Project Report

Brian Ramm and Shunya Sunami

Abstract

This study explores the use of minimax algorithms in the real world. The example Connect Four AI program in this project utilizes minimax algorithms to select the best decision when playing Connect Four against human players. The program is very successful and performs significantly better than creators. The study also investigates how the program performs against a random column selector and weak AI depending on the search depth of the algorithm, and the result reveals that the algorithm with depth of 3 would be the best model, considering its winning rate and run time costs. This research contributes to the understanding of the use of minimax algorithm and provides practical insight for creators and researchers of Ai programs for two player turn placed games.

Introduction

In this project, we developed an artificial intelligence (AI) strategy in the board game, Connect Four. Connect Four is a simple game in which the objective is to align 4 pieces in a row. The “board” is a 7 by 6 grid that in physical form is usually placed vertically, and each player takes turns dropping their pieces into one of the 7 columns. The piece will be placed at the lowest available position in that column either at the bottom or on top of the number of pieces placed in that column. When one player lines up four pieces in a row either horizontally, vertically, or diagonally on the grid, then they win the game. Each player will take turns until the object of 4 sequential pieces is achieved or the entire grid is filled in with no winner. This project designed and implemented an AI Connect Four program using the minimax algorithm, one of the most basic game tree search methods.

The minimax algorithm is used to find the optimal strategy in a two-player game, and assumes that each player chooses his or her own moves while taking into account the best moves of the opponents [1]. It recursively navigates the tree and evaluates the favorability of each game-state node, and then selects the optimal move that results in the best score for the AI player and worst score for the opponent. It has been also shown that applying alpha-beta pruning, a technique that reduces the number of nodes that need to be explored, to the minimax algorithm is effective [2]. Therefore, we also utilized this optimization technique in order to increase computational efficiency.

Furthermore, we tested the algorithm using different search depths in order to develop the best AI program. The algorithm with the search depth of 1,3, and 5 were each tested against random column selector and weak AI (our program with search depth of 1), and the winning rate and the average run time were examined. The findings showed that the program with depth of 3 is the best when both running time and winning rate were taken into account. The primary goal for this research is to develop a deep understanding of minimax algorithm, as well as to inform the utilization of the algorithm for game enthusiasts and programming learners.

The remaining sections of this report are organized as follows. First, how the algorithm was developed was detailed in the Algorithm Development section. Subsequently, the Methodology section provides the process of the evaluation of the program. Finally, the Results and the Discussion sections offer the summaries of the winning rate and running costs, and the interpretation of those results.

Algorithm Development

The minimax algorithm is an algorithm for determining the optimal move in a game in which two players take turns playing moves. It aims to make the best move based on the assumption that the opponent will also make the best move. The transition of the game board can be represented as a tree structure, and this game tree is explored in a depth-first manner to determine the next move. In Connect Four, the algorithm recursively evaluates each move and selects the best option.

In this study, we implemented an AI program for Connect Four using python. The core of this program is the min_max_algorithm function. This function first finds the next column where chips can be dropped. It then recursively computes the evaluation scores for the cases where chips are dropped in each column and chooses the next move based on whether the current player is the maximizing or minimizing player.

The evaluation score is calculated using the evaluate_window function and the evaluate function. evaluate_window function takes four adjacent spaces on the board ("window") and computes the score for a particular player ("piece"). If the window consists only of the player's chips and blank, then the score is the 10 to the power of "number of player's pieces". On the other hand, if the window consists only of the opponent's chips and blanks, then the score is -10 to the power of "number of opponent's pieces". It gives a score of 0 except for those situations as this function only cares when windows contain one type of piece.

The evaluate function uses the evaluate_window function to calculate the score for the entire board. Since Connect Four is a game in which the player wins if four chips are placed in a row, the window size is set to 4. It then computes the scores for all horizontal, vertical, and diagonal (positive and negative slope) windows on the board and calculates the total score.

This algorithm has the disadvantage of having a large time complexity as it considers all possible cases. For example, in the case of Connect Four, there are often seven possible next moves, and thus, if all cases are considered, it will take O(7^m) where m is the depth of the search tree.

Therefore, we used alpha-beta pruning in order to reduce the time complexity. We define alpha to be the best value of the maximizer at the moment and beta to be the best value of the minimizer at the moment. As we are maximizing, the search is terminated when the score is larger than beta, whereas when we are minimizing, the search is terminated when the score is is lower than alpha.

Evaluation Methodology

To evaluate the success of the algorithm there were two areas to evaluate the win rate of the AI and the run time. As an initial step, the creators of the algorithm took turns playing against the AI to ensure it was behaving correctly. This was actually humbling as players but satisfying as developers because the AI was able to beat our human play almost every time.

Once we felt confident that the game logic was successful then the program just needed to be optimized. This is done on a minimax algorithm by controlling the depth of the results tree analyzed. A higher depth will increase the number of future outcomes analyzed and result in smarter potential AI decisions but increase the computation time of the algorithm.

To increase the sample size of our tests without needing human players we needed some dummy algorithms to test our AI against. One such dummy was an algorithm that randomly selected columns but was normally distributed to select middle columns more often. The second dummy algorithm was actually the most stripped down version of our AI algorithm to use a depth of one. This would allow us to see how the algorithm competed against itself. A key component is that we always forced our AI algorithm to go second. This is because going first is a big advantage and would overestimate the AI performance. It also allowed us to randomize the first move of the game to simulate different scenarios.

Results and Discussion

Table 1: Win Rate and Average Run Time Against Random Column Selector

Depth	**Win Rate**	**Average Turns to Win**	**Average Turn Time (s)**	**Total Turn Time (s)**
1	0.95	9.42	0.0028	0.0276
3	0.8	7.44	0.0427	0.3176
5	1	8.3	0.6281	5.130

Table 2: Win Rate and Average Turn Time Against Weak AI

Depth	Win Rate	Average Turns to Win	Average Turn Time (s)	Total Turn Time (s)
1	0.2	11.5	0.0044	0.0481
3	1	5.85	0.0530	0.3099
5	1	11.4	0.5131	5.3925

Figure 1: Win Rate Against Random Column Selector and Weak AI for Each Depth

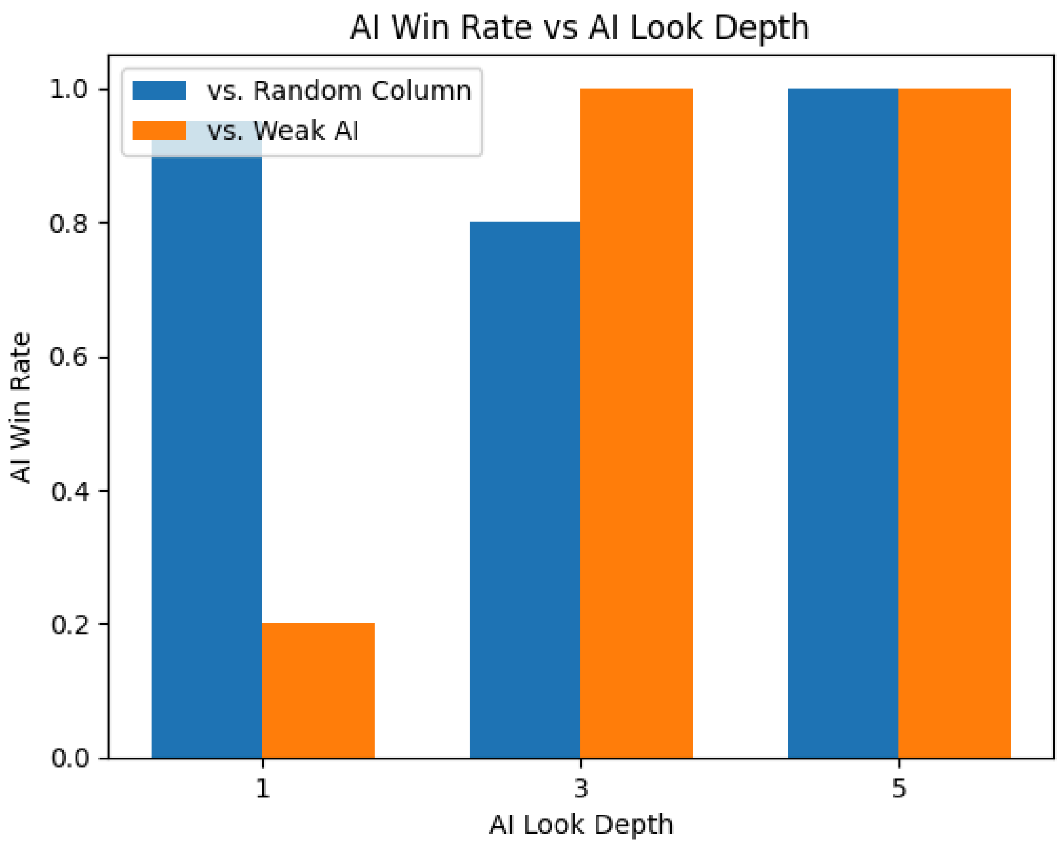
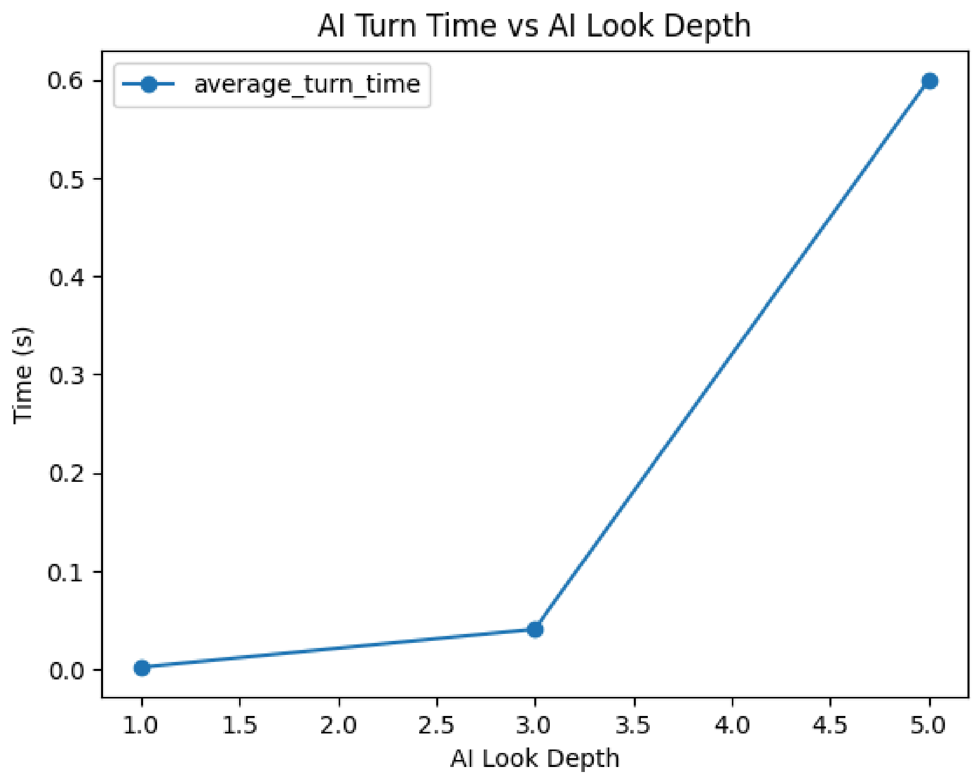


Figure 2: Average Turn Time vs Depth



The results from our tests backed up our observations from playing that the AI is a capable player regardless of the depth level. Whether it was playing against the random distribution column selector or another AI, it won at least 80% of the matches except in one scenario. The only losing scenario was when we set our AI to a search depth of one and it played against another AI with a search depth of one. This is because our AI was at a disadvantage going second.

The turn time results also backed up something that we noticed from our playing experience too. Increasing the search depth has an exponential time cost. Using a depth of 1 and 3 were slightly different but not very perceptible to human players. Increasing to 5 levels had a very noticeable lag when playing. We also tried playing with a depth search of 7 but it was such a tedious and time consuming experiment we were unable to simulate the number of matches needed to obtain results.

Conclusion

The ConnectFour algorithm ultimately was very successful. The results showed that the best overall model would be to play with a search depth of 3. This provided a consistently winning automated player with small run time costs. It also felt the most comparable to going against a human player in terms of reaction speed and decision making. There might be scenarios where a depth of 5 look ahead might be desired if a perfect winning algorithm was needed and the more than 10x run time was not a problem. If someone was using GPUs for instance and had lots of computing power then this might be viable.

The minimax algorithm was a great baseline decision making algorithm for a simple game such as ConnectFour. It actually presented a fairly enjoyable user experience and I’d recommend playing it to anyone. We’ve included some features that display the board to make the user experience a positive one.

References

[1] Shevtekar, Prof & Malpe, Mugdha & Bhaila, Mohammed. (2022). Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 328-333. 10.32628/CSEIT1228644.

[2] B. Swaminathan & R, Vaishali & R, subashri. (2020). Analysis of Minimax Algorithm Using Tic-Tac-Toe. 10.3233/APC200197.