EECS 581 — Project 1: Minesweeper Testing Documentation

Team 26

September 21, 2025

1 Testing Strategy, Cases, and Results

1.1 Scope & Objectives

Our goal was to verify correctness of the core rules and stability of the game loop, independent of any single UI implementation. We emphasized deterministic board setup, rule invariants (first–click safety, unique mines, legal flagging), and state transitions (`START` → `PLAYING` → `END::WIN/LOSS` → `START`).

1.2 Test Environment

- **Language/Runtime:** Python 3.12 (compatible 3.10–3.12).
- **Framework:** `unittest` for unit tests; ad–hoc scripts for integration checks.
- **Determinism:** When verifying mine placement or neighbor counts, tests use fixed coordinates via `BoardManager.place_unique_mines(...)` and/or a seeded RNG.
- **System Under Test (SUT):** `BoardManager`, `GameLogic`, `InputHandler`, and `User Interface` event loop (manual checks).

1.3 Unit Test Coverage

Board Initialization

- **T-BOARD-INIT-01** (*dimensions*) — Construct a board with $(r, c) = (10, 10)$; assert internal grid shape matches (10×10).

- **T-BOARD-INIT-02** (*mine count*) — Call `place_unique_mines(total_mines)` with $m \in [10, 20]$; assert exactly m cells have `is_mine=True`.
- **T-BOARD-INIT-03** (*deterministic placement*) — Provide a fixed set of coordinates (edges, corners, center); assert those cells are mined and all others are not.

Adjacency / Neighbor Counts

- **T-NEIGH-01** (*single mine*) — Place one mine and assert each neighbor increments `neighbor_count` to 1.
- **T-NEIGH-02** (*multi-mine*) — Place a pattern of adjacent mines; verify numbers (1–8) match expected counts at each cell.
- **T-NEIGH-03** (*clear*) — Clear mines via `clear_mines_and_counts`; assert all `neighbor_count` reset to 0.

Reveal Behavior

- **T-REVEAL-01** (*numbered cell*) — Uncover a nonzero cell; assert only that cell flips to uncovered.
- **T-REVEAL-02** (*zero flood*) — Uncover a 0 cell; assert flood/cascade reveals contiguous zero–region and its perimeter numbers.
- **T-REVEAL-03** (*first–click safe*) — On the very first uncover, if the target is mined, ensure logic relocates the mine (or places mines excluding that cell); assert no loss is triggered and the clicked cell is safe.

Win/Loss Detection

- **T-END-01** (*win*) — Reveal all non–mine cells; assert `GameState = EndWin`.
- **T-END-02** (*loss*) — Uncover any mined cell (post–first–click); assert `GameState = EndLose`.

Flags & Counters

- **T-FLAG-01** (*toggle*) — Call `toggle_flagged_cell` twice; assert `flagged` toggles and `flags_remaining` updates accordingly.

- **T-FLAG-02** (*no flags left*) — With `flags_remaining = 0`, attempt to place another flag; assert no additional flags are set.
- **T-FLAG-03** (*covered vs flagged*) — Ensure a flagged cell is never uncovered by a left–click until unflagged.

Input Handling

- **T-INPUT-01** (*mines entry range*) — In `START`, feed keyboard digits; assert acceptance only for 10–20 inclusive; otherwise error response.
- **T-INPUT-02** (*state gating*) — Ensure board clicks are ignored when not in `PLAYING`; ensure mine–count entry is ignored when not in `START`.
- **T-INPUT-03** (*click types*) — Left–click uncovers; right–click toggles flag; assert corresponding state and counters.

Reset / Restart

- **T-RESET-01** (*end–screen restart*) — In `EndWin`/`EndLose`, press `R`; assert transition to `START`, cleared board, and fresh mine–entry prompt.

1.4 Representative Procedures

P1 — Zero Flood Fill

1. Place mines such that cell (r, c) has `neighbor_count = 0`.
2. Call `uncover_cell(r,c)`.
3. **Expected:** All connected zero cells become uncovered; perimeter numbered cells adjacent to that region are uncovered; non–adjacent covered cells remain covered.

P2 — First–Click Safety

1. Initialize game; ensure no prior reveals.
2. Click a cell that is mined under a naive placement.
3. **Expected:** The first click never triggers loss; either mines are placed excluding (r, c) or the original mine is relocated by logic; state remains `PLAYING`.

P3 — Win Detection

1. Arrange m mines and uncover all $100 - m$ safe cells (scripted).
2. **Expected:** `GameState = EndWin`; subsequent press `R` returns to `START`.

P4 — Flag Counter Guard

1. Set `total_mines = k`; place k flags.
2. Attempt to place an additional (k+1)th flag.
3. **Expected:** No change to any cell's `flagged` state; `flags_remaining` remains 0.

1.5 Results Summary

- **Pass:** Board dimensions, unique mine counts, neighbor numbers (1–8), zero–cell flood fill, first–click safety, win/loss detection, flag toggle & count bounds, state–aware input gating, end–screen restart via `R`.
- **Manual UI checks:** Mouse left/right clicks; mine–count entry (10–20) on `START`; on–screen *Mines Remaining* and end–state message visibility; `R` to restart returns to `START`.

1.6 Notes & Observations

- *Mines Remaining* displays immediately after the first flag is placed; it reflects flags placed (candidate mines), not confirmed mines.
- There is no dedicated GUI button for *Reset*; restart is via keyboard (`R`) at end–screen.
- Status indicator is visible during `PLAYING` and on `END::WIN/LOSS`.
- Input range for mine count is validated strictly to $[10, 20]$ at `START`.

1.7 Future Test Enhancements

- Automate UI–level tests (headless) by driving `InputHandler` with synthetic events and asserting rendered state via a thin adapter.
- Add property–based tests for flood fill invariants (e.g., uncovered zero–region forms a connected set; perimeter numbers match counts).