Minesweeper System Development

Course: EECS 581 Software Engineering II, Fall 2025
Professor: Hossein Saiedian

Contributors:

- Nick Grieco
- Zack Corbin
- Saurav Renju
- Muhammad Abdullah
- Maren Proplesch
- Ibrahim Muhammad

Overview

This project implements **Minesweeper**, a classic single-player puzzle game. Players uncover cells on a 10×10 grid to reveal numbers indicating adjacent mines while avoiding detonating any bombs. Flagging allows players to mark suspected mine positions, with victory achieved by uncovering all safe cells.

The implementation is written in **Python** using the **Pygame** library, with a graphical user interface and simple controls.

Features

- 10×10 grid with columns **A–J** and rows **1–10**
- **16 mines** placed randomly at game start (user-configurable between 10–20)

- **Safe first click**: first clicked cell (and surrounding area) is guaranteed to be mine-free
- Recursive uncovering of cells with zero adjacent mines
- Right-click flagging and unflagging of cells
- Display of:
    - Remaining mine count
    - Timer
    - Current game state (Playing, Game Over, Victory)
- Popup window for **Game Over** and **Victory**, with Play Again/Quit options

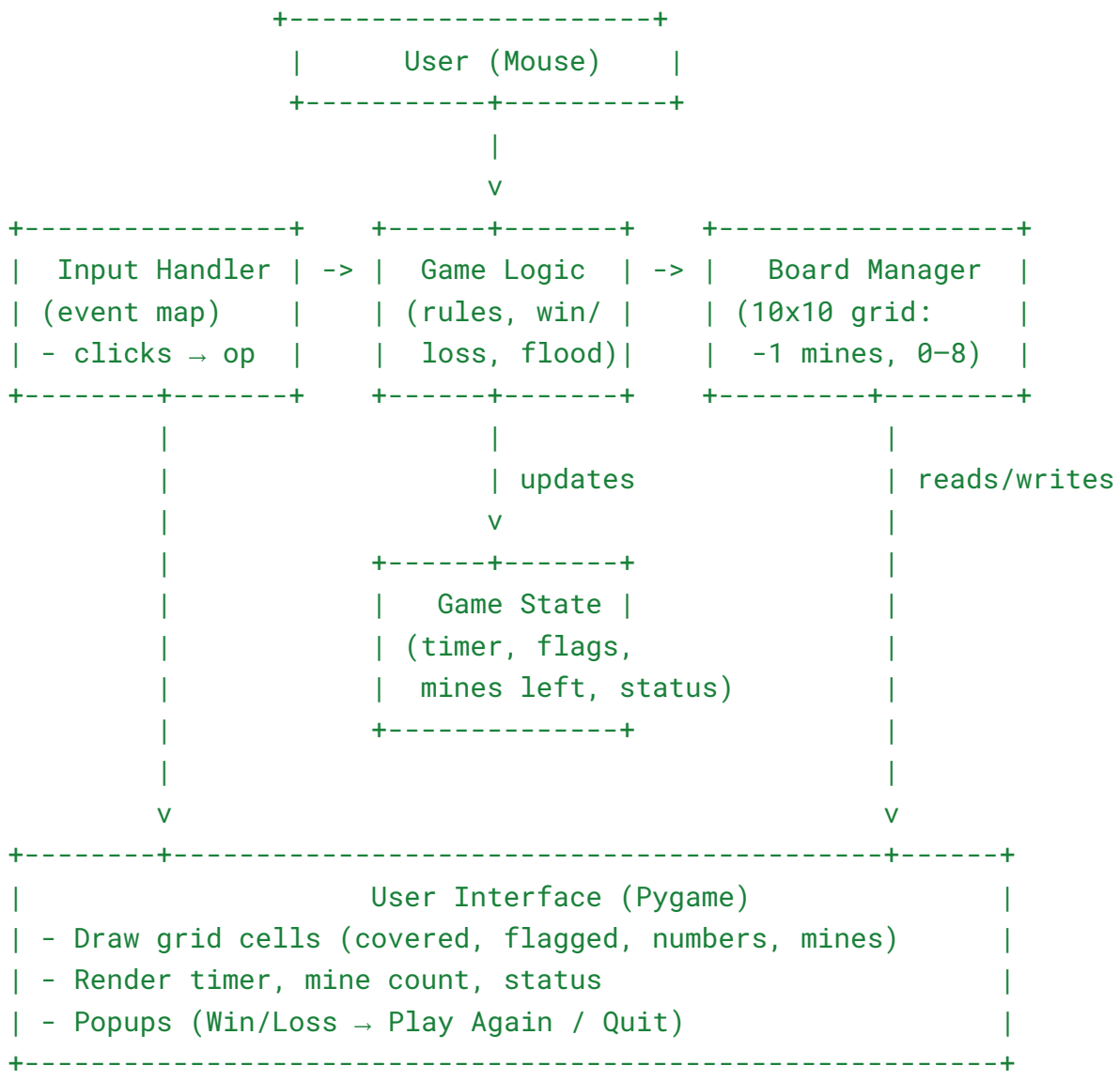
---

System Architecture

Components

- **Board Manager**: Manages the 10×10 grid as a 2D array, storing cell states (covered, flagged, uncovered, mine).
- **Game Logic**: Handles rules such as bomb placement, safe-start guarantee, uncovering logic, recursive flood fill, and win/loss detection.
- **User Interface**: Uses Pygame to render the board, flags, numbers, mines, and popup messages.
- **Input Handler**: Maps user mouse clicks to uncover/flag actions, communicates updates to the Game Logic, and triggers UI updates.

Data Flow Diagram



Key Data Structures

- **2D list (10×10 grid)**: stores each cell's state
    - `-1` = mine
    - `0–8` = number of adjacent mines
- **Sets**:

    - `revealed` = uncovered cells
    - `flagged` = flagged cells
- **Game state object**: tracks mine count, flags remaining, and win/loss state

---

Assumptions

- Fixed **10×10 grid size**
- **User-specified mine count** (between 10 and 20) at game start
- Written in **Python with Pygame**
- Runs locally in terminal or inside an IDE (e.g., VS Code)

Installation & Running

1. Install Python (>= 3.10 recommended).

Install dependencies:

```
pip install pygame
```

2.

Run the game:

```
python main.py
```

3. or open the project in **VS Code** and run `main.py`.

Person-Hours

Estimated Hours (per team member):

Zack Corbin — 1 hrs
Nick Grieco — 1 hrs
Saurav Renju — 1 hrs
Muhammad Abdullah — 1 hrs
Maren Proplesch — 1 hrs
Ibrahim Muhammad — 1 hrs

Methodology: We divided the project into four major components (Board Manager, Game Logic, Input Handler, User Interface). We estimated how long each would take based on past Python/Pygame assignments (about 1–2 hours each), then assigned responsibility across six members to balance workload. Debugging and documentation were included as part of the estimates.

Real Hours Spent

- Name Task Hours
- Zack Corbin Set up repo, initial grid logic 1
- Nick Grieco Bomb placement .5
- Saurav Renju UI drawing with Pygame 1
- Muhammad Abdullah Testing flood fill, bug fixing 1.5
- Maren Proplesch Documentation (README) 1
- Ibrahim Muhammad Popup/game-over logic 1

Grading Artifacts

- **Working Product Demonstration (40 pts):** Complete feature set, intuitive UI, stability tested.

- **System Documentation (40 pts):** Architecture overview (above), documented code with prologue comments.

- **Estimate of Person-Hours (10 pts):** Placeholder section provided above.

- **Actual Accounting of Person-Hours (10 pts):** Placeholder section provided above.
- **Peer Evaluation:** To be submitted individually via Canvas.

Source Attribution

- **Original code** primarily authored by the project team.
- Libraries used: Pygame
- External references: ChatGPT