Design and Implementation of a Minimax Game-Playing Agent for Connect Four

Nebiyu Daniel
Department of Computer Science
Artificial Intelligence Principles

February 15, 2026

1 Introduction

This report presents the design and implementation of an intelligent game-playing agent for **Connect Four**, a classic two-player board game. The agent uses the **Minimax algorithm** with **alpha-beta pruning** and **depth-limited heuristic evaluation** to make optimal decisions in an adversarial environment.

Connect Four is played on a 6×7 vertical grid. Players alternate dropping colored discs into columns, where each disc falls to the lowest available position. The first player to align four consecutive discs—horizontally, vertically, or diagonally—wins. If the board is completely filled with no winner, the game ends in a draw.

The game satisfies the key properties required for Minimax application:

- **Deterministic** — no random elements
- **Turn-based** — players alternate moves
- **Zero-sum** — one player's gain equals the other's loss
- **Perfect information** — the entire board is visible to both players

2 Game Representation

2.1 State Space

The game state is represented as a 6×7 integer matrix B where each cell $B[r][c] \in \{0, 1, 2\}$, with 0 denoting an empty cell, 1 denoting Player 1, and 2 denoting Player 2. Row 0 is the top of the board, and pieces drop to the lowest empty row.

2.2 Legal Actions

A move consists of selecting a column $c \in \{0, 1, \ldots, 6\}$. Column c is a legal move if and only if $B[0][c] = 0$ (the topmost cell in that column is empty).

2.3 Terminal States and Utility

A state is terminal when:

1. A player has four consecutive pieces in any direction (win), or
2. No legal moves remain (draw).

The utility function assigns values from the AI's perspective:

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

3 Minimax Algorithm

3.1 Core Algorithm

The Minimax algorithm models the game as a search tree where the AI (*maximizer*) and the opponent (*minimizer*) alternate turns. At each node, the maximizer selects the action with the highest value, and the minimizer selects the action with the lowest value.

Algorithm 1 Minimax with Alpha-Beta Pruning

```
function MINIMAX(state, depth, isMax, α, β)
    if TERMINAL(state) or depth = 0 then
        return EVALUATE(state)
    end if
    if isMax then
        value ← −∞
        for each action in ACTIONS(state) do
            value ← max(value, MINIMAX(RESULT(state, action), depth − 1, false, α, β))
            α ← max(α, value)
            if α ≥ β then break                                   ▷ β cutoff
            end if
        end for
        return value
    else
        value ← +∞
        for each action in ACTIONS(state) do
            value ← min(value, MINIMAX(RESULT(state, action), depth − 1, true, α, β))
            β ← min(β, value)
            if α ≥ β then break                                   ▷ α cutoff
            end if
        end for
        return value
    end if
end function
```

3.2 Alpha-Beta Pruning

Alpha-beta pruning significantly reduces the number of nodes explored without affecting the result. It maintains two bounds:

- α: the best value the maximizer can guarantee (lower bound)
- β: the best value the minimizer can guarantee (upper bound)

When $\alpha \geq \beta$, the current branch is pruned because the opponent would never allow this path. In the best case, alpha-beta pruning reduces the time complexity from $O(b^d)$ to $O(b^{d/2})$, where b is the branching factor and d is the search depth.

3.3 Depth-Limited Search

Since Connect Four has up to 4.5×10^{12} possible positions, exhaustive search is infeasible. We limit the search to a configurable depth (3, 5, or 7 levels) and apply a heuristic evaluation function at the depth boundary.

3.4 Heuristic Evaluation Function

When the depth limit is reached and the state is non-terminal, a heuristic function estimates the board's favorability. The evaluation function examines every contiguous window of four cells across all directions and scores them based on:

Pattern	**Score**
4 AI pieces (win)	+100
3 AI + 1 empty	+10
2 AI + 2 empty	+5
Center column piece	+6
3 opponent + 1 empty	−8
2 opponent + 2 empty	−4

Table 1: Heuristic scoring for windows of four cells

The center column receives a bonus because center pieces participate in more potential four-in-a-row combinations, providing more strategic flexibility.

4 Design Choices and Challenges

4.1 Move Ordering

Moves are sorted by proximity to the center column before evaluation. Center-first ordering increases the likelihood of early cutoffs in alpha-beta pruning, as central moves tend to produce higher-scoring positions.

4.2 Win Depth Bonus

Terminal utility values include a small depth bonus: $U_{\text{win}} = 1000 + d$ for AI wins and $U_{\text{loss}} = -(1000 + d)$ for losses, where d is the remaining depth. This encourages the AI to prefer *faster* wins and *slower* losses.

4.3 Challenges

1. **Search space size**: The Connect Four game tree is far too large for exhaustive Minimax. Depth limiting with heuristic evaluation was essential, and tuning the heuristic weights required iterative experimentation.

2. **Heuristic balance**: Scoring defense too highly made the agent passive, while scoring offense too highly caused it to miss opponent threats. The final weights reflect a balanced offensive-defensive strategy.

3. **Performance**: Even with pruning, deeper searches are computationally expensive. Move ordering was crucial for reducing the effective branching factor.

5 Conclusion

The implemented Connect Four agent demonstrates the practical application of adversarial search in AI. The combination of Minimax, alpha-beta pruning, depth-limited search, and heuristic evaluation enables the agent to play competitively against human opponents while maintaining reasonable response times. The modular code design separates game logic, AI reasoning, and user interaction, making the system extensible and maintainable.