Laboratory 2

Variant 2

By Sevastian Bortsevich, Yermukhamed Islam

Introduction

Connect Four is a two-player strategy game where players take turns dropping pieces into a vertical grid. The objective is to form a sequence of four of their own pieces either horizontally, vertically, or diagonally before their opponent does. In this project, we implemented an AI opponent that can play Connect Four using the Minimax algorithm with alpha-beta pruning.

The Minimax algorithm is a recursive decision-making strategy used in two-player games to minimize potential losses in a worst-case scenario. It evaluates possible future game states and selects the optimal move based on maximizing or minimizing the player's advantage. Alpha-beta pruning enhances Minimax by eliminating unnecessary branches in the decision tree, reducing computation time without affecting accuracy.

Advantages of Minimax:

- Makes optimal decisions in a deterministic environment.
- Provides a structured approach to evaluating moves.

Disadvantages of Minimax:

- High computational complexity, especially in large state spaces like Connect Four.

Implementation

Connect Four AI - Function Descriptions

`minimax`

Implements the Minimax algorithm with alpha-beta pruning to determine the best move. The main variation from the standard algorithm is how the value is calculated: the difference between the player's and the bot's `evaluate_position()` result.

`evaluate_position`

Evaluates the board state and assigns a score based on the following factors:

- The number of target pieces in a row (up to 4).
- Whether there is a free space after these pieces (up to 1 additional point).
- For diagonal alignments, it ensures that any free space counted towards the score has a supporting piece underneath, preventing inflated scores.

```
def evaluate_position(self, board, piece):
    max_score = 0
    for r in range(ROWS):
        for c in range(COLS - 3):
            if board[r][c] == piece:
                temp_score = 0
                for i in range(4):
                    if board[r][c + i] == piece:
                        temp_score += 1
                    elif board[r][c + i] is None:
                        temp_score += 1
                        break
                    else:
                        break
                if temp_score > max_score:
                    max_score = temp_score
    ...
    return max_score
```

`evaluate_window`

This function was initially intended to be used inside `evaluate_position` to calculate the score of each possible combination. However, during development, it was left in its initial state and now represents an early concept for evaluating board sections.

```
def evaluate_window(self, window, piece):
    max_count = 0
    count = 0
    for i in range(4):
        if window[i] == piece or window[i] is None:
            count += 1
        else:
            if count > max_count:
                max_count = count
            count = 0
    return max_count
```

Discussion

As mentioned in the implementation section, the evaluation function considers:

- The number of consecutive target pieces in a row (up to 4).
- The presence of a free space next to them, which adds up to 1 extra point.
- For diagonal alignments, any counted free space must have a supporting piece underneath to avoid inflated scores.

Strengths:

- Encourages moves that build towards a four-in-a-row win.
- Prevents overestimating potential future moves by requiring structural support for diagonal plays.
- Considers both occupied positions and potential extensions, leading to better strategic play.

Weaknesses:

- Does not explicitly prioritize blocking the opponent's winning moves.
- Lacks weighting for some common strategies.
- Could be improved by incorporating threat detection, although the current evaluation generally performs well.

Conclusion

The most challenging part of this laboratory was designing the evaluation function. Constantly tuning and accounting for various situations was particularly problematic. Because of this, the evaluation function could certainly be improved further. Future improvements may focus on refining heuristic evaluations and incorporating better defensive strategies.