REPORT

Tic-Tac-Toe Solver Using Minimax Algorithm

Name: Harshit Patel

Course: B.Tech

Instructor: Mr. Shivansh

Date: 11-03-2025

Introduction

Tic-Tac-Toe is a popular game played on a 3x3 grid, where two players take turns marking the spaces with 'X' and 'O' symbols. The goal of the game is to get three of one's own symbols in a row, either horizontally, vertically, or diagonally. The game is typically played between two human players or between a human player and a computer.

This report presents the design and implementation of a **Tic-Tac-Toe solver** using the **Minimax algorithm**. The solver is capable of playing the game optimally, meaning it will always either win or force a draw. The algorithm simulates all possible moves in the game, evaluates them based on the game’s rules, and chooses the best move for the AI player.

The project includes the creation of a Python-based solver, which applies the Minimax algorithm to make decisions on the best possible move at any point in the game.

Methodology

Minimax Algorithm

The Minimax algorithm is a decision-making algorithm commonly used in game theory for two-player games. It works by simulating all possible future moves and evaluating each scenario. The algorithm operates recursively by alternating between the maximizer and minimizer.

- **Maximizing Player**: Player X (AI) aims to maximize their score.
- **Minimizing Player**: Player O (AI) aims to minimize the score.

Steps Involved:

1. **Evaluation Function**: The algorithm evaluates the board after each move. A score of `1` indicates a win for Player X, `-1` indicates a win for Player O, and `0` indicates a draw or ongoing game.
2. **Recursive Search**: The Minimax algorithm recursively explores each possible move at every level of the game tree until a terminal state is reached (win, loss, or draw).
3. **Optimal Move Selection**: Based on the evaluation, the algorithm chooses the move that maximizes its score (for Player X) or minimizes its opponent's score (for Player O).

Implementation in Python:

The Tic-Tac-Toe solver was implemented in Python, using a 3x3 list to represent the game board. The algorithm employs recursion to evaluate all possible moves for both players, returning the optimal move for the AI.

Main Features:

- **Game Initialization**: The game board is initialized as a 3x3 grid.
- **Optimal AI Moves**: The AI makes optimal moves by using the Minimax algorithm.
- **Win Conditions**: The game checks for a winner after each move.
- **Draw Condition**: The game checks for a draw if the board is full and no player has won.

Code Typed

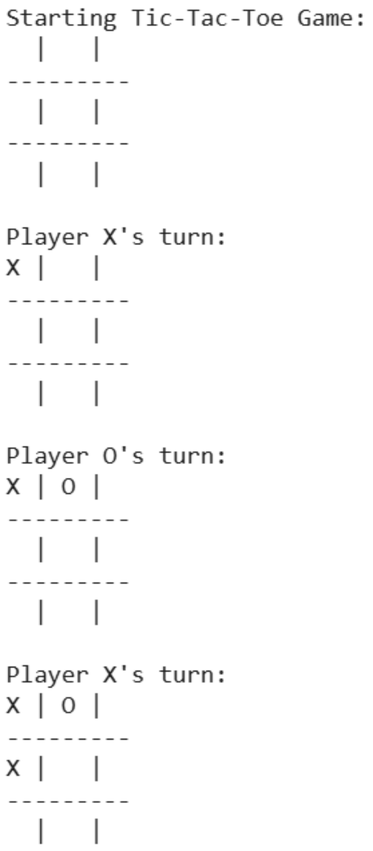
- python
- Copy
- import random

```
# Constants to represent players
PLAYER_X = 'X'
PLAYER_O = 'O'
EMPTY = ' '

# Function to print the current Tic-Tac-Toe board
def print_board(board):
    for row in range(3):
        print(f'{board[row][0]} | {board[row][1]} | {board[row][2]}')
        if row < 2:
            print('---------')

# Check if the current player has won
def check_winner(board, player):
    # Check rows, columns, and diagonals for a win
    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
    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

# Check if the board is full (i.e., a draw)
def is_board_full(board):
    for row in board:
        if EMPTY in row:
            return False
    return True

# Evaluate the board for Minimax algorithm (return scores)
def evaluate(board):
    if check_winner(board, PLAYER_X):
        return 1  # Player X wins
    elif check_winner(board, PLAYER_O):
        return -1  # Player O wins
    else:
        return 0  # Draw or ongoing
```

```
# Minimax algorithm to find the best move
def minimax(board, depth, is_maximizing):
    score = evaluate(board)

    # If the game is over (either player wins or draw)
    if score == 1 or score == -1 or is_board_full(board):
        return score

    if is_maximizing:
        best = -float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = PLAYER_X
                    best = max(best, minimax(board, depth + 1, not
is_maximizing))
                    board[i][j] = EMPTY
        return best
    else:
        best = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == EMPTY:
                    board[i][j] = PLAYER_O
                    best = min(best, minimax(board, depth + 1, not
is_maximizing))
                    board[i][j] = EMPTY
        return best

# Function to find the best move for the current player (Player X)
def find_best_move(board):
    best_val = -float('inf')
    best_move = (-1, -1)

    # Traverse all the cells to find the best move
    for i in range(3):
        for j in range(3):
            if board[i][j] == EMPTY:
                board[i][j] = PLAYER_X
                move_val = minimax(board, 0, False)
                board[i][j] = EMPTY
                if move_val > best_val:
                    best_move = (i, j)
                    best_val = move_val
    return best_move
```

```
# Function to play a game between two players (AI vs AI)
def play_game():
    board = [[EMPTY for _ in range(3)] for _ in range(3)]

    print("Starting Tic-Tac-Toe Game:")
    print_board(board)

    current_player = PLAYER_X  # Player X starts first

    while True:
        if current_player == PLAYER_X:
            print("\nPlayer X's turn:")
            i, j = find_best_move(board)
            board[i][j] = PLAYER_X
        else:
            print("\nPlayer O's turn:")
            i, j = find_best_move(board)
            board[i][j] = PLAYER_O

        print_board(board)

        if check_winner(board, PLAYER_X):
            print("\nPlayer X wins!")
            break
        elif check_winner(board, PLAYER_O):
            print("\nPlayer O wins!")
            break
        elif is_board_full(board):
            print("\nIt's a draw!")
            break

        # Switch players
        current_player = PLAYER_O if current_player == PLAYER_X else PLAYER_X

# Run the game
play_game()
```

Screenshots of Output



```
Starting Tic-Tac-Toe Game:
  |   |  
---------
  |   |  
---------
  |   |  

Player X's turn:
X |   |  
---------
  |   |  
---------
  |   |  

Player O's turn:
X | O |  
---------
  |   |  
---------
  |   |  

Player X's turn:
X | O |  
---------
X |   |  
---------
  |   |  
Player O's turn:
X | O | O
---------
X |   |  
---------
  |   |  

Player X's turn:
X | O | O
---------
X | X |  
---------
  |   |  

Player O's turn:
X | O | O
---------
X | X | O
---------
  |   |  

Player X's turn:
X | O | O
---------
X | X | O
---------
X |   |  

Player X wins!
```