Implementation of a Tic-Tac-Toe game using Adversarial Search

Submitted by:

Alexander Jamatia, 234101007

Amiya Ranjan Panda, 234101010

Problem Definition:

Design and implement a Tic Tac Toe game in Python. The game should allow a user to play against an AI opponent that makes moves based on the minimax algorithm and alpha-beta pruning. The game should provide a graphical user interface (GUI) where users can interact with the game board by clicking on buttons representing the cells. The game should accurately determine the winner or declare a draw when appropriate. Additionally, the game should have a restart button to allow users to start a new game after completing one.

Input Description:

1. Click Events: The main input mechanism for the game is clicking on the buttons representing the cells of the Tic Tac Toe grid. These clicks trigger the **play** function, which determines the player's move and updates the game state accordingly.
2. Restart Button: Clicking on the "RESTART" button triggers the **restart** function, which resets the game state and clears the game board for a new game session.

The game logic also receives implicit inputs from the program state:

1. Current Game State: The state of the Tic Tac Toe board, represented by the **board** dictionary, determines the positions of Xs and Os.
2. Current Turn: The **turn** variable keeps track of whose turn it is to play (either "X" or "O").
3. Game Over State: The **over** variable indicates whether the game has ended (0 for ongoing, 1 for game over). This prevents further moves after a win or draw.

Implementation Description:

1. **Graphical User Interface (GUI) Setup**:
    - The Tkinter library, a standard Python library for creating GUI applications, is imported at the beginning of the script to provide the necessary tools for building the game interface.
    - An instance of Tkinter's **Tk()** class is created to represent the main window of the application (**root**). This window serves as the container for all other GUI elements.

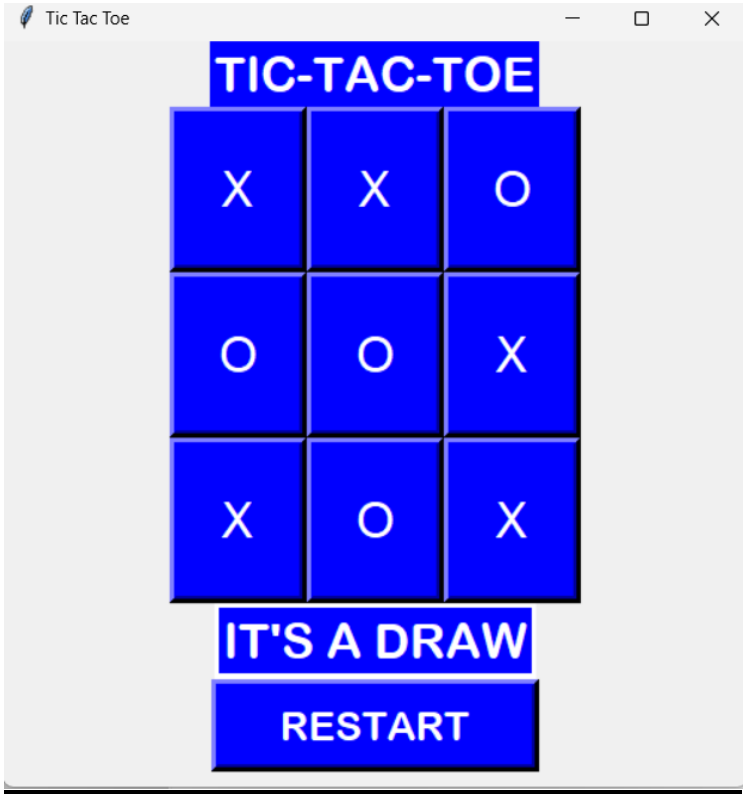
- Upon initialization, the window's geometry is set to a square shape with dimensions of 500x500 pixels to provide a visually appealing layout. Additionally, the title of the window is defined as "Tic Tac Toe" to reflect the nature of the game.

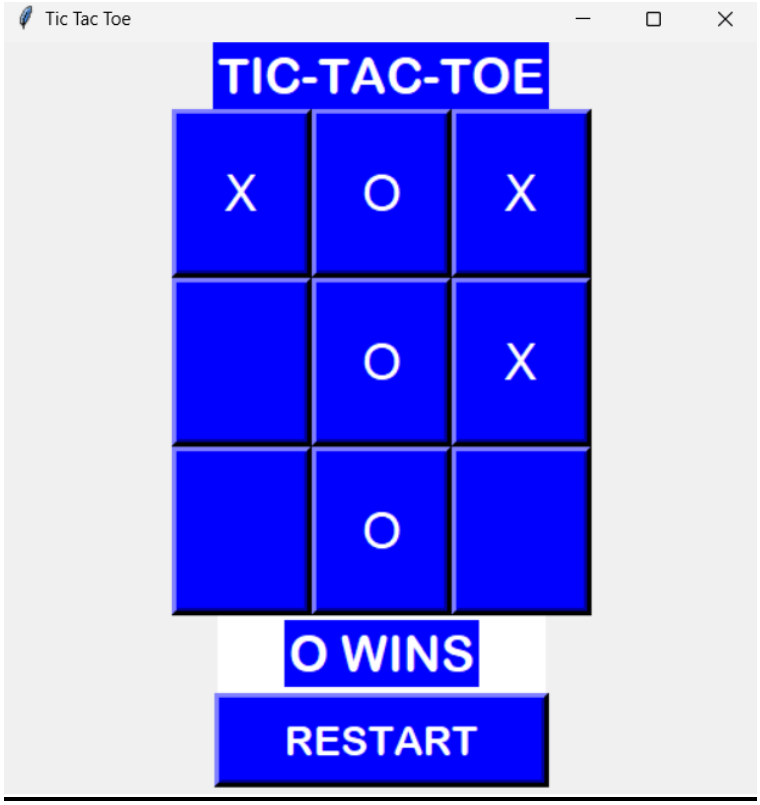
2. **Frame Organization**:
    - To achieve a structured and organized layout, two separate frames, **frame1** and **frame2**, are instantiated within the main window (**root**).
    - **frame1** is positioned at the top of the window and is primarily utilized for displaying the game's title. It serves as a visual anchor to provide context and branding for the game.
    - **frame2** occupies the remaining space within the window and serves as the main area for displaying the Tic Tac Toe grid, buttons, and game outcome labels. This frame is responsible for hosting the interactive components of the game.
3. **Tic Tac Toe Board Representation**:
    - The game board is represented programmatically using a Python dictionary named **board**.
    - Within this dictionary, each key corresponds to a specific cell position on the Tic Tac Toe grid (ranging from 1 to 9), while the corresponding values denote the content of each cell. Initially, all cells are empty and are represented by space characters (" ").
    - This data structure facilitates efficient tracking and manipulation of the game state throughout the course of gameplay.
4. **Game Logic Functions**:
    - The implementation of the game's core logic is encapsulated within a series of functions designed to handle various aspects of gameplay and decision-making processes.
    - **winner(player)**: This function evaluates the current state of the game board to determine if the specified player has achieved a winning configuration. It checks all possible win conditions, including horizontal, vertical, and diagonal alignments, to ascertain the presence of a winning pattern.
    - **draw()**: This function assesses whether the game has concluded in a draw by examining the occupancy status of all cells on the game board. If all cells are filled without any player achieving a winning configuration, the game is declared a draw.
    - **Minimax(board, Alturn, alpha, beta)**: The Minimax algorithm, a classic approach used in game theory and decision-making, is implemented to enable the AI opponent to make strategic moves. This function recursively explores potential future game states by simulating both player and opponent moves, assigning scores to each outcome, and selecting the optimal move based on the calculated scores. The algorithm utilizes alpha-beta pruning to enhance efficiency by eliminating unnecessary branches of the game tree.

- **playAI()**: This function orchestrates the AI opponent's decision-making process by invoking the Minimax algorithm to determine the best possible move. Once the optimal move is identified, it updates the game board accordingly with the AI's move.
- **play(event)**: Responsible for handling player interactions, this function is triggered in response to button clicks corresponding to cell selections on the Tic Tac Toe grid. It updates the game state based on the player's chosen move, evaluates the game's outcome (i.e., win, loss, or draw), and alternates between player turns.
- **restart()**: Upon invocation, this function resets the game state to its initial configuration, effectively initiating a new game session. It clears the game board, resets the game outcome labels, and prepares the environment for the commencement of a fresh game.

5. **Button Creation and Binding**:
    - The graphical representation of the Tic Tac Toe grid is realized through the creation of individual buttons using Tkinter's **Button** widget.
    - Each button is configured with specific attributes, and associated with the **play** function using event binding to ensure that clicking on a button triggers the corresponding gameplay action.
6. **User Interaction**:
    - The primary mode of interaction for players is through the graphical interface, which allows them to make moves by clicking on the buttons representing cells on the game board.
    - Upon selecting a cell, the game logic processes the player's move, updates the game state, evaluates the outcome of the game, and triggers subsequent actions accordingly.
7. **Game Outcome Display**:
    - To provide feedback to players regarding the outcome of each game session, designated labels are utilized to display relevant messages indicating the result (i.e., win, loss, or draw) of the game.
8. **Restart Button**:
    - A dedicated "RESTART" button is incorporated into the user interface to offer players the option to initiate a new game session at their discretion. Upon clicking the "RESTART" button, the associated event triggers the **restart** function.
9. **Mainloop**:
    - The execution of the GUI application is governed by the Tkinter mainloop, which serves as the event handling mechanism responsible for continuously monitoring user interactions and updating the graphical interface accordingly.
10. **Executable file creation:**
    - An executable file of the given python code “tictactoe_game” has been created using the Pyinstaller library.

Output:


Tic Tac Toe
TIC-TAC-TOE
X X O
O O X
X O X
IT'S A DRAW
RESTART


Tic Tac Toe
TIC-TAC-TOE
X O X
O X
O
O WINS
RESTART

References:

- For GUI and MINIMAX algorithm: https://www.youtube.com/watch?v=uVrzuKVus7s&list=PLeb12iNltItGQeGYSZdAIUArzBIDxVqMg&ab_channel=VisheshProgramming

- For alpha-beta pruning: https://stackoverflow.com/questions/65653940/tictactoe-alpha-beta-pruning

- For Executable file creation: https://www.youtube.com/watch?v=Iv_dECet_oM&ab_channel=CodeFirstwithHala