**FUTURENSE**

TIC-TAC-TOE

**\_\_\_**

By Josh Hanley Thomas, Abhilash Santhosh, Arun Siby, Shreyaan S K

# INTRODUCTION

The Tic-Tac-Toe game is a well-known and straightforward two-player game that involves placing 'X' or 'O' symbols on a 3x3 grid. This project focuses on implementing a console-based version of Tic-Tac-Toe in Python. Players take turns making moves until there is a winner with three symbols in a row, column, or diagonal, or the game ends in a tie when the entire board is filled.

# IMPLEMENTATION

The game board is represented as a 3x3 grid using a 2D list, with each cell initially containing an empty space (' '). A `print\_board` function is created to visually display the current state of the board in the console. Players interact with the game by inputting row and column indices to make their moves. Input validation ensures that the entered move is within the board boundaries and that the selected cell is unoccupied.

The game logic is encapsulated within the `tic\_tac\_toe` function, which serves as the main game loop. Players alternate turns, and after each move, the game checks for a winner by examining rows, columns, and diagonals. If a winner is found, the game announces the winning player. If the board is full with no winner, the game declares a tie.

# CODE

import tkinter as tk

from tkinter import messagebox

def print\_board(board):

    """

    Prints the current state of the tic-tac-toe board.

    """

    for row in board:

        print(" | ".join(row))

        print("-" \* 9)

def check\_winner(board):

    """

    Checks if there's a winner on the tic-tac-toe board.

    """

    for row in board:

        if all(cell == row[0] and cell != ' ' for cell in row):

            return True

    for col in range(3):

        if all(board[row][col] == board[0][col] and board[row][col] != ' ' for row in range(3)):

            return True

    if all(board[i][i] == board[0][0] and board[i][i] != ' ' for i in range(3)) or \

       all(board[i][2 - i] == board[0][2] and board[i][2 - i] != ' ' for i in range(3)):

        return True

    return False

def is\_board\_full(board):

    """

    Checks if the tic-tac-toe board is full.

    """

    return all(cell != ' ' for row in board for cell in row)

def on\_click(row, col):

    """

    Handles the player's move when a button is clicked.

    """

    global current\_player, board\_buttons

    if board[row][col] == ' ':

        board[row][col] = current\_player

        board\_buttons[row][col].config(text=current\_player, state='disabled', disabledforeground='black')

        if check\_winner(board):

            messagebox.showinfo("Winner", f"Player {current\_player} wins!")

            reset\_board()

        elif is\_board\_full(board):

            messagebox.showinfo("Tie", "It's a tie!")

            reset\_board()

        else:

            current\_player = 'O' if current\_player == 'X' else 'X'

        update\_turn\_label()

def reset\_board():

    """

    Resets the tic-tac-toe board.

    """

    global board\_buttons, current\_player, board

    for row in range(3):

        for col in range(3):

            board[row][col] = ' '

            board\_buttons[row][col].config(text=' ', state='normal')

    current\_player = 'X'

    update\_turn\_label()

def update\_turn\_label():

    """

    Updates the label to display the current player's turn.

    """

    turn\_label.config(text=f"Turn: Player {current\_player}")

# Initialize the main window

root = tk.Tk()

root.title("Tic Tac Toe")

# Initialize the game variables

board = [[' ' for \_ in range(3)] for \_ in range(3)]

current\_player = 'X'

# Create label for player's turn

turn\_label = tk.Label(root, text=f"Turn: Player {current\_player}", font=('Helvetica', 14))

turn\_label.grid(row=0, column=0, columnspan=3)

# Create buttons for the board

board\_buttons = []

for row in range(3):

    button\_row = []

    for col in range(3):

        button = tk.Button(root, text=' ', font=('Helvetica', 24), width=6, height=2,

                           command=lambda row=row, col=col: on\_click(row, col))

        button.grid(row=row+1, column=col, padx=5, pady=5)

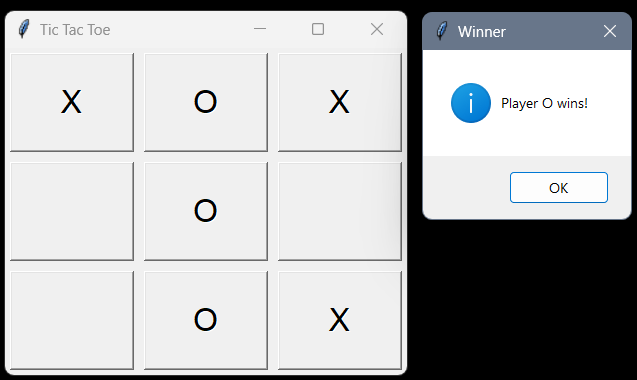
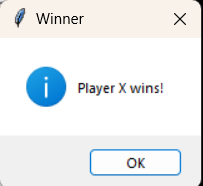
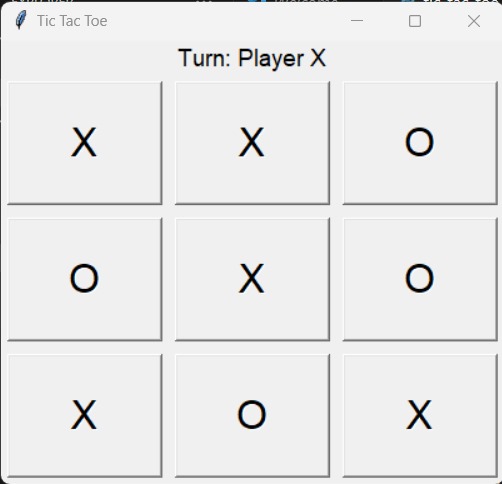
        button\_row.append(button)

    board\_buttons.append(button\_row)

# Start the main event loop

root.mainloop()

# INTERFACE



**Game Board Representation**:

The game board is displayed as a 3x3 grid of buttons, where each button represents a cell on the board.

Players can interact with the game by clicking on the buttons to place their symbols ('X' or 'O').

**Clickable Buttons**:

Buttons are configured with event handlers to detect clicks, allowing players to make their moves by clicking on the desired cell.

Upon clicking a button, the corresponding cell on the game board is updated with the player's symbol ('X' or 'O').

***Packages Used***:

**tkinter**:

Purpose: Tkinter is a standard Python GUI toolkit used for creating graphical user interfaces.

Functionality: It provides a set of tools and widgets for building interactive applications with buttons, labels, frames, and more.

Usage in the Game: Tkinter is used to create the game's graphical interface, including the game board grid represented by buttons and message boxes for displaying game outcomes.

**tkinter.messagebox**:

Purpose: The messagebox module is a sub-module of Tkinter used for displaying various types of message boxes, such as information, warning, error, and question boxes.

**Functionality**: It provides a simple interface for showing dialog boxes with messages to users.

**Usage in the Game**: The messagebox module is used to display messages to players, informing them of game outcomes such as winning conditions or tie situations.

# LOGIC

1. Main Game Loop:

The game revolves around a continuous loop that executes until a winner is determined or the board is full. Players make moves, and the game alternates between displaying the current state of the board and prompting players for their input.

2. Player Moves:

Players input their moves by specifying row and column indices. The game ensures that the move is valid by checking if it is within the board boundaries and if the selected cell is unoccupied.

3. Checking for a Winner:

After each move, the game checks for a winner by examining the rows, columns, and diagonals of the board. It looks for a sequence of three identical symbols in a row, column, or diagonal to declare a winner.

4. Checking for a Tie:

To determine a tie, the game checks if the board is completely filled with symbols and no winner has been declared. If all cells are occupied and no winner is found, the game concludes as a tie.

5. Switching Players:

After each move, the current player is switched to the other symbol ('X' to 'O' or vice versa). This ensures that players alternate turns throughout the game.

6. Conclusion of the Game:

The game loop concludes when either a winner is determined or the board is full, resulting in a tie. The game announces the winner or declares a tie and then exits the loop.

# CONCLUSION

In conclusion, this implementation of Tic-Tac-Toe in Python provides a fundamental understanding of game development concepts. The game offers an interactive and engaging experience for players while adhering to the classic rules of Tic-Tac-Toe. It serves as a foundation for further exploration into more advanced game development concepts and features.