Assignment 4 Report

1 Part 1 : Exploring the minimax Algorithm and Utility Functions

1.1 Introduction to Game Theory and the Minimax Algorithm

Provide a brief overview of game theory and the role of the Minimax algorithm in strategic decision-making for two-player, zero-sum games. Discuss the concept of optimality in moves and how Minimax contributes to achieving it.

Game theory studies how players make decisions in competitive situations. It helps understand and predict the behavior of individuals or groups with conflicting interests. In game theory, players choose strategies to maximize their own benefits while minimizing losses. A zero-sum game is a type of game in which the total gain or loss among all players is zero. This means one player's gain is exactly balanced by the other players' loss. In other words, if one player wins, the other players lose an equivalent amount. In two-player, zero-sum games two players play that game and one player's gain is exactly the other player's loss. Examples include chess and tic-tac-toe. The total ”sum” of gains and losses is zero, meaning if one player wins, the other loses by the same amount. The minimax algorithm is a decision-making algorithm used in zero-sum games. It helps a player choose the best move by assuming the opponent will also play optimally. The algorithm considers all possible moves and their outcomes to find the move that minimizes the possible loss for the worst-case scenario. The optimal move is the move that causes the best possible outcome for the player assuming also another player plays optimally. In zero-sum games, one player tries to maximize its benefit but at the same time, other player tries to minimize opponents' gains. Therefore optimality means maximizing a player's minimum gain. The minimax algorithm helps a player play optimally because it explores all possible moves, evaluates their outcomes, and chooses the best move. The player aims to maximize their minimum gain, assuming the opponent will choose the move that minimizes the player's gain. Therefore, it can determine what a player can gain in the worst-case scenario.

1.2 Understanding Utility Functions

Explain what utility functions are and how they are used in the context of the Minimax algorithm to evaluate the desirability of game states. Discuss factors that might influence the design of a utility function for different types of games.

Utility functions give a number to show how good or bad different states are by evaluating them. These numbers help measure the results of different moves, making it easier to compare and choose the best move. In the minimax algorithm, utility functions are used to evaluate states. The algorithm looks at possible future moves and uses the utility function to decide how good each state is. By doing this, the minimax algorithm can pick the best move that gives the player the highest utility, while assuming the opponent will try to lower it. There are multiple things to consider when designing the utility functions. Game objectives affect the utility function. There can be specific goals in the game to make the state closer to winning state. In chess for example we can check to the values of the pieces on the table, if the player has more valuable pieces it can increase the utility function value, or in xox game the possible winning triple count can be used to calculate the utility function. In the chess example player 1 tries to maximize its valuable piece count and minimize the player2's valuable piece count and the other player would do the opposite of this, or in the xox example player1 tries to increase its winning possible triples and reduce the number of winning triples of player2 also player2 would do the same for itself.

2 Part 2 : Application and Analysis

Selection of a Game: Choose a game (this could be Mancala, chess, or any other game suitable for Minimax analysis). Provide a brief description of the game's rules and objectives.

Connect 4 is a two-player strategy game where players take turns dropping colored discs into a vertical grid. The grid consists of 7 columns and 6 rows. Each player uses discs of a different color(most of the time red and yellow). The objective of the game is to be the first player to form a horizontal, vertical, or diagonal line of four of one's own discs. The game starts with an empty grid. Players take turns to drop one of their discs into any of seven columns. The disc falls down and occupies the lowest available space within the column. If the grid is completely filled with discs without any player connecting four in a row the game ends in a draw.

2.1 Designing a Utility Function

Create a utility function in pseudocode that could be used by a Minimax algorithm to evaluate game states for your selected game. Explain your reasoning behind the design of the utility function, including any assumptions or considerations you made.

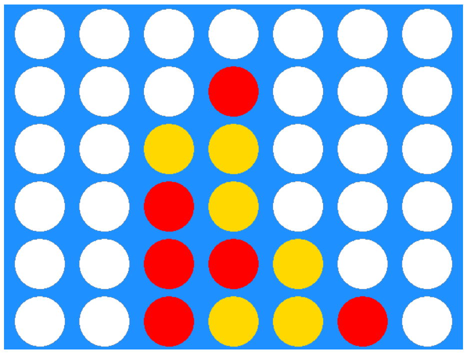

Figure 1: https://miro.medium.com/v2/resize:fit:1280/1*A5b630g96x9PrhwB9Mvf1w.png

For the utility function, several factors can be checked to calculate the desirability of a state. First, it can check if the state is a winning or losing state. If it is a losing state for the player, it gives a very low negative score. If it is a winning state for the player, it gives a very high positive score. After that, it can check the player's three-in-a-row positions with an open fourth spot, as these positions could represent potential winning opportunities for the next move. Therefore, we can assign high points to these positions. Additionally, it can check the opponent's three-in-a-row positions with an open spot for winning, and these positions should receive low negative points in the utility function. After this, it can check the same for two-in-a-row positions, as these could potentially become three-in-a-row positions. In this utility function player 1 would choose the higher-scored states, and player 2 would like to choose the lower-scored states in minimax algorithm,

In this example, the points assigned to the player's positions are higher than the points assigned to the opponent's positions. This can be adjusted based on which strategy results in better solutions.

Algorithm 1 Utility Function Psuedo Code

```
function EVALUATEBOARDSTATE(board,player,opponent)
    utility ← 0
    WIN_SCORE ← 100000
    LOSE_SCORE ← −100000
    THREE_IN_A_ROW_SCORE ← 100
    TWO_IN_A_ROW_SCORE ← 10
    OPPONENT_THREE_SCORE ← −50
    OPPONENT_TWO_SCORE ← −5
    if CHECKWIN(board,player) then
        return WIN_SCORE
    else if CHECKWIN(board,opponent) then
        return LOSE_SCORE
    end if
    utility ← utility + COUNTTHREEINROWWITHOPENSPOT(board,player) × THREE_IN_A_ROW_SCORE
    utility ← utility + COUNTTHREEINROWWITHOPENSPOT(board,opponent) × OPPONENT_THREE_SCORE
    utility ← utility + COUNTTWOINROWWITHOPENSPOT(board, player) × TWO_IN_A_ROW_SCORE
    utility ← utility + COUNTTWOINROWWITHOPENSPOT(board,opponent) × OPPONENT_TWO_SCORE
    return utility
end function
```

2.2 Challenges and Strategies:

Discuss the challenges you might encounter in implementing the Minimax algorithm and your utility function for the selected game. Propose strategies to overcome these challenges, considering the complexity of the game and computational limitations.

State space complexity and depth of search could be problems. Connect 4 has a large state space with 7 columns, allowing for 7 moves per round, and 42 empty places. Approximately 7^{42} represents the maximum state space. The maximum depth of the tree is 42, determined by the number of moves before the game ends. The large state space makes it impossible to search all possible states, but using alpha-beta pruning can reduce the number of nodes evaluated by the Minimax algorithm. This optimization skips evaluating branches that won't affect the final decision, thereby improving efficiency. To solve the issue of depth, we can set a limit on how deep the search goes. When the search reaches this limit, it stops. This way, we keep the search within a more manageable depth to calculate states.

3 Part 3: Reflection and Conclusion

3.1 Reflection

Reflect on the process of designing the utility function and how it has deepened your understanding of strategic decision-making in games. Consider the limitations and potential improvements to your approach.

Designing the utility function for Connect 4 has improved my understanding of strategic decision-making in games. First, it helped me understand how to make moves in a more logical way without relying solely on intuition. In fact, it showed me that intuition for some moves comes from understanding certain factors of the game. It helped me see that future gains are as important as current gains, and decisions should be made with that in mind. While I can try to predict the future, I can't always know if a move will lead to winning. I realized that computers also can't calculate all possible moves due to limitations. However, improvements can be made to help the algorithm choose the best option without searching all states. This can be achieved with a good utility function and effective pruning of the search tree, such as limiting the depth and using alpha-beta pruning. The utility function helps to find the best state for us, and pruning helps to focus on the most relevant states for winning. My utility function might not find the best score for every state, but it can be improved by considering some other new factors like how many discs are blocking the opponent or other elements that show how advantageous a state is for the player.

3.2 Conclusion

Summarize your findings and insights gained from the theoretical exploration of the Minimax algorithm and the practical application of designing a utility function in pseudocode.

Exploring the Minimax algorithm and designing a utility function for Connect 4 has provided valuable insights. Firstly, I learned the importance of evaluating both current and future gains when making strategic decisions. This helped me understand how to break down game states into measurable factors. Connect 4 has a large number of possible game states, making it hard to evaluate all options. This showed me the need for efficient algorithms and cutting some states from search space. The utility function design should look for current gains, future opportunities, and blocking the opponent's good positions, which improved my understanding of game strategy. Using alpha-beta pruning was essential to reduce the number of states the algorithm had to evaluate. This made the Minimax algorithm more efficient and allowed for deeper searches. However, setting a depth limit is important for practical applications because evaluating all possible states is still not feasible. Adding a depth limit helps the Minimax algorithm making it more manageable and practical. The utility function has limitations and may not capture all the details of the game. Therefore, with better utility functions, the Minimax algorithm could choose better states.