Minesweeper Game Specifications Documentation

1. Purpose of the Project

This program is a console-based Minesweeper game inspired by the classic Microsoft design from the '90s. The game allows users to reveal cells, plant flags on suspected mine locations, and track their progress with a visual grid, dynamic smiley icons, and a built-in timer. For the code implementation, the program uses 2D lists to represent board data, mine placements, and cell visibility. The classic board design is modeled on boards and game rules described in more detail here: Minesweeper Online

From an implementation standpoint, the project includes two primary classes: `MinesweeperBoard`, which sets up the game board, mines, flags, and display logic, and `GameLogic`, which manages user input, game flow, win/loss conditions, and timing. The board dynamically updates based on user commands, and leverages breadth-first search (BFS) to recursively reveal adjacent cells when zeros are uncovered. The system supports multiple difficulty levels (beginner, intermediate, expert) and integrates gameplay features like flag tracking, cell validation, and smiley icon feedback for user experience.

This project integrates two key algorithmic concepts from our CSC 255 coursework: *games* and *random number generation*. As a game, Minesweeper provides a rule-based system of user interaction, decision-making, and win/loss conditions. This is modeled through cell reveals, flag placement, and board updates. The game's unpredictability relies on random number generation to determine unique mine positions across the grid. Python's built-in `random` module uses the Mersenne Twister pseudorandom number generator, a widely adopted algorithm known for its high-quality randomness and efficiency (also described in our Weiss course textbook). This ensures each game session has a distinct configuration, reinforcing the gameplay's strategic challenge and emphasizing algorithmic design concepts we learned over the semester.

2. Program Specifications

Expected Inputs

The program does not require any external data file and instead initializes the Minesweeper board in-memory with randomly generated mine placements based on the selected gameplay difficulty. Players choose one of three preset levels (beginner, intermediate, or expert) which determine the number of rows, columns, and mines for the session. (The default gameplay level is beginner.) The game accepts user commands via standard input and interprets each move according to a structured format:

- `R row col` to reveal a cell
- `F row col` to flag or unflag a suspected mine
- `N` to start a new game
- `Q` to quit the game

Internally, the game uses Python's `random` module (which relies on the Mersenne Twister algorithm) to place mines across the board with high-quality pseudorandomness. This ensures that each game session starts with a distinct board configuration, allowing for repeated play with varying challenges.

Expected Outputs

For every turn, the program displays an updated visual representation of the Minesweeper board. Revealed cells display a number corresponding to the count of adjacent mines, while unrevealed cells appear as periods (`.`), and flagged cells display an `F`. The game also provides visual feedback using smiley symbols that change based on game progress: a happy face during play, a cool face on win, and a sad face on loss. When a mine is triggered, the game ends immediately, revealing all mine positions and displaying the total time of gameplay elapsed. If the player successfully reveals all safe cells (i.e., non-mine cells), a congrats message is printed with the elapsed gameplay time in seconds.

Key Functions of the Program

The program is organized into two main classes: `MinesweeperBoard` and `GameLogic`. The `MinesweeperBoard` class handles mine placement, board creation, cell tracking, and display formatting. It also manages flags, timers, and game state transitions. The `GameLogic`class interfaces directly with the player, processes input commands, and handles gameplay actions like revealing cells or planting flags. A breadth-first search (BFS) algorithm is used to efficiently uncover adjacent zero-mine cells, enhancing user experience and mimicking the behavior of classic Minesweeper.

3. Class Design

`MinesweeperBoard` Class

The `MinesweeperBoard` class represents the state and configuration of the Minesweeper grid. It defines gameplay levels, sets up the board dimensions, mines, and display formatting. It also tracks flags, revealed cells, game status, and timer information.

Data Members

Data Member Name	Type	Description
`rows`	int	Number of rows on the board.
`cols`	int	Number of columns on the board.
`num_mines`	int	Total number of mines on the board.
`board`	list	Holds the value of each cell (number or mine).
`mine_positions`	list	Stores which cells contain mines.
`revealed`	list	Tracks whether each cell has been revealed.
`flags`	list	Tracks cells flagged by the player.
`flag_count`	int	Current number of flags placed.
`start_time`	float	Start timestamp of the gameplay session.
`end_time`	float	End timestamp of the gameplay session.
`game_over`	bool	`True` if the game has ended.
`smiley`	string	Status emoji representing game mood.

Methods

Method Name	Return Type	Description
`place_mines`	void	Randomly places mines on the board using Python's `random` module.
`count_adjacent_mines`	int	Returns the number of adjacent mines for a given cell.
`calculate_numbers`	void	Updates all non-mine cells with the number of adjacent mines.
`display_board`	void	Prints the current state of the board to the console.
`start_timer`	void	Begins tracking gameplay time.
`stop_timer`	void	Stops the timer for elapsed time calculation.

`get_elapsed_time`	float	Returns total time played in seconds.
`reset_board`	void	Re-initializes the board for a new game session.

`GameLogic` Class

The `GameLogic` class manages player interaction, input parsing, and control flow of the Minesweeper game. It processes user commands and operations to the board such as revealing cells, flagging, starting a new game, or checking for a win.

Data Members

Data Member Name	Type	Description
`level`	string	Gameplay level selected by the player.
`board`	MinesweeperBoard	The active board instance for the current game.
`first_move_done`	bool	Tracks whether the timer has been started.
`running`	bool	Main loop control flag for the game.

Methods

Method Name	Return Type	Description
`run`	void	Starts and runs the gameplay loop, prompting user input.
`process_turn`	void	Parses user commands and performs corresponding actions.
`parse_input`	tuple	Returns the parsed command and coordinates as (action, row, col).
`reveal_cell`	void	Reveals a selected cell and handles mine logic or BFS.
`plant_flag`	void	Toggles a flag at the selected coordinates.
`check_win`	void	Determines whether the game has been successfully completed.

Minesweeper Test Driver

This is the driver script that initiates the Minesweeper game. It calls the main `run()` loop from the `GameLogic` class, and allows the user to play interactively by entering commands.

Functions

Method Name	Return Type	Description
main	void	Starts the Minesweeper program via command-line interface and initializes gameplay.

4. Algorithm Design Concepts Selection

Our Minesweeper game reflects the two concepts from CSC 255 we chose to highlight: **games** and **random number generation.**

Basically, Minesweeper is a strategy-based **game** where the player uncovers cells on a grid without triggering hidden mines. Each move follows defined rules: cells reveal numbers based on nearby mines, flagged cells mark suspected mines, and the game ends in a win or loss depending on the player's choices. These rules are implemented through our code which performs cell reveals, win/loss checking, and flagging cells to highlight how our game uses algorithms to support user interactions and flow.

The unpredictability of the game relies on **random number generation**. When the board is created, mines are placed at random locations using Python's `random` module which relies on the **Mersenne Twister** algorithm. This ensures that every session starts with a new board configuration, giving the player a unique challenge each time. In other words, randomness makes our game dynamic and fun to replay since each iteration is different.

5. Group Member Contributions

Our group worked together consistently over the course of the project to build this functional Minesweeper console game. Early on, we focused on choosing reliable ways to collaborate and communicate across different schedules. We found that Discord was the most effective tool for staying in touch. It allowed us to share quick updates, ask questions, and coordinate meetings throughout the development process.

For coding, we used a shared GitHub repository to store our files and manage updates. This made it easy to track changes, contribute code, and keep everything version-controlled as we each worked on the program. We held four Zoom meetings during the week to walk through project goals, review each class structure, troubleshoot bugs together, and refine gameplay details like mine placement, flag tracking, and win/loss conditions.

Each teammate played an active role in developing the program. Pallavi created the `MinesweeperBoard.py` file, implemented random mine generation, and started the documentation describing the game's technical specifications. Ojan developed the `GameLogic.py` class to manage input commands and game state, and led debugging efforts to

polish user interactions and win checking conditions. We both collaborated on planning, commenting code, reviewing the test driver, and writing up this final specifications document.

Overall, the project involved a balance of planning, coding, and testing. We effectively managed remote collaboration, designing robust gameplay loops, and applying algorithmic concepts like randomness and game states to complete our project.

6. Progress Log

Date	**Teammate(s)**	**Task Description**
7/20/2025	Ojan, Pallavi	● Met on Zoom to discuss the initial project idea, including an overview of the selected topics (*Random Number Generation* and *Games*) ● Selected project idea suggested by Ojan (Minesweeper simulation game) ● Outlined class structure and test driver required and how the code interacts to achieve the program's goals ● Discussed target dates for the week and implementation details, including the need for a follow-up meeting, and details to include in the specifications document
7/21/2025	Pallavi	● Set up a Google Doc to track project progress ● Set up a Discord server to readily communicate with teammates ● Created a first draft of the group lab report with technical specifications, program goals, work progress, etc. ● Added the class file `MinesweeperBoard.py` to instantiate the Minesweeper board for gameplay with choice of three different levels (beginner, intermediate, and expert) and display it ● Implemented random number generation in `MinesweeperBoard.py` to place the mines randomly ● Set up a dummy test driver file Play`Minesweeper.py`
7/21/2025	Ojan, Pallavi	● Met on Zoom to discuss the class file `MinesweeperBoard.py` and talked through the code and its structure
7/23/2025	Ojan	● Added the class file `GameLogic.py` to define functions for playing the game and defining winning/losing conditions for the user's board ● Edited the test driver file Play`Minesweeper.py` to reflect the structure of `GameLogic.py`
7/23/2025	Ojan, Pallavi	● Met on Zoom to discuss the class file

		`GameLogic.py` and talked through the code and its structure ● Reviewed the test driver file Play`Minesweeper.py` and how it calls `GameLogic.py`
7/23/2025	Ojan	● Debugged `GameLogic.py` file to enhance gameplay functions
7/24/2025	Pallavi	● Added comments to `MinesweeperBoard.py` ● Added more print statements related to user entries to `GameLogic.py`
7/24/2024	Ojan, Pallavi	● Met on Zoom to test and finalize all code files ● Finalized technical specifications document and progress log ● Submitted group project

7. Project Expectations vs. Outcomes

Our project exceeded our expectations overall. Our goal was to build a fun, console-based Minesweeper game that showcased concepts from CSC 255, such as random number generation and game logic. With the two main classes (`MinesweeperBoard` and `GameLogic`) working together to handle board setup, gameplay interaction, and win/loss conditions, the program came together smoothly and was a great representation of our initial idea.

Beyond the basic functions, we put extra effort into writing clean, well-commented code that made our logic easy to follow and modify. We tested the game thoroughly, checking multiple difficulty levels, edge cases (like trying to flag revealed cells), and validating user input to create a more reliable user experience. Our Zoom meetings, use of GitHub for version control, and ongoing communication via Discord helped us stay aligned throughout development.

While we're happy with the end result, we also discussed a few enhancements we would have explored if we had more time. These included support for custom board sizes, a function that could algorithmically solve or assist with playing the game (though tricky due to the randomness of mine placement), and a backtracking feature to undo a move after hitting a mine. We also considered using graphical packages like `tkinter` or `pygame` to create a better visual experience, but went with the console-based game due to time constraints.

Overall, this project gave us great experience in collaborative development, applying algorithms to gameplay, and writing clean Python code as a group.