Joshua Bicera

Professor Tang

CS 4200

08/05/2023

Project 3 CS 4200

Approach:

1. Program interaction: For the program interaction I had it so that it would first prompt the user on who is going first, either the computer or the user. The main function then begins to take turns between the user and the computer where after every move the board is checked for whether or not the game has a winner. For the user interaction I have a getPlayerMove() function that prompts the user for their 2 character input where it reprompts for another input if the input is invalid. So after the player makes a move, the board is checked, the computer makes its move within 5 seconds, the board is checked again, and the cycle repeats.
2. Evaluation Function: The evaluation function plays a very important role as it determines the desirability of the board state of the computer (MAX player). In order to quantify how desirable the board is I assigned different values for the current move trying to be played by evaluating the largest number of consecutive X’s or 0's in the given row or column. I gave anything that can achieve or block a winning move highest priority, achieving or blocking three consecutive spaces' second highest priority, and achieving two consecutive moves the lowest priority. By iterating through every column and row the function

assesses the board and gives a quantified value if a given move will yield the more favorable results.

3. Alpha-Beta Pruning: The alpha-beta pruning algorithm optimizes the search process of finding a move that will yield the most value. It was implemented through a depth-limited search approach so the algorithm stays within the time constraint where there is a maximum depth it is allowed to search until. It also uses the evaluate() function which calculates the desirability of each move where the higher the value the more favorable to the max player (computer) and the lower the value the move is more favored towards the min player (user). The way the algorithm prunes the search tree is by comparing alpha and beta values, if the current move exceeds the beta or falls below alpha, the algorithm removes that subtree of moves and isn't used in the final decision. Additionally, this function limits itself to running for 5 seconds at most in addition to a depth check in order to keep the runtime shorter where if the time runs out or depth is reached it returns the best calculated move so far.

Conclusion:

In conclusion, my implementation of the Alpha-Beta Pruning algorithm is designed to be more simple in terms of evaluation which allows it to explore more moves or greater depth. The use of pruning and depth-limited search allows the program to weigh many different moves in pursuit of achieving the best valued move at the time. I learned a lot from this project as unlike the last project I was able to make my own evaluation function which was very interesting. For a lot of my debugging in order to get the algorithm to work well within the time frame I spent a lot of time adding onto the evaluations and tweaking the weights to increase its efficiency.

Unfortunately due to time I wasn’t able to have the most efficient evaluation function as it is very straight forward, I definitely would spend more time on it if I were to do this project again. It is also worth noting that some unreasonable moves are due to the combination of the 5 second limit and my simple evaluation function. For example, with my code the computer will make moves regardless if it's impossible to make 4 because of the edge of the board such as placing down A6-A8 even though there is no A9. If I had more time I would have implemented the evaluation to better factor in the bounds of the board and possibly weigh the middle of the board with more weight since it provides the most flexibility and options. Overall, I thoroughly enjoyed this project, most likely more than the other two but I do wish I had more time to really optimize my evaluation function but I am happy with what I was able to make.