**VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY**

**INTERNATIONAL UNIVERSITY**

School of Computer Science & Engineering



**DATA STRUCTURES AND ALGORITHMS PROJECT**

**Topic:** Development of an Interactive Minesweeper Game

**Instructor**: Thái Trung Tín

**Members:**

Nguyễn Đức Bảo – ITITIU22238

Nguyễn Quốc Trung – ITITIU22171

[**I. INTRODUCTION**](#_qn551acbgedm) 4

[1. Objectives](#_p645zlw8xr8c) 4

[a. Implement Core Game Mechanics](#_yvzq30uoy1ns) 4

[b. Utilize Fundamental Data Structures](#_p8i4fgpmhan3) 4

[c. Implement Advanced Features](#_3ccxveuh39x8) 4

[2. Used tools](#_841by1sx0fa2) 4

[**II. GAMEPLAY AND GAME RULES**](#_7m0w9xqobvqq) 4

[1. Gameplay](#_fhgkadzhaxv6) 4

[a. Start the Game:](#_1aypr4stfl3k) 4

[b. Reveal Cells:   
The player clicks on any cell to uncover it. If the cell contains a mine, the game is over. If the cell does not contain a mine, it will reveal a number or be blank:](#_a4wsh0qy7ddd) 5

[c. Mark Mines:](#_7ks6ld8bfvkf) 5

[d. Win Condition:](#_igtfs3ibszot) 5

[e. Lose Condition:](#_3bg0xpypcwe6) 5

[2. Game rules](#_4qh9q2qwcuwe) 5

[a. Grid and Mines](#_s0ydew5rglnu) 5

[b. Cell Interaction](#_mxddylr1pgij) 5

[c. Numbers on Cells](#_j7w8jqyzu1wi) 5

[d. Flags](#_3u503qrr3v09) 5

[e. Automatic Uncovering](#_6j723b877m8a) 6

[f. Endgame](#_9zor1z73dqdy) 6

[3. Levels](#_5c9dbfoxyu3o) 6

[**III. DETAILS**](#_ebdkxu34u6l5) 6

[1. Functionality](#_qc7hskjbao6e) 6

[1.1 App class](#_tmc5bb71mnp7) 6

[1.2 MineSweeperFrame class](#_s6co7b3shcum) 6

[1.3 Singleton Pattern](#_x0rel1ml9vcr) 6

[1.4 BoardPanel class](#_czzu0qkwpltx) 7

[1.5 BoardPanelFactory class and IBoardPanel interface:](#_7e887j5w7vl1) 8

[1.6 CareTaker and Memeto](#_k5jns56ih00f) 9

[1.7 MineTile class](#_992o2e2l4jhg) 9

[1.8 SettingMenu and SettingPanel class](#_83kczlpa2wck) 9

[1.9 TextPanel class](#_5daxhdfjbspw) 10

[**IV. DEMONSTRATION**](#_gwfd9f1ljkgu) 10

[1. Launching the Game](#_616ox2l83lvj) 10

[2. User Interface](#_771ktcua3cpy) 11

[3. Revealing Tiles](#_1xrbq9hb4690) 11

[4. Flagging Mines](#_yfloiuqldps3) 13

[5. Winning the Game](#_j1uq7xptyhbh) 13

[6. Difficulty Levels](#_ow8gfsqabhqi) 14

[**V. CONCLUSION**](#_rtiq9tr9qekh) 14

[1. Summary](#_1rij7bdl36q0) 14

[2. Limitation](#_ykj729oihipx) 14

[3. Evaluation](#_getvptsdudga) 14

[**REFERENCES**](#_7lp5e4qshcbn) 15

[1. Tutorial](#_5fl18r4g8ypp) 15

[2. Design patterns](#_wao9uiv70gjc) 15

# INTRODUCTION

Minesweeper is a classic puzzle game that challenges players to clear a grid of hidden mines using logic and deduction. The objective is to uncover all the cells on the board that do not contain mines while avoiding those that do. Each uncovered cell provides clues about the number of neighboring mines, guiding the player's next moves.

## Objectives

### Implement Core Game Mechanics

* Develop the Minesweeper game logic, including mine placement, cell uncovering, and mine detection.
* Create a user interface to interact with the game, allowing players to mark mines and uncover cells.

### Utilize Fundamental Data Structures

* Use **arrays or matrix** data structures to represent the game board.
* Implement efficient algorithms for mine placement and neighboring cell calculations.

## Used tools

* IDEs for programming and debugging: JetBrains IntelliJ IDEA
* Project Management:

# GAMEPLAY AND GAME RULES

## Gameplay

Minesweeper is a single-player puzzle game where the objective is to clear a rectangular grid containing hidden mines without detonating any of them. The gameplay involves the following steps:

### Start the Game:

The game begins with a grid of covered cells, and the player has no initial information about the location of the mines.

### Reveal Cells: The player clicks on any cell to uncover it. If the cell contains a mine, the game is over. If the cell does not contain a mine, it will reveal a number or be blank:

* **Numbered Cells**: A number indicates how many mines are in the eight neighboring cells (horizontal, vertical, and diagonal).
* **Blank Cells**: A blank cell means there are no mines in the adjacent cells. All neighboring cells are automatically uncovered, and this process continues recursively until cells with numbers are uncovered.

### Mark Mines:

If the player suspects a cell contains a mine, they can right-click (or use a designated key or button) to mark the cell with a flag. This helps in keeping track of potential mine locations.

### Win – Lose Condition:

* The game is won when all non-mine cells are uncovered. The player must carefully use the numbers on uncovered cells to deduce the locations of mines and avoid them.
* The game is lost if the player uncovers a cell containing a mine.

## Game rules

### Grid and Mines

* The game grid can vary in size, typically 8x8.

### Cell Interaction

* **Left-Click**: Uncover a cell. If it is a mine, the game ends. If it is not, it shows a number or is blank.
* **Right-Click**: Place or remove a flag to mark a suspected mine.

### Numbers on Cells

* A cell with no neighboring mines is blank and triggers the uncovering of adjacent cells.

### Flags

* Flags are used to mark suspected mine locations. When a cell contains flag then it can not be pressed until unflagged.

### Automatic Uncovering

* When uncovering a blank cell, the game will automatically reveal all adjacent blank and numbered cells in a chain reaction until numbered cells are bordered by covered cells.

### Endgame

* The game ends either when all mines are flagged, and all other cells are uncovered (win) or when a mine is uncovered (loss).

# DETAILS

## Functionality

### Mine Field class

#### ****Purpose****:

The MineField class handles the grid (board) where the game is played, placing mines randomly, and providing methods to interact with the board.

* + 1. **Constructor**

**Purpose**:

* Initializes the game board with dimensions (numRows x numCols) and sets the total number of mines.
  + 1. **initializeBoard()**

**Purpose**:

* Creates a new MineTile object for each cell in the grid.
  + 1. **setMines**()

**Purpose**:

* Randomly assigns mineCount mines to tiles on the board.
  + 1. **isMine(int r, int c)**

**Purpose**:

* Checks if the tile at position (r, c) is a mine.
  + 1. **getBoard()**

**Purpose:**

* Returns the entire board as a 2D array of MineTile objects.
  + 1. **getMineList()**

**Purpose:**

* Returns the list of mine tiles.

### MineTile class

**Purpose**:

The MineTile class represents a single cell (or tile) on the Minesweeper board. It is a specialized version of a JButton that includes row and column indices for easy reference in the game logic.

**1.2.1. Field**

**Purpose**:

* r and c store the row and column indices of the tile on the Minesweeper board.
* These indices allow efficient access and manipulation of the tile within the MineField grid.
  + 1. **Constructor**

**Purpose**:

* Initializes the tile with its row (r) and column (c) indices and applies specific UI configurations.

### Game Control class

**Purpose**:

The GameControl class is a utility class to manage the state of the Minesweeper game. It tracks whether the game is over and counts how many tiles have been clicked.

* + 1. **resetGame(MineField field, JLabel textLabel)**

**Purpose**:

* Resets the game state for a new game session.
  + 1. **isGameOver()**

**Purpose**:

* Returns whether the game has ended.
  + 1. **setGameOver(boolean state)**

**Purpose**:

* Updates the gameOver status to the given state.
  + 1. **incrementTilesClicked()**

**Purpose**:

* Increases the count of tiles that have been clicked during the game.

### Minesweeper class

* + 1. **Minesweeper Constructor**

**Purpose**:

* Initializes the game, sets up the GUI, creates the minefield, and prepares event handling.

**Time Complexity**:

* O(n×m): Initializes the board with n×mn \times mn×m tiles.
* O(1): Adds listeners and creates GUI components.
  + 1. **handleLeftClick(MineTile tile)**

**Purpose**:

* Handles the left-click event on a tile.
* Reveals mines if clicked on a mine or reveals safe tiles recursively.

**Time Complexity**:

* **Best Case**: O(1) (If the game stops after one tile).
* **Worst Case**: O(n×m) (Flood-fill for all tiles).
  + 1. **handleRightClick(MineTile tile)**

**Purpose**:

* Handles the right-click event to toggle a flag on a tile.

**Time Complexity**:

* O(1): Simple toggle operation.
  + 1. **revealMines()**

**Purpose**:

* Reveals all mines on the board when the game ends.

**Time Complexity**:

* O(k), where k is the number of mines.
  + 1. **revealSafeTile(int r, int c)**

**Purpose**:

* Reveals a safe tile and recursively uncovers adjacent tiles if no surrounding mines are found.

**Time Complexity**:

* **Best Case**: O(1) (Reveals only one tile).
* **Worst Case**: O(n×m) (Reveals all tiles in a flood-fill manner).
  + 1. **countMine(int r, int c)**

**Purpose:**

* Checks if a given tile is a mine.

**Time Complexity**:

* O(1): Single check for each tile.
  + 1. **countSurroundingMines(int r, int c)**

**Purpose**:

* Counts the number of mines surrounding a specific tile.

**Time Complexity**:

* O(1): Fixed number of neighbors (8).
  + 1. **checkWinCondition()**

**Purpose**:

* Checks if all non-mine tiles are revealed, signaling the player has won.

**Time Complexity**:

* O(n×m): Checks every tile.
  + 1. **restartGame()**

**Purpose**:

* Resets the game board with the specified number of mines.

**Time Complexity**:

* O(n×m): Reinitializes all tiles.
  + 1. **Main()**

**Purpose:**

* Entry point of the program. Creates an instance of Minesweeper.

**Time Complexity:**

* O(1): Instantiates the game.

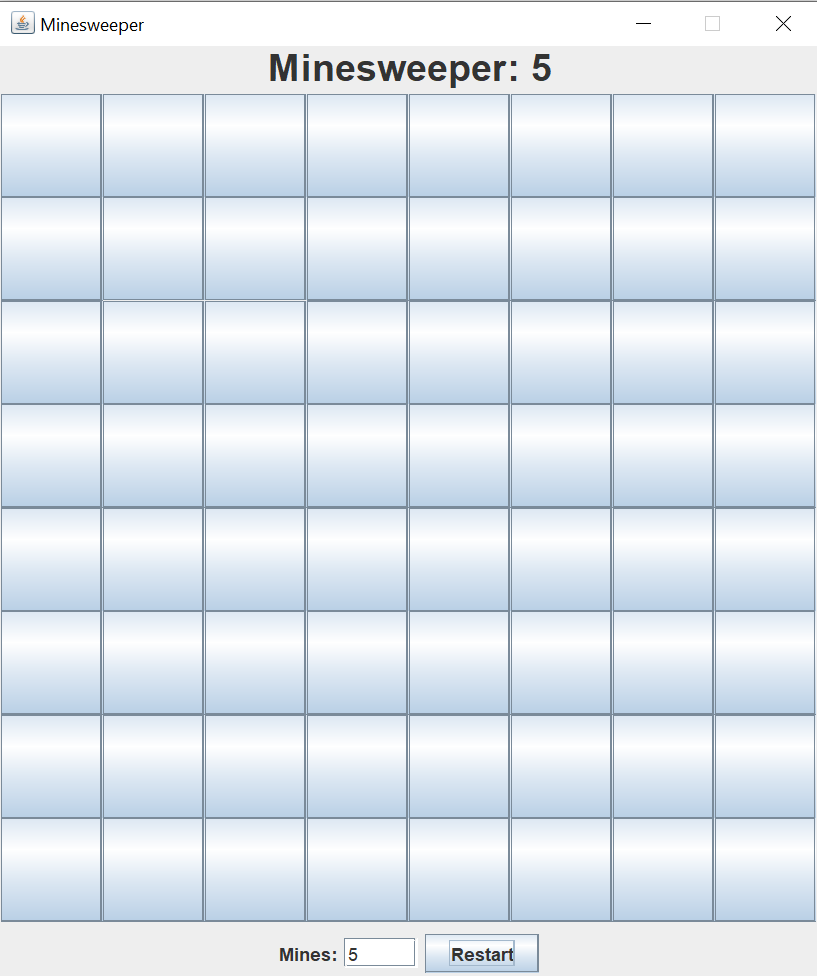
**Summary of Function Complexities**

|  |  |  |
| --- | --- | --- |
| **Function** | **Purpose** | **Time Complexity** |
| Minesweeper() | Constructor to initialize the game | O(n×m) |
| handleLeftClick() | Reveals a tile or handles mine click | O(n×m) |
| handleRightClick() | Toggles a flag on a tile | O(1) |
| revealMines() | Reveals all mines | O(k) |
| revealSafeTile() | Recursively reveals safe tiles | O(n×m) |
| countMine() | Checks if a tile is a mine | O(1) |
| countSurroundingMines() | Counts adjacent mines | O(1) |
| checkWinCondition() | Checks if the player has won | O(n×m) |
| restartGame() | Resets the game | O(n×m) |
| main() | Entry point | O(1) |

# DEMONSTRATION

## Launching the Game

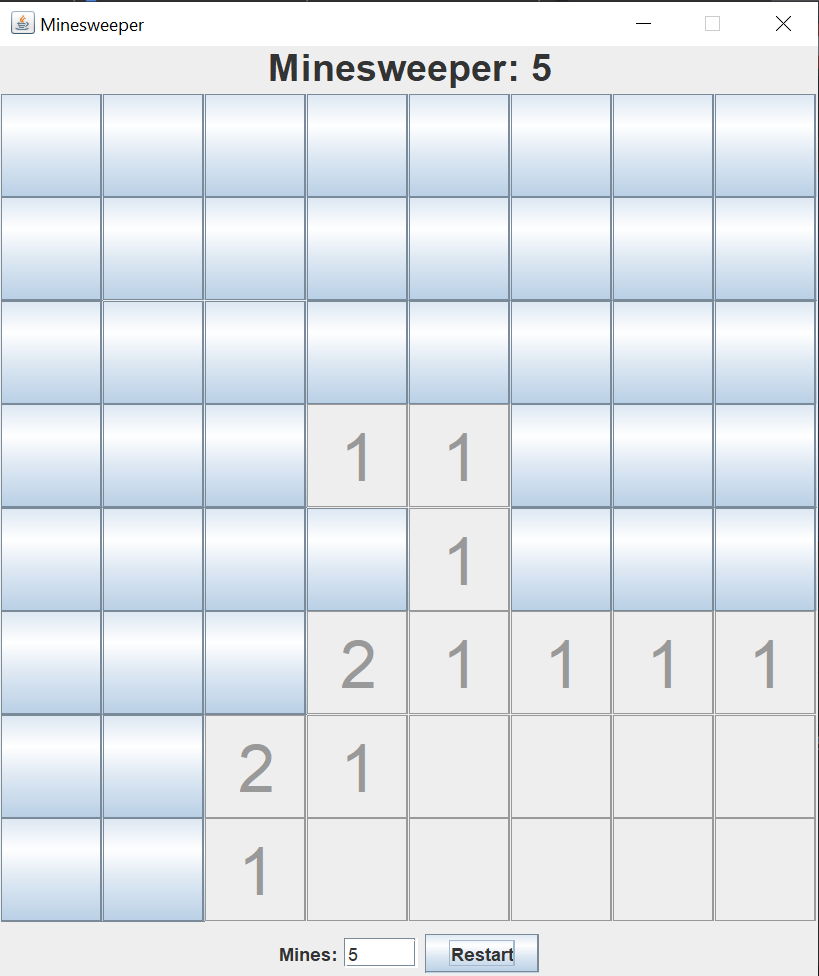
Upon launching the game, the player is presented with a grid of covered tiles.



*Figure 8. 8x8 grid*

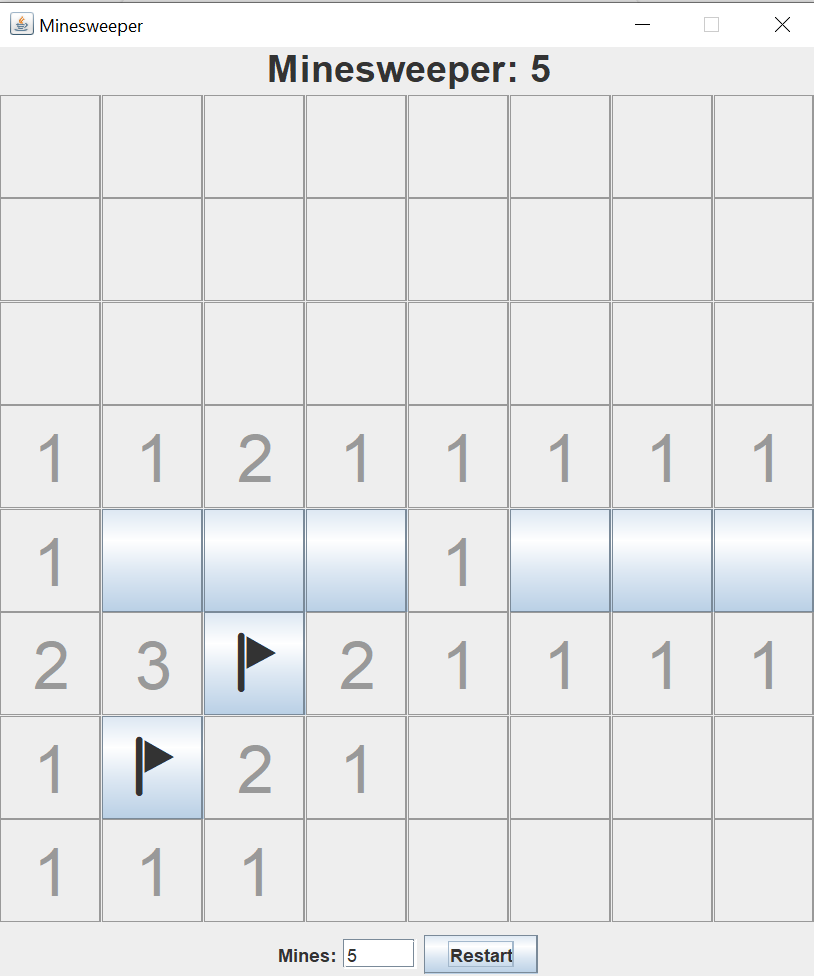
## User Interface

The Minesweeper game interface features an 8x8 grid where players can left-click tiles to reveal numbers indicating nearby mines or right-click to flag suspected mines. At the bottom, there's a **"Mines" input field** where players can specify the number of mines before starting, and a **"Restart" button** to reset the game with the chosen mine count. The top of the interface displays the total number of mines dynamically, providing a clean and simple design for seamless gameplay.

*Figure 9.*

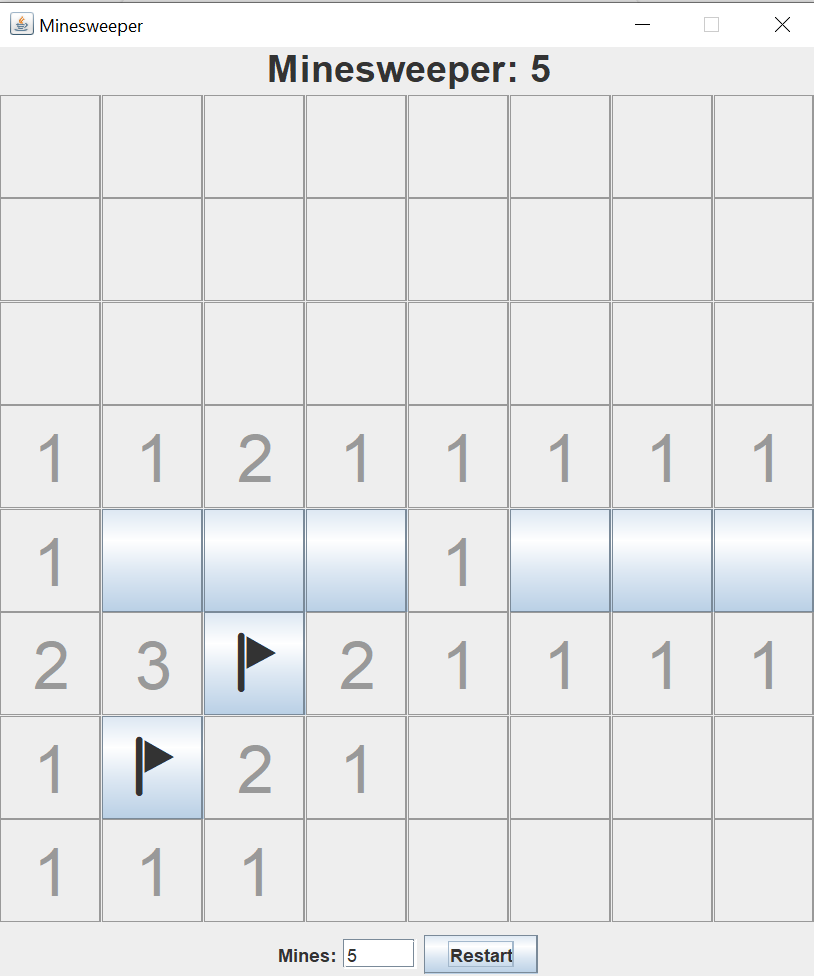
## Revealing Tiles

Players reveal tiles by left-clicking on them. If the tile does not contain a mine, it will show a number indicating how many adjacent tiles contain mines.



If a player clicks on a mine, all mines are not shown up yet. Instead, the board is temporarily hidden and a message appears, notifying that the player clicked on mines and they can either choose to undo that recent step or restart the game. If they choose to undo, they can continue the game keeping in mind that one tile contains mine to avoid clicking it. If they choose to restart, a new game with same difficulty level will be launched. For easy, medium, hard mode, the number of undo time available is 3, 2, 1, respectively.

## Flagging Mines

Players can flag tiles they suspect to contain mines by right-clicking on them. Flagged tiles cannot be revealed until the flag is removed by right-clicking again.

## Winning the Game

The player wins the game by correctly revealing all non-mine tiles. Once this condition is met, a ‘Mines Cleared!’ message is displayed.



# CONCLUSION

The Minesweeper project successfully implemented core game mechanics, leveraging fundamental data structures and design patterns to create an interactive and engaging experience. While the game delivers a functional and enjoyable user interface, further enhancements, such as improved performance optimization and additional features, can elevate the overall experience. This project provided valuable insights into practical problem-solving and software development principles.

# REFERENCES

## Tutorial

Kenny Yip Coding. (2023, August 14). Code Minesweeper in Java [Video]. YouTube. <https://www.youtube.com/watch?v=5VrMVSDjeso>