PROJECT REPORT

ON

Tic-Tac-Toe Game in Python

Submitted by:

Hitansh Sharma

Registration Number: 25BAI11433

November 24, 2025

Contents

Abstract

This project report presents the design and implementation of a classic Tic-Tac-Toe game using Python programming language. The game allows two players to compete against each other in a turn-based format on a 3×3 grid. The implementation demonstrates fundamental programming concepts including data structures, control flow, user input validation, and game logic implementation.

1 Introduction

Tic-Tac-Toe, also known as Noughts and Crosses, is a two-player game played on a 3×3 grid. Players take turns marking spaces with their respective symbols (X and O), with the objective of placing three of their marks in a horizontal, vertical, or diagonal row to win the game.

1.1 Objectives

The main objectives of this project are:

- To implement a fully functional Tic-Tac-Toe game in Python
- To create an interactive command-line interface for gameplay
- To implement game logic for win detection and draw conditions
- To validate user inputs and handle errors gracefully
- To demonstrate proficiency in Python programming

1.2 Scope

This project implements a console-based two-player Tic-Tac-Toe game with the following features:

- Visual representation of the game board
- Turn-based gameplay between two human players
- Input validation and error handling
- Win condition detection
- Draw condition detection

2 System Requirements

2.1 Software Requirements

- Operating System: Windows/Linux/macOS
- Python Version: Python 3.x or higher
- No external libraries required (uses only Python standard library)

2.2 Hardware Requirements

- Processor: Any modern processor
- RAM: 512 MB minimum
- Storage: 10 MB free space

3 Design and Implementation

3.1 Program Structure

The program is structured into the following main components:

3.1.1 Data Structure

The game board is represented as a list of 9 elements, where each element represents a position on the 3×3 grid:

```
board = [" ", " ", " ",
         " ", " ", " ",
         " ", " ", " "]
```

3.1.2 Board Display Function

The print_board() function displays the current state of the game board in a visually appealing format with grid lines separating each cell.

3.1.3 Win Checking Logic

The check_winner() function evaluates all possible winning combinations:

- Three horizontal rows
- Three vertical columns
- Two diagonal lines

3.1.4 Main Game Loop

The main game loop handles:

- Turn management between players
- User input collection and validation
- Move execution
- Win/draw condition checking

3.2 Algorithm and Flowchart

3.2.1 Game Algorithm

1. Initialize an empty 3×3 board
2. Set the current player to X
3. For each turn (maximum 9 turns):
   (a) Display the current board
   (b) Prompt the current player for input
   (c) Validate the input (1-9, position available)
   (d) Place the player's mark on the board
   (e) Check if the current player has won
   (f) If won, display winner and exit
   (g) Switch to the other player
4. If all 9 turns complete without a winner, declare a draw

4 Source Code

```python
# Create the board
board = [" ", " ", " ",
         " ", " ", " ",
         " ", " ", " "]

# Function to print the board
def print_board():
    print()
    print(board[0] + " | " + board[1] + " | " + board[2])
    print("---+---+---")
    print(board[3] + " | " + board[4] + " | " + board[5])
    print("---+---+---")
    print(board[6] + " | " + board[7] + " | " + board[8])
    print()

# Check for a winner
def check_winner(player):
    # All winning combinations
    wins = [
        [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 wins:
        if board[combo[0]] == board[combo[1]] == board[combo[2]] == player:
            return True
    return False

# Main game
current_player = "X"

for turn in range(9):   # Max 9 moves
    print_board()
    print(f"Player {current_player}, choose a position (1-9):")

    while True:
        try:
            move = int(input()) - 1    # convert to index 0-8

            if move < 0 or move > 8:
                print("Invalid position! Choose 1-9.")
            elif board[move] != " ":
                print("Spot already taken! Choose another.")
            else:
                break
        except:
            print("Enter a number from 1 to 9.")

    # Place the move
    board[move] = current_player

    # Check if the player wins
    if check_winner(current_player):
        print_board()
        print(f"Player {current_player} wins!")
        break

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

else:
    print_board()
    print("It's a draw!")
```

5 Features and Functionality

5.1 Key Features

1. **Interactive Gameplay**: Players interact with the game through a simple numeric input system (1-9) corresponding to board positions.
2. **Visual Board Display**: The game board is clearly displayed after each move with grid lines for easy visualization.
3. **Input Validation**: The program validates user inputs to ensure:
   - Input is a number
   - Number is within valid range (1-9)
   - Selected position is not already occupied
4. **Win Detection**: Automatically detects when a player achieves three marks in a row, column, or diagonal.
5. **Draw Detection**: Identifies when all positions are filled without a winner.
6. **Error Handling**: Gracefully handles invalid inputs and provides helpful error messages.

5.2 Game Flow

The game follows this flow:

1. Game starts with Player X
2. Board is displayed
3. Current player enters position (1-9)
4. Input is validated
5. Move is placed on the board
6. Check for win condition
7. If no win, switch to other player
8. Repeat until win or draw

6 Testing and Results

6.1 Test Cases

6.1.1 Test Case 1: Normal Game Flow

Input: Valid moves leading to Player X win
Expected Output: Player X wins message
Result: Pass

6.1.2 Test Case 2: Draw Scenario

Input: Valid moves filling all positions without winner
Expected Output: ”It’s a draw!” message
Result: Pass

6.1.3 Test Case 3: Invalid Input - Out of Range

Input: Number outside 1-9 range (e.g., 10, 0, -1)
Expected Output: ”Invalid position! Choose 1-9.”
Result: Pass

6.1.4 Test Case 4: Invalid Input - Occupied Position

Input: Attempting to place mark on occupied position
Expected Output: ”Spot already taken! Choose another.”
Result: Pass

6.1.5 Test Case 5: Invalid Input - Non-numeric

Input: Letter or special character
Expected Output: ”Enter a number from 1 to 9.”
Result: Pass

6.1.6 Test Case 6: Win Detection - Row

Input: Player completes any horizontal row
Expected Output: Win detected immediately
Result: Pass

6.1.7 Test Case 7: Win Detection - Column

Input: Player completes any vertical column
Expected Output: Win detected immediately
Result: Pass

6.1.8 Test Case 8: Win Detection - Diagonal

Input: Player completes any diagonal
Expected Output: Win detected immediately
Result: Pass

7 Advantages and Limitations

7.1 Advantages

- Simple and intuitive user interface
- No external dependencies required
- Comprehensive input validation
- Clear visual feedback
- Lightweight and fast execution
- Easy to understand code structure
- Platform independent

7.2 Limitations

- Console-based interface only (no GUI)
- Two-player mode only (no AI opponent)
- No score tracking across multiple games
- No option to restart without rerunning the program
- Limited visual appeal compared to graphical versions

8 Future Enhancements

The following enhancements could be implemented to improve the game:

1. **Graphical User Interface (GUI)**
   - Implement using Tkinter or Pygame
   - Add visual themes and animations
2. **Artificial Intelligence**
   - Implement single-player mode with AI opponent
   - Add difficulty levels (Easy, Medium, Hard)
   - Use Minimax algorithm for optimal AI moves
3. **Game Statistics**
   - Track wins, losses, and draws
   - Maintain player profiles
   - Display leaderboards
4. **Additional Features**
   - Undo/Redo functionality
   - Save and load game states
   - Multiplayer over network
   - Customizable board sizes (4×4, 5×5)
   - Sound effects and background music
5. **Code Optimization**
   - Implement object-oriented design
   - Add configuration file support
   - Improve code modularity

9 Conclusion

This project successfully demonstrates the implementation of a functional Tic-Tac-Toe game using Python. The program effectively handles user interactions, validates inputs, and correctly determines game outcomes. The implementation showcases fundamental programming concepts including data structures (lists), control structures (loops and conditionals), functions, and error handling.

The game provides an enjoyable user experience through its clear interface and robust input validation. While the current implementation is console-based, it serves as a solid foundation for future enhancements such as GUI development and AI opponent integration.

Through this project, practical experience was gained in:

- Algorithm design and implementation
- User input handling and validation
- Game logic development
- Python programming best practices
- Problem-solving and debugging

The project meets all its initial objectives and provides a fully functional, playable Tic-Tac-Toe game that can be easily extended with additional features.

References

1. Python Software Foundation. (2024). Python Documentation. Retrieved from https://docs.python.org/3/
2. Lutz, M. (2013). *Learning Python*. O'Reilly Media, Inc.
3. Matthes, E. (2019). *Python Crash Course*. No Starch Press.
4. GeeksforGeeks. (2024). Python Game Development Tutorials. Retrieved from https://www.geeksforgeeks.org/
5. Real Python. (2024). Python Tutorials. Retrieved from https://realpython.com/