**Tic-Tac-Toe Solver Report**

**1. Title Page**

**Project Title**: Tic-Tac-Toe Solver  
**Author**: Atharv Tayal  
**Institution**: Kiet Group of Institutions  
**Date**: 11/03/2025

## 2. Introduction

Tic-Tac-Toe is a classic two-player strategy game played on a 3x3 grid. The players take turns marking their respective positions with 'X' and 'O' until one player forms a row, column, or diagonal with three of their symbols. The game ends either when one player wins or when the board is full, resulting in a draw.

This report presents the implementation of a **Tic-Tac-Toe Solver** that uses the **Minimax algorithm** to create an unbeatable AI opponent. The AI evaluates all possible game states, recursively explores each move, and selects the optimal one. As a result, the AI is always able to select the best move, making the game a challenging experience for the user.

## 3. Methodology

The Tic-Tac-Toe solver uses a structured methodology to simulate the game while incorporating an AI that never loses. Here's the detailed approach:

### 3.1 Game Representation

* The game board is represented as a **3x3 matrix**.
* Each cell of the matrix can be empty (' '), contain 'X', or contain 'O'.
* The board is printed in a user-friendly format after each move.

### 3.2 User Input & AI Response

* The player selects a row and column (1-3) to place their 'X' mark.
* The AI uses the **Minimax algorithm** to evaluate possible future board states and picks the best move for 'O'.

### 3.3 AI Decision Making (Minimax Algorithm)

* **Evaluation of Moves**: The algorithm assigns scores to all possible moves.
* **Recursion & Backtracking**: The AI recursively explores future game states to identify the optimal move.
* **Best Move Selection**: The AI always selects the move that maximizes its chances of winning while minimizing the player's winning opportunities.

### 3.4 Game Loop & Win Conditions

* The game continues in turns until:
  + A player wins (three of the same marks in a row, column, or diagonal).
  + The board is full, resulting in a draw.
* The program then outputs the result (winner or draw) and terminates the game.

## 4. Code Typed

The following Python code implements the **Tic-Tac-Toe Solver** using the **Minimax algorithm**:

python

Copy

# Function to print the game board

def print\_board(board):

for row in board:

print(" | ".join(row))

print("-" \* 9)

# Function to check if a player has won

def check\_winner(board):

for i in range(3):

if board[i][0] == board[i][1] == board[i][2] and board[i][0] != ' ':

return board[i][0]

if board[0][i] == board[1][i] == board[2][i] and board[0][i] != ' ':

return board[0][i]

if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':

return board[0][0]

if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':

return board[0][2]

return None

# Function to check if the board is full (draw condition)

def is\_draw(board):

return all(cell != ' ' for row in board for cell in row)

# Minimax algorithm for optimal AI move

def minimax(board, is\_maximizing):

winner = check\_winner(board)

if winner == 'X': return -1

if winner == 'O': return 1

if is\_draw(board): return 0

if is\_maximizing:

best\_score = -2

for i in range(3):

for j in range(3):

if board[i][j] == ' ':

board[i][j] = 'O'

score = minimax(board, False)

board[i][j] = ' '

best\_score = max(score, best\_score)

return best\_score

else:

best\_score = 2

for i in range(3):

for j in range(3):

if board[i][j] == ' ':

board[i][j] = 'X'

score = minimax(board, True)

board[i][j] = ' '

best\_score = min(score, best\_score)

return best\_score

# Function to determine AI's best move

def best\_move(board):

best\_score = -2

move = (-1, -1)

for i in range(3):

for j in range(3):

if board[i][j] == ' ':

board[i][j] = 'O'

score = minimax(board, False)

board[i][j] = ' '

if score > best\_score:

best\_score = score

move = (i, j)

return move

# Main function to run the game

def play\_tic\_tac\_toe():

board = [[' ' for \_ in range(3)] for \_ in range(3)]

player\_turn = True

while True:

print\_board(board)

if check\_winner(board) or is\_draw(board):

break

if player\_turn:

row, col = map(int, input("Enter row and column (1-3): ").split())

row -= 1

col -= 1

if 0 <= row < 3 and 0 <= col < 3 and board[row][col] == ' ':

board[row][col] = 'X'

player\_turn = False

else:

print("Invalid move! Try again.")

else:

row, col = best\_move(board)

board[row][col] = 'O'

player\_turn = True

print\_board(board)

winner = check\_winner(board)

if winner:

print("Winner:", winner)

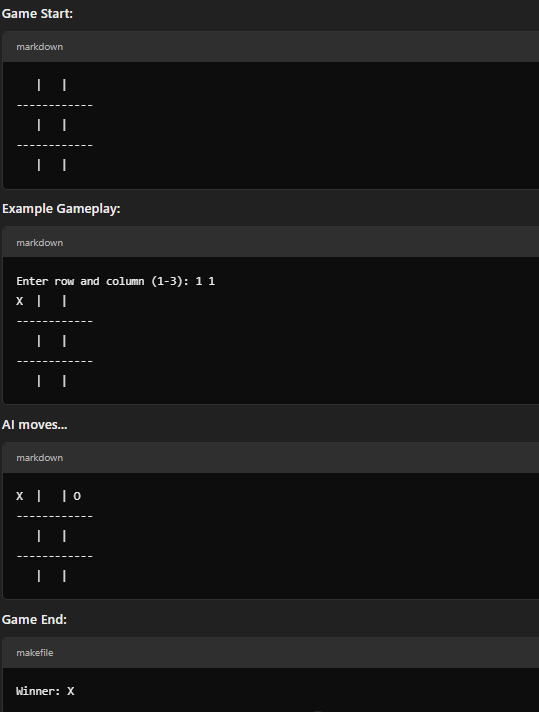
else:

print("It's a draw!")

if \_\_name\_\_ == "\_\_main\_\_":

play\_tic\_tac\_toe()

5. Screenshots / Output Photos



**Conclusion**

The **Tic-Tac-Toe Solver** successfully demonstrates how AI can make optimal decisions in a game using the **Minimax algorithm**. The AI always makes the best possible move, rendering the opponent unable to win (unless the opponent makes a mistake).

Future enhancements for this project could include:

* Implementing a **Graphical User Interface (GUI)** for a more engaging user experience.
* Adding **adjustable difficulty levels** to provide a varied challenge for users.

This project serves as an excellent example of AI-based decision-making, showcasing how algorithms can solve problems in games and real-world scenarios.