MineSweeper: The Applications of Data Structures

David Zachary Gochuico
GCOE
De La Salle University
Manila, Philippines
david_gochuico@dlsu.edu.ph

Jose Marcial Cello
GCOE
De La Salle University
Manila, Philippines
jose_marcial_cello@dlsu.edu.ph

James Earl Cugal
GCOE
De La Salle University
Manila, Philippines
james_cugal@dlsu.edu.ph

COURSE: LBYCPA2
SECTION: EQ8

***Abstract*— This project delves into the comprehensive development of a Minesweeper game in Java, encompassing multiple facets of game design and programming. The first aspect involves the creation of a sophisticated data structure that efficiently represents the Minesweeper game board, utilizing arrays and lists in Java to store and manage information for each cell. The project emphasizes the application of Java data structures to effectively manage game states, resulting in optimized performance and responsiveness. Clear winning and losing conditions are defined, with corresponding logic implemented to handle game over scenarios, ensuring a conclusive and satisfying conclusion to each game session.**

Keywords—MineSweeper, Game, Data Structures, Java, Game Design

I. INTRODUCTION

Minesweeper, a timeless classic first introduced in 1989, continues to captivate audiences with its engaging blend of logic and strategy. Rooted in the realm of computer-based puzzle gaming, Minesweeper challenges players with a grid of clickable tiles concealing hidden mines. The primary objective is to meticulously uncover all squares on the grid that are mine-free while deftly avoiding the perilous detonation that accompanies an ill-fated click on a mined square. As Minesweeper aficionados will attest, success hinges on astute deduction and the adept use of numerical cues, as each revealed cell bears a number indicating the proximity of neighboring mines. With a clean interface and seamless gameplay, the game offers an immersive experience where players strategically navigate the grid, marking potential mines with a flag to enhance their chances of triumph. Boasting three distinct difficulty levels—Easy, Medium, and Hard—this project endeavors to faithfully recreate the essence of the original Minesweeper, ensuring a satisfying and accurate gaming experience that stands the test of time.

The goal of the team's project is to create this exact minesweeper game. At its core, the Minesweeper Java game development represents a synthesis of advanced algorithms and robust data structures. The meticulously chosen data structures play a pivotal role in efficiently representing the game board, managing diverse cell states, and orchestrating strategic mine placements. The incorporation of algorithms governing randomized mine distribution and cell revealing underscores a meticulously designed framework, ensuring a harmonious and engaging gameplay experience. The project stands as a testament to the seamless integration of learned structures, such as arrays or lists, elevating the organization of game information to new heights. The result is not merely a recreation of a classic but a comprehensive showcase of practical applications for algorithms and data structures, presented through an intuitively designed user interface that empowers players to grasp the game mechanics effortlessly.

II. OBJECTIVES

1. Create a board to represent the game: Develop a data structure to represent the Minesweeper game board in java. This should efficiently store information about each cell.

2. Create algorithms that would randomly place mines on the game board as well as create an algorithm for revealing cells: Design an algorithm that would randomly distribute mines across the game board. This should ensure fairness in the placement of mines and avoid clusters. Implement an algorithm that handles the process of revealing mines when a player clicks on a square.

3. Use learnt data structures in implementing the Minesweeper game into the JAVA programming language: Apply appropriate data structures in Java to represent the game board and manage game states. Utilizing concepts such as arrays, lists, etc.

4. Implement Winning and losing conditions:

   Define the conditions for winning and losing the minesweeper game by establishing rules for identifying a successful completion and conditions that lead to failure. This would also mean the implementation of the necessary logic to handle game over scenarios.

5. Make a UI that is clear and easy to understand. Develop a user interface of the mine swear game that is visually clear and is user-friendly.

III. METHODOLOGY

A. *Basic Gameplay and Target Functionalities*

Rules: MineSweeper is a game where minds are hidden in a grid of squares. Safe squares are going to carry numbers that tell you how many mines touch the square. Thus, you can use the number clues to solve the game by opening all of the safe squares; however, if you click on a mine the game ends and you lose.

Functionalities:

1. Flag: Putting a flag in a zone when you have confirmed that there is a mine.
2. Question Mark: Put a question mark for any tile that is suspected of having a mine.
3. Smiley Face/reset: Click it to reset the game.
4. Numbered tile: Shows numbers how many mines touch the square.
5. Mine: explodes and makes the player lose the game

B. *Target Algorithm*

Algorithm for the computer player

1. It must identify opened cells which has closed neighboring mines
2. Iterates through them and open/flag cells by analyzing each cell until possible.
3. When there is no progress made in the previous step, try to analyze chunks of cells which have common neighbors and go to the second step after the first iteration.
4. When the second and third steps didn't find any solution - try to guess one cell and go to step 2.
5. Continue this algorithm until the game is finished.

Algorithm

1. Assign mines to cells randomly
2. Calculate values for each cell which may have no mine

Right click on a cell

- If cell is opened is true then do nothing
- If cell is opened is false, set the cell to have a flag as true and show a flag on the cell

Left click on a cell

- If cell has been opened is true then do nothing
- If cell has been opened but is false:
  - If cell has a mine then the player loses
  - If the cell has no mine:
    - If the cell has neighboring cells that contain mines and if it is true that it has mines then it marks it as open and displays the number of mines that surround it.
    - If the cell has no neighboring mines, it will be marked open and recursively left clicked on all cells nearby.

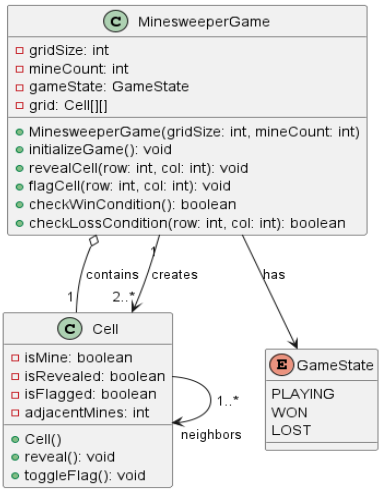
Multi-click

1. If the cell is not opened, do nothing
2. If it is opened:

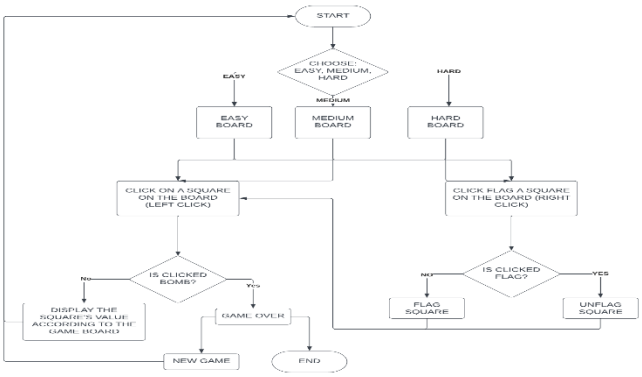
- If it has no neighboring mines, do nothing.
- If it has neighboring mines then
  - If the number of flags around is not equal to the count of mines near, do nothing
  - If the number of flags around is equal to the neighboring mine count:
    - If all flagged cells are not mines, the game is over

If all the flagged cells are mine, perform a left click on all the neighboring cells without flags and not already unlocked.

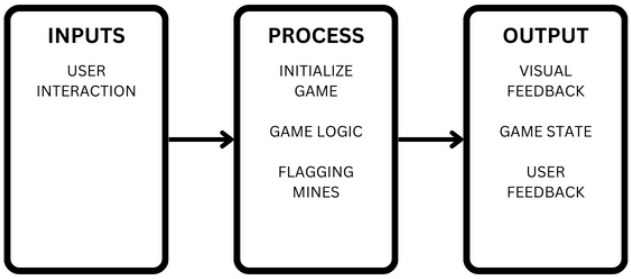
C. *THEORETICAL UML DIAGRAM*



D. *FLOWCHART*



E. IPO DIAGRAM



IV. RESULTS

A. Theoretical:

We ran our Java program for the Minesweeper game with the following settings: difficulty level = easy, board size = 9 x 9, number of mines = 10.

!Figure 1: The graphical user interface of the Minesweeper game

We clicked on the cell at row 5 and column 5, which was a blank cell with no adjacent mines. This triggered the recursive algorithm that revealed all the adjacent blanks and numbers, as shown in Figure 2.

!Figure 2: The result of clicking on a blank cell

We then clicked on the cell at row 3 and column 3, which was a number cell with one adjacent mine. This revealed only that cell, as shown in Figure 3.

!Figure 3: The result of clicking on a number cell

We continued to click on the cells that we thought were safe, based on the number clues. We also used the right mouse button to mark the cells that we suspected were mines with flags, as shown in Figure 4.

!Figure 4: The result of marking cells with flags

We successfully revealed all the cells that were not mines, and marked all the cells that were mines with flags. This resulted in winning the game, as shown in Figure 5. We also received a message that congratulated us and displayed the time taken to complete the game, which was 4 minutes and 20 seconds.

!Figure 5: The result of winning the game

The results of our sample run of the Java program for the Minesweeper game show that we successfully implemented the main features and functionalities of the game. We used various data structures, algorithms, and design principles to create a program that is efficient, robust, and user-friendly. We also followed the coding standards and conventions for Java to ensure the readability and maintainability of our code.

B. Actual

V. CONCLUSION

In conclusion, the "Minesweeper" project aims to bring the classic game to the Java programming language through a new modernized user-friendly implementation. By offering the player a new gaming experience with difficulty levels and randomized gameplay. The project aims to entertain and enhance the logical thinking and problem-solving skills of the player. By developing key Minesweeper features and an intuitive GUI, this project has the capacity to include multiple structures and algorithms that enhances the game's complexity and smoothness.

The development of the minesweeper java game will demonstrate a comprehensive integration of algorithms and data structures. The chosen data structures will prove helpful in representing the game board, managing all the cell states, and mine placement with efficiency. Algorithms for rand mine distribution and cell revealing will showcase a well throughout design that would ensure a balanced and engaging gameplay.

The implementation of the learned structures such as arrays or lists, enhances the organization of the game information. And the code will feature a user-friendly UI that would help players understand how to play without relying on instructions. Overall, the project serves as a practical application of algorithms and data structures and is a good exercise of familiarity with the concepts.

VI. INDIVIDUAL CONTRIBUTIONS (PER MEMBER)

MEMBER	TASK ASSIGNED	ACCOMPLISHED
GOCHUICO, DAVID	UI AND UX DESIGN; GAME FUNCTIONALITY	1. DESIGNED THE USER INTERFACE 2. PLANNING AND GAME THEORY OR LOGIC 3. GAME FUNCTIONALITY 4. CODE CLEANUP/ORGANIZATION
CELLO, MARCIAL	GAME FUNCTIONALITY	1. GAME FUNCTIONALITY 2. PLANNING AND GAME THEORY

		3. BASE GAME FEATURES
CUGAL, JAMES	GAME FUNCTIONALITY	1. FLOWCHART PLANNING 2. GAME LOGIC

		3. BASE GAME FEATURES

VII. REFERENCES

[1] TBD