Project 1: Minesweeper

System Documentation

Person-Hours Estimate (10 Points): Detail your methodology for estimated hours

Methodology: Use Case Points

Beginning & End: (3 steps)

Complexity: Simple

Weight: 5 points

Core Gameplay: (6 steps)

Complexity: Average

Weight: 10 points

User Interface: (5 steps)

Complexity: Average

Weight: 10 points

Total Unadjusted Use Case Weight: 25 points

Actors

User: A person playing the game

Complexity: Complex

Weight: 3 points

Total Unadjusted Actor Weight: 3 points

Unadjusted Use Case Points

25 +3 = 28 points

Final Effort

28 Use Case Points x 1 person hour per UCP = 28 person-hours

Actual Person-Hours (10 Points): Day-by-day accounting of each member's hours (excluding EECS 581 lectures)

Manu:

(9-1-2025) Spent 45 minutes studying the assignment rubric for requirements engineering purposes.

(9-9-2025) Spent 1 hour implementing unflagging capability and game restart functionality.

(9-16-2025) Spent 30 minutes adding in-code comments in main.py.

(9-17-2025) Spent 30 minutes working on system documentation.

(9-21-2025) Spent 45 minutes finalizing the system documentation and prologue comments.

Jackson:

(9-2-2025) Spent 30 minutes researching game rules and possible technologies.

(9-9-2025) Spent 1.5 hours implementing initial functionality and creating requirements.txt

(9-10-2025) Spent 45 minutes implementing "First click will always be valid" into the gameplay loop

(9-17-2025) Spent 15 minutes updating readme with names and markdown formatting.

(9-21-2025) Spent 15 minutes finalizing readme, 45 minutes merging and finalizing github repository.

Riley:

(9-3-2025) Spent 30 minutes researching pygame.

(9-10-2025) Spent 1.5 hours creating sprites (extracurricular) and

(9-16-2025) Spent 1.5 hours implementing UI within the game.

(9-17-2025) Spent 45 minutes adding Mine Counter and Game Status to the program.

(9-21-2025) Spent 15 minutes doing a final pass over the repository.

Evans:

(9-2-2025) Spent 1.5 hours researching the game rules and requirements, and how we can use PyGame to implement the game.

(9-7-2025) Spent 1.5 hours creating a program that prompts the user for the number of mines and a game window that has a status message, instructions for which buttons to use, and the count for mines and flags.

(9-16-2025) Spent 45 minutes adding the slider bar and buttons to the game.

(9-18-2025) Spent 30 minutes adding more options to either quit the game or restart during gameplay.

(9-20-2025) Spent 1.3 hours to integrate the different programs developed by team members to create the final product.

Cole:

(9-1-2025) Spent 30 minutes working on Person-Hours Estimate.

(9-8-2025) Spent 45 minutes working on System Architecture Documentation (System Components)

(9-14-2025) Spent 30 minutes working on System Architecture Documentation(Key Data Structures)

(9-16-2025) Spent 30 minutes identifying and fixing the flagged cell click prevention bug - prevented players from accidentally revealing flagged cells.

(9-18-2025) Spent 30 minutes working on System Architecture Documentation (Data Flow)

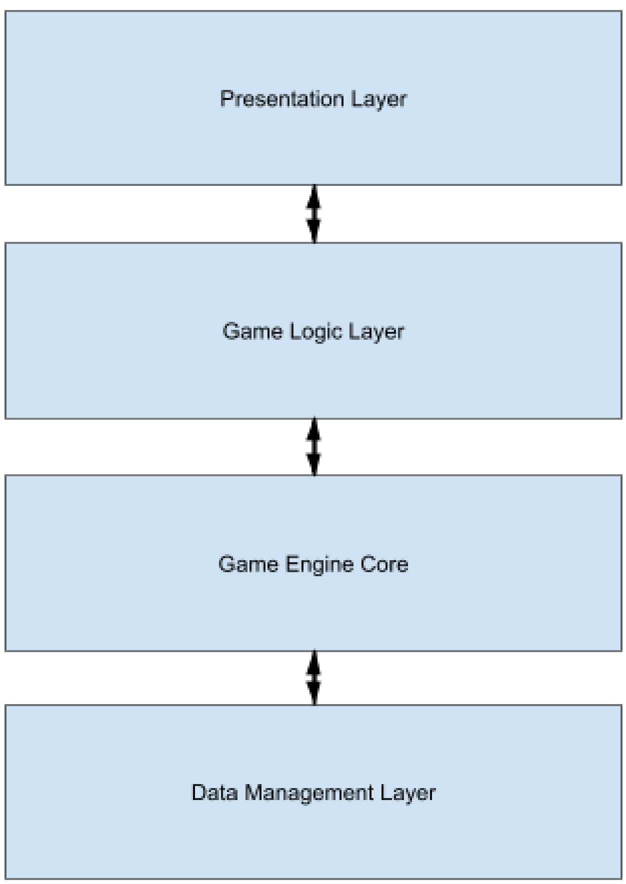
System Architecture Overview

The Minesweeper game project is implemented using Python and Pygame. The system uses a modular architecture with separation between game logic, state management, and presentation layers. The application operates on a game loop pattern, continuously processing user input events and updating the display accordingly.

Key Technologies:

- Python 3.x
- Numpy
- Pygame

System Components



Data Management Layer

- Board Data Structure: 2D numpy array stores mine locations and adjacent counts
- State Tracking Arrays: Boolean arrays for revealed and flagged cell states
- Game State Variables: Control flags for game flow management

Presentation Layer

- Sprite Management: Loads and manages visual assets
- Rendering Engine: Draws game board, cells, and visual setup
- UI Elements: Grid layout and visual indicators

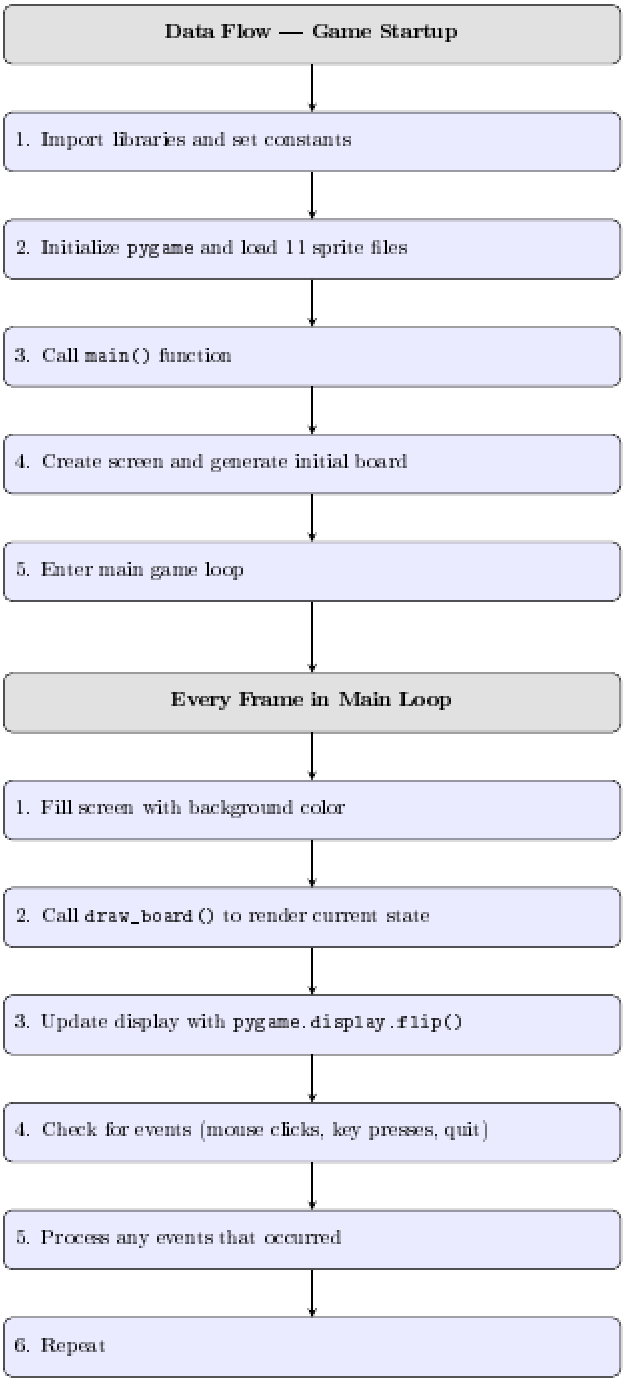
Game Logic Layer

- Board Generator: Defines the board size, creates randomized mine placement and calculates adjacent mine counts
- Game State Manager: Tracks game progression, win/loss conditions
- Action Processors: Handles reveal, flag, and restart operations

Game Engine Core

- Main Game Loop: Central control flow manages the application cycle
- Event Handler: Processes user input (mouse clicks, keyboard input)
- Display Manager: Handles screen rendering and visual updates

Data Flow



Game Startup

1. Import libraries and set constants
2. Initialize pygame and load 11 sprite files
3. Call main() function
4. Create screen and generate initial board
5. Enter main game loop

Every Frame in Main Loop

1. Fill screen with background color
2. Call draw_board() to render current state
3. Update display with pygame.display.flip()
4. Check for events (mouse clicks, key presses, quit)
5. Process any events that occurred
6. Repeat

When User Clicks

Left Click:
Mouse position → Convert to grid coordinates → Check if valid → Call reveal() → Update revealed array

Right Click:
Mouse position → Convert to grid coordinates → Call flag() → Toggle flagged array

'R' Key:
Call restart_game() → Generate new board → Reset all arrays

Key Data Structures

Primary Data Arrays

Board Array (board: np.ndarray[int])

Type: 2D NumPy integer array (GRID_SIZE × GRID_SIZE)
Purpose: Stores the core game state
Values:

-1: Mine location
0-8: Number of adjacent mines
Generated once per game, remains constant after creation

Revealed Array (revealed: np.ndarray[bool])

Type: 2D NumPy boolean array (GRID_SIZE × GRID_SIZE)
Purpose: Tracks which cells have been revealed by the player
Values:

True: Cell is revealed and visible
False: Cell is hidden from player

Flagged Array (flagged: np.ndarray[bool])

Type: 2D NumPy boolean array (GRID_SIZE × GRID_SIZE)
Purpose: Tracks which cells have been flagged by the player
Values:

True: Cell is flagged
False: Cell is not flagged

Configuration Constants

GRID_SIZE = 10 # Board dimensions (10×10 grid)
NUM_MINES = 10 # Total mines on board
CELL_SIZE = 40 # Pixel size of each cell
WIDTH = HEIGHT = 400 # Window dimensions

Game State Variables

start: bool: Indicates if it's the first click (prevents first-click mine hits)

game_over: bool: Tracks game completion state

running: bool: Controls main game loop execution

Sprite Dictionary

The system uses a mapping structure to associate numerical values with corresponding sprite assets:

```
num_sprite = {
   1: sprite_grid1,   # Cells with 1 adjacent mine
   2: sprite_grid2,   # Cells with 2 adjacent mines
   ….
   8: sprite_grid8,   # Cells with 8 adjacent mines
        }
```