Tic-Tac-Toe Solver Report

Tic-Tac-Toe Solver in Python

A project report on implementing a one-player Tic-Tac-Toe game using Python.

Prepared by: KRISHNA SINGH

CSE AI B (ROLL NO 65)

Date: 03/11/2025

Introduction

Tic-Tac-Toe is a classic two-player game played on a 3x3 grid. The objective is to align three symbols (either 'X' or 'O') horizontally, vertically, or diagonally. This project presents a Python-based Tic-Tac-Toe solver where a human player competes against an AI opponent. The AI makes moves by randomly selecting an available position on the board. The program follows a structured approach to check for a winner and handle user input.

Methodology

1. Board Representation: The game board is a 3x3 list of lists initialized with empty spaces (' ').

2. Game Flow: The game alternates between the player and the AI until a win or draw condition is met.

3. User Input Handling: The player selects a position by entering row and column numbers.

4. AI Move Selection: The AI randomly picks an available cell.

5. Win Condition Check: The game verifies if either the player or AI has won after each move.

6. Draw Condition Check: If all cells are filled with no winner, the game ends in a draw.

7. Loop Execution: The game continues running until a win or draw is detected.

Code Implementation

Tic-Tac-Toe Game in Python (1 Player vs Computer)

```
# Tic-Tac-Toe Game  (1 Player vs Computer)

import sys
import random

def print_board(board):
    """Function to print the Tic-Tac-Toe board in a formatted
way."""
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    """Function to check if a player has won the game."""
    # Check rows and columns
    for i in range(3):
        if all(board[i][j] == player for j in range(3)) or
all(board[j][i] == player for j in range(3)):
            return True

    # Check diagonals
    if all(board[i][i] == player for i in range(3)) or
all(board[i][2 - i] == player for i in range(3)):
        return True
```

```
    return False

def is_full(board):
    """Function to check if the board is full, resulting in a
draw."""
    return all(board[i][j] != " " for i in range(3) for j in
range(3))

def get_valid_input(board):
    """Function to get valid input from the player."""
    while True:
        try:
            row, col = map(int, input("Enter row and column (0-2)
separated by space: ").split())
            if 0 <= row < 3 and 0 <= col < 3:
                if board[row][col] == " ":
                    return row, col
                else:
                    print("Cell is already occupied. Try again.")
            else:
                print("Invalid input. Please enter numbers between 0
and 2.")
        except ValueError:
            print("Invalid input. Enter two numbers separated by
space.")

def get_computer_move(board):
    """Function for the computer to make a move by choosing a random
empty cell."""
    empty_cells = [(i, j) for i in range(3) for j in range(3) if
board[i][j] == " "]
    return random.choice(empty_cells)

def tic_tac_toe():
    """Main function to run the Tic-Tac-Toe game with one player vs
computer."""
    board = [[" " for _ in range(3)] for _ in range(3)]
    player_symbol = "X"
    computer_symbol = "O"
```

```
    turn = 0

    while True:
        print_board(board)
        if turn % 2 == 0:
            print("Player's turn")
            row, col = get_valid_input(board)
            board[row][col] = player_symbol
        else:
            print("Computer's turn")
            row, col = get_computer_move(board)
            board[row][col] = computer_symbol

        # Check for a winner or draw
        if check_winner(board, player_symbol):
            print_board(board)
            print("Player wins!")
            break
        if check_winner(board, computer_symbol):
            print_board(board)
            print("Computer wins!")
            break
        if is_full(board):
            print_board(board)
            print("It's a draw!")
            break

        turn += 1

# Run the game
if __name__ == "__main__":
    try:
        tic_tac_toe()
    except KeyboardInterrupt:
        print("\nGame interrupted. Exiting...")
        sys.exit(0)
```

https://github.com/krishna24up/krishnasinghcseaib/blob/main/KRISHNA%20SINGH%20CSEAIB

Screenshots of Output

Below are sample screenshots showing various stages of gameplay:

1. Initial Game Board:

```
  |   |
---------
  |   |
---------
  |   |
```

2. Player and AI Moves:

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

O | | X

```

3. Game End Conditions:

- If a player wins, the board is displayed along with a victory message.

- If all spaces are filled with no winner, a draw message is shown.

```
PS C:\Users\dell> python -u "c:\Users\dell\Downloads\n11"
  | |

  | |

  | |

Player's turn
Enter row and column (0-2) separated by space: 1 1
  | |

  | X |

  | |

Computer's turn
  | |

  | X |

O | |

Player's turn
Enter row and column (0-2) separated by space: 2 2
  | |

  | X |

O | | X

Computer's turn
  | |

```
```

```
Player's turn
Enter row and column (0-2) separated by space: 2 2
  |   |
---------
  | X |
---------
O |   | X
---------
Computer's turn
  |   |
---------
O | X |
---------
O |   | X
---------
Player's turn
Enter row and column (0-2) separated by space: 0 0
X |   |
---------
O | X |
---------
O |   | X
---------
Player wins!
PS C:\Users\dell>
```

Conclusion

This Python-based Tic-Tac-Toe solver provides an interactive gameplay experience against an AI opponent. The AI uses a simple random move strategy. Future improvements could include implementing a minimax algorithm to make the AI more competitive.

Possible Enhancements:

- Implementing a GUI using Tkinter or Pygame.

- Improving AI decision-making using the minimax algorithm.

- Adding difficulty levels for varied gameplay.

End of Report

THANKS