Addis Ababa University

College of Built Environment and Engineering Department of Artificial Intelligence

Assignment Title: Design and Implementation of a Minimax Game-Playing Agent

Course: Artificial Intelligence: Principles and Techniques
Program: Master of Science in Artificial Intelligence
Topic: Adversarial Search – Minimax Algorithm

Submitted by:-
Name: Fuad Nuri
ID:GSE/1569/18

Submitted to:
Dr. Natnael Argaw Wondimu

1. Game Representation and Formal Modeling

To implement the Minimax algorithm, I first formally define the environment as follows

- **States (S):** A 3 X 3 grid where each cell is empty, "X", or "O".
- **Actions (A):** Placing a mark in any empty cell.
- **Terminal States:** A state where one player has three marks in a row (horizontally, vertically, or diagonally) or the board is full.
- **Utility Function (U):**
    - **AI Wins:** +1
    - **Human Wins:** -1
    - **Draw:** 0

2. The minimax algorithm logic

The primary objective of the implementation is to design an agent capable of making optimal decisions under competition. The agent assumes that the human opponent also plays optimally and selects moves based on the Minimax decision rule.

2.1 The decision rule

The implementation uses a recursive search through the game tree to evaluate potential future states. The value of a state V(s) is defined by:

$$V(s) = \begin{cases} \text{Utility}(s) & \text{if} s \text{is terminal} \\ \max_{a \in \text{Actions}(s)} V(\text{Result}(s,a)) & \text{if player is MAX} \\ \min_{a \in \text{Actions}(s)} V(\text{Result}(s,a)) & \text{if player is MIN} \end{cases}$$

2.2 Algorithms Flow

1. **Search:** The algorithm starts at the current state and explores all possible legal actions.
2. **Recursion:** For each move, it simulates the opponent's best response by recursively calling the minimizing function.
3. **Backtracking:** Values from the terminal states (leaf nodes) are passed back up the tree.
4. **Selection:** The AI(Maximizer) chooses the move that leads to the highest possible value, assuming the human (Minimizer) will always choose the move that leads to the lowest value for the AI.

3. Discussion of Design Choices and Challenges

The implementation process focused on reinforcing theoretical concepts of adversarial search through practical application.

3.1 Design Choices

A critical design decision in the minimax method was the use of in-place state modification followed by backtracking.

- **Readable code:** The Minimax logic was kept distinct from the game interface (Human–Agent Interaction) to ensure clarity and meet technical requirements.
- **User interaction UI**: It have easy web-based ui with fastapi backend for appealing ui/ux and
- **User interaction terminal:** Theres terminal based implementation for quick test in ipynb
- **Mechanism:** instead of creating a new copy of the board for every possible move which would be memory-intensive. The agent places its mark(state[i][j] = self.ai), recurses to evaluate that path, and then immediately “undoes” the move (state[i][j] = “ “).
- **Reasoning:** This ensures the game-tree search remains efficient while maintaining the integrity of the actual board state during the AI’s “thinking” phase.

3.2 Challenges Encountered

- **Handling the “Draw” State:** Initially, determining if a state was a terminal draw required a separate check. Integrating this into check_winner using Python’s all() function allowed for a clean recursive base case in the minimax function.
- **Recursive Branching:** Tic-Tac-Toe has a maximum of 9!(362,880) possible move sequences. While this is manageable for modern hardware, the recursive depth can lead to performance lags if the terminal checks are not optimized.
- **Grid Coordinate Mapping:** Ensuring the best_move function turned a tuple (i,j) compatible with the human interface was necessary to satisfy the requirement for functional human agent interaction.