Minesweeper System Architecture

1. Architectural Overview

The Minesweeper application follows a modular, object-oriented design built on the Model-View-Controller pattern. While not a strict MVC implementation, the project separates its concerns into distinct components:

- **Model:** The Board and Tile classes manage the game's core data and state.
- **Controller:** The GameLogic class handles all gameplay rules and state transitions.
- **View & Input:** The GameLogic class also serves as the User Interface and Input Handler for the terminal-based application.

This separation ensures that game rules and data management are independent of the user interface, making the project extensible. For example, if the Project 2 team wanted to create a graphical interface, they could simply replace the current terminal-based UI with a new one without altering the core Board or GameLogic components.

2. Components

Board Manager (Board.py and Tile.py)

The Board Manager is the foundation of the game, responsible for the representation of the Minesweeper grid.

- **Board (**Board.py**):** This class manages the 10x10 grid as a 2D array. It handles mine placement, initializes all tiles, and provides methods for interacting with the grid. It acts as the central data store for the game's state.
- **Tile (**Tile.py**):** This class represents a single cell on the grid. Each Tile object tracks its individual state, including whether it's a mine, if it has been revealed, if it's flagged, and the number of adjacent mines.

Game Logic (GameLogic.py)

The Game Logic component contains all the rules for the game's progression. It acts as the controller, taking user input and converting it into actions on the board.

- **Core Responsibilities:**
    - **Game Stat:** Tracks the overall status of the game ("playing", "win", "loss").
    - **Gameplay Rules:** Executes rules for uncovering a tile (game over if a mine is hit), toggling flags, and initiating the recursive reveal for empty cells.
    - **Win/Loss Detection:** Checks the board state to determine if all non-mine tiles have been uncovered (a win) or if a mine has been hit (a loss).

User Interface & Input Handler (within GameLogic.py**)**

In this terminal-based implementation, the User Interface and Input Handler are tightly coupled with the GameLogic component.

- **User Interface:** This part is responsible for rendering the game board and status indicators. It displays the 10x10 grid in the terminal using ASCII characters (#, F, 1-8, .,).
- **Input Handler:** This part processes raw user commands ("uncover A1", "flag B2"), validates them, and passes the parsed information (action, row, column) to the game logic.

3. Data Flow

The flow of data through the system is a simple, unidirectional loop. This ensures a predictable and easy to follow sequence of events for a given action.

1. **User Input:** The player enters a command (uncover C5, flag D7) into the terminal.
2. **Input Handler:** The GameLogic's handle_input() method receives the command, parses the action and coordinates, and validates the input format.
3. **Game Logic:** The validated input is used to call a method on the Board (e.g., board.show_tile(row, col) or board.flag_on_off(row, col)).
4. **Board State Update:** The Board's state changes based on the action. For example, a tile's is_revealed or flaggedattribute is updated.
5. **UI Update:** The GameLogic's display_board() method is called to re-render the board, reflecting the new state to the player. This loop continues until a win or loss condition is met.

4. Key Data Structures

- **2D Array (10x10 Grid):** The primary data structure is the grid, which is a list of lists. Each element in this 2D array is an instance of the Tile class. This structure is efficient for board traversal and direct access to any tile by its coordinates.
- **Tile Object:** Each Tile object serves as a small, self-contained data structure storing all the information needed for a single cell.

5. Assumptions

- **Fixed Grid Size:** The game board is a fixed 10x10 grid. The code is hardcoded to this dimension, simplifying development.
- **Mine Count:** The number of mines is user-specified at the start and is between 10 and 20.
- **Terminal Environment:** The application is designed to run exclusively within a command-line terminal, using basic text output for the user interface.