RAJALAKSHMI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

ACADEMIC YEAR 2025 - 2026

SEMESTER III

ARTIFICIAL INTELLIGENCE LABORATORY

MINI PROJECT REPORT

REGISTER NUMBER: 21172440030132

NAME: SANTHOSH KR

PROJECT TITLE: Implementation of an Unbeatable AI Game Bot for Tic-Tac-Toe using the Minimax Algorithm

FACULTY IN-CHARGE: Bhavani M

INTRODUCTION

Artificial Intelligence (AI) is a branch of computer science dedicated to creating systems that can perform tasks that typically require human intelligence. These tasks include problem-solving, decision-making, learning, and perception. A key area within AI is the study of search algorithms, which are methods for navigating through a set of potential solutions (a "state space") to find an optimal one.

Game playing has historically been a significant testbed for AI. Games like Tic-Tac-Toe or Chess are "adversarial" environments where two or more agents have conflicting goals. This project focuses on creating an AI bot for the classic game of Tic-Tac-Toe.

This project aims to design, implement, and test an AI agent that can play Tic-Tac-Toe perfectly. The agent will use the Minimax algorithm to ensure it never loses a game, meaning it will always win or, at worst, force a draw against any opponent.

PROBLEM STATEMENT

To design and implement a deterministic AI agent for the two-player, zero-sum game of Tic-Tac-Toe. The agent must be capable of exploring the complete game tree from any given state to select the move that guarantees the best possible outcome (a win or a draw), assuming the opponent also plays optimally.

GOAL

The primary goal is to develop a fully functional Python application where a human player can compete against an AI agent. The expected result is an 'unbeatable' AI.

THEORETICAL BACKGROUND

Tic-Tac-Toe is a perfect information, two-player, zero-sum game. The Minimax algorithm is a recursive decision-making algorithm designed to find the optimal move in such games. It evaluates all possible moves and returns the move that maximizes the AI's chance of winning while minimizing the opponent's advantage.

LITERATURE SURVEY

Rule-Based (Heuristic) Bot, Alpha-Beta Pruning, and Reinforcement Learning are common AI methods for game playing. However, Minimax was chosen for this project because it guarantees the optimal solution for Tic-Tac-Toe.

ALGORITHM EXPLANATION WITH EXAMPLE

The Minimax algorithm recursively explores all possible moves from the current state until it reaches a terminal state (win, loss, or draw). The AI is the 'Maximizer' (MAX), and the human is the 'Minimizer' (MIN). Each terminal state is given a score: +10 for AI win, -10 for human win, and 0 for draw.

IMPLEMENTATION AND CODE

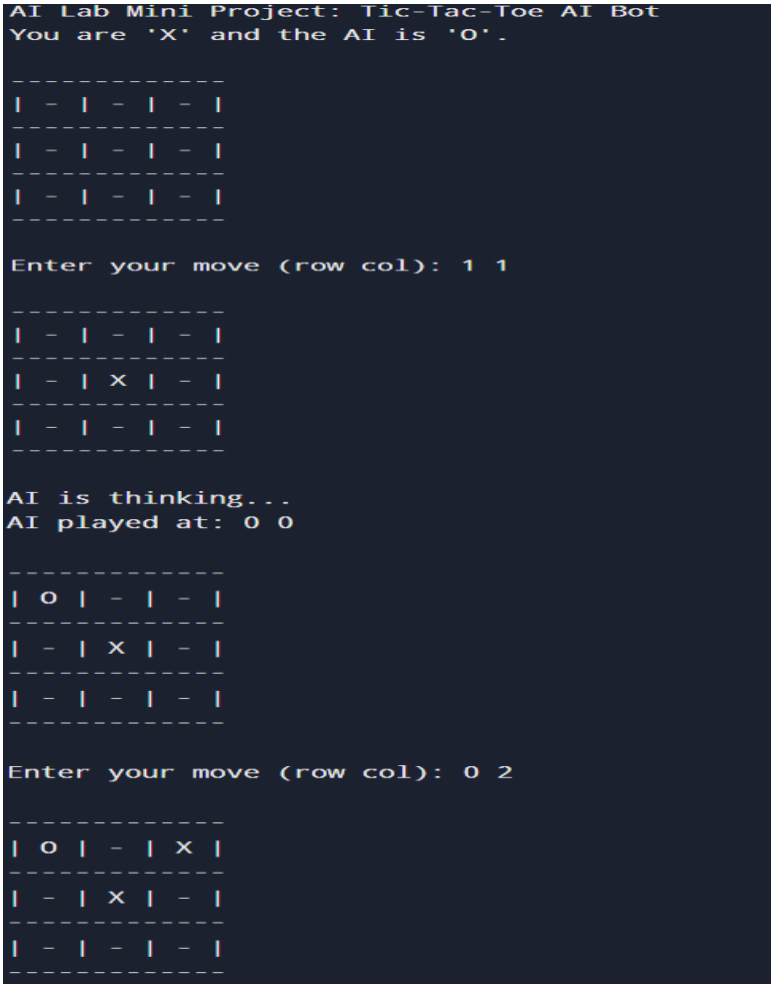
```
import math

PLAYER = 'X'
AI = 'O'
EMPTY = '-'

def print_board(board):
    print("\n-------------")
    for row in board:
```

```
        print(f"| {row[0]} | {row[1]} | {row[2]} |")
        print("-------------")
    print()

def is_moves_left(board):
    for row in board:
        if EMPTY in row:
            return True
    return False

def evaluate(b):
    for r in range(3):
        if b[r][0] == b[r][1] == b[r][2]:
            if b[r][0] == AI: return 10
            elif b[r][0] == PLAYER: return -10
    for c in range(3):
        if b[0][c] == b[1][c] == b[2][c]:
            if b[0][c] == AI: return 10
            elif b[0][c] == PLAYER: return -10
    if b[0][0] == b[1][1] == b[2][2]:
        if b[0][0] == AI: return 10
        elif b[0][0] == PLAYER: return -10
    if b[0][2] == b[1][1] == b[2][0]:
        if b[0][2] == AI: return 10
        elif b[0][2] == PLAYER: return -10
    return 0

def minimax(board, is_max):
    score = evaluate(board)
    if score in [10, -10]: return score
    if not is_moves_left(board): return 0

    if is_max:
        best = -math.inf
        for r in range(3):
            for c in range(3):
                if board[r][c] == EMPTY:
                    board[r][c] = AI
```

```
                    best = max(best, minimax(board,
False))
                    board[r][c] = EMPTY
        return best
    else:
        best = math.inf
        for r in range(3):
            for c in range(3):
                if board[r][c] == EMPTY:
                    board[r][c] = PLAYER
                    best = min(best, minimax(board,
True))
                    board[r][c] = EMPTY
        return best

def find_best_move(board):
    best_val = -math.inf
    best_move = (-1, -1)
    for r in range(3):
        for c in range(3):
            if board[r][c] == EMPTY:
                board[r][c] = AI
                move_val = minimax(board, False)
                board[r][c] = EMPTY
                if move_val > best_val:
                    best_move = (r, c)
                    best_val = move_val
    return best_move

def play_game():
    board = [[EMPTY]*3 for _ in range(3)]
    print("AI Lab Mini Project: Tic-Tac-Toe AI Bot")
    print("You are 'X' and the AI is 'O'.")
    print_board(board)

    while True:
        move = input("Enter your move (row col): ")
        r, c = map(int, move.split())
```

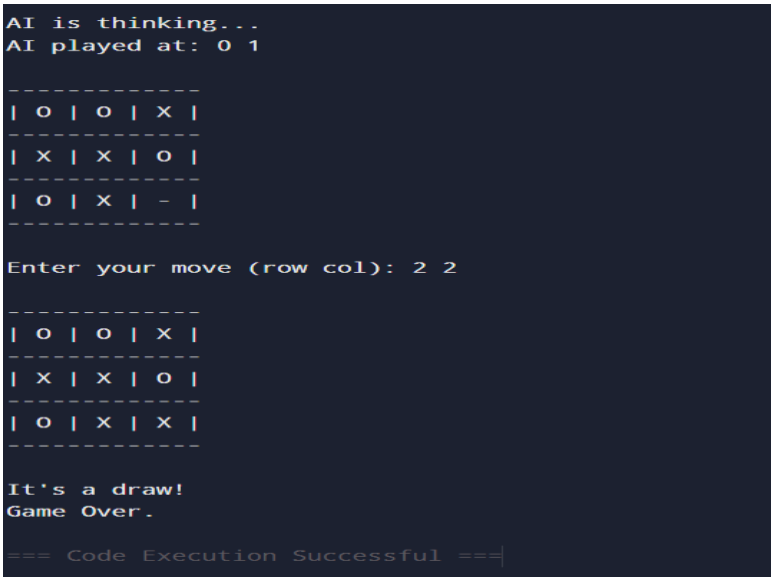
```
            if board[r][c] == EMPTY:
                board[r][c] = PLAYER
            print_board(board)

            if evaluate(board) == -10:
                print("You won!")
                break
            if not is_moves_left(board):
                print("Draw!")
                break

            ai_r, ai_c = find_best_move(board)
            board[ai_r][ai_c] = AI
            print(f"AI played at {ai_r}, {ai_c}")
            print_board(board)

            if evaluate(board) == 10:
                print("AI won!")
                break
            if not is_moves_left(board):
                print("Draw!")
                break

if __name__ == "__main__":
    play_game()
    print("Game Over.")
```

OUTPUT


AI Lab Mini Project: Tic-Tac-Toe AI Bot
You are 'X' and the AI is 'O'.
Enter your move (row col): 1 1
AI is thinking...
AI played at: 0 0
Enter your move (row col): 0 2

```
AI is thinking...
AI played at: 2 0

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

Enter your move (row col): 1 0

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

```
AI is thinking...
AI played at: 1 2

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

Enter your move (row col): 2 1

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



RESULTS AND FUTURE ENHANCEMENT

The implemented Minimax bot plays optimally, never losing a game. Enhancements such as Alpha-Beta Pruning can make the algorithm faster, and GUI integration can make it more interactive.

REFERENCES

1. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.
2. GeeksforGeeks. (2023). Minimax Algorithm in Game Theory.
3. DataCamp. (2025). Implementing the Minimax Algorithm for AI in Python.
4. Luger, G. F. (2008). Artificial Intelligence: Structures and Strategies for Complex Problem Solving (6th ed.). Addison-Wesley.
5. Coppin, B. (2004). Artificial Intelligence Illuminated. Jones and Bartlett Publishers.