CS329 Foundations of AI: Multiagent Systems

Neeldhara Misra Manisha Padala

Kandarp Jani **22110104**

IIT Gandhinagar

Tic-Tac-Toe with MiniMax AI – A Two-Player Adversarial Game Project

1 Introduction

This project implements a classic Tic-Tac-Toe game enriched with an AI opponent based on the MiniMax algorithm. The work is carried out as part of the course Foundations of AI: Multiagent Systems, and focuses on modelling the game as a two-agent, turn-based, adversarial interaction in a deterministic, fully observable environment.

Tic-Tac-Toe is a simple yet complete example of a zero-sum game between two rational agents with strictly competing goals. This makes it ideal for studying core topics of the course such as:

- Game representation as states and actions
- Multiagent interaction in competitive settings
- Adversarial search
- Optimal policies under perfect information

The primary objective is to design an AI agent that plays optimally against a human player using MiniMax and to understand the mathematical and conceptual foundations behind this decision-making process.

2 Problem Definition and Game Model

2.1 Game Description

Tic-Tac-Toe is played on a 3×3 grid by two players, traditionally denoted as X and O. Players alternate turns placing their symbol in an empty cell. The game terminates when:

- One player has three of their symbols in a line (horizontal, vertical, or diagonal), or
- All cells are filled and no player has three in a row (draw).

2.2 Multiagent System View

From a multiagent systems perspective:

- There are two agents: the human player and the AI agent.
- The interaction is sequential (turn-taking).
- The environment is:
  - Deterministic – the result of each action is predictable.
  - Fully observable – both agents see the complete board state.
  - Static during decision – the state does not change while an agent is choosing an action.
- The game is zero-sum: one agent's gain is exactly the other agent's loss.

2.3 State and Utility Representation

A state is represented by a 1D array of length 9 (or equivalently a 3×3 board) where each position can be:

- X (occupied by player X)
- O (occupied by player O)
- empty (undefined or null)

Terminal states are states where either player has won, or the board is full (draw).

A utility function $U(s)$ is defined on terminal states:

$$U(s) = \begin{cases} +1 & \text{if the AI (computer player) wins} \\ -1 & \text{if the human player wins} \\ 0 & \text{if the game ends in a draw} \end{cases}$$

3 System Design and Implementation

3.1 Components

The system is implemented as a browser-based application with three main layers:

- **User Interface (UI):**
  - A 3×3 grid rendered using HTML and styled with CSS.
  - Buttons to choose game mode (Human vs Human or Human vs Computer) and player symbol (X or O).
  - A status display showing the result of the game and a reset option.
- **Game Logic (JavaScript):**
  - Maintains the board state in a 9-element array.
  - Handles player turns, move validation, win detection, and draw detection.
- **AI Agent (MiniMax):**
  - Evaluates possible moves in the game tree and selects the optimal action when playing against a human.

3.2 Game Modes

Human vs Human Mode:

- Both agents are human players taking turns on the same device.
- The system only enforces rules and detects outcomes.

Human vs Computer Mode:

- The human selects their symbol (X or O).
- The AI plays as the opponent and uses MiniMax to choose moves.
- The environment becomes a classic human–AI two-player adversarial game.

4 MiniMax Algorithm and Mathematical Background

4.1 Adversarial Search Framework

In a two-player zero-sum game with perfect information, decision making is modelled as an adversarial search problem. The game forms a tree where:

- Nodes represent game states.
- Edges represent legal actions (placing X or O in an empty cell).
- MAX corresponds to the AI agent (trying to maximize utility).
- MIN corresponds to the human agent (assumed to play optimally and minimize the AI's utility).

4.2 MiniMax Value Definition

Let $V^*(s)$ denote the optimal value of state s:

$$V^*(s) = \begin{cases} U(s) & \text{if } s \text{ is a terminal state} \\ \max\limits_{a \in \text{Actions}(s)} V^*(\text{Result}(s,a)) & \text{if it is MAX's turn (AI)} \\ \min\limits_{a \in \text{Actions}(s)} V^*(\text{Result}(s,a)) & \text{if it is MIN's turn (human)} \end{cases}$$

Where:

- Actions(s) is the set of legal moves in state s.
- Result(s, a) is the state obtained after applying action a to state s.

The MiniMax algorithm recursively computes $V^*(s)$ from the leaves (terminal states) back to the root and chooses the move that leads to the best value for the AI at the current state.

4.3 Complexity Analysis

For Tic-Tac-Toe:

- Maximum depth d of the game tree is at most 9 (one move per empty cell).
- The branching factor b (average number of legal moves) starts at 9 and decreases.
- The upper bound on the number of leaf nodes is on the order of 9! (362,880 possible sequences), but the effective number of states is smaller due to:
  - Early terminations (wins before the board is full)
  - Symmetries of the board (rotations and reflections)
- Because the game tree is relatively small, a full-depth MiniMax search is computationally feasible without needing pruning or heuristics. This guarantees that the AI plays optimally and cannot be beaten if it does not make mistakes. At worst, a rational opponent can force a draw.

4.4 Relation to Course Concepts

This project demonstrates:

- Representation of a multiagent interaction as a formal game.
- Use of utility functions and optimality criteria.
- Design of a rational agent that reasons about an opponent's possible actions.
- Application of adversarial search (MiniMax) in a finite, deterministic, two-player zero-sum game.

5 Results and Observations

Empirically, the AI agent based on MiniMax:

- Never loses when playing with perfect information and full-depth search.
- Either wins or forces a draw, regardless of whether it plays as X or O.
- Encourages the human player to think strategically and understand optimal play patterns (such as always taking the center or corners in certain situations).
- The human vs human mode additionally serves as a baseline, allowing comparison between purely human play and human–AI play, and helping to verify correctness of win/draw detection logic.

6 Conclusion

This project successfully implements a Tic-Tac-Toe game as a two-agent adversarial system and integrates a MiniMax-based AI agent that plays optimally. By modelling the game as a zero-sum, deterministic, fully observable environment, we applied core ideas from multiagent systems and adversarial search to a simple but complete domain.

The project illustrates how:

- Game states, actions, and utilities can be formalized in a multiagent setting.
- Rational agents can reason about an opponent's possible responses.
- The MiniMax algorithm computes optimal strategies in finite, perfect-information games.

Although Tic-Tac-Toe is a small game, the same principles generalize to more complex competitive environments that arise in AI and multiagent systems, where reasoning about other agents' behaviour is essential for intelligent decision making.