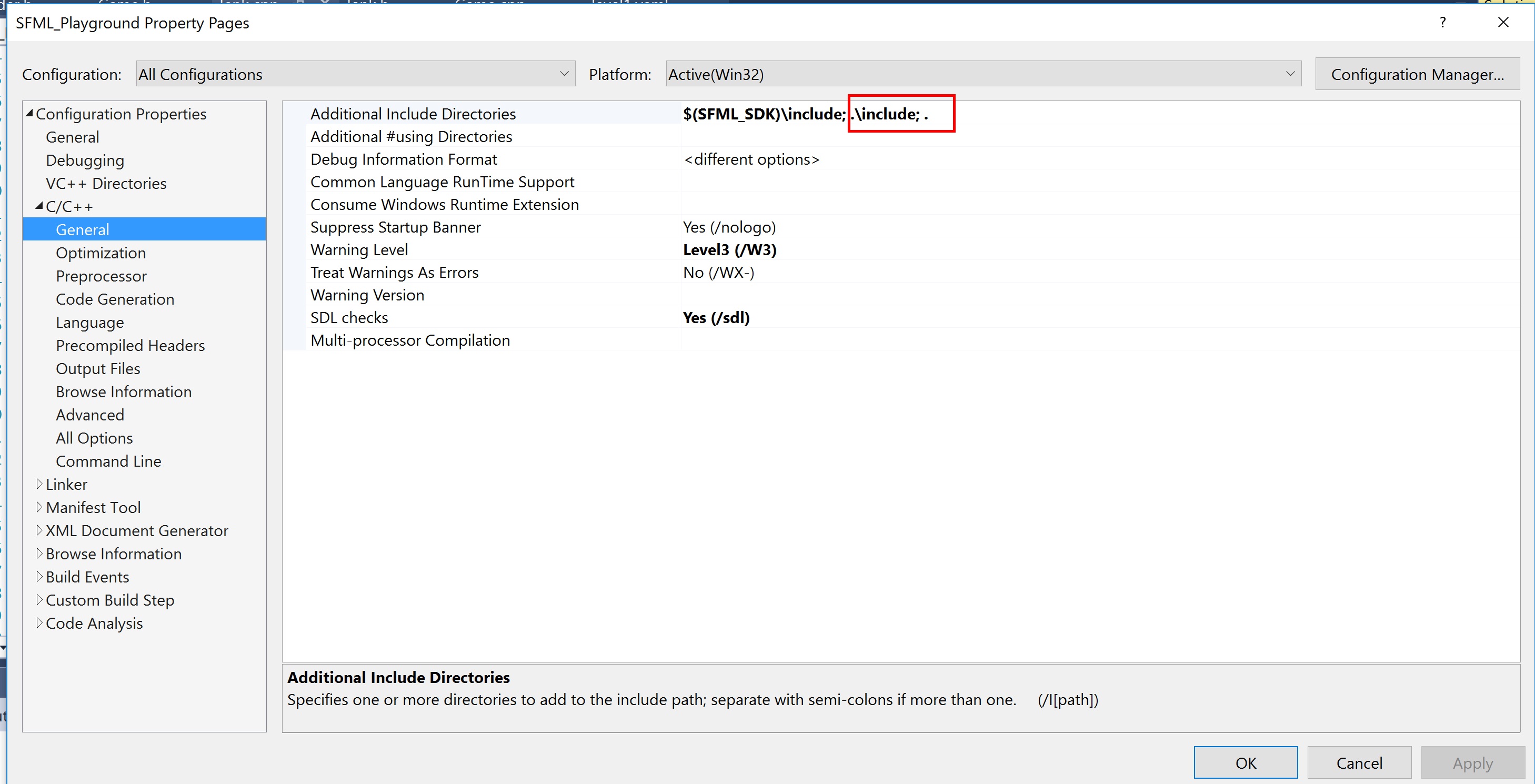
Right click on the Source files filter and choose 'Add->Existing Item'. Browse to your src folder and add the source files.

For large projects, it is useful to create your own custom filters. For example, inside the Header files filter you could create further filters for different aspects of your project, e.g. Sound, AI, Physics etc as appropriate. This structure would again be mirrored inside the Source files filter.

**1.5** Now we need to tell the compiler where to locate our header files (remember, we have moved them in the file system into a folder called include). Right click on your project name, then choose **Project Properties**, then:

C/C++->General->Additional Include Directories

Add the entry **;.\include\;.** as indicated in the following screenshot under **Additional Include Directories**



The semi-colon (;) is used to separate directory paths. The .\include means search under your project folder for a directory called include, while a . on it's own, means search the current (project) folder.

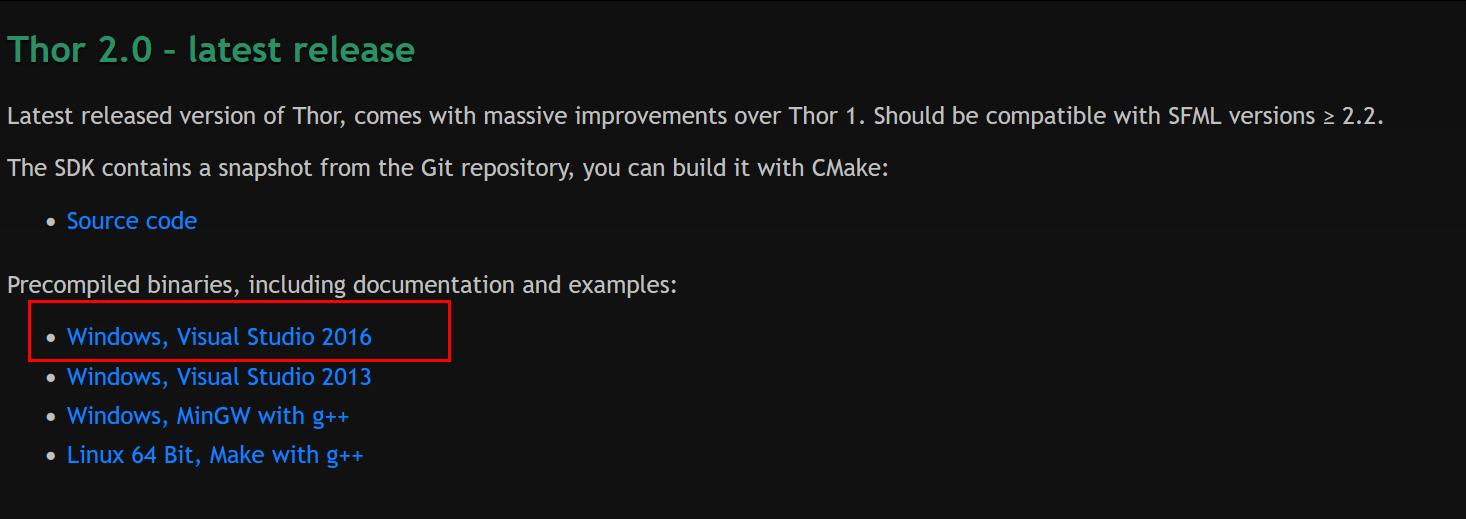
**2. Add thor functionality**

Thor is an external library that brings additional functionality to SFML. In this lab, we will use the Resources module of the Thor library to manage the textures in our project.

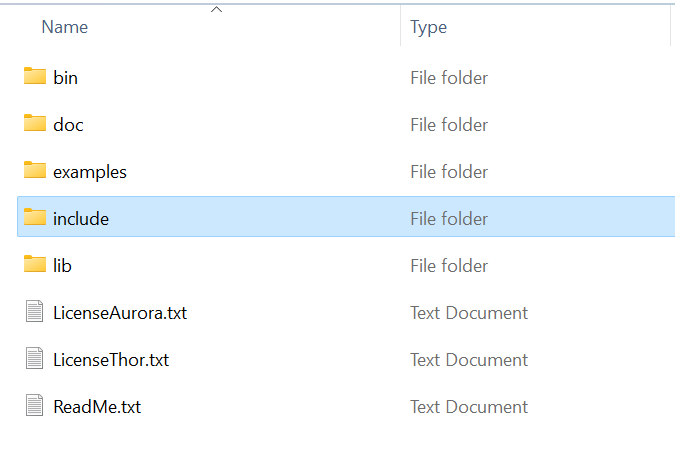
Download the thor library from here:

<https://bromeon.ch/libraries/thor/download/index.html>

Get the **Windows, Visual Studio 2016** precompiled binaries:



Inside the archive you will find an **include** folder



Look inside the include folder, in here copy the Thor and Aurora folders to your own include folder.

Next, create a **lib** folder in your project directory. Do not use the lib folder from what you downloaded in the previous step - I have rebuilt the thor library for the latest version of SFML.

Download the contents of this folder from onedrive which contains the Debug library (.lib) and dll (.dll) files for Thor:

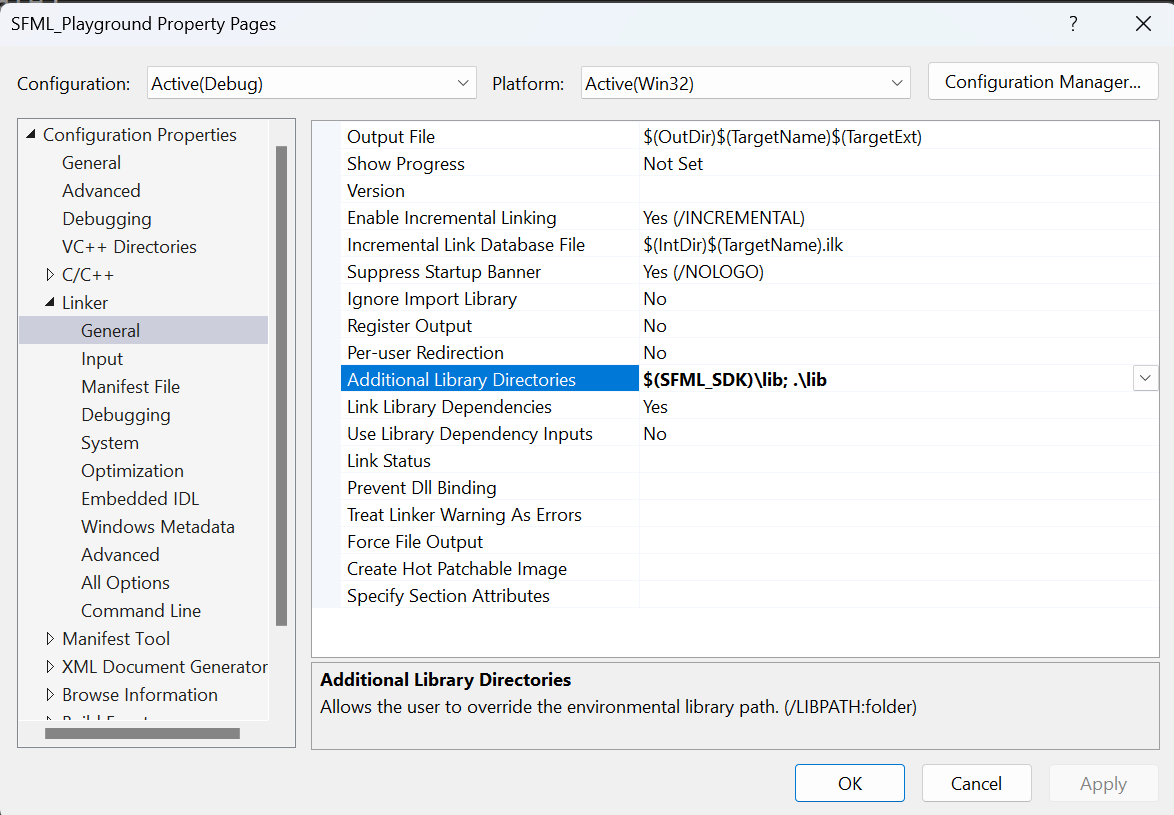
[Debug](https://setuo365-my.sharepoint.com/:f:/g/personal/ross_palmer_setu_ie/EtE743IocW5GiJGQLUlFPKkBItMoYy4VuPzYL5boOgQI3w?e=MDuhrj)

Now copy the thor library file thor-d.lib into your lib folder. Finally, copy the Thor runtime library file thor-d.dll into your Debug folder.

Open **main.cpp** and add the following lines to tell the linker to use the thor library files.

// Add this line under the #ifdef \_DEBUG directive (for Debug builds)  
#pragma comment(lib,"thor-d.lib")   
  
// Add this line under the #else directive (for Release builds)  
#pragma comment(lib,"thor.lib")

Next, instruct the Linker to search your newly created lib folder – in Project Properties, go to Linker->Additional Library Directories and add the entry **.\lib** as shown below:



We will test out the thor functionality shortly.

**3. Creating image assets**

The next step is to create some image assets. In the same onedrive folder as this document, you will find five image files for a tank themed game. These files are taken from here:

<https://free-game-assets.itch.io/free-2d-tank-game-assets>

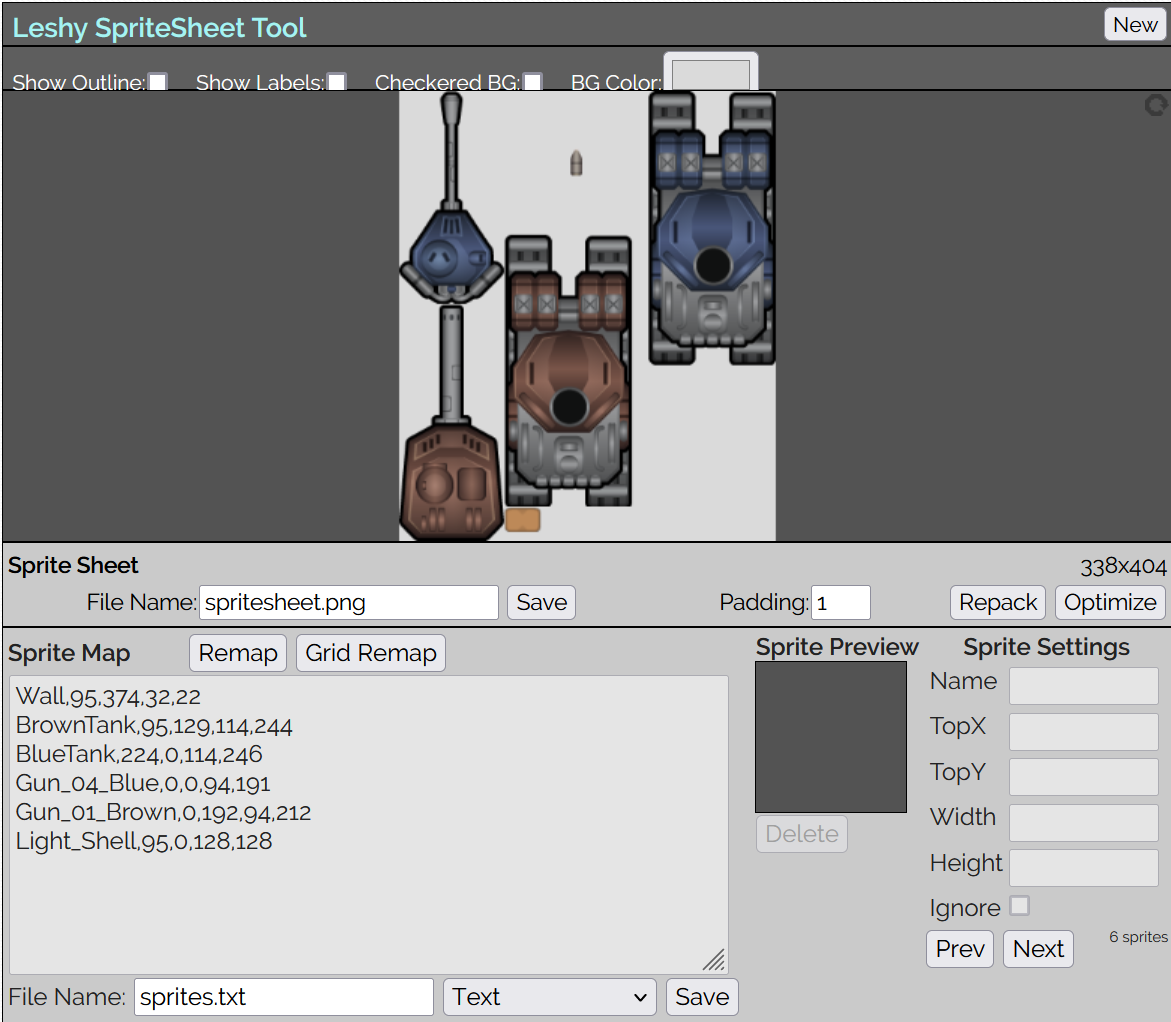
Instead of dealing with five separate textures, we want to create a single texture atlas. A texture atlas is essentially a single texture file that contains multiple textures. Using a single atlas instead of managing many individual texture files provides two clear benefits:

1. Reduces the amount of disk I/O overhead (I/O operations are slow) as you are loading fewer texture files.
2. Having fewer images files decreases the amount of switches between different textures at runtime. Texture switching is a relatively slow operation on the GPU, using texture atlases may result in performance increases.

There are various tools we can use to create a texture atlas, one simple way is to use the Leshy SpriteSheet Tool available here:

<https://www.leshylabs.com/apps/sstool/>

When you have opened this page, follow the instructions by dropping the images files onto the specified location. The result is something like this:



Save the file spritesheet.png and also the file named sprites.txt

The .png file is the texture atlas, while the .txt file tells you where each texture image is located within the atlas. For example, the BrownTank is located at 95,129 (x,y) and has a width of 114 pixels and a height of 244 pixels. We will need this information for each texture later on.

NB! Before you go any further, there is a problem with the orientation of these assets. Currently, the tank images "face" up the y-axis. By default, in SFML, 0° corresponds to "right", i.e. facing or oriented along the positive x-axis. If we leave the artwork as is, it would mean the tank would face along the x-axis when rotated 90° when it should in fact be facing straight down the positive y-axis. How can this be fixed? The solution is simple, open each image file (except for Wall.png) and manually rotate it 90° clockwise and save the file.

Now use the rotated images to recreate the sprite sheet using the Leshy SpriteSheet Tool. Copy the resulting files spritesheet.png and sprites.txt to your Projects **resources/images** folder.

**4. Creating and drawing a sprite.**

To create and draw a sprite, we first need to load a texture for that sprite. This is where we use the thor Resources module to manage our texture atlas.

**4.1** Open Game.h, at the top under the other #include statements, add the line:

#include <Thor/Resources.hpp>

Next, in the section where the member variables are declared, add the following declaration:

thor::ResourceHolder<sf::Texture, std::string> m\_holder;

This declares an instance of the thor ResourceHolder. It is a template class that uses an SFML texture (sf::Texture) and a C++ string.

Underneath this, add a member variable of type Sprite:

sf::Sprite m\_tankSprite;

Open Game.cpp, inside the Game::init() function, add the following after the code that loads the font:

m\_holder.acquire("tankAtlas", thor::Resources::fromFile<sf::Texture>("resources/images/spritesheet.png"));

sf::Texture& texture = m\_holder["tankAtlas"];

The first line loads the texture atlas from the file system into the resource holder and assigns an identity “tankAtlas” to the texture. The second line uses the assigned identity to look up the texture and assign it to a local reference of type sf::Texture.

This variable must be a reference because we do not want to copy the texture!

**4.2** Add a further two lines:

m\_tankSprite.setTexture(texture);

m\_tankSprite.setPosition(100, 100);

The line m\_tankSprite.setTexture(texture) causes the sprite to hold a pointer to the texture, so the texture **must exist** so long as the sprite is using it. This is why the thor ResourceHolder is a good idea, it provides a single object to manage all the textures we may need in a project. If the ResourceHolder object is in memory, the textures will stay loaded. We then position the sprite at 100,100

Now draw the sprite inside the Game::render() function. If you are not sure how to do this, consult the help page here:

<https://www.sfml-dev.org/tutorials/2.6/graphics-sprite.php>

**4.3** Build the application. Because we have made a Debug build, a Debug folder is created inside your Project folder (you get a Release folder created for Release builds, but more on that later). Inside Debug, you will find your SFML executable. When you run this program, it will hook into the SFML DLLs that are in C:\SFML-2.6.1\bin because we add this directory to the PATH. If you ever want to distribute your SFML application (so it runs on another PC), you will need to copy the required DLLs from C:\SFML-2.6.1\bin to your Debug or Release folder as appropriate.

Note the render function clears the screen to black and displays the contents of whatever draw() calls we have made to the SFML window on-screen. Because we have made a debug build, you should see the FPS counters showing updates per second (UPS) and draws per second (DPS), the rest of the screen will stay black.

What do you observe when the sprite is drawn?

(Put your answer as a comment in your code)

**Make it so only the blue tank is drawn with the turret positioned on top. Hint: you need to search for the setTextureRect function in the web link above.**

**4.4** In Game::init(), add a line at the end of the function to release the texture resource:

m\_holder.release("tankAtlas");

Build and execute the project. What happens now? (Put your answer as a comment in your code). Now comment out the line you added above.

**5. Setting the sprite origin.**

By default the origin of the sprite is set to the top left corner. Set the origin of the sprite to it's centre point. Add a comment after your code to explain where the sprite is now drawn. Also explain why part of the sprite is missing.

**6. Rotating a sprite.**

Draw the sprite anywhere so it is fully visible (i.e. translate the sprite somewhere). Rotate the sprite by 90 degrees. Add a comment to say in which direction the tank rotated (clockwise or anti-clockwise) and why you think it is oriented this way. Remember, by default, in SFML, 0° corresponds to "right", i.e. along the positive x-axis.

All SFML classes (sprites, text, shapes) inherit from the same interface for transformations which is called sf::Transformable.

<https://www.sfml-dev.org/documentation/2.6.0/classsf_1_1Transformable.php>

Using your sprite, experiment with the Transformable functions:

rotate()

setRotation()

Write comments in your code to explain the differences between them.