# Tic-Tac-Toe Solver

Using Minimax Algorithm

| | |
|---|---|
| Name | Anurag Singh |
| Roll No | 202401100400041 |
| Date | 10/03/2025 |

## Description and Goal

Tic-Tac-Toe is a classic two-player game where the goal is to get three marks in a row, column, or diagonal. This report presents a Python program that utilizes the Minimax algorithm to implement an AI-based Tic-Tac-Toe solver. The AI ensures an optimal move, making it unbeatable. The problem statement is to design an intelligent agent that plays Tic-Tac-Toe optimally against a human opponent.

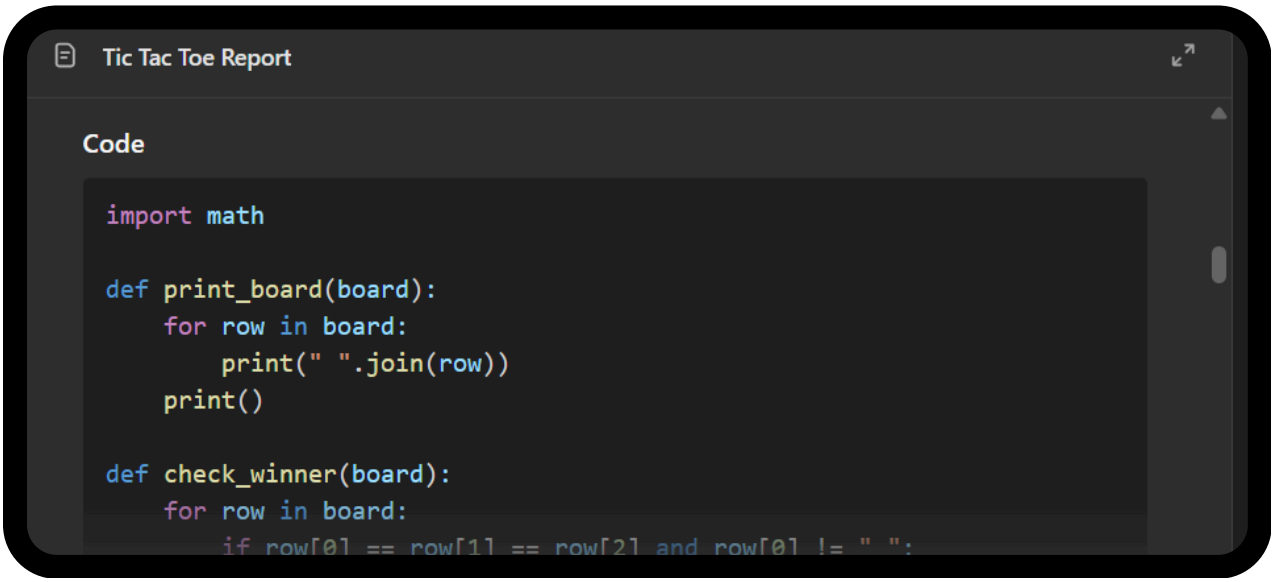
# Scope and Methodology

The approach used to solve the problem includes the following steps:

1. Represent the Tic-Tac-Toe board as a 3x3 matrix.
2. Implement a function to check for a winner or a tie.
3. Use the Minimax algorithm to evaluate all possible game states and choose the best move for the AI.
4. Allow the player to make a move and then let the AI respond with the optimal move.
5. Display the board after each move and declare the winner when the game ends.

The Minimax algorithm works by recursively evaluating all possible game outcomes. The AI aims to maximize its score (+1 for AI win, -1 for player win, and 0 for a tie) while assuming that the opponent plays optimally.

# Output




# References/ Credits

- The Minimax algorithm was referenced from AI game theory principles.
- Python documentation was used for syntax and function implementation.
- (Include any additional sources used)

```python
import math

# Function to print the current board state
def print_board(board):
    for row in board:
        print(" ".join(row))
    print()

# Function to check if there is a winner or a tie
def check_winner(board):
    # Check rows for a win
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != " ":
            return row[0]

    # Check columns for a win
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != " ":

            return board[0][col]

    # Check diagonals for a win
    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]

    # Check if the board is full (tie)
    if all(board[row][col] != " " for row in range(3) for col in range(3)):
        return "Tie"

    return None

# Minimax algorithm to evaluate the best move
def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == "X":
        return 1  # AI wins
    if winner == "O":
        return -1  # Player wins
    if winner == "Tie":
        return 0  # Tie

    if is_maximizing:
        best_score = -math.inf
        for row in range(3):
            for col in range(3):
                if board[row][col] == " ":
                    board[row][col] = "X"  # AI move
                    score = minimax(board, depth + 1, False)
                    board[row][col] = " "  # Undo move
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = math.inf
        for row in range(3):
            for col in range(3):
                if board[row][col] == " ":
                    board[row][col] = "O"  # Player move
                    score = minimax(board, depth + 1, True)
                    board[row][col] = " "  # Undo move
                    best_score = min(score, best_score)
        return best_score

# Function to find the best move for AI
def best_move(board):
    best_score = -math.inf
    move = None
```

```python
    move = None
    for row in range(3):
        for col in range(3):
            if board[row][col] == " ":
                board[row][col] = "X"
                score = minimax(board, 0, False)
                board[row][col] = " "  # Undo move
                if score > best_score:
                    best_score = score
                    move = (row, col)
    return move

# Main function to play the game
def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Tic Tac Toe AI Solver (X is AI, O is Player)")
    print_board(board)

    while True:
        # Player move
        row, col = map(int, input("Enter row and column (0-2): ").split())
        if board[row][col] != " ":
            print("Invalid move! Try again.")
            continue
        board[row][col] = "O"

        # Check if the player wins or game is tied
        if check_winner(board):
            print_board(board)
            print("Winner:", check_winner(board))
            break

        # AI move
        move = best_move(board)
        if move:
            board[move[0]][move[1]] = "X"

        print_board(board)

        # Check if AI wins or game is tied
        if check_winner(board):
            print("Winner:", check_winner(board))
            break

# Run the game
if __name__ == "__main__":
    main()
```