

|  |
| --- |
| **RAJALAKSHMI INSTITUTE OF TECHNOLOGY** |
| (An Autonomous Institution, Affiliated to Anna University, Chennai) |

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**ACADEMIC YEAR 2025 - 2026**

**SEMESTER III**

**ARTIFICIAL INTELLIGENCE LABORATORY**

**MINI PROJECT REPORT**

|  |  |
| --- | --- |
| **REGISTER NUMBER** | 2117240070032 |
| **NAME** | Vijaykumar A |
| **PROJECT TITLE** | Tic Tac Toe using Artificial Intelligence |
| **DATE OF SUBMISSION** |  |
| **FACULTY IN-CHARGE** | **Mrs. M. Divya** |

**Signature of Faculty In-charge**

**INTRODUCTION**

Artificial Intelligence (AI) enables machines to mimic human decision-making and problem-solving abilities. Game-playing is one of the earliest and simplest applications of AI.  
This project implements an AI-powered Tic Tac Toe game where the computer uses logic and prediction to play optimally against a human opponent. The project demonstrates core AI techniques like game trees, search algorithms, and decision-making under constraints.

**PROBLEM STATEMENT**

**To design an intelligent system capable of playing Tic Tac Toe against a human user, analyzing possible game states, and selecting the best move automatically to win or draw the game.**

**GOAL**

**To develop an AI that never loses a Tic Tac Toe match.**

**To apply the Minimax algorithm for decision-making.**

**To provide a simple interactive interface for human–AI gameplay.**

**THEORETICAL BACKGROUND**

**Tic Tac Toe is a two-player, zero-sum game, making it ideal for applying adversarial search.  
The entire game can be represented as a state-space tree, where each node represents a possible board configuration.**

* **Algorithms Explored: Random move, Greedy move, Minimax.**
* **Chosen Algorithm: Minimax (guarantees the best move).**
* **Why Minimax: It systematically explores all possible moves, assuming the opponent plays optimally, ensuring the AI never loses.**

**ALGORITHM EXPLANATION WITH EXAMPLEAlgorithm Used: Minimax with optional Alpha-Beta Pruning  
Steps:**

1. **Generate all possible moves from the current board.**
2. **For each move, simulate the opponent’s response recursively.**
3. **Assign a score: +1 (AI win), -1 (player win), 0 (draw).**
4. **Choose the move that maximizes the AI’s score.  
   Example:  
   If the player places ‘X’ in the center, the AI checks all empty cells, evaluates future outcomes, and chooses the cell that leads to a win or draw.**

**IMPLEMENTATION AND CODE**

**import math**

**# --- Board Setup ---**

**board = [" " for \_ in range(9)]**

**# --- Print the board ---**

**def print\_board():**

**print("\n")**

**for i in range(3):**

**print(" | ".join(board[i\*3:(i+1)\*3]))**

**if i < 2:**

**print("--+---+--")**

**print("\n")**

**# --- Check for winner ---**

**def check\_winner(player):**

**win\_combinations = [**

**(0, 1, 2), (3, 4, 5), (6, 7, 8), # rows**

**(0, 3, 6), (1, 4, 7), (2, 5, 8), # columns**

**(0, 4, 8), (2, 4, 6) # diagonals**

**]**

**return any(board[a] == board[b] == board[c] == player for a, b, c in win\_combinations)**

**# --- Check if board is full ---**

**def is\_full():**

**return " " not in board**

**# --- Minimax Algorithm ---**

**def minimax(is\_maximizing):**

**if check\_winner("O"): # AI wins**

**return 1**

**if check\_winner("X"): # Human wins**

**return -1**

**if is\_full():**

**return 0**

**if is\_maximizing:**

**best\_score = -math.inf**

**for i in range(9):**

**if board[i] == " ":**

**board[i] = "O"**

**score = minimax(False)**

**board[i] = " "**

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

**return best\_score**

**else:**

**best\_score = math.inf**

**for i in range(9):**

**if board[i] == " ":**

**board[i] = "X"**

**score = minimax(True)**

**board[i] = " "**

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

**return best\_score**

**# --- AI Move ---**

**def ai\_move():**

**best\_score = -math.inf**

**move = None**

**for i in range(9):**

**if board[i] == " ":**

**board[i] = "O"**

**score = minimax(False)**

**board[i] = " "**

**if score > best\_score:**

**best\_score = score**

**move = i**

**board[move] = "O"**

**# --- Human Move ---**

**def human\_move():**

**while True:**

**try:**

**move = int(input("Enter your move (1-9): ")) - 1**

**if move < 0 or move > 8 or board[move] != " ":**

**print("Invalid move, try again.")**

**else:**

**board[move] = "X"**

**break**

**except ValueError:**

**print("Please enter a number between 1 and 9.")**

**# --- Main Game Loop ---**

**def play\_game():**

**print("Welcome to Tic Tac Toe! You are 'X' and the AI is 'O'.")**

**print\_board()**

**while True:**

**human\_move()**

**print\_board()**

**if check\_winner("X"):**

**print("🎉 You win!")**

**break**

**if is\_full():**

**print("It's a draw!")**

**break**

**print("AI is making a move...")**

**ai\_move()**

**print\_board()**

**if check\_winner("O"):**

**print("💻 AI wins!")**

**break**

**if is\_full():**

**print("It's a draw!")**

**break**

**# --- Start the game ---**

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

**play\_game()**

**OUTPUT**

A screen shot of a computer

AI-generated content may be incorrect.

**RESULTS AND FUTURE ENHANCEMENT**

**Results:**

* **Successfully implemented an AI that plays optimally.**
* **Demonstrated use of Minimax for game decision-making.  
  Future Enhancements:**
* **Add multiple difficulty levels.**
* **Use Reinforcement Learning (Q-Learning) for adaptive play.**
* **Extend to 4×4 or 5×5 Tic Tac Toe boards.**

|  |  |
| --- | --- |
| **Git Hub Link of the project and report** | [**link**](https://github.com/avulavijay96/Ai_miniproject96) |

**REFERENCES**

**Stuart Russell & Peter Norvig, *Artificial Intelligence: A Modern Approach*, 4th Edition.**

**GeeksforGeeks – “Tic Tac Toe AI using Minimax Algorithm.”**

**Towards Data Science – “Understanding Minimax and Alpha-Beta Pruning.”**

**Wikipedia – “Minimax Algorithm.”**

**Real Python – “Creating Games with Python and Pygame**