Python Project: Detailed Problem Statement Document

Each of the 5 Python projects requires a Tkinter-based GUI. All projects follow common general requirements and involve file handling using CSV. The projects should be developed with modularity, error handling, and user-friendliness in mind.

General Requirements:

Each project will be a Tkinter-based GUI application and must adhere to the following basic requirements:

1. **At Least 3 Classes in Different Modules:**
   - Classes should be distributed across different Python modules (separate files). The application must be structured with a clear separation of concerns, including, but not limited to:
     - `main.py`: The main file which integrates the other classes and manages the overall control flow.
     - `game_logic.py` (or equivalent): Manages the core game/system logic.
     - `ui.py`: Handles Tkinter-based UI components.
     - `file_manager.py`: Handles file input/output using CSV for storing and retrieving data.
2. **Tkinter-based GUI:**
   - The application must have at least 3 different user interfaces (UI pages), created using Tkinter, for interaction and navigation.
3. **File Handling Using CSV:**
   - All data (such as game results, history, orders, borrowing records, etc.) must be stored in CSV format. No databases should be used.
   - CSV files should be well-formatted, with appropriate headers for each column.
   - The system should include error handling for file operations such as:
     - Handling missing or corrupted files (missing fields or invalid data).
     - Providing appropriate error messages when file read/write fails.
4. **User-Friendliness:**
   - The application must be user-friendly and intuitive, with clear instructions and navigations.
   - Implement basic error handling and validation, such as managing invalid inputs and handling exceptions gracefully.
5. **Main Entry Point:**
   - The project should be structured such that `main.py` serves as the entry point. It should import necessary modules and manage interactions between UI, game/system logic, and file handling components.
6. **UI Responsiveness:**

- Ensure the UI is responsive, adjusting elements like buttons and grids to various screen sizes if necessary (optional).

7. **CSV Data Formats:**
   - Ensure all CSV files are well-structured with appropriate headers.
   - Example for a Tic-Tac-Toe results file:

```
Game Number, Player 1 Name, Player 2 Name, Winner, Date, Time
           1, Alice, Bob, Alice, 2024-09-18, 14:30
```

Project Tracks:

1. Tic-Tac-Toe Game

Problem Statement: Design and implement a Tic-Tac-Toe game where two players (or a player and a computer) can play against each other. The game displays a 3x3 grid, and players take turns placing their marks (X and O). The first player to align three marks horizontally, vertically, or diagonally wins. If all spaces are filled and no player has won, the game ends in a draw.

Requirements:

- **UI 1 (Game)**:
  - Display a 3x3 grid.
  - Players can click on grid cells to place their marks.
  - Display whose turn it is (Player 1 or Player 2).
  - Option to restart the game.
- **UI 2 (History of Results)**:
  - Show the history of all games played (win/loss/draw) stored in a CSV file.
  - Each entry displays the result (Player 1 wins, Player 2 wins, or Draw) and the time the game was played.
- **UI 3 (Select UI)**:
  - This screen allows the user to select whether they want to play a new game or view the history of results.
  - Include buttons: “New Game” and “View History”.
  - If New Game is selected, take player names as inputs.
- **File**:
  - Store the results of each game in a CSV file with columns like “Game Number”, “Winner”, and “Date/Time”.

Example Modules & Classes:

- **Game Logic Module**: Handle game logic (e.g., determine the winner or draw).

    - Class: `TicTacToe`

- **UI Module**: Create and manage the Tkinter interface.
    - Class: `TicTacToeUI`
- **File Handling Module**: Read/write game history to/from a CSV file.
    - Class: `HistoryManager`

---

2. Memory Game

Problem Statement: Develop a Memory Game where the player must match pairs of cards (can be colours or images also). The cards are displayed face down, and the player can flip two at a time. If the cards match, they stay face up; if they don't, they flip back. The game ends when all pairs are matched.

Requirements:

- **UI 1 (Game)**:
    - Display cards in a grid (4x4 or 5x5).
    - Allow the player to flip two cards at a time.
    - Track the number of moves made by the player.
- **UI 2 (History of Scores)**:
    - Show the player's previous scores (number of moves to complete the game).
    - Store scores (moves taken) and completion time in a CSV file.
- **UI 3 (Select UI)**:
    - Allow the player to either start a new game or view previous scores.
    - Include buttons: "New Game" and "View History".
    - If New Game is selected, take player names as inputs.
- **File**:
    - Store scores and completion times in a CSV file with columns like "Game Number", "Moves", and "Time".

Example Modules & Classes:

- **Game Logic Module**: Handle the card flipping and matching logic.
    - Class: `MemoryGame`
- **UI Module**: Create and manage the Tkinter interface.
    - Class: `MemoryGameUI`
- **File Handling Module**: Manage the reading/writing of scores to/from a CSV file.
    - Class: `ScoreManager`

---

3. Library Management System

Problem Statement: Develop a Library Management System where users can borrow books and an admin can manage borrowing records. Users can view available books and borrow them. Admins can manage which books are available or mark them as borrowed.

Requirements:

- **UI 1 (Student Borrowing)**:
    - Display a list of available books.
    - Allow students to borrow books by selecting from the list.
    - Display confirmation once a book is borrowed.
- **UI 2 (Admin Manages Borrowing)**:
    - Admin can view a list of borrowed books.
    - Admin can mark books as returned, updating their availability.
- **UI 3 (Select UI)**:
    - Display two options: for students to borrow books or for the admin to manage borrowing.
    - Include buttons: "Borrow Books" and "Manage Borrowing".
- **File**:
    - Store borrowing records in a CSV file with columns like "Book ID", "Book Title", "Borrower", and "Borrow Date".
    - Update the file when books are borrowed or returned.

Example Modules & Classes:

- **Book Module**: Manage the list of books and their borrowing status.
    - Class: `Book`
- **UI Module**: Create and manage the Tkinter interface.
    - Class: `LibraryUI`
- **File Handling Module**: Manage the reading/writing of borrowing history to/from a CSV file.
    - Class: `BorrowingHistoryManager`

---

4. Hospital Management System

Problem Statement: Develop a Hospital Management System where patients can book appointments, and doctors can manage those appointments. Patients can see available time slots and book appointments. Doctors can view their appointments and manage them.

Requirements:

- **UI 1 (Patient Appointment Booking)**:
    - Display a list of available doctors and time slots.
    - Allow patients to book appointments by selecting a doctor and time.
    - Confirm the appointment once booked.
- **UI 2 (Doctor Manages Appointments)**:
    - Doctors can view their scheduled appointments.
    - Allow doctors to mark appointments as completed or canceled.
- **UI 3 (Select UI)**:
    - Provide options for patients to book appointments or for doctors to manage their schedule.
    - Include buttons: "Book Appointment" and "Manage Appointments".
- **File**:
    - Store appointment details in a CSV file with columns like "Appointment ID", "Doctor", "Patient Name", "Time Slot", and "Status".
    - Update the file when appointments are booked, completed, or canceled.

Example Modules & Classes:

- **Appointment Module**: Manage appointment booking and status.
    - Class: `Appointment`
- **UI Module**: Create and manage the Tkinter interface.
    - Class: `HospitalUI`
- **File Handling Module**: Manage the reading/writing of appointment records to/from a CSV file.
    - Class: `AppointmentHistoryManager`

5.Restaurant Management System

Problem Statement: Create a Restaurant Management System where customers can place food orders, and an admin can manage the menu and orders. Customers select food items, place orders, and see the total cost. Admins can add/remove menu items and view current orders.

Requirements:

- **UI 1 (Customer Orders)**:
    - Display the menu with food items and prices.
    - Allow customers to select items, place an order, and see the total cost.

- **UI 2 (Admin Manages Menu & Orders)**:
  - Admin can add/remove menu items and manage the current list.
  - Admin can view current orders and mark them as completed.
- **UI 3 (Select UI)**:
  - Provide options for customers to place orders or for the admin to manage the menu.
  - Include buttons: “Place Order” and “Manage Menu”.
- **File**:
  - Store order records in a CSV file with columns like “Order ID”, “Customer Name”, “Items Ordered”, “Total Cost”, and “Status”.
  - Update the file when orders are placed or completed.

Example Modules & Classes:

- **Menu Module**: Manage the list of food items and prices.
  - Class: `Menu`
- **UI Module**: Create and manage the Tkinter interface.
  - Class: `RestaurantUI`
- **File Handling Module**: Manage the reading/writing of menu and order history to/from a CSV file.
  - Class: `OrderHistoryManager`