Tic-Tac-Toe Game in Python

A Detailed Documentation

Submitted by:
Anish Khadka (080BEL013)
Bigyan Adhikari (080BEL021)
Bikalpa Bhattarai (080BEL023)
Hritesh Rai (080BEL031)

Submitted to:
PCP Sir(Department of Electronics and Computer Engineering)

September 12, 2024

Abstract

This document provides in-depth documentation for a Python-based Tic-Tac-Toe game. The project is implemented using object-oriented programming, and this report covers the problem statement, the design, implementation, detailed code walkthrough, algorithms, flowcharts, and possible future improvements. The aim of this document is to provide a comprehensive understanding of how the game works both from a coding and a theoretical perspective.

Contents

Chapter 1

Introduction

1.1 Overview

Tic-Tac-Toe is a classic two-player game where players take turns marking spaces on a 3x3 grid. The objective of the game is for a player to place three of their marks in a horizontal, vertical, or diagonal row. This documentation outlines the Python implementation of the game using an object-oriented approach.

1.2 Problem Statement

The problem is to create a simple command-line version of Tic-Tac-Toe where:

- Two players take turns.
- The game continues until there's a winner or a tie.
- The game checks for a winner after each move.
- Invalid moves are handled gracefully.

1.3 Objective

The primary objective is to develop a Python program for the Tic-Tac-Toe game that:

- Is easy to play.
- Implements basic game mechanics.
- Provides a clear and understandable code structure.

Chapter 2

Design and Implementation

2.1 Class Design

The game is implemented using a single class, `TicTacToe`, that manages the game state, board, player moves, and winner determination.

2.1.1 TicTacToe Class

The `TicTacToe` class handles all the core logic of the game:

1. **Board initialization:** The board is initialized as a list of 9 empty spaces.
2. **Printing the board:** The game board is visually presented in a 3x3 layout.
3. **Move validation:** The game checks if a move is valid (i.e., if the chosen square is available).
4. **Winner determination:** The game checks if the latest move results in a win either through rows, columns, or diagonals.

2.2 Algorithms

2.2.1 Algorithm for Board Initialization

The board initialization involves creating a list of 9 spaces, each initialized to an empty string. This forms our 3x3 grid.

Algorithm 1 Board Initialization

```
Initialize board as an empty list of length 9.
for each position i from 0 to 8 do
    Set board[i] to ' ' (empty).
end for
Set current_winner to None.
```

2.2.2 Algorithm for Printing the Board

This algorithm prints the current state of the board in a 3x3 format.

Algorithm 2 Board Printing

```
for each row r in the board do
    Slice the board into rows.
    Print the row, separating each item with '—'.
end for
```

2.2.3 Algorithm for Making a Move

This algorithm checks if the move is valid (i.e., the square is empty) and places the player's mark on the board.

Algorithm 3 Make Move

```
Input: square, player_letter
if board[square] is empty then
    Place player_letter in board[square].
    if Check for a winner after the move then
        Set current_winner to player_letter.
    end if
    return True
else
    return False
end if
```

2.2.4 Algorithm for Checking a Winner

This algorithm checks if the current player has won by forming a line of three marks either horizontally, vertically, or diagonally.

Algorithm 4 Check Winner

```
Input: square, player_letter
Check the row of the square for three identical marks.
Check the column for three identical marks.
if the square is in a diagonal then
    Check both diagonals for three identical marks.
end if
if any of the above checks match then
    return True
else
    return False
end if
```

2.3 Flowchart

Below is the flowchart representing the overall flow of the Tic-Tac-Toe game. It shows how the game alternates between players, checks for a winner, and handles invalid moves.

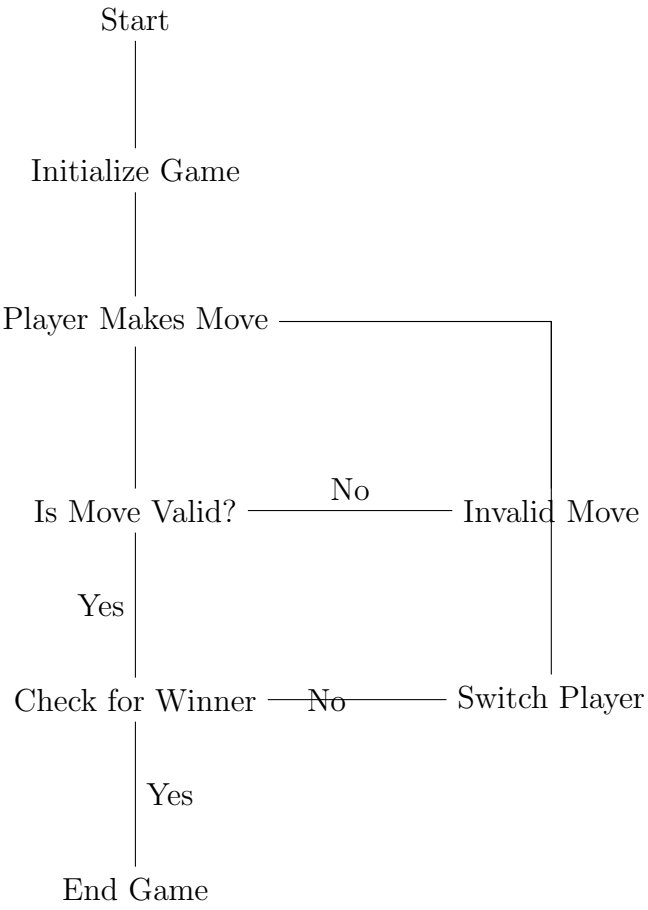

Figure 2.1: Flowchart of the Tic-Tac-Toe Game

This flowchart illustrates the following: - The game starts with board initialization. - The player makes a move. - The game checks if the move is valid. - If valid, the game checks for a winner. - If no winner is found, the turn switches to the next player. - If the move is invalid, the player is prompted to try again.

Chapter 3

Code Explanation

The following section explains the code in detail, with comments and explanations for each part of the implementation.

3.1 Imports

The project does not require any external libraries except for built-in Python functions.

3.2 TicTacToe Class

3.2.1 Class Initialization

Listing 3.1: Class Initialization

```
class TicTacToe:
    def __init__(self):
        self.board = [' ' for _ in range(9)]  # 3x3 board initialized
        self.current_winner = None  # No winner at the start
```

The constructor initializes the game board and sets the current winner to `None`.

3.2.2 Printing the Board

Listing 3.2: Printing the Board

```
def print_board(self):
    for row in [self.board[i * 3:(i + 1) * 3] for i in range(3)]:
        print('| ' + ' | '.join(row) + ' |')
```

This method slices the board list into rows and prints each row in a 3x3 grid format.

3.2.3 Available Moves

Listing 3.3: Available Moves

```
def available_moves(self):
    return [i for i, spot in enumerate(self.board) if spot == ' ']
```

This function returns a list of indices where the board is still empty.

3.2.4 Making a Move

Listing 3.4: Making a Move

```
def make_move(self, square, letter):
    if self.board[square] == ' ':
        self.board[square] = letter
        if self.winner(square, letter):
            self.current_winner = letter
        return True
    return False
```

This function places the player's letter ('X' or 'O') on the board, checks if the move results in a win, and returns `True` if the move is valid.

3.2.5 Checking for a Winner

Listing 3.5: Checking for a Winner

```
def winner(self, square, letter):
    row_ind = square // 3
    row = self.board[row_ind * 3:(row_ind + 1) * 3]
    if all([spot == letter for spot in row]):
        return True

    col_ind = square % 3
    column = [self.board[col_ind + i * 3] for i in range(3)]
    if all([spot == letter for spot in column]):
        return True

    if square % 2 == 0:
        diagonal1 = [self.board[i] for i in [0, 4, 8]]
        if all([spot == letter for spot in diagonal1]):
            return True
        diagonal2 = [self.board[i] for i in [2, 4, 6]]
        if all([spot == letter for spot in diagonal2]):
            return True
    return False
```

This function checks for three consecutive matching marks in rows, columns, or diagonals.

Chapter 4

Game Flow

4.1 Main Game Loop

The game loop alternates between two players and continues until there is a winner or a tie.

Listing 4.1: Main Game Loop

```
def play_game():
    game = TicTacToe()
    game.print_board()

    letter = 'X'
    while game.empty_squares():
        square = int(input(f"{letter}'s turn. Input move (0-8): "))
        if game.make_move(square, letter):
            game.print_board()
            if game.current_winner:
                print(f"{letter} wins!")
                return
            letter = 'O' if letter == 'X' else 'X'
        else:
            print("Invalid move. Try again.")

    print("It's a tie!")
```

The `play_game` function manages the game loop. It prompts players for input, updates the board, checks for a winner, and handles invalid moves.

Chapter 5

Testing and Results

5.1 Test Cases

The following test cases were performed:

- Test Case 1: Player 'X' wins via a horizontal row.
- Test Case 2: Player 'O' wins via a diagonal.
- Test Case 3: The game ends in a tie with no winner.

5.2 Results

The game successfully handled all scenarios, including detecting a winner and recognizing a tie.

Chapter 6

Future Improvements

6.1 Graphical User Interface (GUI)

A future enhancement could involve adding a graphical user interface (GUI) using libraries like `tkinter` or `pygame` to make the game more interactive.

6.2 AI Opponent

The current version is a two-player game. Adding an AI opponent using algorithms like minimax could be a valuable improvement.

Chapter 7

Conclusion

In this project, we successfully implemented a command-line version of the Tic-Tac-Toe game. The game is designed with clear logic to handle valid and invalid moves, determine a winner, and detect ties. This report serves as detailed documentation of the design, implementation, and testing process for the game.

Chapter 8

References

- Python official documentation: `https://docs.python.org/3/`
- Tic-Tac-Toe rules: `https://en.wikipedia.org/wiki/Tic-tac-toe`