Minesweeper 2.0

Use Case Description

Our project aims to recreate the classic game “Minesweeper” in C++. The game challenges players to uncover all the safe tiles while avoiding hidden mines. If a mine is clicked, the game ends in defeat. To help logically locate the mines, the game shows numbers on safe squares indicating how many adjacent tiles contain mines and players are allowed to flag potential mine locations. The goals of the project are to build a fully functional version of Minesweeper while implementing object-oriented programming concepts using C++.

Minesweeper 2.0 is aimed for those who enjoy solving puzzles in a game-like environment and also students who are interested in learning c++, object oriented programming and game design/development.

User tasks and goals include:

Gameplay: Players will uncover tiles, flag suspected mines, and rely on the numbers revealed in adjacent squares to deduce mine locations.

Winning condition: To win, players must safely reveal all non-mined squares without triggering a mine.

Post game options: After winning or losing, users can restart the game, they can also see their high scores listed.

Responsive features: The game will have a tutorial/how to play section and show win/loss messages and a timer will allow users to track their speed.

Powerup: When clicked, it flags the neighboring mines for the player. It may or may not spawn in a game randomly.

Class List

Cell - Represents the foundational unit of each cell, has attributes such as location, color, is_reveal, is_flagged and has functions such as draw(), reveal(),flag() and on_reveal() which are all virtual functions that can be overridden by derived class

CellMatrix - Represents the board and has attributes such as matrix which is an array of cells, num_cols, num_rows, num_mines and mine_locations which is a vector. It has functions such as display(), set_gameboard(), display_overlay() and game_over().

Game - Represents a game which controls the overall game logic and user interaction. It has attributes like game_window and game_matrix and functions such as run().

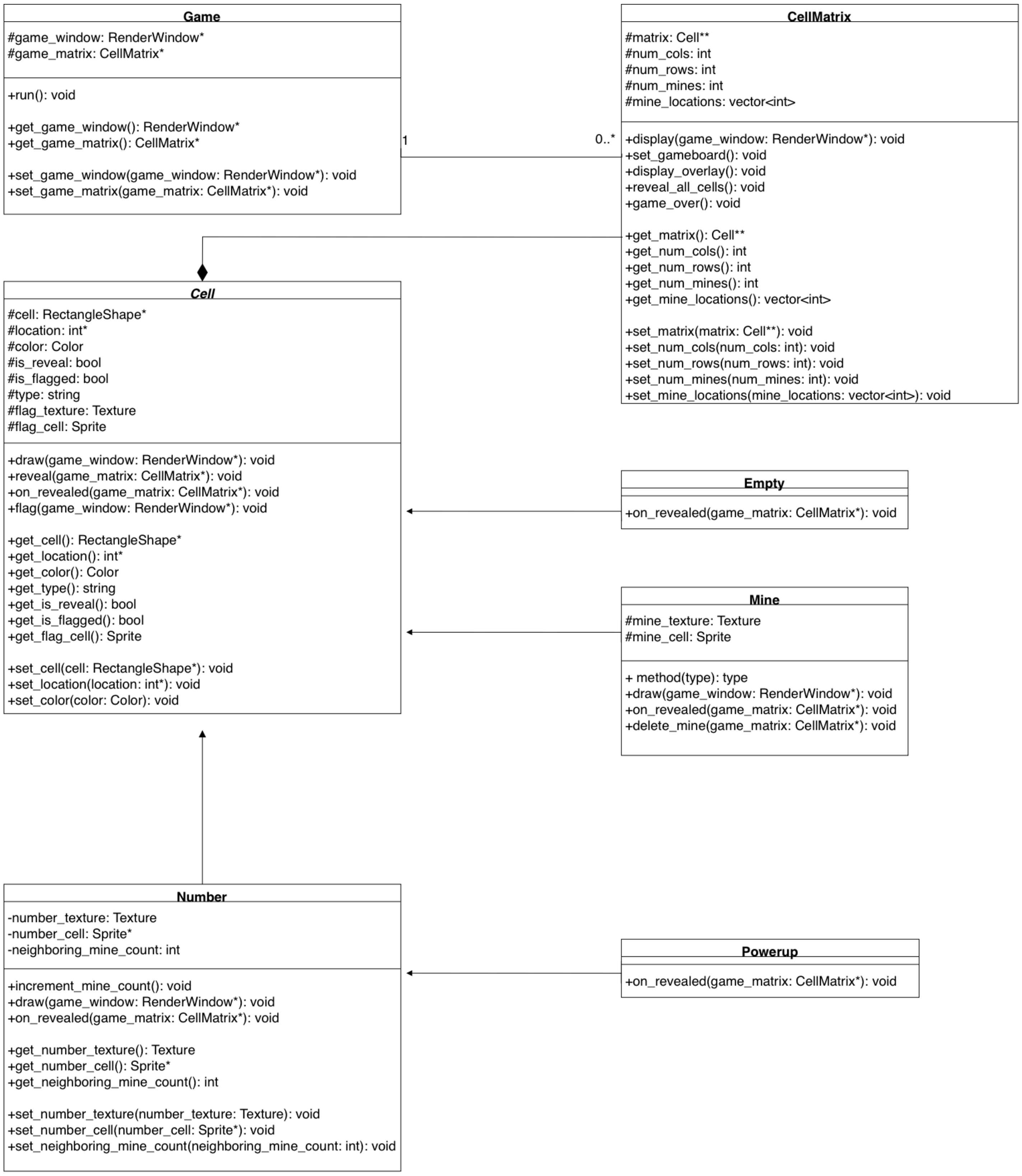
Mine - Represents a mine cell, it's an inheritance of Cell class and contains overridden functions such as on_revealed()

Number - Represents a number cell, it's an inheritance of Cell class and contains functions such as increment_mine_count() and overridden functions such as on_revealed() and draw(). It also has an attribute neighboring_mine_count.

Powerup - Represents a powerup cell, its an inheritance of number and contains an overridden function on_revealed().

Empty - Represents an empty cell, it's an inheritance of Cell class and contains functions such as overridden functions such as on_revealed().

Data and Function Member & Relationships between Class



```
Game
#game_window: RenderWindow*
#game_matrix: CellMatrix*

+run(): void

+get_game_window(): RenderWindow*
+get_game_matrix(): CellMatrix*

+set_game_window(game_window: RenderWindow*): void
+set_game_matrix(game_matrix: CellMatrix*): void
```

```
CellMatrix
#matrix: Cell**
#num_cols: int
#num_rows: int
#num_mines: int
#mine_locations: vector<int>

+display(game_window: RenderWindow*): void
+set_gameboard(): void
+display_overlay(): void
+reveal_all_cells(): void
+game_over(): void

+get_matrix(): Cell**
+get_num_cols(): int
+get_num_rows(): int
+get_num_mines(): int
+get_mine_locations(): vector<int>

+set_matrix(matrix: Cell**): void
+set_num_cols(num_cols: int): void
+set_num_rows(num_rows: int): void
+set_num_mines(num_mines: int): void
+set_mine_locations(mine_locations: vector<int>): void
```

```
Cell
#cell: RectangleShape*
#location: int*
#color: Color
#is_reveal: bool
#is_flagged: bool
#type: string
#flag_texture: Texture
#flag_cell: Sprite

+draw(game_window: RenderWindow*): void
+reveal(game_matrix: CellMatrix*): void
+on_revealed(game_matrix: CellMatrix*): void
+flag(game_window: RenderWindow*): void

+get_cell(): RectangleShape*
+get_location(): int*
+get_color(): Color
+get_type(): string
+get_is_reveal(): bool
+get_is_flagged(): bool
+get_flag_cell(): Sprite

+set_cell(cell: RectangleShape*): void
+set_location(location: int*): void
+set_color(color: Color): void
```

```
Empty

+on_revealed(game_matrix: CellMatrix*): void
```

```
Mine
#mine_texture: Texture
#mine_cell: Sprite

+ method(type): type
+draw(game_window: RenderWindow*): void
+on_revealed(game_matrix: CellMatrix*): void
+delete_mine(game_matrix: CellMatrix*): void
```

```
Number
-number_texture: Texture
-number_cell: Sprite*
-neighboring_mine_count: int

+increment_mine_count(): void
+draw(game_window: RenderWindow*): void
+on_revealed(game_matrix: CellMatrix*): void

+get_number_texture(): Texture
+get_number_cell(): Sprite*
+get_neighboring_mine_count(): int

+set_number_texture(number_texture: Texture): void
+set_number_cell(number_cell: Sprite*): void
+set_neighboring_mine_count(neighboring_mine_count: int): void
```

```
Powerup

+on_revealed(game_matrix: CellMatrix*): void
```

Project Task List and Timeline

Task	Time	Author
Design the class structure and relationships	2 Days	Hiten, Joe & Josheen
setting up Git repository , Makefile and collaboration	1 Day	Joe
Create Game() and Cell()	1 Day	Hiten
Create CellMatrix() and displaying the board on window	1 Day	Hiten & Joe
Finding cursor coordinates and implementing user interaction	2 Days	Joe
Create base class Mine(), Number() and Empty()	1 Days	Hiten & Joe
Setting game board and randomly placing mines	3 Days	Joe
Creating unique functions Empty() and Mine()	3 Days	Hiten
Creating unique function for Number()	1 Day	Joe
Adding sprites and graphics and displaying graphics	3 Days	Joe
Adding flags and its functionality	1 Day	Joe
Adding win conditions and lose condition	1 Day	Josheen
Adding Powerups	2 Days	Josheen
Creating main menu	2 Days	Josheen
Adding timer	1 Day	Josheen
Storing Scores and best times in CSV files	2 Days	Joe
Displaying High Score in main menu	1 Day	Hiten
Testing and Debugging code	3 Days	Josheen

User Interaction Description

The user will interact with the game through a graphical user interface created using the SFML library. The interface will allow users to interact with the gameboard using their mouse to reveal and flag cells. Users can also use the ESC key to end the game and go back to the main menu.

Left click: Users can left mouse click on a cell to reveal it. If the cell contains a number cell,it will reveal the number of mines neighboring it. If the cell is an empty cell it will also reveal the neighboring empty cells until it comes across a number cell. If the user clicks on a mine, then the game ends and reveals all the mines. If user clicks on a power up, then the user gets the respective powerup.

Right click: Users can right click a cell to flag it, which places a flag graphic over the cell. This prevents the users from accidentally revealing a mine cell and it can help users remember where the mines are located. Right clicking a flagged cell causes the cell to be unflagged

The game will show real time visual feedback such as the cells revealing to show what cell is underneath. Number cells , Powerups and mines have textures that reveal a graphic on revealing.

Unit Testing and Debugging Plan

The program will be thoroughly tested using a combination of peer review, well documented commit logs, stress testing and unit testing. These steps will allow us to detect and resolve any potential issues early. Before we push to the main branch, we will conduct peer reviews to ensure that the code works as intended and there is no unexpected behavior. Each commit will have well-written commit logs which will include a clear description of all the changes. This lets us traceback to any errors making it easier to debug.

To ensure that the gameboard is reproducible and hence making the test cases more accurate , we will use a fixed random seed which will be used for mine placement. This allows us to reproduce bugs and verify that the fixes work consistently. We will also stress testing by changing around the constants and seeing how they effect the program which in turn will reveal any unexpected behavior and any edge cases.

We will unit test our code by writing tests for each class and function to ensure that they perform as expected. Each unit test will focus on verifying the correctness of specific functionality, such as the behavior of the Cell, Mine, Number, and CellMatrix classes, along with the reveal() and flag() methods. Additionally, we will test edge cases, such as boundary conditions on the game board, and ensure that random mine placement are consistent by using a fixed seed. Automated tests will be integrated into our workflow, ensuring that any new changes do not break existing functionality.

The project will include a Makefile which will manage dependencies and compilation ensuring that all required components are built correctly. This will also handle different platforms requiring different instructions to compile the game.