**System Architecture Overview**

**High-Level Description**

This program implements a Tkinter-based Minesweeper game that allows both human and AI-controlled gameplay. The architecture follows an MVC-style (Model–View–Controller) pattern, separating game logic, user interface, and control flow.

At a high level, the system consists of four primary layers:

1. UI Layer (View & Controller):
   * Built with Tkinter, containing windows, frames, and buttons.
   * Manages user interactions (clicks, dialogs, resets) and updates the screen accordingly.
   * The MinesweeperApp class acts as the main window, while MineCountDialog provides a startup configuration menu.
2. Logic Layer (Model):
   * Managed by the BoardManager class (imported from Classes.minesweeper).
   * Stores the true state of the game board, including mine locations, revealed cells, and flags.
   * Handles all core game logic (mine placement, uncovering cells, flood-fill, win/loss conditions).
3. AI Layer (Automation & Simulation):
   * Provides automated gameplay through the easyai, mediumai, and hardai methods inside MinesweeperApp.
   * Each AI level uses progressively more sophisticated decision-making, ranging from random guessing to deterministic inference.
4. Timer System (Optional Gameplay Mode):
   * Added within the UI Layer to support Time-Attack gameplay.
   * Displays and updates a countdown timer in the toolbar.
   * Starts when the first cell is uncovered and stops upon win, loss, or timeout.
   * Adds bonus seconds per valid uncover and triggers a time-up event when time expires.
   * Controlled by methods: \_start\_timer(), \_tick(), \_add\_time(), \_stop\_timer(), \_time\_up().

Together, these components create a modular and extendable system, where the UI and AI interact through well-defined interfaces to update the underlying board model and refresh the display in real time.

**Key System Components**

|  |  |  |
| --- | --- | --- |
| Component | Role | Key Methods / Features |
| MineCountDialog | Startup menu that prompts for mine count and AI difficulty. | submit(), aisubmit(), addbuttons() |
| MinesweeperApp | Main application class controlling the Tkinter window and user interface. Handles events, AI turns, and updates the display. | on\_left\_click(), on\_right\_click(), update\_view(), reset\_game() |
| BoardManager | Backend model managing the board state and game logic. | uncoverCell(), flagCell(), expandOpenCells(), placeMines() |
| AI Modules | Simulate automated gameplay behavior based on difficulty. | easyai(), mediumai(), hardai() |
| Tkinter Components | GUI elements such as grid buttons, toolbar, and status labels. | Buttons in a 2D array, dynamically styled per state |
| Timer System | Manages countdowns, bonus additions, and timeout behavior for Time-Attack mode. | \_start\_timer(), \_tick(), \_add\_time(), \_stop\_timer(), \_time\_up() |

**Key Data Structures**

|  |  |  |
| --- | --- | --- |
| Data Structure | Description | Example |
| boardContent[r][c] | 2D list storing mine placement and adjacent mine counts. | [[0,1,M], [1,2,1], ...] |
| boardState[r][c] | 2D list of integers representing the visual state of each tile (0 = covered, 1 = flagged, 2 = uncovered). | [[0,0,1], [2,0,0], ...] |
| buttons[r][c] | 2D list of Tkinter Button widgets, each tied to a position on the board. | [ [Button, Button, Button], ... ] |
| NUMBER\_COLORS | Dictionary mapping numbers (0–8) to color hex codes for visual differentiation. | {1:"#1976d2", 2:"#388e3c", ...} |
| ai\_diff, ai\_mode | Strings that determine AI difficulty and behavior mode ("vs" or "sim"). | "m", "sim" |

In Time-Attack mode, the timer begins after the first uncover action. Each valid uncover grants bonus time, and the countdown decrements every second. If time reaches zero, \_time\_up() triggers a time-out loss, revealing all mines and resetting the game.

**Data Flow and System Behavior**

The system operates through an event-driven cycle:

1. Initialization Phase:
   * The user launches the game and configures parameters (mine count, AI mode) through MineCountDialog.
   * The main window (MinesweeperApp) is created with a grid of Tkinter buttons and a linked BoardManager.
2. Gameplay Phase:
   * User or AI performs left or right clicks.
   * Click events are handled by on\_left\_click() or on\_right\_click(), which call BoardManager functions to update the game state.
   * Once the model is updated, update\_view() redraws the UI to reflect uncovered cells, flags, or revealed mines.
3. Endgame Phase:
   * \_check\_win() continuously monitors progress.
   * When all safe cells are uncovered or a mine is revealed, a dialog appears and the game resets.

This diagram now includes a Timer System component in the UI layer, showing how it connects to event handlers and affects game flow.

**Diagram 1 – System Component Architecture**

Purpose:  
Illustrates the major architectural components and their interactions within the MVC + AI integrated system.

Description:  
The diagram separates the program into the UI Layer, Logic Layer, and AI Layer.

* The UI Layer manages Tkinter components like buttons, labels, and frames, as well as the startup dialog.
* The Logic Layer is encapsulated by BoardManager, which holds the true state of the game (mines, uncovered cells, flags).
* The AI Layer automates decision-making by generating click and flag commands that mimic user input.
* The Event System connects these layers, translating clicks into model updates and triggering a UI refresh after every change.

Key Flow:  
User → Event Handler → BoardManager → UI Update → Win/Loss Check  
AI operates as an alternate event source feeding into the same cycle.

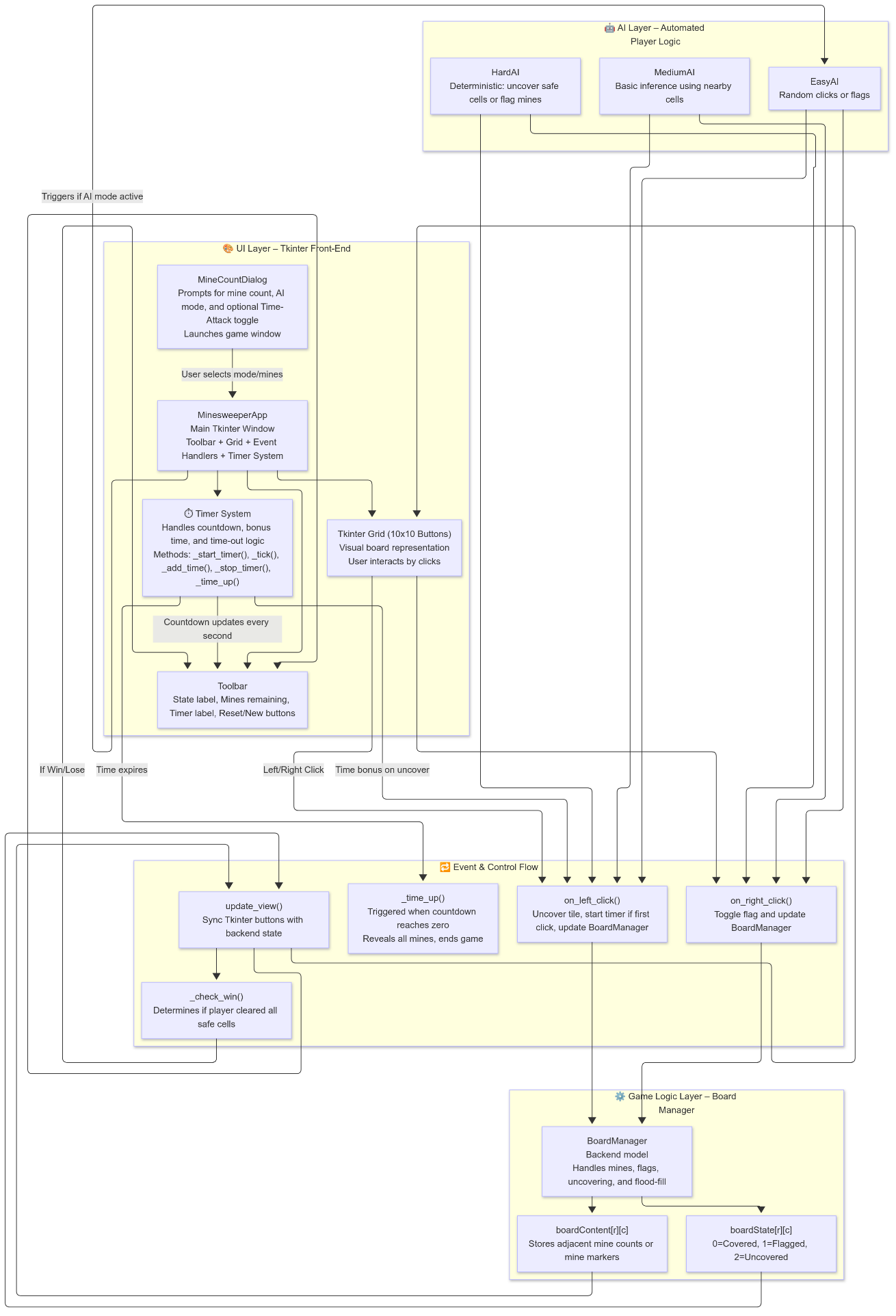


Diagram 2 now includes Time-Attack interactions, showing timer initialization, countdown, bonus addition, and timeout flow.

**Diagram 2 – Data Flow and Game Lifecycle**

Purpose:  
Demonstrates how data and control flow through the system over time — from initialization to game conclusion.

Description:  
The diagram breaks the game process into four distinct stages:

1. Initialization Stage:
   * MineCountDialog collects configuration values and creates the main application instance.
   * The first click is guaranteed safe by regenerating the board if necessary.
2. Gameplay Loop:
   * Each user or AI click triggers an event handled by on\_left\_click() or on\_right\_click().
   * BoardManager updates boardState and boardContent, which are immediately reflected in the GUI via update\_view().
3. Game End Conditions:
   * \_check\_win() verifies victory or defeat.
   * When the game ends, reset\_game() reinitializes the board and restarts the UI.
4. AI Interaction:
   * When active, the AI replaces the player as the input source.
   * Depending on difficulty, the AI chooses moves randomly or strategically, using the same event handling pipeline as a human player.

This flow demonstrates a continuous feedback loop between player actions, backend logic, and visual rendering. This forms the reactive core of the Minesweeper experience.

