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.

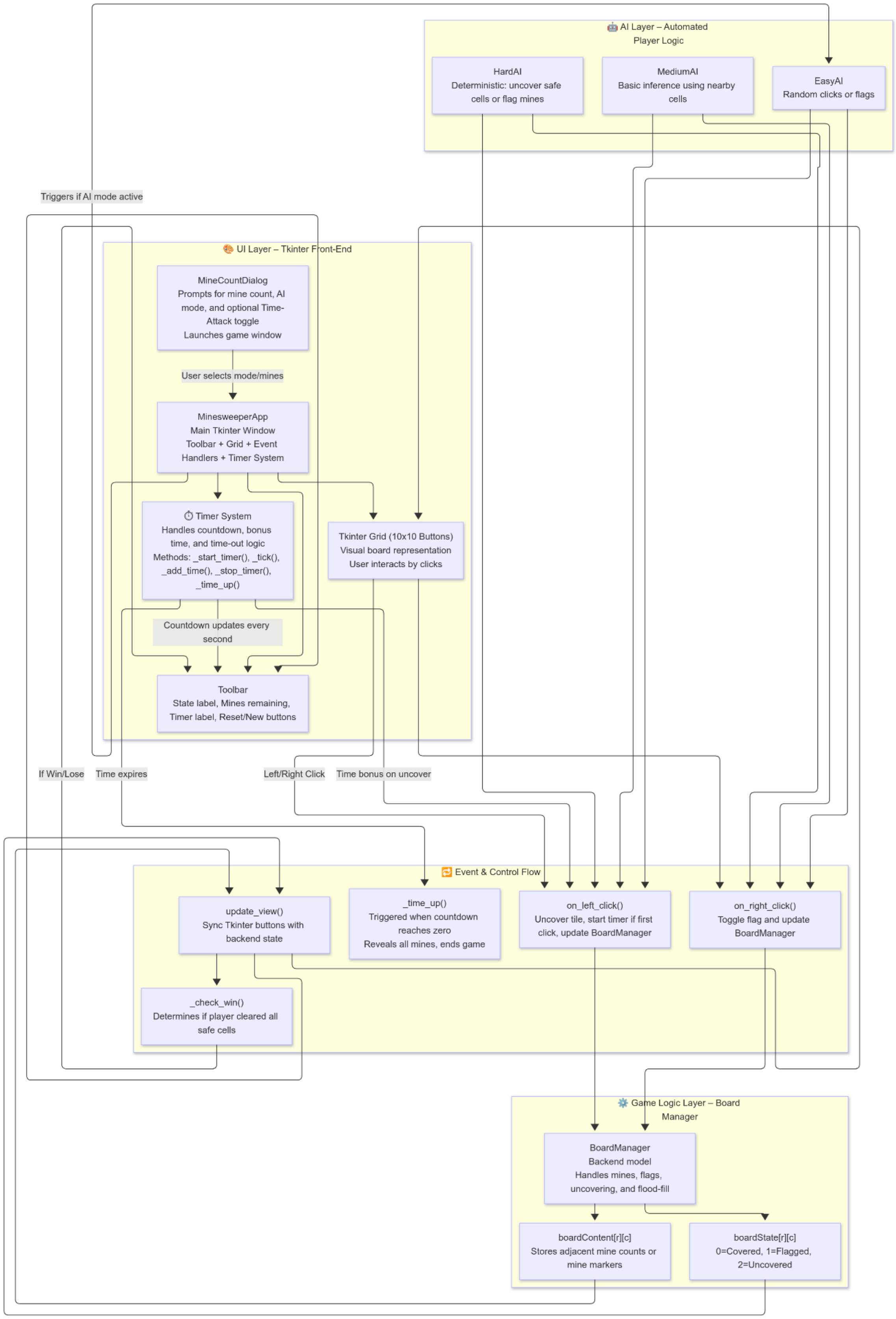

🤖 AI Layer – Automated Player Logic
HardAI Deterministic: uncover safe cells or flag mines
MediumAI Basic inference using nearby cells
EasyAI Random clicks or flags
Triggers if AI mode active
🎨 UI Layer – Tkinter Front-End
MineCountDialog Prompts for mine count, AI mode, and optional Time-Attack toggle Launches game window
User selects mode/mines
MinesweeperApp Main Tkinter Window Toolbar + Grid + Event Handlers + Timer System
⏱ Timer System Handles countdown, bonus time, and time-out logic Methods: _start_timer(), _tick(), _add_time(), _stop_timer(), _time_up()
Tkinter Grid (10x10 Buttons) Visual board representation User interacts by clicks
Countdown updates every second
Toolbar State label, Mines remaining, Timer label, Reset/New buttons
If Win/Lose
Time expires
Left/Right Click
Time bonus on uncover
🔁 Event & Control Flow
update_view() Sync Tkinter buttons with backend state
_time_up() Triggered when countdown reaches zero Reveals all mines, ends game
on_left_click() Uncover tile, start timer if first click, update BoardManager
on_right_click() Toggle flag and update BoardManager
_check_win() Determines if player cleared all safe cells
⚙️ Game Logic Layer – Board Manager
BoardManager Backend model Handles mines, flags, uncovering, and flood-fill
boardContent[r][c] Stores adjacent mine counts or mine markers
boardState[r][c] 0=Covered, 1=Flagged, 2=Uncovered

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

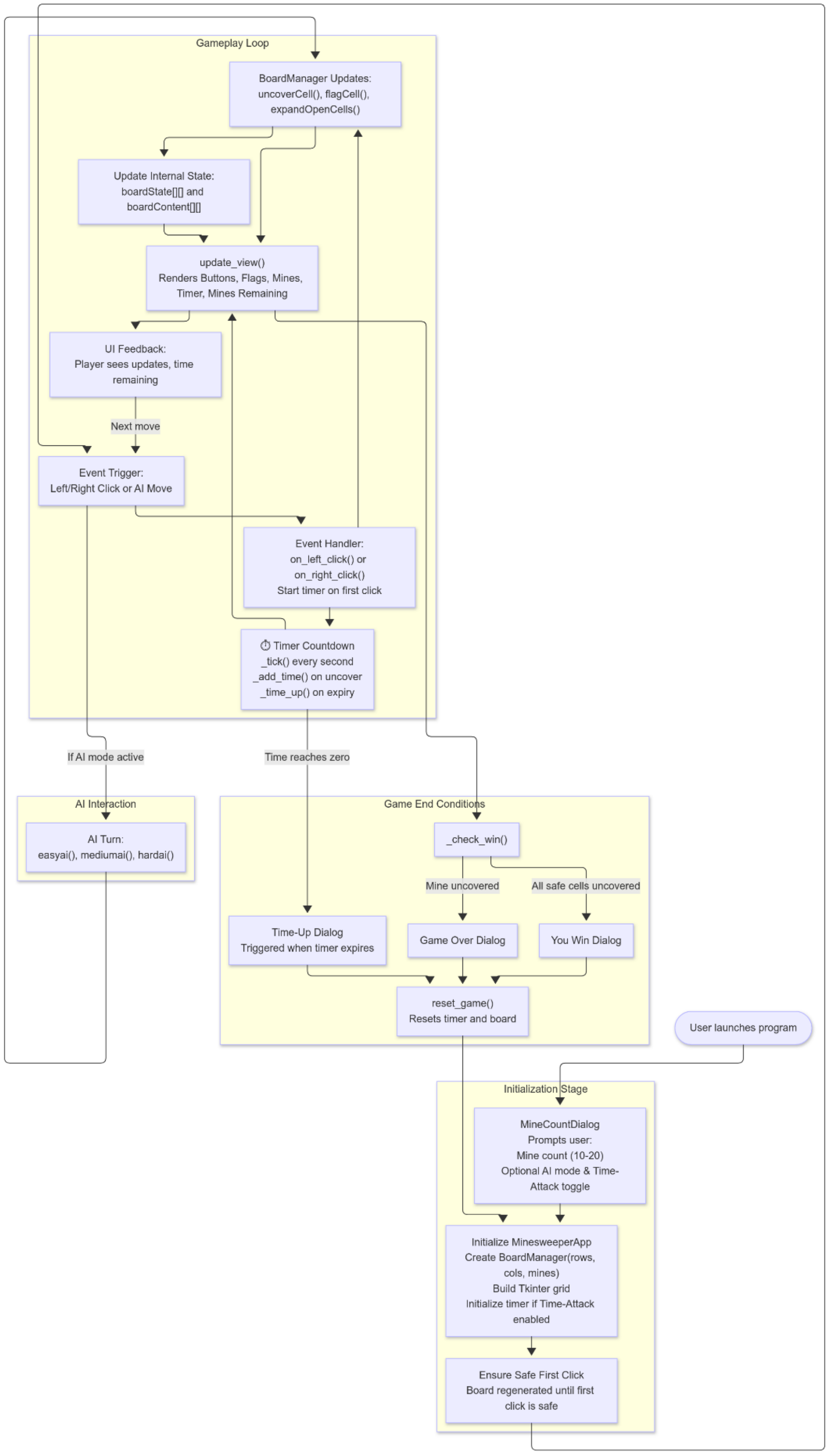
Diagram 2 – Data Flow and Game Lifecycle

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.


Gameplay Loop
BoardManager Updates: uncoverCell(), flagCell(), expandOpenCells()
Update Internal State: boardState[][] and boardContent[][]
update_view() Renders Buttons, Flags, Mines, Timer, Mines Remaining
UI Feedback: Player sees updates, time remaining
Next move
Event Trigger: Left/Right Click or AI Move
Event Handler: on_left_click() or on_right_click() Start timer on first click
⏱ Timer Countdown _tick() every second _add_time() on uncover _time_up() on expiry
If AI mode active
Time reaches zero
AI Interaction
AI Turn: easyai(), mediumai(), hardai()
Game End Conditions
_check_win()
Mine uncovered
All safe cells uncovered
Time-Up Dialog Triggered when timer expires
Game Over Dialog
You Win Dialog
reset_game() Resets timer and board
User launches program
Initialization Stage
MineCountDialog Prompts user: Mine count (10-20) Optional AI mode & Time-Attack toggle
Initialize MinesweeperApp Create BoardManager(rows, cols, mines) Build Tkinter grid Initialize timer if Time-Attack enabled
Ensure Safe First Click Board regenerated until first click is safe