Exploring the Minimax Algorithm and Utility Functions

Tansylu Akhmetova

17/05/2024

Introduction to Game Theory and the Minimax Algorithm

Game Theory

Game theory is a field in mathematical economics that focuses on the analysis of conflict resolution between agents (players) and the calculation of optimality of the strategies employed in the process. The situations studied range from plain games to financial and sociological concepts. Conflict is any situation in which two or more participants have opposing or intersecting interests. Each player possesses a certain set of possible moves and strategies to apply. Together these strategies produce a certain resolution of the game, sometimes beneficial or harmful to one or more parties.

The main idea is that for reaching optimality in choosing a game strategy, one must not only consider the best-profit path for themselves, but also predict the opponent's possible choices and their impact on the situation as a whole.

Minimax Algorithm

Specifically, for two-person zero-sum games, which are the ones in which one player wins all and the opponent loses all, one of the most prominent algorithms is Minimax.

The essence of the Minimax algorithm lies in the assumption that in a game with two players, each will make the choice of maximum profit or least loss. Let X represent the player who is looking for maximum profit, and Y represent the player looking for minimum loss. As such, a typical game employing the Minimax algorithm would go as follows:

1. If the game is over, return the score.
2. Otherwise, obtain a list of all new game states for each possible move.
3. Measure max profit/least loss for each state.
4. For each of the available future states, add the Minimax score for the current state.
5. If it's the X player's turn: return the move that leads to the maximum score.
6. If it's the Y player's turn: return the move that leads to the minimum score.

The algorithm is recursive, and calculations are performed in a sequence for each of the players until the final score is found.

Understanding Utility Functions

A game is defined by three components:

1. Players: The set of players, $I = \{1, \ldots, n\}$.
2. Strategies: Each player i has a set of strategies S_i.
3. Utility Functions: For each player i, there is a utility function u_i.

The utility function $u_i : S \to R$ maps each combination of strategies (one chosen by each player) to a real number, representing the payoff or satisfaction that player i receives. The combination of strategies is represented as $s = (s_1, \ldots, s_n)$, where s_i is the strategy chosen by player i.

Thus, utility functions assign numeric values, or game scores, to the available game states in the state space. The decision in Minimax algorithm are based on the values of the utiity function in a given state.

Part 2: Application and Analysis

Selection of a Game

I chose Russian checkers, which is a more complex complex version of English checkers. Here is a brief description of the rules for the game:

- The game is played on a standard 8x8 chess board.
- Checkers move only on black squares diagonally.
- A regular checker in Russian checkers moves forward one square and can capture both forward and backward by jumping over one square (in English checkers, it captures only forward).
- A regular checker becomes a queen when it reaches the farthest row from its starting position.
- A queen in Russian checkers moves and captures both forward and backward across any number of squares (in English checkers, it moves one square forward or backward and captures by jumping over one square).
- Players must capture if possible. If multiple capturing options are available, the player can choose any of them.
- A player loses if they cannot make a move.

Designing a Utility Function

Here is the reasoning process I followed in order to develop my utility function for the game and the assumptions that I have made:

1. Piece Value: Regular checkers and queens have different values. Queens are more powerful, so they should valued higher.
2. Y-Bonus: Pieces closer to promotion (the farthest row) should have higher value, rewarding aggressive forward movement.
3. Piece Count: The total number of pieces for each player affects the overall utility, as with low number of pieces it is very difficult to win against a higher number of pieces.
4. Positioning: For position specific reward I mainly considered Y-location, which represents the closeness to converting a regular piece(s) to a queen. This is given by the Y-bonus coefficient variable.

Pseudocode for Utility Function

Initialize Constants:
RegularCheckerValue = 100
QueenValue = 500
YBonus = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7]

Define evaluate_position(board):

player_score = 0
opponent_score = 0
Iterate through each row on the board:
    Iterate through each column on the board:
        Check the piece at the current position.
        If piece is player's regular checker:
            Add (RegularCheckerValue $\times$ (1 + YBonus[row])) to `player_score`
        If piece is player's queen:
            Add (QueenValue $\times$ (1 + YBonus[row])) to `player_score`
        If piece is opponent's regular checker:
            Add (RegularCheckerValue $\times$ (1 + YBonus[7 $-$ row])) to `opponent_score`
        If piece is opponent's queen:
            Add (QueenValue $\times$ (1 + YBonus[7 $-$ row])) to `opponent_score`
Return (`player_score` $-$ `opponent_score`)

Challenges and Strategies

Implementing the Minimax algorithm and designing a utility function for Russian checkers posed several challenges:

1. **Complexity of Game States:** Russian checkers has a large number of possible game states, especially as the game progresses and more pieces are on the board. This complexity increases the computational cost of evaluating each state.

2. **Branching Factor:** The branching factor in Russian checkers can be quite high, especially considering the various possible moves for each piece. This leads to a large search space and significantly increases the time complexity of the Minimax algorithm.

3. **Dynamic Nature of the Game:** Russian checkers is a dynamic game where the board state can change rapidly with each move. This dynamic nature makes it challenging to accurately evaluate the utility of each state, as the significance of certain positions may change throughout the game.

To overcome these challenges, the following could be applied:

- **Pruning Techniques:** Implement alpha-beta pruning to reduce the search space and improve the efficiency of the Minimax algorithm. It is a way to optimize the search process by pruning branches of the game tree that are guaranteed to be worse than previously examined branches.

- **Heuristic Evaluation:** Develop smarter heuristic evaluation functions that take into account not only piece values and positions but also strategic board configurations and potential future moves. For example, it might be helpful to consider how close to the center a player's pieces are located as it is a more advantageous position.

Part 3: Reflection and Conclusion

Designing the utility function for Russian checkers and exploring the Minimax algorithm has led me to research and and learn a lot about decision-making in games. It has deepened my understanding of how to evaluate game states and make optimal moves, considering both immediate gains and long-term strategies.

However, there are limitations to the current approach, such as the simplification of positional bonuses to Y-bonus and the lack of consideration for specific board configurations. Additionally, the computational complexity of the Minimax algorithm can lead to a failure for large search spaces.

Moving forward, improvements could be made by refining the utility function to incorporate more nuanced evaluations of board states and exploring more advanced algorithms, such as Monte Carlo Tree Search, to handle the complexity of Russian checkers more effectively.

Overall, exploring the Minimax algorithm and creating a utility function for Russian checkers provided me with a broader knowledge of game theory and strategic decision-making. Through this process, I gained the ability to navigate the complexities of two-player zero-sum games, where each move influences both my own gains and my opponent's losses. Developing the utility function required a comprehensive understanding of the game's mechanics, including factors such as piece values, positioning, and strategic objectives, to effectively guide my decision-making process. It was quite captivating to experience playing a game from the point of view of a perfect player - a computational machine.