Project Report: Tic-Tac-Toe Game

Field	Detail
Project Title	Graphical Tic-Tac-Toe Application
Course Code	CSE 1021
Student Name	Akshit
Registration No.	25BAI10954
Submission Date	November 2025

2. Introduction

This report documents the design and implementation of a console-style graphical Tic-Tac-Toe game. The project aims to provide a simple, intuitive, and interactive experience for playing the classic 3x3 grid game. The application is built using Python, leveraging the **Tkinter** library for the Graphical User Interface (GUI), and employing a modular design to separate the UI and core game logic. The application supports two modes: Player vs. Player (Friend) and Player vs. Computer.

3. Problem Statement

The goal is to develop a functional and user-friendly Tic-Tac-Toe game that runs as a desktop application. The system must accurately manage game state, enforce game rules, determine win/draw conditions, and provide a choice between a two-player mode and a single-player mode against a rudimentary computer opponent.

4. Functional Requirements

ID	Requirement	Description
FR1	Game Initialization	The application must start with a menu allowing the user to select the game mode (Friend or Computer).
FR2	Board Display	Display a clear 3x3 grid

		representing the game board.
FR3	Player Input	Allow players to click on empty cells to make a move.
FR4	Turn Management	Alternate turns between 'X' and 'O'.
FR5	Rule Enforcement	Prevent moves on occupied cells.
FR6	Win Condition	Accurately detect and announce the winner based on horizontal, vertical, or diagonal alignment.
FR7	Draw Condition	Accurately detect and announce a draw if all cells are filled without a winner.
FR8	Computer AI (Single Player)	The computer player ('O') must automatically make a move in the single-player mode.
FR9	Game Reset	After a game concludes, prompt the user or automatically return to the main menu.

5. Non-functional Requirements

ID	**Requirement**	**Description**
NFR1	Usability	The GUI must be intuitive, with clear labels and large, clickable buttons.
NFR2	Modularity	The application code must

		be separated into logical modules (GUI, Logic) for easy maintenance and testing.
NFR3	Performance	The game must respond instantly to player clicks and the computer move should occur with minimal delay.
NFR4	Portability	The application should run on any operating system that supports Python and Tkinter.

6. System Architecture

The project follows a simple **Three-Tier Architecture** (Presentation, Logic, Data) implemented using a **Modular Design Pattern**:

1. **Presentation Layer (gui.py, main.py):** Tkinter-based user interface. This layer handles drawing the board, capturing button clicks, and updating the display based on game state.
2. **Business Logic Layer (game_logic.py):** Contains the core algorithms for checking win/draw conditions and generating the computer's move. This layer is independent of the GUI.
3. **Data Layer (In-Memory Array):** The game state is maintained as a simple list/array (self.board in gui.py), which holds the values of the 9 cells.

7. Design Diagrams

Use Case Diagram (Textual Description)

- **Actors:** Player (Human/User), System (Computer AI).
- **Use Cases:**
  - **Player:** Start Game, Select Game Mode (vs Friend/vs Computer), Make Move, View Board.
  - **System:** Check for Winner, Switch Turn, Determine Draw, Make Computer Move (for vs Computer mode).

Workflow Diagram (Textual Description)

1. **Start:** Application loads and displays the **Menu Screen**.

2. **Selection:** Player clicks "Play vs Friend" or "Play vs Computer".
3. **Game Setup:** The board is initialized, and Player 'X' starts.
4. **Loop:**
   - Current Player clicks an empty cell (via on_click).
   - make_move updates the board and UI.
   - check_winner is called.
   - **If Winner/Draw:** Go to **End Game**.
   - **If Computer Turn:** computer_turn calls get_computer_move.
   - The loop continues until End Game.
5. **End Game:** A message box displays the result. The application returns to the Menu Screen.

Sequence Diagram (Textual Description for a Player vs Computer turn)

1. **User** -> **GUI:** on_click(index)
2. **GUI** -> **GUI:** make_move(index, 'X') (Updates board, UI)
3. **GUI** -> **Logic:** check_winner(board)
4. **Logic** -> **GUI:** Returns None (No winner yet)
5. **GUI** -> **GUI:** Updates info_label (Switches turn to 'O')
6. **GUI** -> **GUI:** computer_turn() (Delayed call)
7. **GUI** -> **Logic:** get_computer_move(board)
8. **Logic** -> **GUI:** Returns move_index
9. **GUI** -> **GUI:** make_move(move_index, 'O') (Updates board, UI)
10. **GUI** -> **Logic:** check_winner(board)

Class/Component Diagram (Textual Description)

- **TicTacToeApp (Class in gui.py):**
  - **Properties:** root, turn, board, game_mode, game_over, buttons, info_label.
  - **Methods:** __init__, create_menu, start_game, create_board, on_click, computer_turn, make_move, end_game, clear_window.
- **game_logic (Module):**
  - **Functions:** check_winner(board), get_computer_move(board).
- **main (Module):**
  - **Functions:** Execution Entry (if __name__ == "__main__":).

ER Diagram

- **Not Applicable.** This application uses in-memory storage (self.board array) and does not require an external relational database, thus an Entity-Relationship (ER) Diagram is not relevant.

8. Design Decisions & Rationale

Decision	Rationale
Tkinter for GUI	Tkinter is the standard Python GUI library, requiring no external installations, ensuring high portability and simplicity for a small desktop application.
Modular Separation	Separating gui.py from game_logic.py adheres to the Single Responsibility Principle (SRP). This makes the core game logic testable without relying on the UI framework.
Simple Computer AI	The initial implementation of the computer move (get_computer_move) uses random.choice. This satisfies the basic requirement for a single-player mode while keeping the logic simple and quickly deployable.
Tkinter after()	Used in on_click to introduce a slight delay (self.root.after(500, self.computer_turn)). This improves the user experience by making the computer's response feel more natural and less instantaneous.

9. Implementation Details

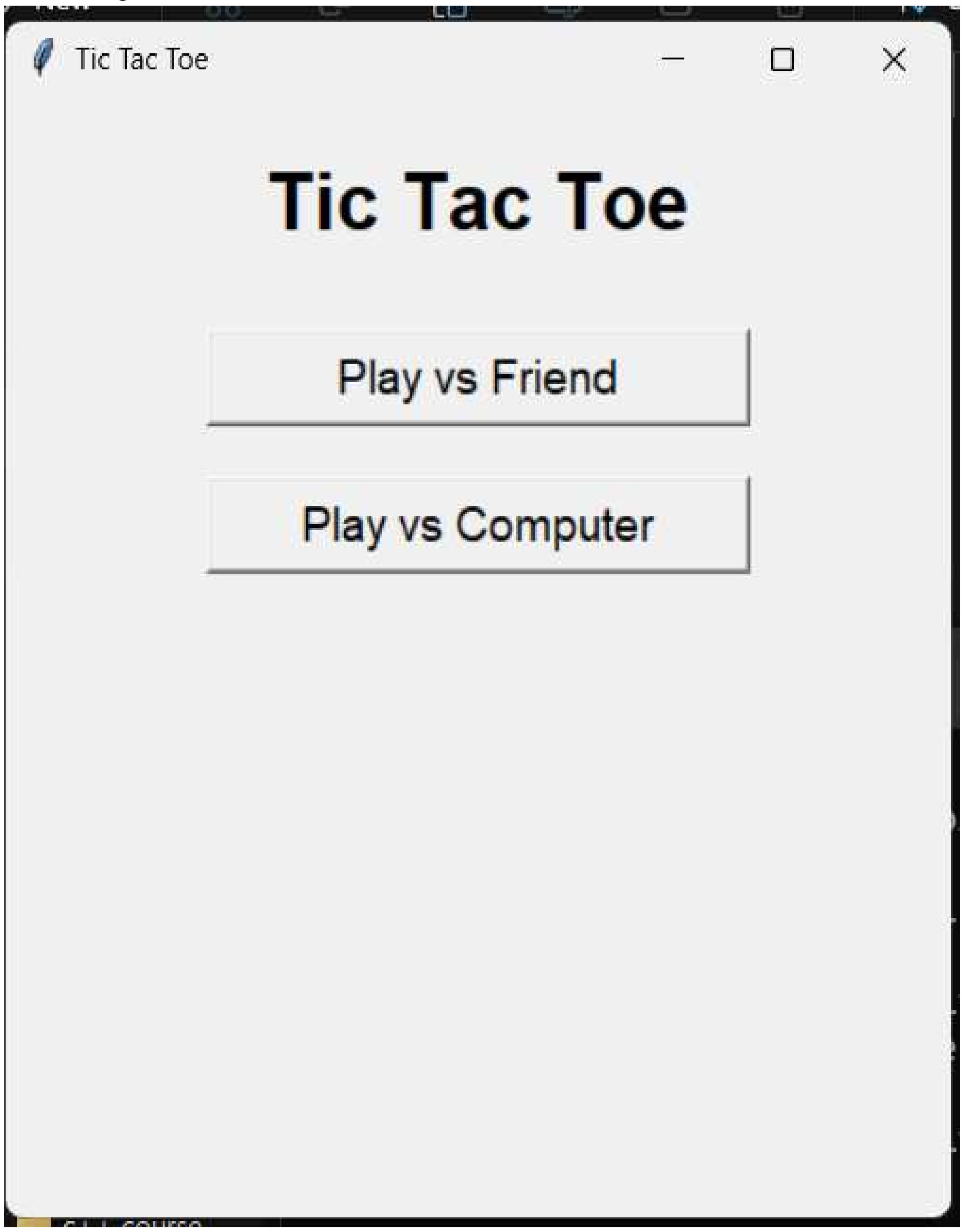
The application is structured into three files:

1. **main.py**: Serves as the boilerplate code, initializes the Tkinter environment, and instantiates the main application class.
2. **gui.py**: Contains the TicTacToeApp class, managing all visual components and event handling. It orchestrates the game flow by calling methods in game_logic.py and updating the UI accordingly.
   - The on_click(index) method is the primary event handler, responsible for validating moves, making the move, checking for the game end, and initiating the computer's turn if necessary.
3. **game_logic.py**: Encapsulates the core Tic-Tac-Toe rules.
   - check_winner: Iterates over 8 predefined winning combinations to check if 'X' or 'O' occupies all three positions. It also checks for a draw (no empty cells).
   - get_computer_move: A simple AI that finds all empty cells and randomly selects one

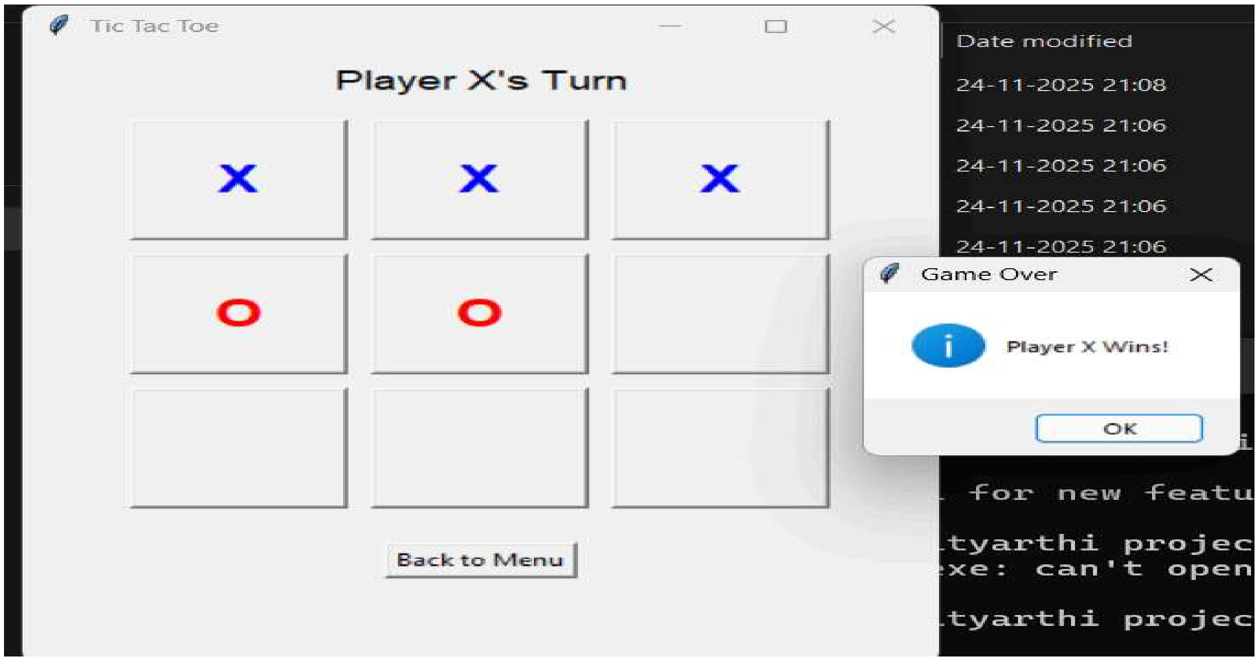
for the computer's move.

10. Screenshots / Results

Home Page



Game Screen



11. Testing Approach

The testing was approached in two phases: Unit Testing for the logic and Integration Testing for the GUI.

1. **Unit Testing (game_logic.py):**
   - **Win Conditions:** Manually constructed test boards to verify if check_winner correctly identifies all 8 winning combinations (rows, columns, diagonals) for both 'X' and 'O'.
   - **Draw Condition:** Tested a full board with no winner to ensure a "Draw" result is returned.
   - **Ongoing Game:** Tested partially filled boards to ensure None is returned.
   - **Computer Move:** Verified that get_computer_move always returns an index that is currently empty on the board.
2. **Integration Testing (gui.py and main.py):**
   - **Flow Test:** Confirmed smooth transition between the Menu, Game Board, and End Game screens.
   - **Interaction Test:** Ensured buttons correctly update the board with the current player's symbol ('X' or 'O') and disabled after a move.
   - **Turn Test:** Verified that the info_label correctly changes between "Player X's Turn" and "Player O's Turn."
   - **Computer Interaction Test:** Ensured the computer automatically makes a move after the human player ('X') moves in the single-player mode.