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. *Data*

1. Game Initialization:
    - The application starts by initializing the minesweeper grid based on the selected difficulty (‘Easy’, ‘Medium’, or ‘Hard’). Bombs are then randomly placed on the grid and neighboring bombs counts for each non-bomb tile are calculated.
2. User Interaction:
    - Users interact with the game through mouse clicks:
        - Left click opens a tile
        - Right click flags a tile
3. Tile Updates:
    - Opening a tile would reveal its contents:
        - If the tile is a bomb the game would end
        - If the tile is empty, adjacent tiles are also revealed recursively.
    - Flagging a tile would prevent accidental clicks.
4. Game Progress:
    - The game continues either all non-bomb tiles are revealed or a bomb tile is opened.
5. Timer and Scoring:
    - A timer starts when the game begins, it would track the time elapsed.
    - Scoring and game completion time are recorded for user feedback.

Square Class:

- Attributes:
    - ‘isBomb’: Indicates whether the square contains a bomb
    - ‘Revealed’: Indicates whether the square has been revealed
    - ‘Flagged’: Indicates whether the square has been flagged
    - ‘numBombsAround’: Stores the count of bombs around the square:
- Methods:
    - ‘Square()’: Constructor initializes the square attributes with default values

SquareNode Class:

- Attributes:
    - ‘square’: Represents a Square object containing data for a particular square.
    - ‘previous’: Reference to the previous SquareNode in the linked list.
    - ‘next’: Reference to the next SquareNode in the linked list.
- Methods:
    - ‘SquareNode(Square square)’: Constructor to initialize a SquareNode with a given Square object.
    - ‘getSquare()’: Returns the Square object stored in the SquareNode.
    - ‘setSquare(Square square)’: Sets a new Square object in the SquareNode.
    - ‘getPrevious()’: Returns the previous SquareNode in the linked list.
    - ‘setPrevious(SquareNode previous)’: Sets the previous SquareNode in the linked list.
    - ‘getNext()’: Returns the next SquareNode in the linked list.
    - ‘setNext(SquareNode next’): Sets the next SquareNode in the linked list.

C. *Planned Modules*

1. MenuApplication:

- Attributes:
    - window: Represents the main window (Stage) for the application.
- Methods:
    - start(Stage stage): Initializes the main window, sets its properties, and loads the initial scene (login-view.fxml).
    - changeScene(String fxml): Changes the current scene to the one specified by the provided FXML filename.

2. MenuController:

- Methods:
    - easyMode(ActionEvent event): Handles the event when the "Easy" button is clicked.
    - mediumMode(ActionEvent event): Handles the event when the "Medium" button is clicked.

- hardMode(ActionEvent event): Handles the event when the "Hard" button is clicked.
- credits(ActionEvent event): Handles the event when the "Credits" button is clicked.

- Responsibilities:
  - Launches the game for different difficulty levels or navigates to the credits scene based on user interactions.
  - Manages the creation of new stages for game difficulty levels.
  - Invokes methods from Difficulty classes to set up the game for different difficulty levels.
  - Uses MenuApplication to change scenes.

3. LoginController:
   - Handles user interactions related to the login process.
   - Invokes the MenuApplication to transition to the "intro-view.fxml" scene when the login button is clicked.
4. LogoutController:
   - Handles user interactions related to logging out.
   - Manages the scene transition to the login view.
   - Methods:
     - onLogout(ActionEvent e): Triggered when the user clicks the logout button.
5. DifficultyEasy, DifficultyMed, DifficultyHard:
   - Attributes:
     - TILE_SIZE: Size of each tile in the game board.
     - W: Width of the game board.
     - H: Height of the game board.
     - bombThreshold: Threshold for flagging tiles based on the number of neighboring bombs.
     - X_TILES: Number of tiles along the X-axis.
     - Y_TILES: Number of tiles along the Y-axis.
     - grid: 2D array representing the game board with tiles.
     - scene: JavaFX scene for the game board.
     - timerText: Text element to display the elapsed time.
     - timeline: JavaFX timeline for updating the timer.
     - elapsedTimeInSeconds: Counter for elapsed time.
   - Methods:
     - createContent(): Creates the game board layout and initializes the tiles.
     - getNeighbors(Tile tile): Returns a list of neighboring tiles for a given tile.
     - Tile: Inner class representing a single tile on the game board.
       - Attributes:
         - x, y: Coordinates of the tile.
         - hasBomb: Boolean indicating whether the tile has a bomb.
         - isOpen: Boolean indicating whether the tile is open.
         - isFlagged: Boolean indicating whether the tile is flagged.
         - border: Rectangle representing the border of the tile.
         - text: Text element displaying the content of the tile (number, bomb, flag).
       - Methods:
         - toggleFlag(): Toggles the flag status of the tile.
         - open(): Opens the tile, revealing its content and potentially triggering a game over.
         - setColorBasedOnNumber(int number): Sets the text color based on the number of neighboring bombs.
     - getNeighbors(Tile tile): Returns a list of neighboring tiles for a given tile.
     - start(Stage stage): Overrides the start method to initialize and show the JavaFX stage.
     - updateTimer(ActionEvent event): Updates the timer text to display the elapsed time.
     - main(String[] args): Main method to launch the application.
6. Credits:
   - Methods:
     - goBack(ActionEvent e): Handles the action event triggered by the user clicking the "Go Back" button. It initiates a transition back to the introductory view of the Minesweeper game

D. Pseudocode (Short)

Input:
- square: a Square object to be inserted

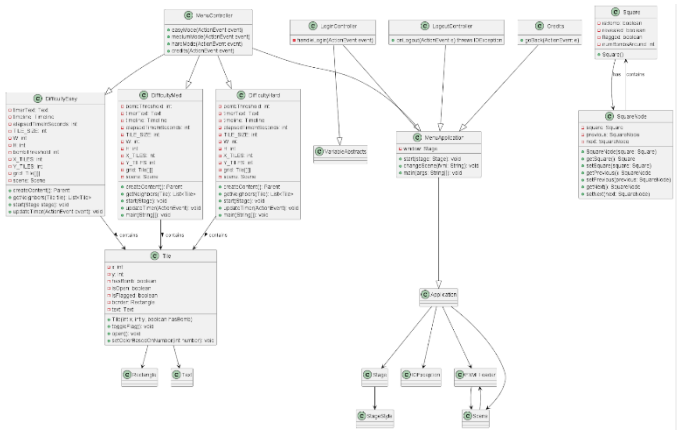
Output:
- Updated linked list with a new SquareNode containing the provided square

Procedure SquareNode(square):
1. Create a new SquareNode object, let's call it newNode, with square as its data.
2. If the linked list is empty (head is null):
   a. Set head to newNode.
3. Else:
   a. Initialize current to head.

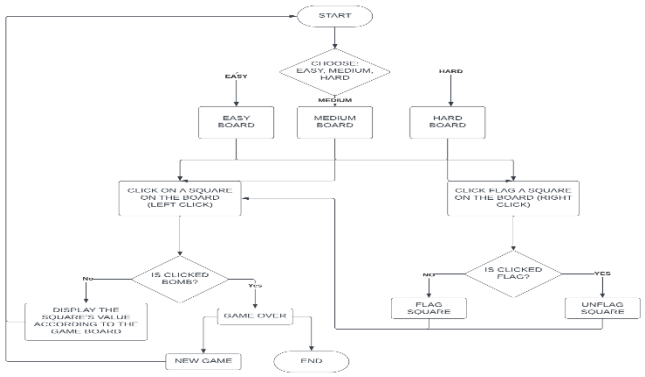
b. Loop until the next of current is null:
   i. Set current to the next of current.
c. Set the next of current to newNode.
d. Set the previous of newNode to current.

4. End procedure.

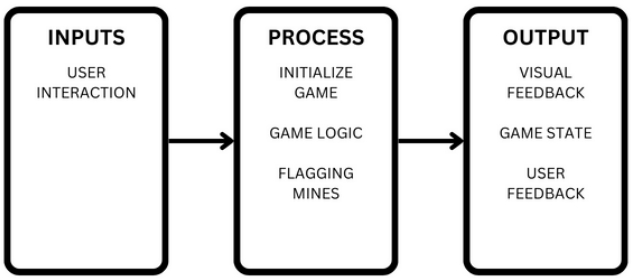
E. UML DIAGRAM



F. FLOWCHART



G. 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.

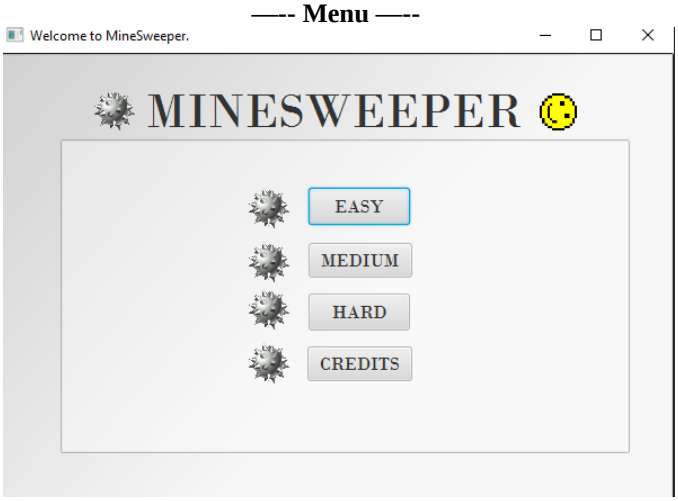
B. Actual

The game revolves around finding all of the clearing a board that contained mines or bombs without detonating any of them, based on the clues provided by the numbers uncovered.There are three types of difficulties to play, (Easy, Medium, or Hard). Each difficulty increases the board and bombs.

—-- Login —--

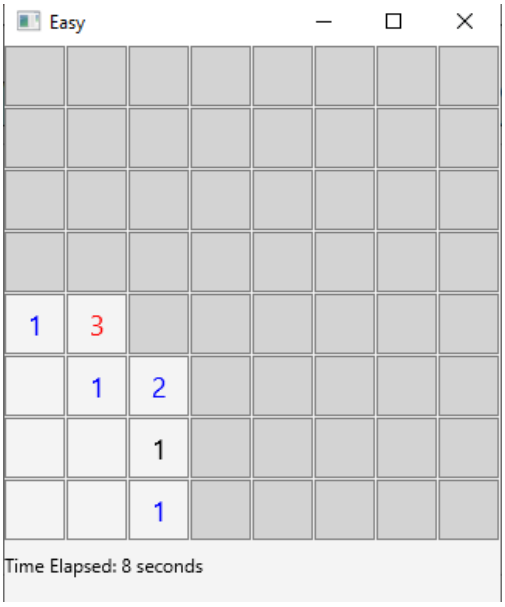
When the game is launched, the user is greeted with the login screen. The login screen contains the title, and a button that changes the scene to the menu.

—-- Menu —--



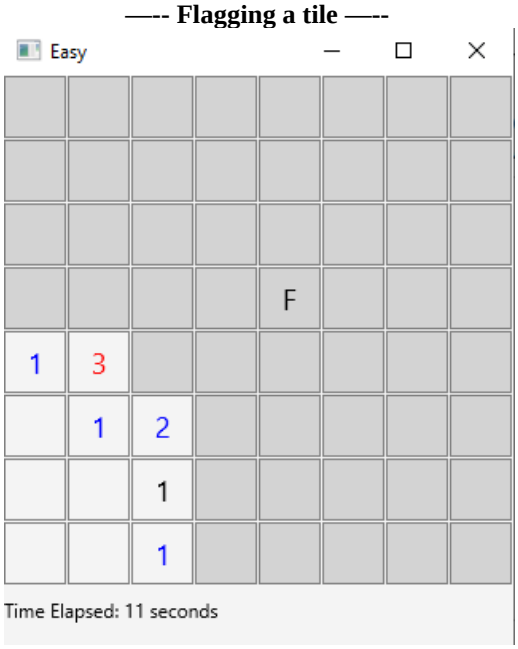
Once the user has entered the game, the user is shown the menu with three different difficulties, easy, medium, or hard. The option for seeing the credits is shown as well.

—--Opening a tile—--



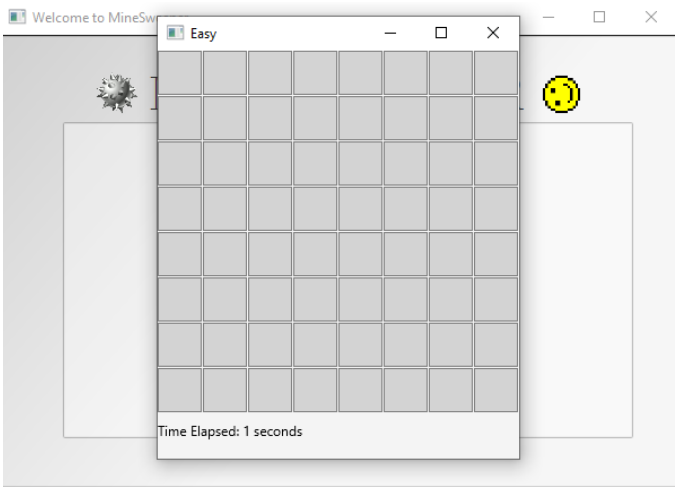
When opening a tile, if it is empty and has no adjacent mines, it would reveal itself and all adjacent empty tiles. If the opened tile contains a number, that number is revealed and indicates the number of mines around it.

—-- Flagging a tile —--



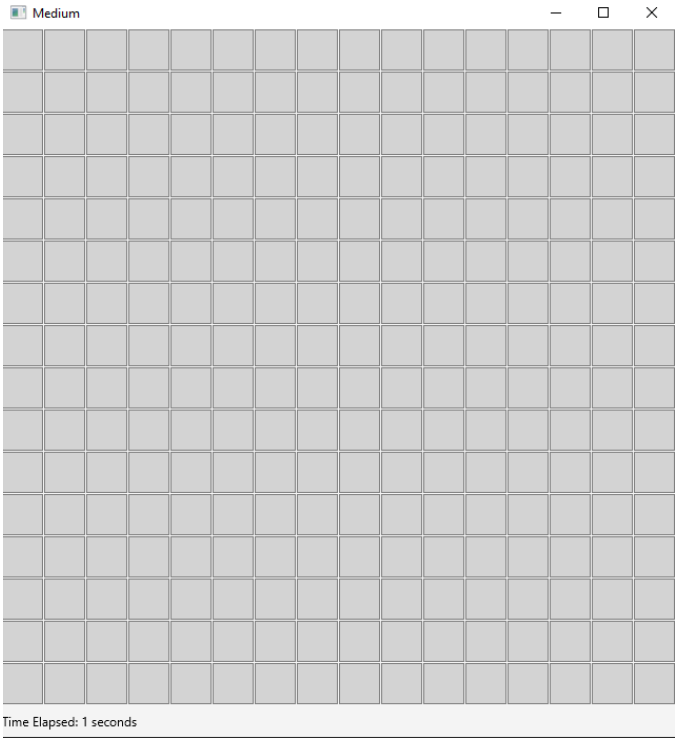
Flagging a tile is a helpful tool for keeping track of potential mines, and you may not be able to open it.

—-- Easy —--



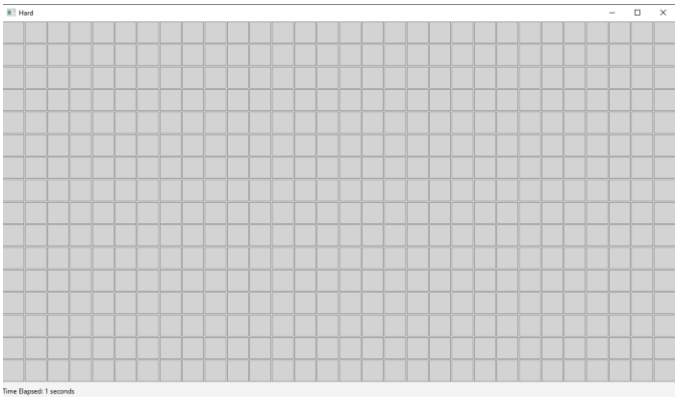
When choosing the easy difficulty, it would show you a game consisting of a grid of cells and each cell can be in one of three states: uncovered, covered, or marked with a flag. The timer in the bottom keeps track of the time.

—-- Medium —--



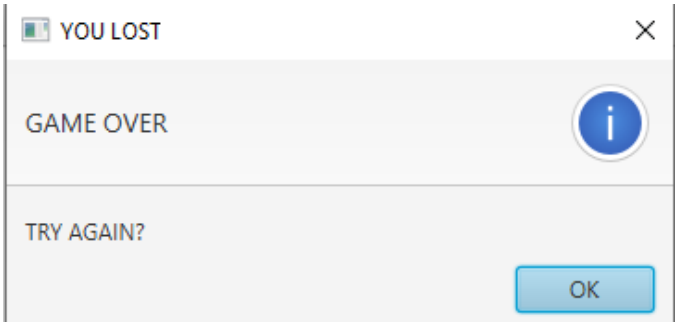
Same as the easy difficulty, however for the medium difficulty it is moderately larger with more bombs

—-- Hard —--



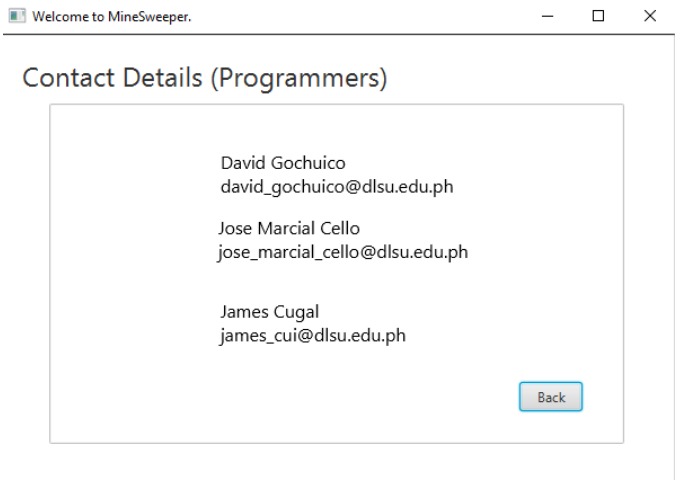
Same as the previous two but larger than medium and easy.

—-- Game Over —--



The game over screen is to be seen when clicking on a tile that has a mine. It would make the game over and force you to restart the game.

—-- Credits —--



Here are all of the names of the members who contributed to making the game.

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 5. Code Finalization 6. UI/UX and Gameplay design
Cello, Marcial	Game Functionality	1. Game Functionality 2. Planning and Game Theory 3. Base Game Features 4. Code Cleanup/Organization 5. Game Engine
Cugal, James	Game Functionality	1. FlowChart planning 2. Game Logic 3. Base Game Features

VII. References

[1] Walker, J. (2010)

Minesweeper and Hypothetical Thinking Action Research & Pilot Study. Effective Education Project. https://files.eric.ed.gov/fulltext/ED509464.pdf