Game Project: Minesweeper

University of San Carlos
School of Arts and Sciences
Department of Computer, Information Sciences and Mathematics
Talamban Campus, Cebu City, Philippines

CIS 2103 - Object-Oriented Programming

Submitted By:
Kurt Russel Carrillo
Kurt Zander Kaw
Joanna Alyssa Mondelo
Cheska Gayle Ouano

Submitted To:
Sir Gran Sabandal

December 28, 2024

CHAPTER I
INTRODUCTION

Background of the Study

Minesweeper is a classic puzzle game that has been popular for decades, known for its simple yet challenging gameplay. The objective of the game is to clear a grid without detonating hidden mines, using numerical hints to identify the location of the mines. Originally developed as part of early computer operating systems, Minesweeper has become a staple game that tests logic and problem-solving skills. Its appeal lies in the balance between luck and strategy, making it both frustrating and addictive for players of all ages. Over the years, Minesweeper has been recreated in various programming languages, showcasing its versatility as a project for developers. For this project, the focus is on building the game from scratch using modern web technologies like Java. This approach not only recreates the classic experience but also provides an opportunity to practice front-end development, game logic implementation, and user interface design.

Despite its simplicity, recreating Minesweeper involves solving several technical challenges, such as implementing the core game logic, managing dynamic user interactions, and ensuring a responsive and visually appealing design. The main goal is to deliver a fully functional, interactive, and responsive version of Minesweeper that can run smoothly using Java. Additionally, the project will focus on creating a clean and intuitive interface to enhance the user experience. Although the scope is limited to single-player functionality, the project aims to replicate the original gameplay as accurately as possible while adding an extra feature to the game and some modern design elements to improve visual appeal.

CHAPTER II
GAME FEATURES

Minesweeper is a classic puzzle game that challenges players to clear a grid while avoiding hidden mines. In this version of minesweeper, the goal is to uncover all the tiles without clicking one of the mines using the numerical clues to deduce the location of the mines, another way to win the game is to find the treasure.

Unique Addition:

- **Treasure Tile:** Randomly placed on the board. Finding it grants an immediate win.
- **Dynamic Grid Generation:** Grid adjusts based on difficulty level selection.

Core Features:

- Three Difficulty Levels:
    - Easy (8x8 grid, 10 mines).
    - Medium (12x12 grid, 20 mines).
    - Hard (16x16 grid, 40 mines).
- Custom Design Elements:
    - Alternate Tile colors for improved visibility.
    - Mines represented by "💣" and treasures by "💎".
    - Flags placed with right-click, represented by "🚩".
    - Timer in seconds and record your best score.
- Game Rules:
    - Left-click to reveal tiles.
    - Right-click to place or remove flags.
    - Revealing a mine ends the game.
    - Finding treasure results in an instant win.

How To Play

To start the game, click anywhere on the grid to reveal a tile. If the tile is a number (1-8), it indicates how many mines are surrounded by it. You can place a flag by using the right-click to mark a suspected mine, to remove the flag, just right-click it again. An alternate way to win the game is to find the treasure hidden among the grid.

CHAPTER III
TECHNOLOGIES & TOOLS

Visual Studio Code

For writing and editing the code for the Minesweeper game, Visual Studio Code was used. It's lightweight and super flexible, with lots of extensions that made managing and organizing the project files a lot easier. The researchers were already familiar with it, so it was the best fit for us to work with, making coding more efficient.

Google Docs

Google Docs was used to draft and organize important parts of the project, like the project requirements, design specs, and progress reports. The best part was the collaboration feature, which let us all edit the documents at the same time. This helped keep everything structured and ensured the development process was well-documented.

Github

GitHub served as the primary platform for sharing and managing the project's codebase. By utilizing Git for version control, the team could track changes, merge contributions, and resolve conflicts. GitHub also acted as a repository for project backups, preventing data loss.

LucidChart

Lucidchart was used to create the UML diagrams necessary for designing and visualizing the system architecture. Diagrams such as class diagrams and sequence diagrams helped in planning the structure of the game, identifying relationships between different components, and streamlining the development process.

Colorhunt.co or Figma.com

These platforms were utilized to select and apply color schemes for the game's interface. Providing ready-made color palettes that helped in creating an appealing and cohesive design. These tools ensured that the game's visual elements were both aesthetically pleasing and consistent.

CHAPTER IV
METHODOLOGY

Class Structure

Main Class (main.java): Launches the game and handles the difficulty selection.

- **main()**: Starts the game by showing a difficulty selection dialog.
- **createAndShowDifficultyDialog()**: Displays a dialog to choose between easy, medium, and hard modes. Depending on the selection, the game initializes with corresponding settings.

Minesweeper Class (minesweeper.java): Core game logic, GUI setup, and event handling.

- Inner Class:
    - **MineTile**: Represents individual tiles on the board (inherits JButton). Each tile tracks its row and column (r, c).
- Fields:
    - **numRows, numCols**: Board dimensions.
    - **mineCount, treasureCount**: Number of mines and treasures.
    - **board[][]**: 2D array to store tiles.
    - **mineList, treasureList**: Lists of mine and treasure tiles.
    - **timer, elapsedTime, highScore**: Timer and score tracking.
- Methods:
    - **setupGame()**: Initializes the board, mines, treasures, and UI.
    - **initializeBoard()**: Creates the grid layout and tiles.
    - **setMines(), setTreasures()**: Randomly places mines and treasures.
    - **checkMine()**: Recursive method to reveal tiles and count adjacent mines.
    - **revealMines(), revealTreasure()**: Handles end-game conditions by revealing mines or treasures.
    - **startTimer()**: Starts and updates the game timer every second.
    - **loadHighScore(), saveHighScore()**: Reads/writes high scores to text files.

UML Diagram

A visual representation of a system, showcasing its structure, components, and relationships of the classes. The UML Diagram provides a clear, graphical view of the minesweeper components, making it easier to understand the system's functionality and interactions.

(Ibutang lang diri ang uml alyssa)

Figure 1: UML Diagram of Minesweeper

UI Components

- JFrame: Main game window.
- JPanel: Divides the UI into sections (textPanel, boardPanel).
- JLabel: Displays game information (mine count, timer).
- JButton (MineTile): Each button acts as a tile that can be clicked.

Flow of the Code

Start Game (Main)

The Main class invokes the (createAndShowDifficultyDialog()), which is the difficulty selection dialog. The selected difficulty initializes the Minesweeper class with different configurations (easy, medium, hard).

```
public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> createAndShowDifficultyDialog());
    }

    public static void createAndShowDifficultyDialog() {
        JFrame frame = new JFrame("Difficulty");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create a custom panel for the dialog
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(Color.decode("#27214f"));

        // Add a title label
        JLabel label = new JLabel("Choose a Difficulty", SwingConstants.CENTER);
        label.setFont(new Font("Arial", Font.BOLD, 18));
        label.setForeground(Color.WHITE);
        label.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0));
        panel.add(label, BorderLayout.NORTH);

        // Create a button panel
        JPanel buttonPanel = new JPanel(new GridLayout(1, 3, 10, 10));
        buttonPanel.setBackground(Color.decode("#27214f"));
```

Figure 2: Main Class & createAndShowDifficultyDialog() method

Board Setup

setupGame() initializes the frame, layout, and text panels (textLabel, timerLabel). The board is created with initializeBoard(), and tiles are added to the grid layout. The mine and treasure will then be placed using the setMines(), which randomly places mines and setTreasures(), which places a treasure not occupied by a mine.

```
public void setupGame(int choice) {
    switch (choice) {
        case 0:
            mineCount = 10;
            numCols = 8;
            break;
        case 1:
            tileSize = 60;
            emojisize = 40;
            mineCount = 20;
            numCols = 12;
            break;
        case 2:
            tileSize = 48;
            mineCount = 40;
            numCols = 16;
            emojisize = 30;
            break;
        default:
            break;
    }
    numRows = numCols;
    boardWidth = numCols * tileSize;
    boardHeight = numRows * tileSize;
    board = new MineTile[numRows][numCols];
    frame.setSize(boardWidth, boardHeight + 50); // Add space for timer
    frame.setLocationRelativeTo(null);
    frame.setResizable(false);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(new BorderLayout());

    textLabel.setFont(new Font("Arial", Font.BOLD, 25));
    textLabel.setHorizontalAlignment(JLabel.CENTER);
    textLabel.setText("Minesweeper: " + mineCount);
    textLabel.setOpaque(true);

    timerLabel.setFont(new Font("Arial", Font.BOLD, 25));
```

Figure 3: setupGame() Method

```
private void initializeBoard() {
    for (int r = 0; r < numRows; r++) {
        for (int c = 0; c < numCols; c++) {
            MineTile tile = new MineTile(r, c);
            board[r][c] = tile;

            if ((r + c) % 2 == 0) {
                tile.setBackground(Color.decode("#2A0055"));
            } else {
                tile.setBackground(Color.decode("#47008E"));
            }

            tile.setBorder(BorderFactory.createLineBorder(Color.decode("#1C0039"), 2));  // White gridlines, 2px thick
            tile.setFocusable(false);
            tile.setMargin(new Insets(0, 0, 0, 0));
            tile.setFont(new Font("Arial Unicode MS", Font.PLAIN, emojisize));
            tile.addMouseListener(new MouseAdapter() {
                @Override
                public void mousePressed(MouseEvent e) {
                    if (gameOver) {
                        return;
                    }

                    MineTile tile = (MineTile) e.getSource();

                    if (e.getButton() == MouseEvent.BUTTON1) {
                        if (tile.getText().equals("")) {
                            if (mineList.contains(tile)) {
                                revealMines();
                            } else if (treasureList.contains(tile)) {
                                revealTreasure(tile);
                            } else {
                                checkMine(tile.r, tile.c);
                            }
                        }
                    } else if (e.getButton() == MouseEvent.BUTTON3) {
```

Figure 4: initializedBoard() Method

```
public void setMines() {
    mineList = new ArrayList<>();
    int mineLeft = mineCount;
    while (mineLeft > 0) {
        int r = random.nextInt(numRows);
        int c = random.nextInt(numCols);

        MineTile tile = board[r][c];
        if (!mineList.contains(tile)) {
            mineList.add(tile);
            mineLeft -= 1;
        }
    }
}

public void setTreasures() {
    treasureList = new ArrayList<>();
    int treasureLeft = treasureCount;
    while (treasureLeft > 0) {
        int r = random.nextInt(numRows);
        int c = random.nextInt(numCols);

        MineTile tile = board[r][c];
        if (!mineList.contains(tile) && !treasureList.contains(tile)) {
            treasureList.add(tile);
            treasureLeft -= 1;
        }
    }
}
```

Figure 5: setMines & setTreasures

GamePlay() (Mouse Settings)

The left-click reveals tiles, checks for mines/treasures, or recursively reveals adjacent tiles. The right-click flags/unflags tiles with a flag emoji (🚩). If a mine is clicked, revealMines() ends the game.

```
tile.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        if (gameOver) {
            return;
        }

        MineTile tile = (MineTile) e.getSource();

        if (e.getButton() == MouseEvent.BUTTON1) {
            if (tile.getText().equals("")) {
                if (mineList.contains(tile)) {
                    revealMines();
                } else if (treasureList.contains(tile)) {
                    revealTreasure(tile);
                } else {
                    checkMine(tile.r, tile.c);
                }
            }
        } else if (e.getButton() == MouseEvent.BUTTON3) {
            if (tile.getText().equals("") && tile.isEnabled()) {
                tile.setText("🚩");
                tile.setForeground(Color.RED);
            } else if (tile.getText().equals("🚩")) {
                tile.setText("");
            }
        }
    }
});
```

Figure 6: Mouse Settings

Time & Scoring

The game tracks elapsed time and compares it to the high score, which is saved upon winning. Game over or victory messages are displayed using JOptionPane.

```
private void startTimer() {
    timer = new Timer(1000, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            elapsedTime++;
            timerLabel.setText("Time: " + elapsedTime + "s");
        }
    });
    timer.start();
}

private void loadHighScore(int difficulty) {
    String filename = "highscore_" + getDifficultyName(difficulty) + ".txt";
    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        highScore = Integer.parseInt(reader.readLine());
    } catch (IOException | NumberFormatException e) {
        highScore = 0;
    }
}

private void saveHighScore(int difficulty) {
    String filename = "highscore_" + getDifficultyName(difficulty) + ".txt";
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        writer.write(Integer.toString(highScore));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 7: Timer ang HighScore Method

```java
public void gameOver(String message, boolean win) {
    timer.stop();

    if (win) {
        if (elapsedTime < highScore) {
            highScore = elapsedTime;
            saveHighScore(numCols == 8 ? 0 : numCols == 12 ? 1 : 2);  // Determine difficulty
            JOptionPane.showMessageDialog(frame, message + "\nNew High Score: " + elapsedTime + "s");
        } else {
            JOptionPane.showMessageDialog(frame, message + "\nTime: " + elapsedTime + "s\nHigh Score: " + highScore + "s");
        }
    } else {
        JOptionPane.showMessageDialog(frame, message + "\nTime: " + elapsedTime + "s\nHigh Score: " + highScore + "s");
    }

    SwingUtilities.invokeLater(() -> Main.createAndShowDifficultyDialog());
    frame.dispose();
}
```

Figure 8: gameOver() Method

CHAPTER V
CONCLUSION & RECOMMENDATION

Conclusion

The Minesweeper game project was developed to recreate the classic puzzle game while improving the overall design and functionality. The main goal was to implement key features such as random mine placement, interactive cells, and clear win/loss conditions. Through continuous development and testing, the project met the necessary requirements and provided a fun and engaging experience for users. Working on this project allowed us to improve our skills in object-oriented programming and user interface design. It also gave us a better understanding of how to solve problems and apply logical thinking in game development.

Future Improvements

- Multiplayer Mode: Add a competitive mode where players can compete to clear the board fastest.
- Leaderboard System: Implement a scoring and leaderboard system to track player performance.
- Save/Load Feature: Enable users to save and resume their game progress.
- Graphical Enhancements: Improve cell animations and sound effects for a more immersive experience.
- Accessibility Features: Add colorblind mode and keyboard navigation support.

Recommendations

To ensure the long-term success and scalability of the Minesweeper game, a modular approach to development should be adopted. This will make it easier to maintain the project and add new features in the future. Prioritizing thorough code documentation will help future developers understand and modify the project more efficiently. Implementing version control systems, such as Git, will enable better tracking of changes and facilitate smoother collaboration during development.