Project 3:

The Minesweeper

COP3503C: Programming Fundamentals II
University of Florida

Laura Cruz Castro, Chaitanya Nulu, Joshua Fox

Welcome to the third project! The minesweeper will be the first game you will develop completely from scratch. Take this as a fun learning opportunity and give yourself enough time to think about this project.

Contents

1 Overview

For this project, you will create a version of the classic game, Minesweeper. Your final version will look something like this:

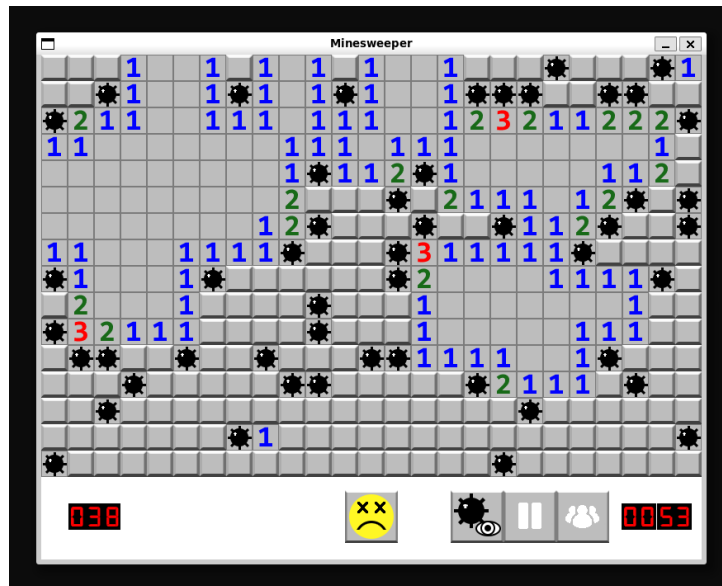Figure 1.1: Game Window.

If you've never played the game before, you can find several playable versions of this online:

- http://minesweeperonline.com/

- http://www.freeminesweeper.org/minecore.html

To create this project, you are going to use SFML (the Simple and Fast Multimedia Library) to do the work of drawing images to the screen and getting mouse input, while you will be responsible for everything else.

2 SFML

2.1 About

The library you will use in this project is SFML—Simple Fast Multimedia Library. The first thing you need to do is compile an application using this. Building an application using an external library can be a difficult thing, but it's something that you typically only have to do once at the start of a project, and then you're good to go until that project is complete. To get started with SFML, visit this site:

- https://www.sfml-dev.org/download/sfml/2.5.1/

2.2 Installation

You want to download the appropriate version for the IDE/compiler that you are using. You can refer to this link https://www.sfml-dev.org/download/sfml/2.5.1/ and pick the IDE which is most comfortable for you.

You have the option to use Visual Studio (for Windows), CLion (for Mac/Windows), or any other code editor to develop your program and run it via the terminal. It's up to you to decide which IDE or code editor you feel the most comfortable using. The link provided above has pre-compiled versions of the library that will function seamlessly with the suitable compiler.

Installing and compiling your first "Hello World" program can be a bit tricky, especially if you've never done it before. There are guides here: https://www.sfml-dev.org/tutorials/2.5/, and certainly elsewhere online, but the best source is often from the developers themselves. We recommend the following tutorials for some of the IDEs mentioned above:

- Visual Studio 2022

  https://www.youtube.com/watch?v=1_EUJjy5IBo

- CLion

  https://www.youtube.com/watch?v=BILAqQ-ZFDA

- Clion with windows

  https://dev.to/danielmelendezz/how-to-get-smfl-to-work-on-windows-using-clion-2bef

When installing the library, make sure to use the appropriate version and to have all your developer tools up to date.

2.3 SFML Basics

There are many guides and tutorials on how to use SFML, but the key features that you will be utilizing for this project are concepts like sf::Font, sf::Text, sf::Sprite, sf::Texture, etc. All these concepts will be discussed in the later sections wherever they are being used.

2.4 SFML Tutorials

This document will not replicate the wealth of information already out there about this library. The primary list of examples/tutorials can be found here: https://www.sfml-dev.org/tutorials/2.5/

From that page, a few, in particular, you will find useful for this project:

- https://www.sfml-dev.org/tutorials/2.5/window-window.php

- https://www.sfml-dev.org/tutorials/2.5/window-events.php

- https://www.sfml-dev.org/tutorials/2.5/graphics-sprite.php

- https://www.sfml-dev.org/tutorials/2.5/graphics-text.php

Anything beyond that will not be applicable for this project (networking, audio, etc will NOT be used). Everything you see on the screen (each tile, number, button, etc.) will be created and drawn the same way: load a texture, create one or more sprites from that texture, and then draw them to the screen, or if it's a text, we load a font once, and then create a Text object, set its size and display it at a certain position.

3 Description

3.1 Windows

In this project, you will be working on three different windows: the welcome window, the game window, and the leaderboard window. A window in SFML is a graphical window that displays graphics and allows user interaction in a computer program.

We use the **sf::RenderWindow** to create a new object of the type render window. It is a subclass of **sf::Window** that provides a 2D rendering context. It allows you to draw graphics and handle user input events, such as mouse clicks and keyboard presses, using the SFML graphics module. It is a fundamental class for creating 2D games and interactive applications using SFML.

All the windows you will be creating will have fixed dimensions and cannot be resized. Here's how you can create a simple sf::RenderWindow of size 800x600 pixels with the title "SFML Window" and the resize option disabled. Read more about sf::RenderWindow here.

```
1  sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window", sf::Style::Close);
```

For all three windows, you will be setting their width and height according to the guidelines. Each of the three values below will be configured by a text document, **"config.cfg"**. There are three lines representing the number of columns, the number of rows, and the number of mines, respectively. You can find more information about the config file in section 3.3.5.

3.2 Welcome Window

3.2.1 What is this window about?

This will be the first thing the player should see when you run the game. It will prompt the user for their name. The user will type their name using the keyboard as input.

Figure 3.1: Example of the welcome window.

3.2.2 What features/behaviors does this window have?

Here are the features of this window:

- The user cannot enter more than 10 characters while typing their name.

- The user has to see the characters as they type and pressing backspace should remove the last character.

- You should also display a cursor indicator '|' as the user types in their name. This cursor is just to indicate where the text is being typed. You don't have to implement any blinking effect or use arrow keys to navigate between the text as this is not an actual cursor. But you can implement these features if you want to.

- The characters should only contain alphabets. So when a user types a number/special character, you shouldn't read it.

- No matter how the user enters their name when storing it, you have to capitalize the first letter, while the other letters have to be in lowercase. Here are some examples:

  - alex -> Alex
  - bRUCe -> Bruce
  - Carl -> Carl

- When the user presses the "Enter" key, you will close this window, and open the game window.

- It is also important to understand that if the user manually closes this window, the program must stop executing and the game window should not load.

- You will be using certain SFML libraries for displaying text which will be discussed later in the next section.

3.2.3 Related Concepts

Here are some of the SFML concepts you will be using for this window:

SFML Class	Description
sf::Font	You will need to load a font before you can display text.
sf::Text	This will be used to draw text onto the screen. It will allow you to change a lot of things like the text position, size, style (bold/underlined/italic), etc.
sf::FloatRect	It is a class in SFML that represents a rectangular area with floating-point coordinates, defined by its top-left corner and its width and height in pixels. You can use this to store the local bounds of a sf::Text, which is helpful in setting the text to the center.
sf::Color	It is a simple class that represents an RGBA color. It allows you to define colors for drawing shapes, sprites, and text in your SFML application. You will mainly use this to set the text and background color.
sf::Vector2f	It is a class in SFML that represents a 2D vector with floating-point coordinates.
sf::Keyboard	It allows you to check if a particular key is currently pressed or released, and take actions accordingly.
std::tolower() and std::toupper()	Converts a character to its corresponding lower case or upper case, if applicable.
std::isalpha()	It is a C++ function from the <cctype> header that checks if a character is an alphabet (uppercase or lowercase).
sf::Event	It is an SFML data structure that represents user or system events, like mouse clicks or key presses, and enables you to handle them in your program. You will be using events like "sf::Event::Closed" and "sf::Event::TextEntered".

3.2.4 Additional Notes

The Welcome window will have the same width and height as the Game Window. The specific guidelines regarding the dimensions can be found in the section 3.3.5. The font you are going to use is **"files/font.ttf"**. Here are some details regarding the components you will be displaying in this window as shown in Figure 3.1:

- **The Welcome Text:** This is the welcome message that you can see on the top i.e., "WELCOME TO MINESWEEPER!". Here are its properties:

  - **Style:** Bold and Underlined
  - **Color:** White (Default)
  - **Size:** 24 px
  - **Position:** width/2.0f, height/2.0f - 150

- **The Input Prompt Text:** This is the prompt message that you can see on the second line i.e., "Enter your name:". Here are its properties:

  - **Style:** Bold
  - **Color:** White (Default)
  - **Size:** 20 px
  - **Position:** width/2.0f, height/2.0f - 75

- **The Input Text:** This is the text that is typed by the user i.e., their name. Here are its properties:

  - **Style:** Bold
  - **Color:** Yellow
  - **Size:** 18 px

      – **Position:** width/2.0f, height/2.0f - 45

- **Note:** All text needs to be aligned to the center.

- **Background Color:** Blue

3.3   Game Window

3.3.1   What is this window about?

This is where you would actually implement the minesweeper game as shown in Figure 3.4. The rules of the game are as follows:

- There exists a board that has a grid of tiles. A tile could contain a mine, or not. The player clicks on a tile, and it gets revealed. The goal of the game is to reveal all the tiles that are not mines while avoiding the tiles that are.

- When a tile is revealed:
    - If it's a mine, the game ends.
    - If it's not a mine, it shows the number of mines adjacent to that tile (anywhere from 0 to 8, with 0 just showing as an empty space).
    - If a tile has no adjacent mines, all non-mine tiles adjacent to it are also revealed. The player uses the numbers as clues to figure out where other mines may be located.

- The player can also put flags on tiles to indicate potential mines.

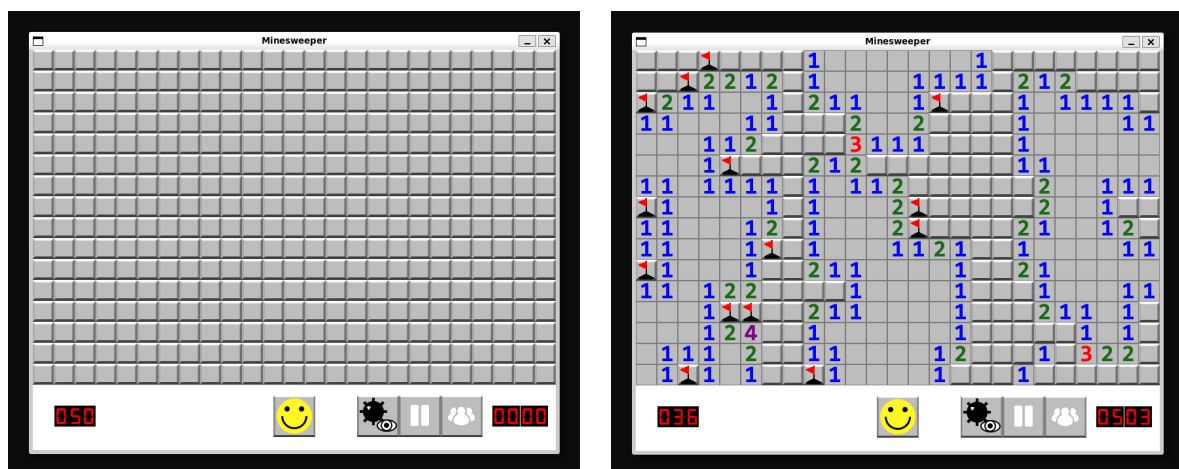- When all of the non-mine tiles have been revealed, the player wins!

Figure 3.2: Example of the Game Window; Left: The window when the game just starts; Right: The window during the gameplay

3.3.2   Images

All the images shown below are available inside the "**files/images**" folder.
Game Images

✹	mine.png	The star of the game (although if you play properly, you'll never see one!)
▢	tile_hidden.png	What all tiles look like before they are clicked/revealed
▢	tile_revealed.png	A revealed tile with no adjacent tiles
1	number_#.png	Tiles with the numbers 1-8 on them (replace # with the appropriate number. Used for tiles that have 1-8 adjacent mines

UI Images

☺	face_happy.png	Click this button to reset the map. New mines, everything hidden, it's like you restart the program.)
😎	face_win.png	Victory!
😵	face_lose.png	The opposite of Victory! (It's cool, no smiley faces were harmed during the creation of this project)
0123456789-	digits.png	Used for the digits on the "remaining mines" display and the "timer". You can use this one texture for all the numbers, and change the "Texture Rect" of a sprite to draw a different portion of the image. The size of each digit (and the size of the "Texture Rect" you should use) is 21 x 32 pixels, and each digit would be offset by 21 (the width) times the digit you wanted. You can do this by using sf::IntRect. See https://www.sfml-dev.org/tutorials/2.5/graphics-sprite.php for more information.
⚙	debug.png	Used to toggle debug mode
⏸	pause.png	Used to pause the game.
▶	play.png	Used to play the game when it's paused.
👥	leaderboard.png	Used to launch the leaderboard window.

3.3.3 What features/behaviors does this window have?

There are many things to do in this window. But we can break them down into the following components:

- **Tiles:** By default all tiles use the "**tile_hidden.png**" sprite. A tile can be left-clicked to reveal it, or right-clicked to place a flag/remove a flag that's already placed. The **"flag.png"** will be placed on top of the the "**tile_hidden.png**" sprite, while the "**tile_revealed.png**" sprite is displayed when the tile is left clicked.
  Depending on the following conditions, you should display another sprite on top of the revealed tile sprite:

  - If the tile has a mine, draw the "**mine.png**" sprite. Then you reveal all the tiles with mines and end the game. If there's a flag on a tile, then you need to draw the mine sprite on top of it i.e., the flag sprite stays in the background while the mine sprite is drawn in the foreground.
  - For tiles that have no mines, check all its neighbors for the number of mines around them and display this value using the "**number_#.png**" sprite.
  - When the revealed tile has no adjacent mines, you will just display a revealed tile and all non-mine tiles surrounding it will be revealed. This happens recursively.
  - When all non-mine tiles are revealed, the player wins. So you MUST place flags on top of all tiles with mines.

- **Happy Face Button:** This button lets you restart a new board with everything reset and mines randomized at any point in time. It also has two additional functions: the sprite changes to "**face_win.png**" when the player wins, and to "**face_lose.png**" when the player loses.

- **Counter:** Helps to track the number of mines that are on the board. Every time the player places a flag, the counter goes down by one. Every time they remove a flag, the counter goes up by one. The counter CAN go negative!

- **Debug Button:** Clicking this button will toggle the visibility of the mines on the board. Use this to help you test/debug various features of the game. Having to play the game properly each time you want to test something is very time-consuming. Creating these developer shortcuts helps speed up the development process. The debug button shouldn't work when the game ends (victory/loss).

- **Pause/Play Button:** Pauses or Plays the game. All functionality except the Happy Face button and Leaderboard button should be disabled. All tiles (regardless of debug mode status) should display **"tile_revealed.png"** sprite. The sprites for this button switch from **"pause.png"** to

"play.png" and vice-versa depending on whether the game is paused or not. The timer will also stop when the pause button is pressed, and it will start again when the play button is pressed. The tiles must go back to their previous states once play is pressed. The pause/play button won't work when the game ends (victory/loss).

- **Leaderboard Button:** Will pause the game, and all tiles (regardless of debug mode status) should display **"tile_revealed.png"** sprite and display the leaderboard window. During this stage, the game window shouldn't be intractable and the timer should stop. When the leaderboard window is closed, all tiles go back to their previous states, and the timer resumes. Note that if the pause button is pressed before the leaderboard button, then closing the leaderboard button should not resume the game.

- **Timer:** Displays the amount of time that has passed since the beginning of the game.
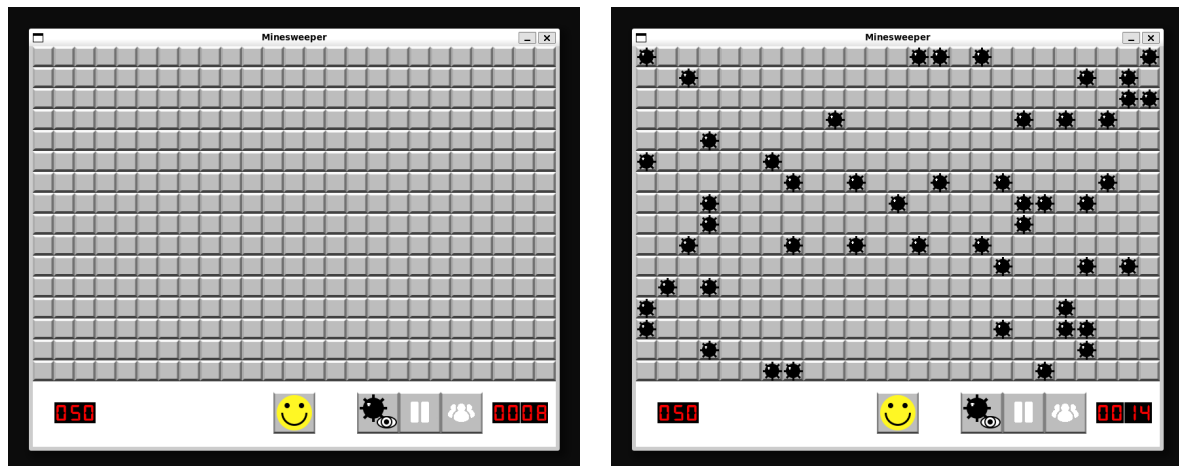

Figure 3.3: Example of the Debug button functionality; Left: Before pressing debug button; Right: After pressing debug button;
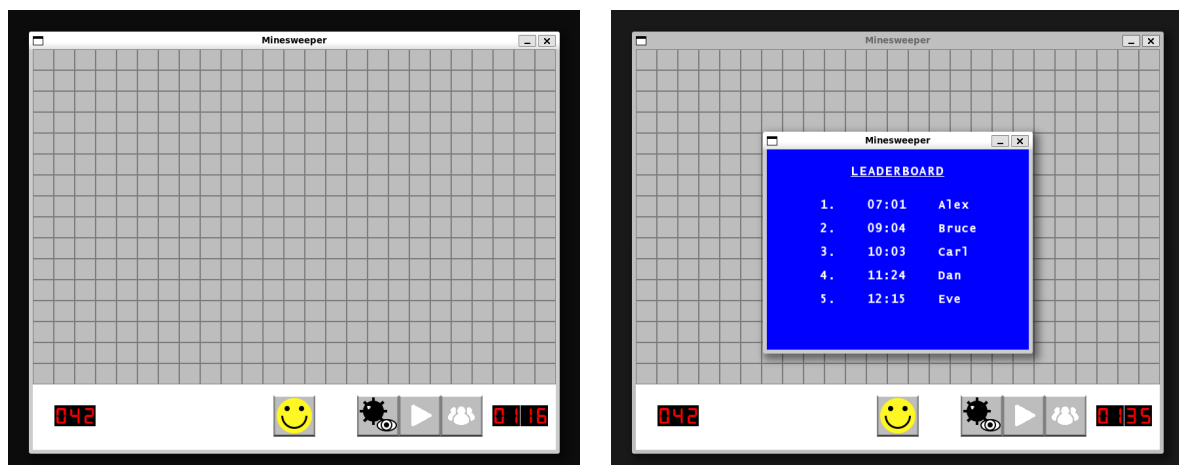

Figure 3.4: Example of the Paused Game Window; Left: The window when the pause button is pressed; Right: The window when the leaderboard button is pressed; (Even when debug mode is active, all tiles should display **"tile_revealed.png"** sprite)
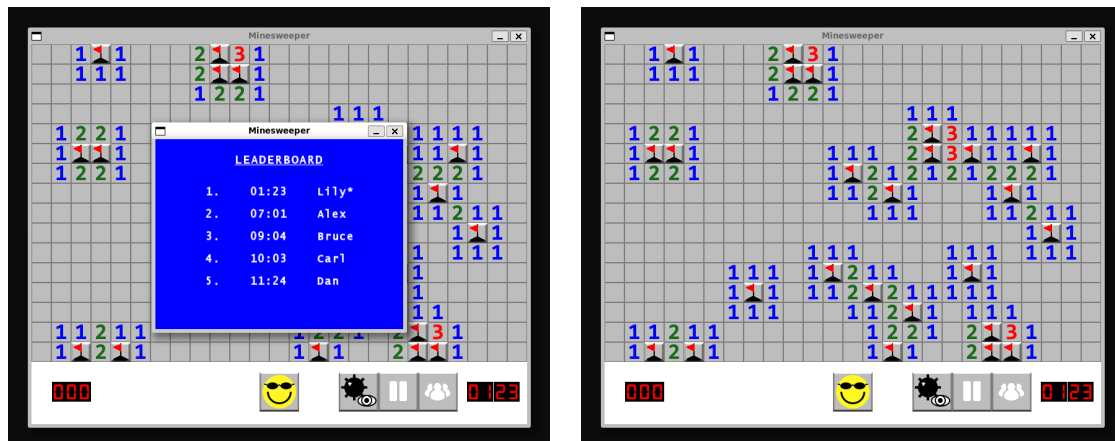
9

Figure 3.5: Example of the Victory Condition; Left: The leaderboard window pops up when the game is won; Right: The game board after the victory condition is met; Note: This is a 25x16 board and has 25 mines!

3.3.4 Related Concepts

Here are some of the SFML concepts you will be using for this window:

SFML Class	Description
std::chronos	It is a part of the C++ standard library that provides support for date and time manipulation, including clocks, timers, time points, and durations. You will use this for implementing the timer functionality. • **chronos::high_resolution_clock::now()** is a C++ function that returns the current time point of the high-resolution clock. • **chronos::duration_cast** is a C++ function that converts a duration value from one unit to another, using the rules of the specified duration type. It allows you to convert between different time units, such as seconds, milliseconds, and microseconds, with precision and type safety.
sf::Texture	It is an object that represents an image loaded into memory and allows you to create sprites that display the image. It is a fundamental class for working with graphics in SFML.
sf::Sprite	It is a 2D object that displays a texture or a portion of a texture on the screen. It allows you to manipulate the position, scale, and rotation of the texture and is commonly used to display game characters and objects.
sf::IntRect	It is a simple rectangle class that represents a rectangular area of pixels, defined by its top-left corner and its width and height in pixels. It is often used to specify a portion of a texture that should be displayed by a sprite.
sf::Vector2f	It is a class in SFML that represents a 2D vector with floating-point coordinates.
sf::Mouse	It is a static class that provides access to the current state of the mouse, such as its position, buttons, and wheel and is used to handle mouse input.
sf::Event	It is an SFML data structure that represents user or system events, like mouse clicks or key presses, and enables you to handle them in your program. You will be using events like "sf::Event::Closed" and "sf::Event::MouseButtonPressed".

3.3.5 Addtional Notes

- **Reading the config file:** First, you will be working on creating the board. For that, you will be reading a text document "**files/board_config.cfg**". There are three lines representing the number of columns, the number of rows, and the number of mines, respectively. You read these values and create a board of the given dimensions. These values will not be changed during a game.

Also, the dimensions will never be less than 22 columns x 16 rows, so that all of the menu buttons will be able to fit with the face-centered and the buttons space correctly. The number of mines will be a valid number less than or equal to the number of tiles. We will provide an example, but we will test with different values.

Window Size	Width = number of columns * 32
	Height = (number of rows * 32) + 100
	32 is the number of pixels on the side of a tile. The extra 100 pixels are for the space for the buttons on the bottom.
Mine Count	Equal to the number of mines set by the "board_config.cfg" text file.
Tile Count	Equal to the number of columns * the number of rows.

Feel free to modify the config file to test out things. For example, setting the "mine count" value to 1 in the config file will make it really easy to test.

For example, if the text document is:

```
25
16
50
```

Then the width is 25 * 32 = 800, the height is (16 * 32) + 100 = 600, mine count is 50, and the tile count is 25 * 16 = 400.

Note: The Welcome window in the previous section will have the same width and height as the Game Window.

- **Buttons:** A button is just an image that you can click on to make something happen. A more complex UI system would use an event/messaging system, but on a basic level you just need a sf::Sprite to represent the button, and every time the player clicks on something, you need to check if that mouse click occurred inside the boundaries of the sf::Sprite you're using as the button.
  If you're drawing a sprite somewhere, you know its position (it's 0, 0 by default, or whatever you set it to). You can get the width/height of the sprite through its textureRect, and then it's just a matter of checking if the mouse position is inside that box.

- **Mouse Interactions:** The application can do "nothing" until the user clicks their mouse. Typically games and many other applications clear/redraw the screen regularly (often dozens of times a second). Until the player clicks the mouse somewhere in the window, the program will appear to just sit there, idle. You won't see anything moving other than the timer.
  Once the player has clicked, however (you can check for this in the event loop), you then need to do some checks about that click.

- **Adjacent Mines and Tiles:** To calculate the number of nearby mines, as well as when revealing tiles, each tile should store a list of neighboring tiles. A tile could have UP TO 8 neighbors. An easy way to do this is with pointers. Since the number is a variable, a dynamically sized container would be perfect for this. You could also use a fixed-length array since no tile will ever have more than 8 neighbors.

  - **vector<Tile*> adjacent tiles;** // Store each tile near us, the size() of each vector will vary.

  - **Tile* neighbors[8];** // Always 8 pointers, some of which might be nullptr.

- **UI Locations:** Here is a table with all positions for the buttons and other UI components. Use these values as they have been tested to work on all board dimensions.

Component	Position
Happy Face Button	((width / 2.0) * 32) - 32, 32 *(height+0.5f)
Debug Button	(width * 32) - 304, 32 * (height+0.5f)
Pause/Play Button	(width * 32) - 240, 32 * (height+0.5f)
Leaderboard Button	(width * 32) - 176, 32 * (height+0.5f)
Counter	Start drawing the digits from (33, 32 * (height+0.5f)+16). In case the counter is negative, draw the '-' sprite in **digits.png** at location (12, 32 * (height+0.5f)+16). Note that all digits are 21px wide, so the next digit sprite should be drawn after 21px.
Timer	It consists of two parts: minutes and seconds. – Minutes has two digits. Start drawing them from ((width*32)-97, 32*(height+0.5f)+16). – Seconds also has two digits. Start drawing them from ((width*32)-54, 32*(height+0.5f)+16). Note that all digits are 21px wide, so the next digit sprite should be drawn after 21px.

3.4 Leaderboard Window

3.4.1 What is this window about?

The Leaderboard window will appear whenever the user presses the "Leaderboard" button, or when the user wins the game. You will be loading "**files/leaderboard.txt**" text file. This file will be already filled with some players and their corresponding times. Here is the default leaderboard file you are provided with:

leaderboard.txt

```
07:01,Alex
09:04,Bruce
10:03,Carl
11:24,Dan
12:15,Eve
```

Note: There is no space after the comma.

3.4.2 What features/behaviors does this window have?

Here are the features of this window:

- You need to read the "leaderboard.txt" file and display its contents according to the format shown in Figure 3.6. Do note that you will be also printing the ranks on the left-hand side as shown in the figure.

- If the player clicked on the "Leaderboard" button, all you have to do is display the contents of the file.

- But in the case where the player wins a game, you need to check if their time is better than any other person on this list. And if that's the case, then you will need to insert the User details (name and time) into the Leaderboard. Indicate this change by displaying the new entry with a '*' at the end as shown in Figure 3.6

- You will also need to update the "leaderboard.txt" file according to the format shown above (TIME,NAME) so that when you open the leaderboard next time, you will see this result properly.

Figure 3.6: Example of the Leaderboard Window.
Left: The window when the player clicks the leaderboard button
Right: The window when the player overtakes someone, after winning the game.

3.4.3 Related Concepts

Here are some of the SFML concepts you will be using for this window:

Concept	Description
sf::Font	You will need to load a font before you can display text.
sf::Text	This will be used to draw text onto the screen. It will allow you to change a lot of things like the text position, size, style (bold/underlined/italic), etc.
sf::Color	It is a simple class that represents an RGBA color. It allows you to define colors for drawing shapes, sprites, and text in your SFML application. You will mainly use this to set the text and background color.
sf::FloatRect	It is a class in SFML that represents a rectangular area with floating-point coordinates, defined by its top-left corner and its width and height in pixels. You can use this to store the local bounds of a sf::Text, which is helpful in setting the text to the center.
sf::Vector2f	It is a class in SFML that represents a 2D vector with floating-point coordinates.
sf::Event	It is an SFML data structure that represents user or system events, like mouse clicks or key presses, and enables you to handle them in your program. You will be using events like "sf::Event::Closed".
File I/O	In C++, file I/O operations are performed using the standard library classes and functions, such as std::fstream and std::ifstream for reading from files, and std::ofstream for writing to files.
StringStream	Allows you to read and write formatted data to and from strings.
SubString	substr() is a member function of the std::string class that allows you to extract a substring from a string, defined by a starting index and a length or by a starting index and an ending index.

3.4.4 Additional Notes

Here are some other details:

- The Leaderboard window's dimensions will be half of the width and height values of the Game Window i.e., width/2 and height/2. Read more details at section 3.3.5

Width	16 * number of columns
Height	(number of rows * 16) + 50

- You will use the same font for the title and contents for this window as the one used in the Welcome Window, i.e., **"files/font.ttf"**.
  You will display the title "LEADERBOARD" with the following specifications:

  - **Style:** Bold and Underlined
  - **Color:** White (Default)
  - **Size:** 20px
  - **Position:** width/2.0f, height/2.0f - 120 (Refers to the width and height of the Leaderboard window)

  You will also display the contents of the "leaderboard.txt" file with the following specifications:

  - **Style:** Bold
  - **Color:** White (Default)
  - **Size:** 18px
  - **Position:** width/2.0f, height/2.0f + 120
  - **Note:** It will be easier for you if you store all the rows in a string separated by two line spaces (\n\n). Then you can use this string to create a sf::Text and set its position as specified above.

  Note: All text needs to be aligned to the center.
  Background Color: Blue

- When the Leaderboard window is open, it just pops up above the actual Game Window. Unless the Leaderboard window is closed, you cannot interact with the Game Window.

- Whenever a player beats another player in the Leaderboard, you have to insert the player's information above that particular row. As we will only be displaying the top 5 players, we can discard the extra player's information when writing the data into the file.
  Note: Using vectors will save you a lot of time and effort.

- Unlike in other games, the ranking system here is independent of the number of rows, columns, and mines. So no matter how you modify the config file, you can update the same "leaderboard.txt" file.

4   Additional information for implementation

4.1   Storing Resources

While a program is running, it needs RESOURCES to get the job done—things like icons, textures, sound files, etc. Many of the resources need to be stored for long-term use, as they may be called upon time and time again... but you don't always know when they'll be needed when you compile your code. A great storage container for assets that you want to reference by name is the map<>. Storing something that you can access by its name with a container["NameOfAsset"] is vastly preferable to that of dealing with arrays—was "GameOver.png" stored in an array[25], or array[26]? You may find it helpful to create a single storage container for all of the sf::Texture objects and then pass that around to any class which might need those files.

4.2   Global Variables

In this project, you should not, under any circumstances, attempt to use **sf::Texture** and **sf::Font objects** in global space. Globals in general should be avoided, but if you try to load a sf::Texture/sf::Font in global space it may be initialized before other parts of SFML are initialized, causing it to crash before main() even starts. To make matters worse, this problem might not occur on your machine, but it could be on someone else's (i.e. the person grading your project).

4.3 Paths

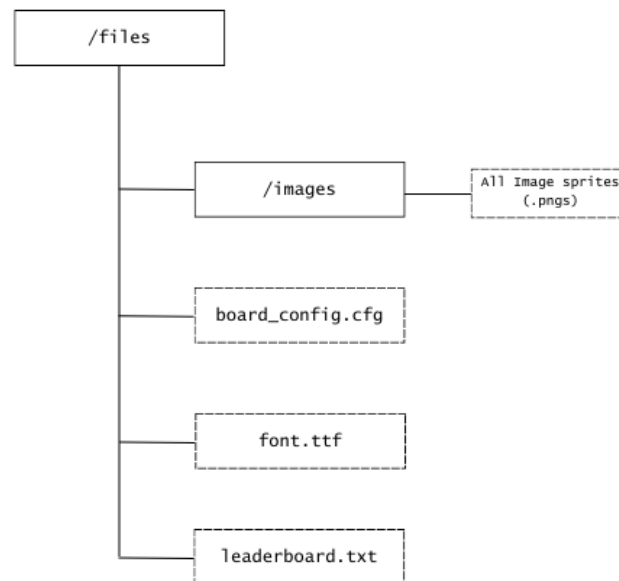Here's the "files" folder structure you will be provided with:

Figure 4.1: "files" Folder Structure

In this project, any operations involving files (loading textures and boards) should use RELATIVE paths, NOT absolute paths. Your code should be based on the folder structure shown above. When you load a texture, it should be from the images folder, like "files/images/mine.png", and when you load the config file, it should be from "files/board_config.cfg"

Be sure to use FORWARD slashes in your paths for compatibility.

For example:

- "files\\images\\mine.png" – this will work on Windows, but not elsewhere. Don't use this.

- "files/images/mine.png" – this will work everywhere. Do use this!

4.4 Code Structure

With larger programs, you can accomplish the goal in any number of ways. There isn't a single way to write this that works better than all others. From the outside perspective (i.e., that of a player), your application needs to DO various things. HOW you choose to accomplish those things is up to you. If you want to write a single, gigantic main() function, you are free to do so—that approach is not recommended, however. A few suggestions:

- A class to represent the board. This represents the core data object in the game.

- A class for tiles. The board is made up of a whole lot of these things. Each one of these can be a mine, have a flag, some number of adjacent tiles/mines, etc.

- You can have a separate Texture Manager class for loading textures and getting them at any point in time.

- Many programs (games or otherwise) do the same things over and over again while the application is running. The ability to easily (in code) reset everything is critical. Think about what sorts of helper functions you might want to make that happen. Things like:

  - Restarting the board.

15

– Setting or clearing tiles of flags.

– Setting or clearing mines (singly or in large quantities).

– Recalculating the number of adjacent mines, etc.

- Separate classes for the implementation of the Welcome window and the Leaderboard window.

4.5   Sample Makefile

To execute your program via terminal, you can use the following Makefile:

```
1  default:
2  g++ src/*.cpp -o project3.out -lsfml-graphics -lsfml-window -lsfml-system
```

Once you have the Makefile in your project directory, navigate to the directory in your terminal and use the appropriate Make command based on your environment:

- For Unix-based systems (such as Linux or macOS), use the "make" command.

- For Windows systems with Microsoft Visual Studio installed, use the "nmake" command.

- For Windows systems with MinGW installed, use the "mingw32-make" command.

After running the appropriate Make command, an executable file named "project3.out" will be generated. You can then run this file to start the game. Note that this Makefile assumes that all your source files are located inside the "src" folder.

5   Milestones

5.1   Milestone 1: Due April 6 (5 points / 150 points)

For this milestone, you must (1) fill out the first questionnaire and (2) show your peer mentor that you have to install SFML successfully on your own machine. For the questionnaire: there are no right or wrong answers in the questionnaire; express your thinking. This assignment is meant to help you succeed in future projects and organize your time in this project. The link will be sent to your email, as you have a unique link, on April 3 at 9:00 am. You will have until April 9 at 11:59 pm to complete the questionnaire. For the SFML installation: The details of how and when this check will be done will be discussed directly by your peer mentor, but the grading is as follows. If your library is installed and you have provided evidence to your peer mentor of the installation, this will give you 5 points. The evidence will be shown in the following window.
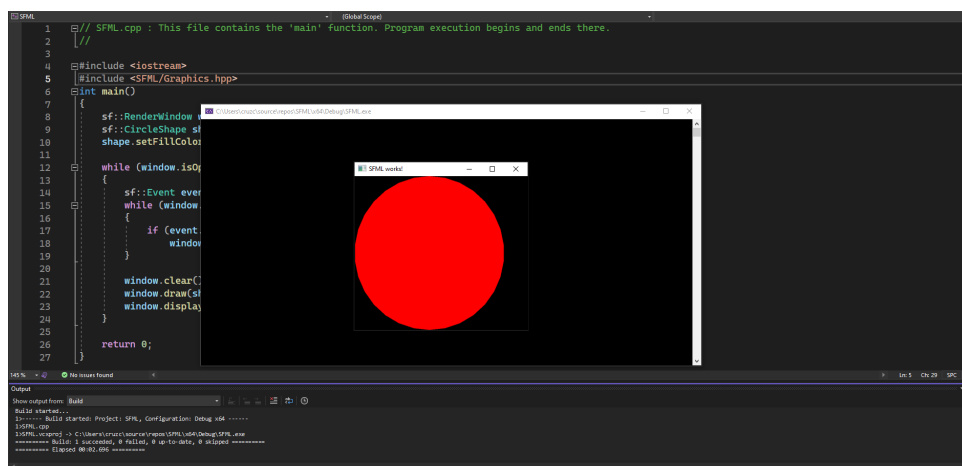

Figure 5.1: Milestone 1

In order for you to generate the window shown in 5.1 you will need to copy and paste the following code into your main.cpp:

```cpp
#include <iostream>
#include <SFML/Graphics.hpp>
int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Red);

    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;
}
```

! Notice that this is not extra credit. If you miss this milestone, your will lose 5 points of your final project grade. In addition, for the questionnaire, you must comply with a minimum of 80 words per question. If this questionnaire is not completed or does not comply with the minimum requirements, you will not receive your milestone points even if you complete the rest of the milestone.

5.2 Milestone 2: Due April 13 (15 points / 150 points)

(1) Complete the second questionnaire. The link will be sent to your email, as you have a unique link, on April 10 at 9:00 am. You will have until April 16 at 11:59 pm to complete the questionnaire. (2) show your peer mentor that you have already implemented at least one of the following behaviors (one setup and one specific) from the following table:

Summary of features to be implemented (Milestone 2)		
Window	Name	Description
Welcome window	player name	Player can enter the name successfully, which includes: (1) cannot enter more than 10 characters (only alphabet, no numbers/special characters allowed), (2) user can see the characters as they type and pressing backspace should remove the last character (3) User can see the cursor (4) names are converted to the right format. (5) Then, once the user presses the "enter" key, the welcome window closes, and the game window launches. (6) In addition, when the player closes the welcome window, the game window won't open, and if there are no characters entered, the "enter" key should not do anything.
Game window	board	It is possible to see visually all the tiles in the board, and the number of tiles and dimensions of the board change according to the configuration file. At least one tile in the board shows the correct functionality (1) if you click and is a number, the number gets revealed (2) you can set a flag by left-clicking the tile (3) you can left-click a flag to remove the flag
Game window	mines	It is possible to see visually tiles in the board and reveal them. In addition, mines are populated into the board and randomly assigned to tiles which one can see because there are at least two buttons that implement the debug and face buttons (with at least the restart game).
Game window	counter	It is possible to see visually all the tiles in the board, and the number of tiles and dimensions of the board change according to the configuration file. In addition, the number of mines appears on the counter (even if mines are not assigned randomly), and the counter decreases or increases.
Leaderboard window	leaderboard	The leaderboard reads from a "leaderboard.txt" file and displays the content of the file as shown in 3.6. In addition, when closing the leaderboard, the game window appears and shows the timer (the times do not have to work yet at this point)

Table 5.1: Summary of features to be implemented for Milestone 2

Other combinations of features might be considered for this milestone; however, as a general, at least a 10-point and a 15 points item of the Milestone 3 rubric will be required for your to achieve full points in Milestone 2.

! Notice that this is not extra credit. If you miss this milestone, you will lose 10 points of your final project grade. In addition, for the questionnaire, you need to comply with a minimum of 80 words per question. If this questionnaire is not completed or does not comply with the minimum requirements, you will not receive your milestone points even if you complete the rest of the milestone.

5.3 Early Birds: Due April 20

Any student who submits the final folder of the project by April 20 and obtains full marks (more than 90 points) will be able to (a) Resubmit one assignment (quiz, lab, or debugging survey) by April 26 if any of the assignments have less than 80% OR (b) Get 15 points of extra credit towards the final grade (if all assignments (quiz, lab, or debugging survey) have been completed with more than 80%). Note: if you have an assignment that you did not submit, this will count as a 0, and therefore, you will enter the first option. In addition, we remove one quiz at the end of the semester; therefore, if you have only one quiz with less than 80% and no other labs or survey with less than 80%, you are eligible for option B.

If you want to participate in this opportunity, you will need to fill out the following form `https://forms.gle/9MCWQDF54sGytSdGA` on April 21 after uploading your final folder to be graded. If you do not achieve full marks, your peer mentor will inform you about it, and you will get feedback on your submission. You will be able to resubmit by April 26.

! Early birds must upload the Milestone 3 questionnaire. The link will be sent to your email, as you have a unique link, on April 20 at 9:00 am. You will have until April 29 at 11:59 pm to complete the questionnaire. For the questionnaire, you must comply with a minimum of 80 chars per question. If this questionnaire is not completed or does not comply with the minimum requirements, you will not receive your milestone points even if you complete the rest of the milestone.

5.4   Milestone 3: Due April 26 (130 points / 150 points)

For this milestone, you will (1) answer the Milestone three questionnaire in this link. The link will be sent to your email, as you have a unique link, on April 20 at 9:00 am. You will have until April 29 at 11:59 pm to complete the questionnaire. You must comply with a minimum of 80 chars per question for the questionnaire. (2) Submits a zip folder to canvas (you will be directed to gradescope similar to your experience with Project 2). This folder with have all .cpp and .h files. This project is manually graded, and you must ensure that your project complies with all rubric requirements before submitting your folder.

Standard Points		
Item	Points	Description
board_config.cfg file	10	Board changes size and its number of mines based on values set in board_ config.cfg file.
Tile Revealing	15	Clicking on a tile reveals it. If it is a mine, game over. If it has 0 adjacent mines, reveal all neighboring tiles which are not currently revealed, not mines and not flagged, and then each of those neighbors goes through this process as well. Depending on the board layout, a single click could reveal nearly the entire board!
Tile Display	10	Tiles display depending on their state: Unrevealed (the default state), Revealed, and empty (no adjacent mines), Revealed, and near 1-8 mines (showing the appropriate number), Revealed, and showing a mine.
Flags	10	Right-clicking on a hidden tile sets a flag on it. Right-clicking a flagged tile removes the flag. Left-clicking a tile with a flag has no effect. Flagged tiles cannot be revealed in any way (by the player or by a revealing algorithm). The flag must be removed first.
Mines Remaining	10	A counter of how many mines are on the board, minus the number of flags placed. Adding/removing flags from tiles affects this. The remaining flags CAN go negative!
Victory	10	Revealing all non-mine tiles ends the game and marks all remaining tiles (i.e., the mines) with flags. Flagging mines will affect the counter, so the final counter will be a 0 after you win. Smiley face changes to sunglasses version. - No further interactions with the game board are possible (you did it all already!) - The player CAN click the smiley face to start a new game or use check the leaderboard by pressing the leaderboard button. (The debug/pause-play buttons shouldn't do anything at this point... game's over, you know where the mines are!) - If the player wins a game, you need to check if their time is better than any other person on this list. And if that's the case, then you will need to insert the User details (name and time) into the Leaderboard. Indicate this change by displaying the new entry with a '*' at the end, as shown in
Defeat	10	Clicking on a mine ends the game. What should happen: - All tiles with mines are revealed (and displayed on top of any flags you may have placed) - The smiley face changes to the dead face (he's just acting, don't worry!) - No further interactions with the game board are possible. - The player CAN click the dead smiley face to start a new game or check the leaderboard using the leaderboard button. (The pause-play buttons shouldn't do anything at this point... game's over, you know where the mines are!)

Table 5.2: Detailed rubric for milestone 3 part I.

Standard Points		
Random Mine	10	When the game starts and when the board is reset (by clicking the smiley face button), a number of mines equal to the number specified in the .cfg file are randomly placed on the map–no more, no less.
Welcome window	15	Player can enter the name successfully, which includes: (1) cannot enter more than 10 characters (only alphabet, no numbers/special characters allowed), (2) user can see the characters as they type and pressing backspace should remove the last character (3) User can see the cursor (4) names are converted to the right format. (5) Then, once the user presses the "enter" key, the welcome window closes and the game window launches. (6) In addition, when the player closes the welcome window the game window won't open, and if there are no characters entered the "enter" key should not do anything.
Timer and pause button	15	The timer is correctly implemented in the game window, and when a player clicks the pause button, the counter pauses, and all tiles are shown with the tile_revealed.png. When paused, the only buttons that should work are the leaderboard and the face. And all tiles should display "tile_revealed.png" sprites (regardless of debug mode status). When the player clicks the play sprite, the game continues where it was, all tiles revert back to their prior states, and the counter starts where it was before.
leaderboard board and button	15	When clicking the leaderboard button, the leaderboard window appears and shows the current leaders in the board (top 5). Clicking this button pauses the timer. And all tiles should display "tile_revealed.png" sprites (regardless of debug mode status). Closing the leaderboard button will close the leaderboard window and show the game window beneath, along with all tiles returning to their prior states.
Total	130	

Table 5.3: Detailed rubric for milestone 3 part II.

! Notice that this is not extra credit. You will lose 85 points of your final project grade if you miss this milestone. In addition, for the questionnaire, you must comply with a minimum of 80 words per question. If this questionnaire is not completed or does not comply with the minimum requirements, you will not receive your milestone points even if you complete the rest of the milestone.

6   Submission

You will turn in your source code and source code only. No images, no SFML libraries, only the .h and .cpp files you wrote to complete the project. Zip up JUST YOUR SOURCE CODE.

Submits a zip folder to canvas (you will be directed to gradescope similar to your experience with Project 2). This folder with have all .cpp and .h files. This project is manually graded, and you must ensure that your project complies with all rubric requirements before submitting your folder.

! If the name or your folder does not comply with the naming convention, you will have a -10 deduction in your project.

7   Grading

In general, the features you need to implement for this project are listed in the feature summary in the Milestone 3 section 5.2 and 5.3, along with point values. You can get full, half, or no points for each feature.

- Full points – Feature works perfectly, no bugs of any kind

- Half points – Feature has any bugs at all or is partially implemented (if your program is slow, you might get deductions here)

- No points – Feature not implemented at all, or so minimally implemented that no functionality exists (for example: drawing a button to the screen that you can't click on at all doesn't count as partial implementation)

The general breakdown of the Project three grade can be found in 7.1

Standard Points		
Item	Points	Description
Milestone 1	5 points	Successful completion of Milestone 1
Milestone 2	10 points	Successful completion of Milestone 2
Milestone 3	85 points	Successful completion of Milestone 3
Total	100	
Deductions	-10	Using global variables (sf::Texture objects or otherwise).
Deductions	-10	Not using relative paths
Deductions	-10	There exist files in the submitted folder that were not requested

Table 7.1: Rubric for the entire project.