RIPHAH INTERNATIONAL UNIVERSITY

Design and analysis of algorithm Project

Presented By:-

Bilal Ahmed 46473

M Ali Meesam: 45289

Project Overview: Tic Tac Toe Game

This project is a graphical user interface (GUI) implementation cof the classic **Tic Tac Toe** game using **Python** and **Tkinter**. It allows two players to play the game on a 3x3 grid, with players alternating turns to place their respective marks ("X" and "O") on the grid. The game checks for winning conditions or a tie after each move and notifies the players accordingly.

How the Game Works

1. **Game Setup:**
   - The game window is created using Tkinter, a Python library for building GUI applications.
   - The game initializes a 3x3 grid using buttons. Each button represents a cell in the Tic Tac Toe grid.
2. **Game Flow:**
   - Player "X" starts the game by clicking on an empty cell in the grid.
   - When a player clicks a cell, the button text is updated with the player's symbol ("X" or "O").
   - The game checks whether the current player has won or if there is a tie after every move.
3. **Winning Conditions:**
   - The game checks the following conditions for a winner:
     - Three matching symbols ("X" or "O") in any row.
     - Three matching symbols in any column.
     - Three matching symbols in either of the two diagonals.
   - If a player achieves one of these winning conditions, a message box appears notifying the winner.
4. **Tie Conditions:**

   - If all cells are filled and no player has won, the game detects a tie and displays a message.
5. **Switching Turns:**
   - After each valid move, the game switches the active player, alternating between "X" and "O".
   - This process continues until either a player wins or the game ends in a tie.
6. **Resetting the Game:**
   - After a win or tie, the game board is reset to its initial empty state.
   - The first player ("X") is set to start the new game.

Key Components of the Code

1. **Tkinter Buttons:**
   - The buttons represent the cells of the Tic Tac Toe grid. Clicking a button updates the board and triggers a check for a winner or tie.
2. **Board Representation:**
   - The game board is represented as a 2D list (`self.board`), where each element corresponds to a cell. An empty string (`""`) represents an empty cell, and "X" or "O" represent the respective player's mark.
3. **Game Logic:**
   - The `check_winner()` method checks if a player has won by verifying rows, columns, and diagonals for three matching symbols.
   - The `is_tie()` method checks if the game board is full and there is no winner, signaling a tie.
4. **Messages and Alerts:**
   - The `messagebox.showinfo()` function is used to display a dialog box when a player wins or when the game ends in a tie.

Project Flow:

1. When the game starts, the window is displayed with a 3x3 grid of empty buttons.
2. Players take turns by clicking an empty button, which is then filled with their symbol ("X" or "O").
3. After each move, the game checks for a winner or tie:
   - If a player wins, a message is shown, and the game resets.
   - If the game ends in a tie, a message is shown, and the game resets.
4. The game continues until the user decides to close the window.

RIPHAH
INTERNATIONAL UNIVERSITY

Code:-

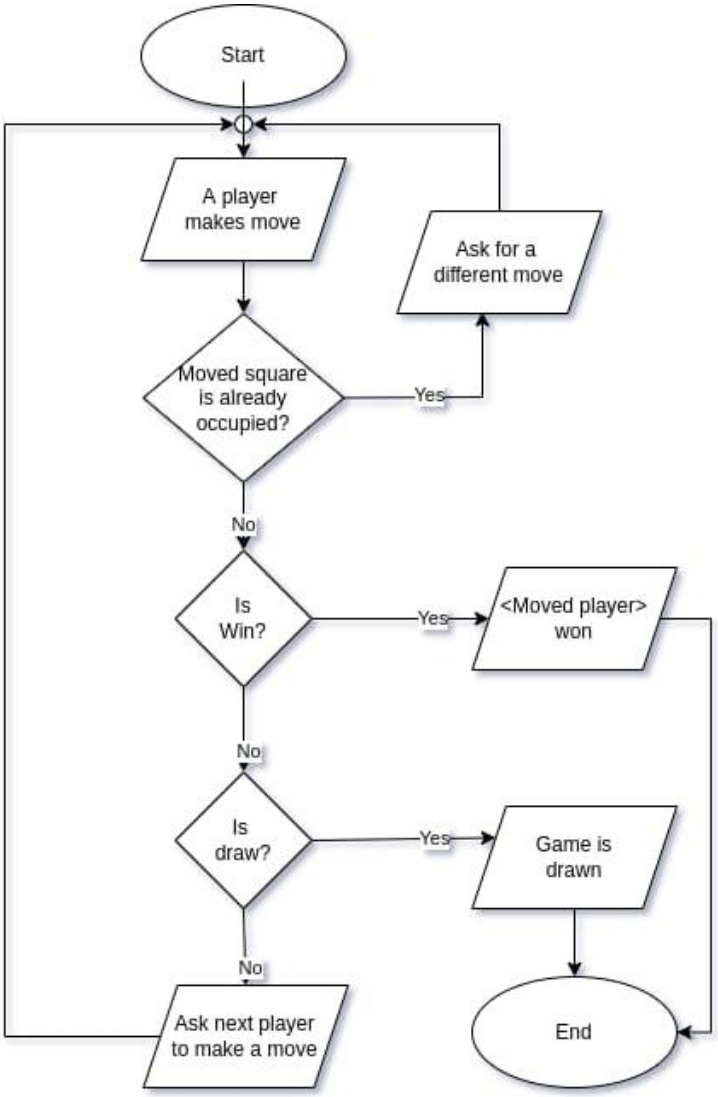
```
import tkinter as tk
from tkinter import messagebox


class TicTacToe:
    def __init__(self):
        self.window = tk.Tk()
        self.window.title("Tic Tac Toe")
        self.current_player = "X"  # Player "X" starts the game
        self.board = [["" for _ in range(3)] for _ in range(3)]  # 3x3 board
        self.buttons = [[None for _ in range(3)] for _ in range(3)]  # Button grid

        # Create the grid of buttons
        for row in range(3):
            for col in range(3):
                self.buttons[row][col] = tk.Button(
                    self.window, text="", font=("Arial", 24), width=5, height=2,
                    command=lambda r=row, c=col: self.on_click(r, c)
                )
                self.buttons[row][col].grid(row=row, column=col)

    def on_click(self, row, col):
        # Check if the cell is empty
        if self.board[row][col] == "":
```

```
        self.board[row][col] = self.current_player
        self.buttons[row][col].config(text=self.current_player)

        # Check for a win or a tie
        if self.check_winner(self.current_player):
            self.show_winner(self.current_player)
        elif self.is_tie():
            self.show_tie()
        else:
            # Switch players
            self.current_player = "O" if self.current_player == "X" else "X"

    def check_winner(self, player):
        # Check rows, columns, and diagonals for a win
        for i in range(3):
            if all(self.board[i][j] == player for j in range(3)):  # Check rows
                return True
            if all(self.board[j][i] == player for j in range(3)):  # Check columns
                return True
        # Check diagonals
        if all(self.board[i][i] == player for i in range(3)):
            return True
        if all(self.board[i][2 - i] == player for i in range(3)):
            return True
        return False

    def is_tie(self):
```

```
        # Check if the board is full
        for row in self.board:
            if "" in row:
                return False
        return True

    def show_winner(self, player):
        messagebox.showinfo("Tic Tac Toe", f"Player {player} wins!")
        self.reset_game()

    def show_tie(self):
        messagebox.showinfo("Tic Tac Toe", "It's a tie!")
        self.reset_game()

    def reset_game(self):
        # Reset the board and buttons for a new game
        self.board = [["" for _ in range(3)] for _ in range(3)]
        for row in range(3):
            for col in range(3):
                self.buttons[row][col].config(text="")
        self.current_player = "X"

    def run(self):
        self.window.mainloop()

# Run the game
if __name__ == "__main__":
```

```
game = TicTacToe()

game.run()
```

Flowchart:-



RIPHAH
INTERNATIONAL UNIVERSITY

Time Complexity:

1. **Game Initialization (`__init__` method):**
   - Creating the board and button grid: The nested list comprehension to create a 3x3 grid of empty strings and buttons takes constant time O(1)O(1)O(1) since the grid size is fixed (3x3).
2. **Button Click (`on_click` method):**
   - After a button click, the game checks for a win or tie.
   - **Check for winner (`check_winner` method):** This method checks each row, column, and diagonal, iterating through the grid. There are 3 rows, 3 columns, and 2 diagonals, so the checks are constant in time, O(1)O(1)O(1).
   - **Check for tie (`is_tie` method):** This checks if the board is full, which involves checking 9 cells, so it's O(1)O(1)O(1) in terms of grid size.
3. **Overall Time Complexity:**
   Every action (button click) involves checking for a winner and checking for a tie, both of which have constant time complexity O(1)O(1)O(1). Therefore, the overall time complexity for each button click action is O(1)O(1)O(1).

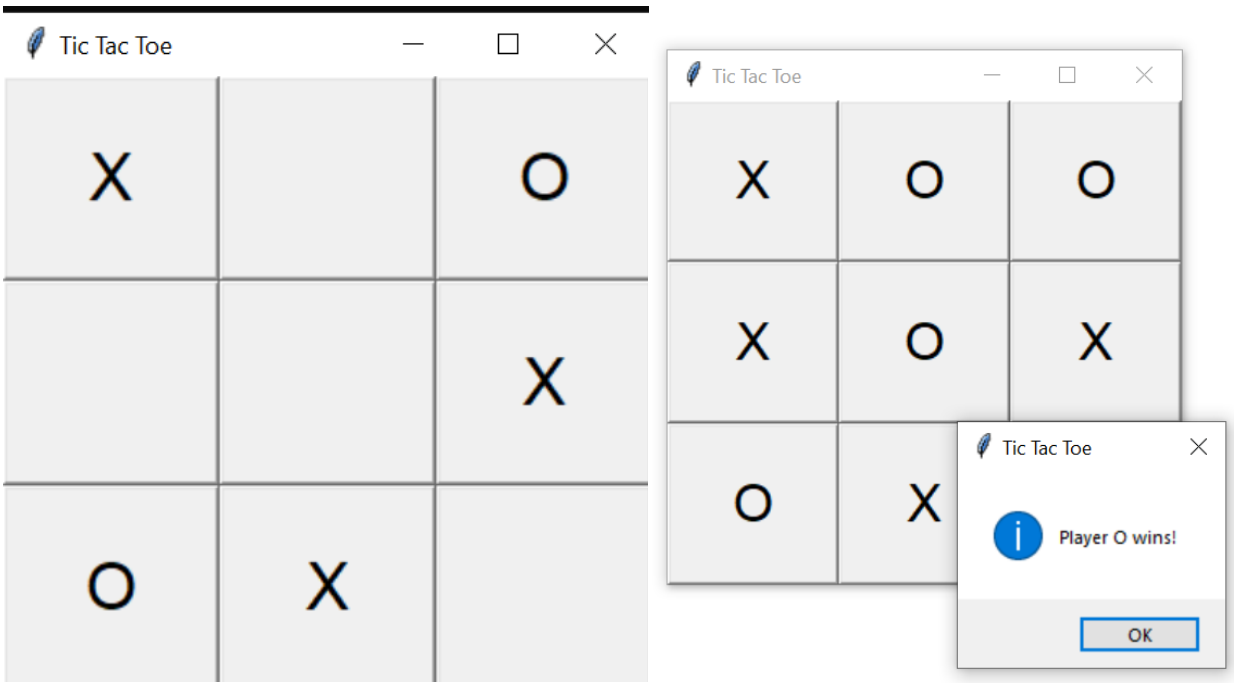
Space Complexity:

1. **Board and Buttons:**
   - The board is a 3x3 grid, so it requires space for 9 elements. This takes O(1)O(1)O(1) space since the grid size is constant.
   - The buttons are also arranged in a 3x3 grid, requiring space for 9 button objects, which also takes O(1)O(1)O(1) space.
2. **Overall Space Complexity:**
   The space complexity is constant O(1)O(1)O(1), as both the board and button grid have a fixed size (3x3). There is no dynamic space allocation that grows with input size.

Summary:

- **Time Complexity:** O(1)O(1)O(1) per button click.
- **Space Complexity:** O(1)O(1)O(1)

Screenshots:-



Conclusion:

This project demonstrates how to implement a simple yet functional game using Tkinter in Python. It emphasizes key programming concepts such as event handling, GUI components, and game logic. The result is an interactive and enjoyable game for two players to engage in.