Artificial intelligence Project2: Report

Malik DAHMANI

June 22, 2024

Contents

1 Explain how you implement Minimax algorithm and MCTS for this game in detail

1.1 Minimax

First, the Minimax algorithm is initialized with a board_state, the current depth, a boolean should_be_maxed indicating whether we are maximizing or minimizing, the color of the current player, and the alpha and beta values for alpha-beta pruning.

Then, for the maximizing player, it initializes value to negative infinity and it generates all possible moves from the current board state and iterates through them. For each move, it recursively calls the minimax function with decreased depth and the roles reversed (minimizing player). Finally, it updates the value with the maximum value obtained from the child nodes and updates the alpha value for alpha-beta pruning. If beta is less than or equal to alpha, it breaks out of the loop, performing pruning.

For the minimizing player, it initializes value to positive infinity and it generates all possible moves from the current board state and iterates through them. For each move, it recursively calls the minimax function with decreased depth and the roles reversed (maximizing player). It updates the value with the minimum value obtained from the child nodes and updates the beta value for alpha-beta pruning. If beta is less than or equal to alpha, it breaks out of the loop, performing pruning.

Finally, the algorithm returns the best score and the corresponding movement for the current player.

1.2 Monte Carlo Tree Search

MCTS is initialized with the current board state and the color of the current player. It maintains dictionaries for tracking wins (self.wins), plays (self.plays), and children nodes (self.children). Then, MCTS runs for a specified number of iterations. In each iteration, it performs four steps: selection, expansion, simulation, and backpropagation:

- Selection: Starting from the root node, MCTS selects nodes to explore using the UCT (Upper Confidence bound applied to Trees) metric until it reaches a leaf node.
- Expansion: If the selected node is not terminal and not fully expanded,

it generates all possible child nodes (moves) and adds them to the children dictionary.

- Simulation: It simulates a random game from the current node until a terminal state is reached. The outcome of this simulation is used to calculate the reward.
- The reward from the simulation is backpropagated through the path from the leaf node to the root, updating the win and play counts for each node in the path.

After the specified number of iterations, MCTS selects the move with the highest play count from the root node as the best move.

1.3 Explain the most difficult thing you faced when implementing the code. Why? And how you solved it?

1.4 Difficulty

The most difficult challenge was integrating the heuristics and evaluation function to accurately assess the board state in the Minimax algorithm. The complexity of the game dynamics in Strands made it difficult to define a heuristic that could effectively differentiate between advantageous and disadvantageous states.

1.5 Solution

To address this, I experimented with various heuristic functions and evaluated their performance in different game scenarios. I eventually settled on a heuristic that combined several factors, such as the number of connected components controlled by each player, the potential future moves, and the immediate threats. Additionally, thorough testing and iterative refinement of the heuristic function helped in fine-tuning its accuracy and reliability.

2 Compare Minimax algorithm and MCTS, what is the difference between them? Which one do you think is better for "Strands"?

2.1 Minimax vs MCTS

First, Minimax is a depth-first search algorithm that explores all possible moves up to a certain depth and uses a heuristic evaluation function to assess the board states. MCTS, on the other hand, is a probabilistic algorithm that builds a search tree using random simulations and selects moves based on their average outcomes.

To continue, Minimax relies on a predefined depth limit and evaluates all nodes up to that depth, leading to exhaustive exploration within the limited depth. On the other hand, MCTS balances exploration and exploitation using the UCT metric, which helps it explore less promising moves initially but exploit better-performing moves more frequently over time.

Finally, Minimax can be computationally expensive, especially with deeper searches, but it provides deterministic results based on the evaluation function. In contrary, MCTS is more efficient in handling larger search spaces and can provide good results with fewer iterations, but it relies on randomness and may have varying performance.

2.2 Which is Better for Strands:

For Strands, the Minimax algorithm might be better suited for several reasons. First, Minimax provides deterministic results based on the evaluation function, making it more predictable and reliable in strategic games like Strands. With a well-designed heuristic evaluation function, Minimax can effectively explore the search space within a limited depth, making informed decisions based on potential future moves.

To continue, the use of alpha-beta pruning optimizes the Minimax algorithm, reducing the number of nodes evaluated and making it more efficient. Therefore, although MCTS has its advantages in handling larger search spaces and probabilistic exploration, the deterministic and strategic nature of Minimax makes it better suited for Strands, especially when combined with an effective heuristic evaluation function.