System Architecture Form

1) Project Overview

Project Name: PySweeper

Version: 1.0.0

Scrum Master: Josh Dwoskin

Developers: Asa Maker, Zach Sevart, Ebraheem AlAamer

Product Owner: Brandon Dodge

Creation Date: September 1, 2025

Last Edited: September 19, 2025

Purpose:
A desktop Minesweeper game written in Python using Pygame. Implements a 10×10 board, user-selected mine count (10–20) entered via console, first-click-safe placement, recursive reveal of empty regions, timer, flags remaining, restart button, and a simple “last game result” display.

Primary Inputs/Outputs:

- **Inputs:**
  - Mouse left-click: reveal a cell
  - Mouse right-click: toggle a flag
  - Console input at startup: number of mines (10–20)
  - Restart button (mouse click)
- **Outputs:**
  - Pygame window (grid, labels A–J and 1–10, header HUD)

  - On-screen status: “Playing…”, “Victory!”, or “Game Over”

  - Timer (capped to 999) and mines remaining (three-digit counter)

2) High-Level Architecture

Architecture Style: Single-process, local, event-driven Pygame application.

Major Components:

- **Rendering & Input Module:** Pygame window creation, event polling, drawing header, labels, and cells (implemented within Game + Cell.draw).

- **Game Coordination Module:** Game loop, state transitions, win/loss detection, restart, timing, flag count (class Game).

- **Board & Rules Module:** Grid management, mine placement (first-click safe), adjacent count computation, recursive revealing, reveal-all-mines on loss (class Board and class Cell).

External Libraries / Assets:

- **Pygame** for graphics, windowing, fonts, timing, and input.

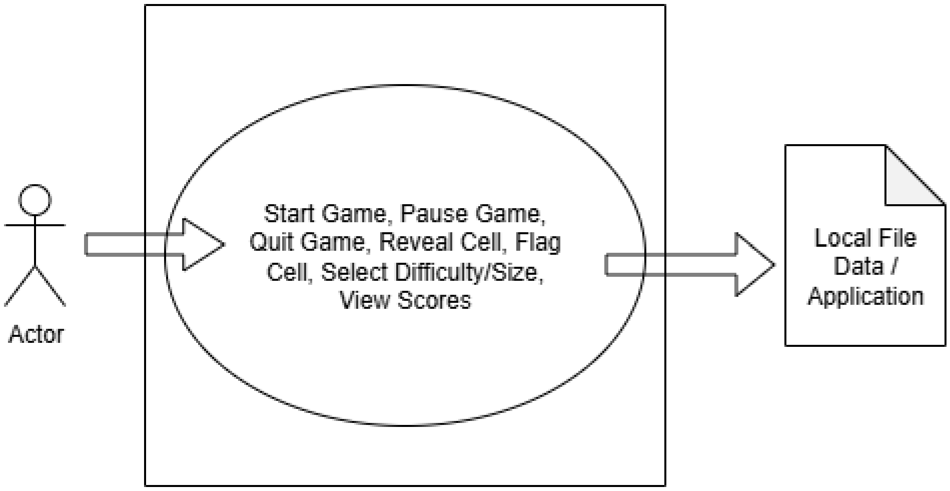
- **Image assets** loaded from images/ (cell states, digits).

APIs/Services: None (fully offline/local).

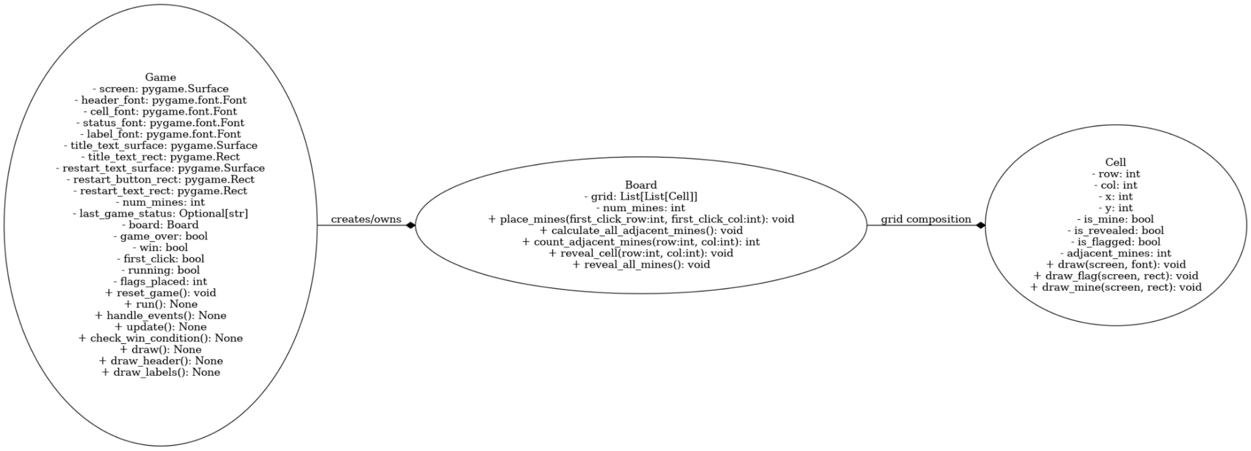
Persistence: None to disk. The prior game result (“Victory!” or “Loss”) is kept in memory only for display until the next reset.

Diagrams:

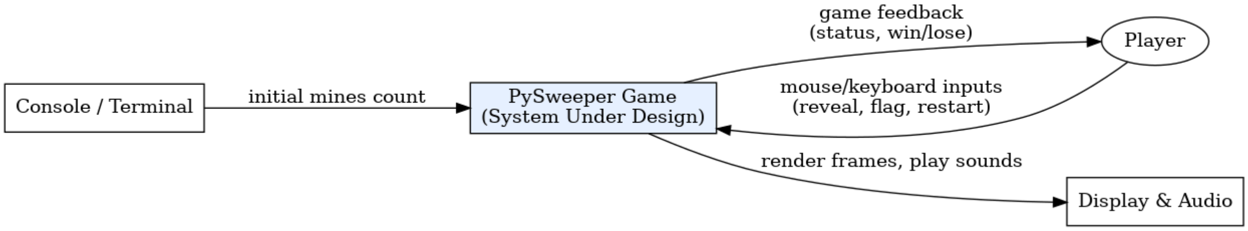
- **Use Case:**


Actor
Start Game, Pause Game, Quit Game, Reveal Cell, Flag Cell, Select Difficulty/Size, View Scores
Local File Data / Application

- **Class:**


Game
- screen: pygame.Surface
- header_font: pygame.font.Font
- cell_font: pygame.font.Font
- status_font: pygame.font.Font
- label_font: pygame.font.Font
- title_text_surface: pygame.Surface
- title_text_rect: pygame.Rect
- restart_text_surface: pygame.Surface
- restart_button_rect: pygame.Rect
- restart_text_rect: pygame.Rect
- num_mines: int
- last_game_status: Optional[str]
- board: Board
- game_over: bool
- win: bool
- first_click: bool
- running: bool
- flags_placed: int
+ reset_game(): void
+ run(): None
+ handle_events(): None
+ update(): None
+ check_win_condition(): None
+ draw(): None
+ draw_header(): None
+ draw_labels(): None
creates/owns
Board
- grid: List[List[Cell]]
- num_mines: int
+ place_mines(first_click_row:int, first_click_col:int): void
+ calculate_all_adjacent_mines(): void
+ count_adjacent_mines(row:int, col:int): int
+ reveal_cell(row:int, col:int): void
+ reveal_all_mines(): void
grid composition
Cell
- row: int
- col: int
- x: int
- y: int
- is_mine: bool
- is_revealed: bool
- is_flagged: bool
- adjacent_mines: int
+ draw(screen, font): void
+ draw_flag(screen, rect): void
+ draw_mine(screen, rect): void

- **System Context:**



3) Component Descriptions

A) Game Coordination Module:

- **Key Responsibilities:**
  - Initialize Pygame, fonts, and main window
  - Maintain game state: running/over, win flag, first-click gating, flags placed, elapsed time, last game status
  - Process events (mouse clicks, restart button)
  - Drive update loop and trigger renders
- **Notable Methods:**
  - run() – main loop
  - handle_events() – input handling (quit, restart button, left/right clicks mapped to board actions)
  - update() – win check, finalize elapsed time on end
  - check_win_condition() – all non-mine cells revealed → victory
  - draw() / draw_header() / draw_labels() – render HUD, labels, and grid
  - reset_game() – initialize a new Board and reset state; caches previous game result in last_game_status

B) Board & Rules Module

- **Board Responsibilities:**
  - Allocate a 10×10 grid of Cell objects
  - Place mines after the first click with a 3×3 “safe zone” centered on the first clicked cell
  - Compute adjacent mine counts for all non-mine cells
  - Recursively reveal empty regions (classic flood-fill behavior)
  - Reveal all mines on loss
- **Board Methods (as in code):**
  - place_mines(first_click_row, first_click_col) – random mine placement excluding safe zone
  - calculate_all_adjacent_mines() – set adjacent_mines on every non-mine cell
  - count_adjacent_mines(row, col) – helper for neighbor counts
  - reveal_cell(row, col) – reveal cell, recursively expand when adjacent_mines == 0
  - reveal_all_mines() – show all mines (on loss)
- **Cell Responsibilities/State:**
  - Coordinates (row/col) and on-screen position (x, y) with label offsets
  - is_mine, is_revealed, is_flagged, adjacent_mines
  - draw(screen, font) uses pre-scaled images for covered, revealed, flags, numbers, and mines

4) Data Flow

1. **Startup:**
   - Console prompts for mine count (10–20).
   - Game.reset_game() creates a new Board with no mines placed yet.

2. **First Click:**
    - On first left-click inside the grid, Board.place_mines(row, col) runs, excluding the clicked cell and its neighbors from mine placement.
    - Timer starts.
3. **Reveals & Flags:**
    - Left-click on a covered cell reveals it.
        - If it’s a mine → game_over = True, win = False, and reveal_all_mines()
        - If it’s empty (0 adjacents) → recursive expansion via reveal_cell()
    - Right-click toggles flag on a covered cell; flags placed are bounded by num_mines.
4. **Win Check:**
    - Each update, check_win_condition() verifies whether all non-mine cells are revealed. If yes → win = True, game_over = True.
5. **Rendering:**
    - Header shows title, status, “mines remaining” (3 digits), and timer (3 digits).
    - Grid shows images for covered/revealed/flagged/numbered/mine cells.
    - Column labels A–J; row labels 1–10.
    - Restart button is clickable; if used, previous result is shown as “Last Game: Victory!/Loss”.

Note:

- No keyboard shortcuts or chord (mass-reveal by number) exist in the provided code.
- No file I/O, no config, no disk persistence.

5) Deployment

- **Environment:** Local desktop, Python 3.11+ with Pygame 2.x installed.
- **Packaging:** Tools like PyInstaller can be used to create standalone builds for Windows/macOS/Linux. (Not required by code.)
- **Network:** None (offline).
- **Assets:** Ensure images/ directory is present with expected filenames; window dimensions depend on CELL_SIZE, labels, and header constants.

6) Non-Functional Requirements

- **Performance:**
  - Responsive reveal and drawing at interactive frame rates on typical desktops
  - Mine placement and neighbor counting complete quickly on a 10×10 grid
- **Startup:**
  - Window opens promptly after entering a valid mine count
- **Reliability/Robustness:**
  - Graceful end states (win/loss) and restart without app crash
  - Safe handling of clicks outside grid or on labels/header
- **Security/Privacy:**
  - No networking; no external data exchange
- **Maintainability:**
  - Clear class separation (Game, Board, Cell) with straightforward responsibilities
- **Accessibility:**
  - Mouse-only interaction supported (left/right click); labels A–J and 1–10 for orientation

7) Risks & Assumptions

Risks (with mitigations):

- **Asset mismatch (missing images):** App will error upon load. → Include asset checks or fallbacks if needed.
- **Event handling edge cases:** Mis-clicks on borders/labels. → Bounds checks already in handle_events().
- **Timer accuracy:** Uses pygame.time.get_ticks(); capped at 999 on HUD.

Assumptions:

- Standard mouse is available (supports left and right click).
- Display resolution accommodates the window (SCREEN_WIDTH × SCREEN_HEIGHT).
- Pygame and required images are installed/present.

8) Testing Strategy

- **Unit Tests (logic-level):**
  - First-click safety: no mine in clicked cell or its 8 neighbors
  - Exact mine count equals num_mines
  - Adjacent counts correct for all non-mine cells
  - reveal_cell() expands correctly for 0-adjacent areas and respects bounds/visited state
  - Win condition triggers only when all non-mine cells are revealed
- **Integration Tests (event + render loop):**
  - Left/right clicks update flags/reveals as expected
  - Restart button resets board and shows prior result in header

    - Timer starts on first reveal and freezes on game end

- **Smoke Tests:**
    - Various mine counts (10–20)
    - Random seeds to spot-check distribution (optional dev harness)

9) Error Handling

- **Input Validation:** Console mine count must be an integer in [10, 20]; reprompts on errors.
- **Runtime:** Graceful quit on window close; ignores clicks outside grid; avoids revealing flagged cells; prevents flag count from exceeding num_mines.
- **No logging subsystem** and **no disk recovery** paths in the current code.

10) State Definitions

- **Game Lifecycle:**
    - ready (before first click) → running (after first click & mine placement) → won or lost → ready (on restart)
- **Cell Lifecycle:**
    - covered ↔ flagged (toggle via right-click)
    - covered → revealed (left-click; recursive expansion for zero-adjacent cells)

11) Board & Mines

- **Layout:** Fixed 10×10 grid
- **Mines:** User-specified 10–20; positions sampled uniformly from cells not in the first-click safe zone

- **Counts:** Numbers 1–8 reflect adjacent mines; zero triggers recursive expansion

12) References

- Pygame documentation
- Dr. Saiedian EECS Slides
- Project source (this implementation) and bundled image assets
- **Terminology**
    - **Flood Fill:** Recursive reveal from a zero-adjacent cell through connected zero-adjacent regions and their numbered borders.
    - **Cell:** Single tile; may be covered, flagged, revealed (empty/numbered/mine).
    - **Board:** 10×10 grid of Cells managed by Board.
    - **Pygame:** Library for windowing, input, timing, and drawing.