Minesweeper Game – Complete Project Report

Abstract

This report presents the full development and analysis of an advanced Minesweeper game built using Java and JavaFX. The system features multiple Data Structures and Algorithms (DSA) including BFS-based flood-fill, stack-driven undo/redo, priority queue ranking, and deterministic AI inference. The architecture is structured across UI, service, model, and utility layers, ensuring modularity and extensibility. This document includes system requirements, architecture, algorithmic analysis, GUI overview, testing strategy, evaluation, and future enhancements.

1. Introduction

The primary objective of this project is to implement a feature-rich, efficient, and extensible version of the classic Minesweeper game while applying key DSA concepts. The system incorporates GUI interaction, an AI solver, undo/redo functionality, persistent score saving, and intelligent flooding of empty regions. This project demonstrates practical implementation of data structures, algorithms, and clean architectural design in a real-world software context.

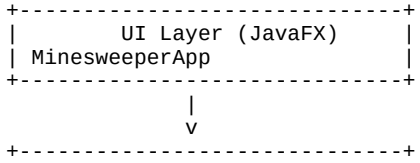
2. System Requirements

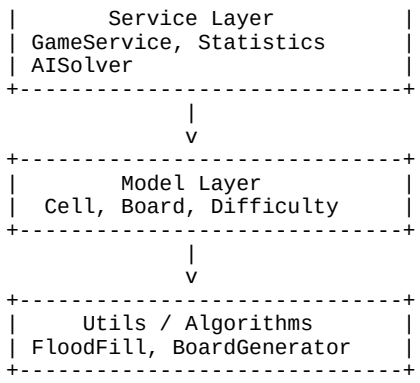
Functional Requirements: • Fully interactive Minesweeper gameplay through JavaFX GUI • Four difficulty modes: Easy, Medium, Hard, Extreme • Automatic flood-fill using BFS for zero-valued cells • Stack-based Undo and Redo • AI Solver capable of deterministic logical inference • Persistent Top-10 best times per difficulty • Real-time timer and mine counter • Tutorial window with gameplay instructions Non-Functional Requirements: • Implemented in Java 17 and JavaFX • High performance for boards up to Extreme difficulty • Modular architecture supporting future extensions • Responsive and intuitive user interface

3. System Architecture

The system is divided into four distinct layers: UI Layer, Service Layer, Model Layer, and Utility Layer. This separation ensures clarity, testability, and maintainability.

Architecture Diagram





4. Data Structures and Algorithms

The system uses multiple core data structures including 2D arrays for the board, queues for BFS, stacks for undo/redo, and priority queues for top-time rankings. Algorithms include deterministic AI logic and BFS-driven flood-fill.

Algorithms

4.1 BFS Flood-Fill Expansion

Flood-fill is implemented using Breadth-First Search to expand empty (zero-value) regions safely and efficiently.

```
Queue<int[]> q = new LinkedList<>();
q.offer(new int[]{r,c});
while (!q.isEmpty()) {
    int[] p = q.poll();
    for (each neighbor) {
        if (not visited and isZero) q.offer(neighbor);
        reveal(neighbor);
    }
}
```

4.2 Undo/Redo (Two-Stack Approach)

The undo/redo mechanism uses two stacks: undoStack and redoStack. Each move is stored as a GameAction containing previous and next states, implementing the Command Pattern.

```
undoStack.push(action);
redoStack.clear();

action = undoStack.pop();
applyPreviousState(action);
redoStack.push(action);
```

4.3 AI Solver (Deterministic Logic)

The AI evaluates number cells and determines whether surrounding hidden cells are safe or mines based on logical inference. If blocked, it performs a safe random guess.

```
if (hiddenCount == number - flaggedCount) hidden are mines;
if (flaggedCount == number) hidden are safe;
```

4.4 Persistent Ranking System

The ranking system uses PriorityQueue configured as a max-heap to store the best 10 completion times for each difficulty. Results are serialized to a local file for persistence.

5. GUI Design

The GUI is developed using JavaFX, featuring interactive board tiles, toolbar controls, and pop-up windows for tutorials and rankings. The layout uses BorderPane, GridPane, HBox, and VBox containers.

GUI Overview

```
[Difficulty] [New Game] [AI Move] [Undo] [Redo] [Help] [Best Times]
-------------------------------------------------------------------
| 1 | 1 | ■ | 2 | ■ | ... |
| 0 | 0 | 1 | 1 | 1 | ... |
-------------------------------------------------------------------
Mines left: X                     Time: Ys
```

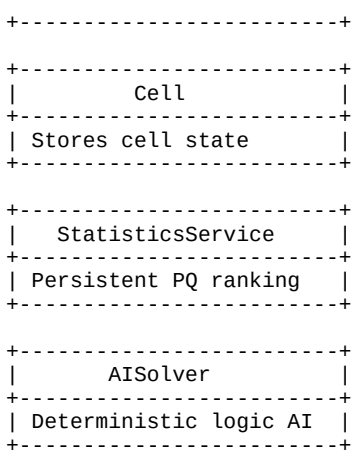
6. Classes Overview

The system consists of well-organized classes under model, service, ui, and utils packages. • model: Cell, Board, Difficulty, GameState • service: GameService, StatisticsService, AISolver • ui: MinesweeperApp • utils: FloodFill, BoardGenerator

Class Diagram

```
+-------------------------+
|       GameService       |
+-------------------------+
| Controls core logic     |
| Undo/Redo, reveal, flag |
+-------------------------+

+-------------------------+
|        Board            |
+-------------------------+
| Holds Cell[][] grid     |
```



7. Testing Strategy

Testing includes: • Unit tests for reveal, flag toggle, flood-fill, undo/redo • AI logic verification on controlled boards • Manual GUI testing on all modes • Persistence tests ensuring saved rankings are correctly loaded • Stress tests with 50+ undo/redo operations

8. Conclusion

This project successfully integrates DSA principles into a real interactive game. Key achievements include a modular architecture, efficient BFS expansion, strong undo/redo system, deterministic AI, and persistent ranking. The project demonstrates how theoretical algorithms can be used to build complete and engaging applications.

Evaluation

Performance is stable across all difficulty modes. BFS ensures fast region expansion. Undo/Redo maintains accurate state history. The AI solver is effective for deterministic logic, although ambiguous scenarios may require guessing. The GUI is intuitive and responsive.

Future Work

Planned future enhancements include: • Probabilistic AI inference • Custom board sizes • Multiplayer / online leaderboard • Enhanced graphics, animations, and themes • Mobile version porting

References

1. Oracle Java Documentation – https://docs.oracle.com 2. JavaFX Documentation – https://openjfx.io 3. Research on Minesweeper logic solving 4. Standard Data Structures & Algorithms textbooks 5. Design Patterns (GoF)