TIC TAC TOE GAME

A PROJECT REPORT

Submitted by
NISHU YADAV 24BCS12967
ANUSHI SHARMA 24BCS12976
MANYATA 24BCS12998

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

Month, 2025

TIC TAC TOE GAME

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1. Indroduction to Project:

The project **"Tic Tac Toe Game using Python"** is a simple console-based game that aims to demonstrate fundamental programming logic, algorithm design, and problem-solving techniques. Tic Tac Toe, also known as "Noughts and Crosses", is a two-player game played on a 3×3 grid. Each player takes turns marking a space with either "X" or "O", and the objective is to get three marks in a row—either horizontally, vertically, or diagonally.

This project has been developed using **Python**, a high-level programming language known for its simplicity and readability. The game operates entirely through the terminal and uses basic control structures such as loops, conditionals, and functions.

The primary motivation behind this project is to create an interactive, educational, and enjoyable program that strengthens understanding of **Python fundamentals**, logic-building, and game flow control. The project demonstrates:

- Use of functions for modular programming.
- Decision-making using conditionals.
- Data structure management using lists.
- Logical checking for win and draw conditions.

This project serves as an introductory programming task suitable for beginners and highlights the use of algorithmic thinking to simulate real-world game logic.

1.2 Identification of Problem:

Traditionally, the Tic Tac Toe game is played on paper, but manual play does not always ensure fairness or instant result validation. The identified problems include:

- Difficulty in manually tracking all winning conditions.
- No automated mechanism to detect invalid moves.
- Lack of immediate feedback for draw or victory.
- Time-consuming setup and restart after every match.
- No record of player turns and move validation.

To overcome these challenges, a Python-based automated Tic Tac Toe program has been developed, which can accurately:

- Validate each move in real-time.
- Detect wins, losses, and draws.
- Display updated boards after every move.
- Ensure smooth switching between players "X" and "O".

CHAPTER 2
BACKGROUND STUDY

2.1 Existing solutions:

Many existing Tic Tac Toe implementations exist in various programming languages, ranging from simple console-based scripts to advanced GUI applications. Some use Python's built-in terminal for interaction, while others utilize libraries like **tkinter** or **pygame** for a graphical interface.

However, many of these versions are either too complex for beginners or overly simplistic without explanations of logic. This project focuses on a **clean, console-based version** that prioritizes **understanding over aesthetics**, making it ideal for learning core programming concepts.

2.2 Problem Definition:

The main goal is to design a simple, efficient, and user-friendly Python program that allows two players to play Tic Tac Toe interactively in the console environment. The program should:

- Represent a 3×3 game board using Python data structures.
- Validate moves and prevent overwriting existing cells.
- Alternate turns between two players automatically.
- Detect all possible win or draw scenarios accurately.
- Provide instant feedback and a clear game interface**.**

2.3Goals/Objectives:

- Develop a functional Tic Tac Toe game using Python.
- Use **modular programming** through functions for clarity.
- Implement **input validation** for accurate gameplay.
- Display a **real-time updated game board** after each turn.
- Determine the winner or declare a draw automatically.
- Strengthen logical and analytical problem-solving skills.

CHAPTER 3

DESIGN FLOW/PROCESS

3.1 Evaluation & Selection of Specifications / Features

After careful evaluation, the following features were finalized for the project:

- **Interactive Console Interface:** User-friendly text-based display for both players.
- **Automatic Turn Handling:** Alternating turns between “X” and “O”.
- **Move Validation:** Ensures the player cannot choose an already occupied cell.
- **Win Checking Mechanism:** Detects all eight possible winning patterns.
- **Draw Condition Detection:** Identifies when no further moves are possible.
- **Restart Option:** Allows restarting the game without re-running the script.

Each feature was designed to enhance usability, clarity, and educational value for Python learners.

3.2 Analysis of Features and finalization subject to constraints:

To To ensure efficiency and correctness, the following features and constraints were incorporated:

1. **Data Structure:**
   - The game board is implemented as a Python **list** with 9 elements representing 9 cells.
   - Each cell holds a value: "X", "O", or " " (empty).
2. **Input Validation:**
   - User inputs are checked to ensure valid numeric range (1–9).
   - Moves are rejected if the selected cell is already occupied.
3. **Turn Control:**
   - The game alternates between Player X and Player O after every valid move.
4. **Winning Condition Check:**
   - The program checks eight possible combinations:
     - 3 rows, 3 columns, and 2 diagonals.
   - A function check_winner() validates if the current player has achieved any of these patterns.
5. **Draw Condition:** ○ If all nine cells are filled and no player has won, the game declares a **Draw**.
6. **Reusability:** Functions are designed modularly (e.g., print_board(), check_winner(), is_full()), making the code reusable and easy to expand into GUI or AI versions

3.3 Design Flow:

Phase 1: Requirement Analysis

- Understanding game rules and user interaction flow.
- Defining core functions needed to execute gameplay.

Phase 2: Database Initialization

Creating modular functions for each process:

- Display board
- Handle input
- Check for win/draw
- Switch turns

Phase 3: Algorithm Flow

- Initialize an empty board of 9 cells.
- Assign Player X as the starting player.
- Prompt the player for a move (1–9).
- Validate the move. If invalid, prompt again.
- Update the board with the player's symbol.
- Display the updated board.
- Check for winning conditions.
- If a player wins → announce winner and end game.
- If board is full → announce draw.
- Switch player and repeat steps 3–9.

Phase 4: Testing and Debugging

- Tested for valid/invalid inputs.
- Checked detection for all possible win conditions.
- Ensured draw recognition and correct player switching.

CHAPTER 4
RESULTS ANALYSIS AND VALIDATION

4.1 Implementation of the Solution

main.py

Share Run

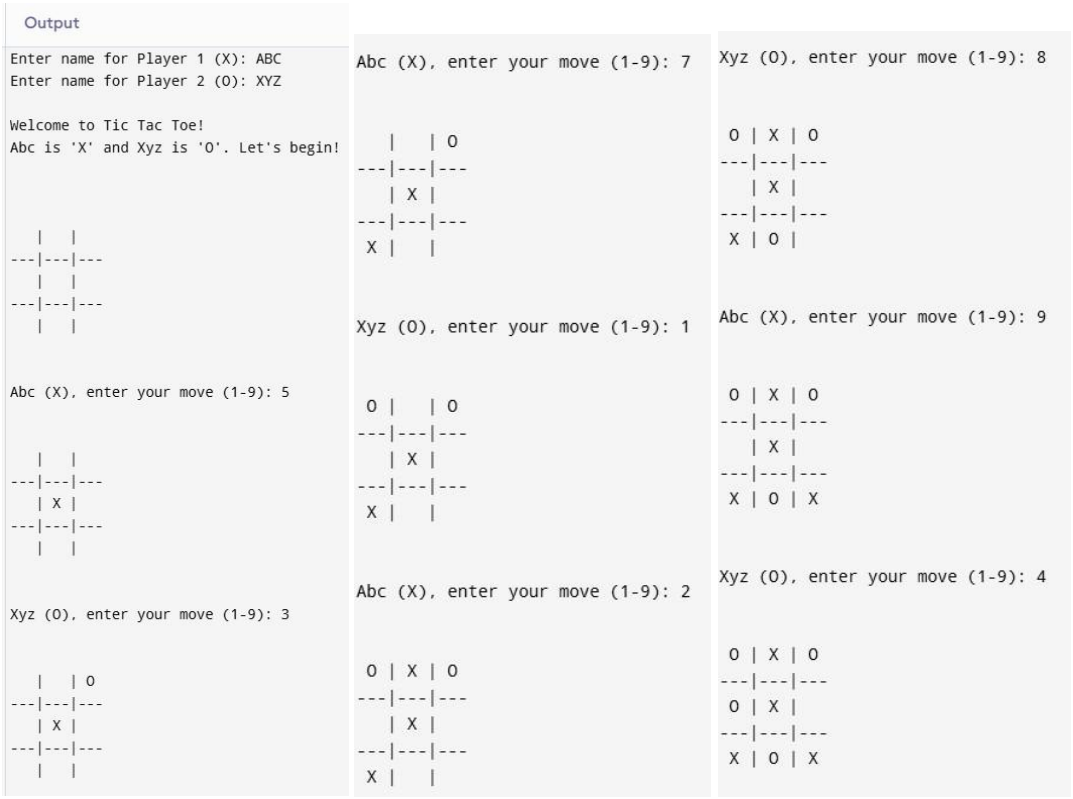
```
# Tic Tac Toe Game

# Function to display the board
def print_board(board):
    print("\n")
    print(f" {board[0]} | {board[1]} | {board[2]} ")
    print("---|---|---")
    print(f" {board[3]} | {board[4]} | {board[5]} ")
    print("---|---|---")
    print(f" {board[6]} | {board[7]} | {board[8]} ")
    print("\n")

# Function to check for a win
def check_winner(board, player):
    win_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],  # rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8],  # columns
        [0, 4, 8], [2, 4, 6]              # diagonals
    ]
    for combo in win_combinations:
        if all(board[i] == player for i in combo):
            return True
    return False

# Function to check if the board is full
def is_full(board):
    return all(cell != " " for cell in board)

# Main game loop
def play_game():
    board = [" "] * 9

    # Ask for player names
    player1 = input("Enter name for Player 1 (X): ").strip().capitalize() or "Player 1"
    player2 = input("Enter name for Player 2 (O): ").strip().capitalize() or "Player 2"

```

```
    players = {"X": player1, "O": player2}
    current_player = "X"

    print("\nWelcome to Tic Tac Toe!")
    print(f"{player1} is 'X' and {player2} is 'O'. Let's begin!\n")
    print_board(board)

    while True:
        try:
            move = int(input(f"{players[current_player]} ({current_player}), enter your
                move (1-9): ")) - 1
            if move < 0 or move > 8 or board[move] != " ":
                print("Invalid move! Try again.")
                continue
        except ValueError:
            print("Please enter a valid number between 1 and 9.")
            continue

        board[move] = current_player
        print_board(board)

        if check_winner(board, current_player):
            print(f"🎉 {players[current_player]} ({current_player}) wins! 🎉")
            break

        if is_full(board):
            print("It's a draw! 🤝")
            break

        # Switch player
        current_player = "O" if current_player == "X" else "X"

# Run the game
if __name__ == "__main__":
    play_game()
```

Testing and Validation

- Tested all 8 win conditions (rows, columns, diagonals). ☑
- Tested invalid inputs (non-numeric, repeated moves). ☑
- Verified draw detection after 9 moves. ☑
- Ensured correct alternation of turns. ☑

OUTPUT:



```
Output

Enter name for Player 1 (X): ABC
Enter name for Player 2 (O): XYZ

Welcome to Tic Tac Toe!
Abc is 'X' and Xyz is 'O'. Let's begin!

   |   |
---|---|---
   |   |
---|---|---
   |   |

Abc (X), enter your move (1-9): 5

   |   |
---|---|---
   | X |
---|---|---
   |   |

Xyz (O), enter your move (1-9): 3

   |   | O
---|---|---
   | X |
---|---|---
   |   |

Abc (X), enter your move (1-9): 7

   |   | O
---|---|---
   | X |
---|---|---
 X |   |

Xyz (O), enter your move (1-9): 1

 O |   | O
---|---|---
   | X |
---|---|---
 X |   |

Abc (X), enter your move (1-9): 2

 O | X | O
---|---|---
   | X |
---|---|---
 X |   |

Xyz (O), enter your move (1-9): 8

 O | X | O
---|---|---
   | X |
---|---|---
 X | O |

Abc (X), enter your move (1-9): 9

 O | X | O
---|---|---
   | X |
---|---|---
 X | O | X

Xyz (O), enter your move (1-9): 4

 O | X | O
---|---|---
 O | X |
---|---|---
 X | O | X
```

```
Abc (X), enter your move (1-9): 6

 O | X | O
---|---|---
 O | X | X
---|---|---
 X | O | X

It's a draw! 🤝

=== Code Execution Successful ===
```

CHAPTER 5.
CONCLUSION AND FUTUREWORK

5.1 Conclusion:

The **Tic Tac Toe Game in Python** project successfully demonstrates the use of basic programming concepts such as loops, lists, conditionals, and functions. It provides a practical example of algorithmic logic and turn-based interaction. The project meets all defined objectives, ensuring accurate game outcomes and smooth player interaction.

The simplicity of the implementation makes it an excellent beginner project for understanding structured programming and game design logic.

5.2 Future Work Future enhancements may include:

Future enhancements can include:

- **Graphical Interface:** Integrate GUI using tkinter or pygame.
- **AI Opponent:** Implement Minimax algorithm for single-player mode.
- **Scoreboard:** Maintain scores across multiple rounds.
- **Sound & Animation:** Add visual and audio feedback for better engagement.
- **Online Multiplayer:** Enable gameplay over a local or cloud network.

These improvements would make the project more interactive, modern, and closer to real-world gaming applications.