System Architecture Form

1. Project Overview

Project Name: PySweeper
Version: 1.0.0
Scrum Master: Josh Dwoskin
Developer: Asa Maker, Zach Sevart, Ebraheem AlAamer
Product Owner: Brandon Dodge
Creation Date: September 1, 2025
Last Edited: September 17, 2025

Purpose:
A desktop Minesweeper game implemented in Python, providing classic gameplay (left-click reveal, right-click flag), 10x10 board size and 10-20 mines, timer, and recent-score tracking with a clean, responsive UI.

2. High-Level Architecture

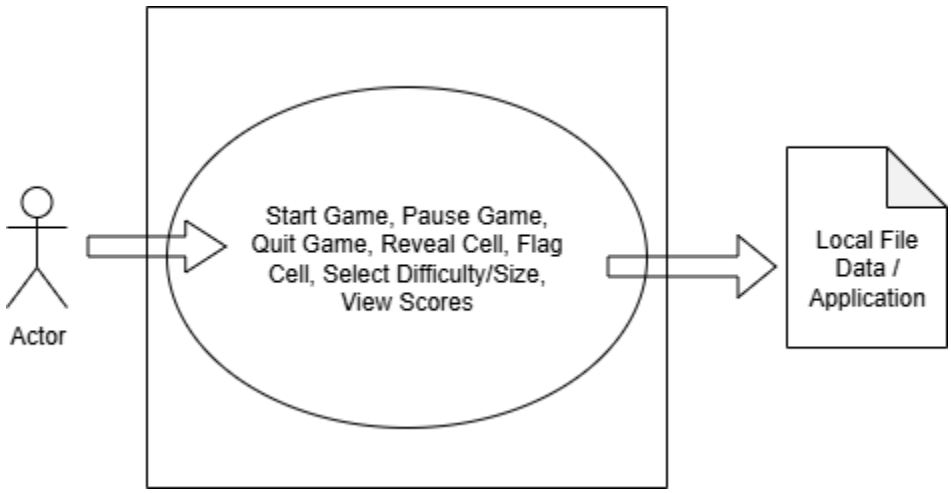
Architecture Style: PyGame-based application

Major Components:

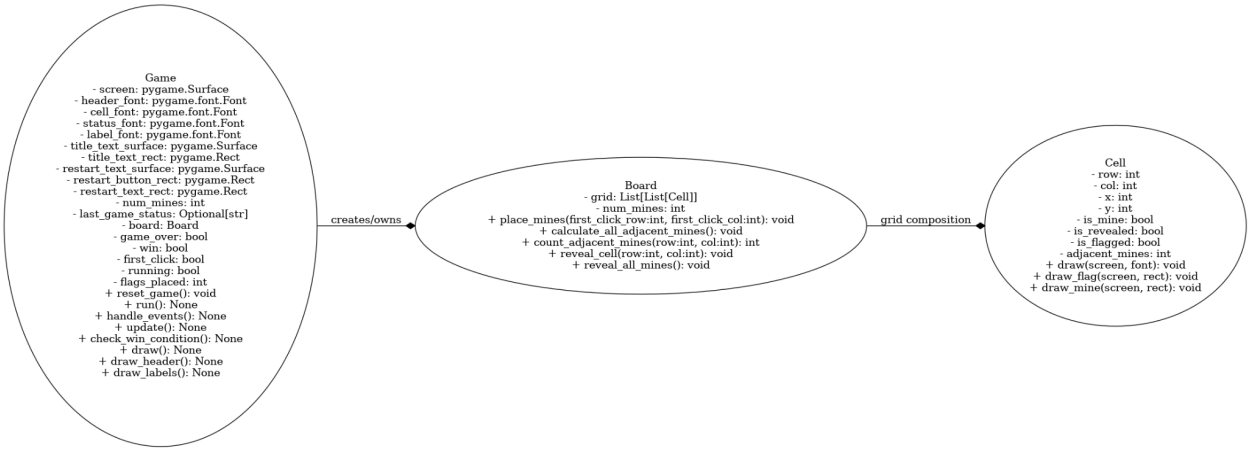
- **Frontend (View):** PyGame UI (window, board rendering, HUD)
- **Backend (Model & Controller):** Game logic (board generation, flood-fill, win/lose detection), settings & high-score service
- **Persistence:** Local recent score stored with PyGame
- **APIs / Services:** None (offline/local only)
- **Third-Party Integrations:** PyGame for rendering

Diagrams:

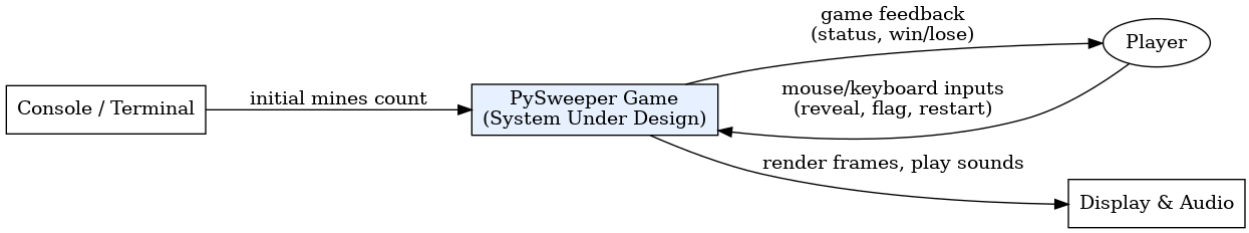
Use Case Diagram:



Class Diagram:



System Context Diagram:



3. Component Description

Frontend (View)

- **Tech:** Python 3.11+, PyGame 2.x
- **Responsibilities:** Draw grid, numbers/bombs/flags, HUD (timer, mines left), buttons, handle animations & sounds.
- **Interfaces:**
    - render_tiles(coords)
    - render_hud(mines_left, time, state)
    - play_sfx(kind)

Backend (Model & Controller)

- **Language/Framework:** Python (stdlib)
- **Model:**
    - Cell (state, mine, adj_mines)
    - Board (grid, mine placement, neighbor counts, flood-fill, chord)
    - Game (state machine: ready, running, won, lost; timer; flag count)
- **Controller:** Maps input events → Game methods.
- **Interfaces:**
    - reveal(x,y)
    - toggle_flag(x,y)
    - chord(x,y)
    - new_game(width, height, mines)

Persistence

- **Storage:** PyGame for recent score
- **Schema:**
    - PyGame { "Number of Mines":"10" }
- **Interfaces:**
    - Start()
    - Set.Mines()

4. Data Flow

Input Sources: Mouse (reveal, flag, chord), keyboard (restart, difficulty hotkeys).
Processing: Controller interprets input → Model updates (board, timer, win/lose) → Persistence saves if high score.
Output: Render frames, HUD updates, play sounds, write JSON.

Special Sequences:

- **First-click safe:** Mines placed after first click; clicked cell + neighbors excluded.
- **Chord:** If the number cell has N, and N flags around, reveal all neighbors. Incorrect flags may cause loss.

5. Deployment Architecture

- **Environments:** Dev (Python), Release (standalone).
- **Infrastructure:** Local desktop only.
- **Packaging:** PyInstaller for Windows, macOS, Linux.
- **Code Signing:** Windows Authenticode, macOS Developer ID.
- **Config Migration:** If schema version mismatch, respond error.

6. Non-Functional Requirements

- **Performance:** ≤50 ms board gen (Expert), ≤10 ms reveal/chord (95th percentile).
- **Startup:** ≤500 ms window open.
- **Availability:** Stable sessions; error handling with recovery.
- **Reliability:** Reset defaults if config corrupt.
- **Security:** Sandbox file writes; no network.
- **Maintainability:** 70%+ test coverage; docstrings.
- **Accessibility:** Mouse-only play

7. Risks & Assumptions

Risks:

- System lag → *Unit tests, optimized rendering*
- Wrong neighbor counts → *Unit tests & invariant checks*
- Timer drift if tied to FPS → *Use real-time counter*
- Save corruption → *Atomic writes + backups*
- RNG fairness complaints → *Optional dev-mode seed*

Assumptions:

- Standard mouse input
- ≥480×480 screen
- PyGame installed
- Local-only play

8. Testing Strategy

- **Unit:** Placement excludes first click; counts correct; flood-fill edges; chord correctness.
- **Property-based:** Invariants hold (mine count, adj counts).
- **Integration:** Timed clicks, flag/unflag, pause/resume.
- **Regression:** Fixed RNG seeds.
- **Performance:** Reveal/chord microbenchmarks; frame budget checks.

9. Logging & Error Handling

- **Local logs only:** pysweeper.log in config dir.
- Levels: INFO (startup, shutdown), WARN (recoverable JSON issues), ERROR (tracebacks).
- **Error Recovery:** Backup corrupted file + restore defaults.

10. State Machines

Game State:
ready → running → (won|lost) → ready (on new game)

Cell State:
covered ⇄ flagged, covered → revealed

11. Custom (Number of Mines)

- **Layout:** 10x10 Grid, 10-20 mines

12. References

- GitHub repo
- PyGame documentation
- Terminology:

- **Flood Fill**: An algorithm that spreads from a starting cell to reveal all adjacent empty cells and their borders, similar to a paint bucket tool.

- **Cell**: A single tile in the Minesweeper grid that can be empty, numbered, flagged, or contain a mine.

- **Board**: The entire Minesweeper grid is made up of cells, which defines the play area.

- **Chord**: A player action that uncovers all adjacent cells of a numbered tile when the correct number of flags are placed around it.

- **PyGame**: A Python library used to build games by handling graphics, sound, and input.

- **Persistence**: The ability to save and reload data (e.g., settings or scores) so it remains across sessions.

- **RNG** (Random Number Generator): A function that produces random values, used here for unpredictable mine placement.

- **Scalability**: The system's capacity to handle larger boards or more complex tasks without performance loss.

- **Deployment**: The process of packaging and delivering the game so it can be installed and run on user machines.