Connect 4 using AI: Evaluating Minimax, MCTS, and Negamax

Chinmayee Dharmastalam Sreedhar Surabhi Sunil

12/11/2024

Abstract

This project explores the evaluation and application of different AI algorithms to play Connect 4, a two-player game that is played on a vertically suspended 6x7 grid. The objective of Connect 4 is to be the first player to align the four discs consecutively in a row -horizontally, diagonally, or vertically. The goal of the project is to implement and compare the performance of three AI algorithms - Minimax(with alpha beta pruning), Monte Carlo Tree Search (MCTS), and Negamax to identify the most effective solution. The algorithms are evaluated based on two criteria: win rates against each other and the time taken to make decisions. The Minimax algorithm, with and without alpha-beta pruning, explores all possible moves to choose the best outcome. MCTS uses probabilistic techniques and random sampling to explore new strategies, while Negamax simplifies the decision-making process by optimizing a single function for both players. Additionally, a custom heuristic was implemented with alpha-beta pruning to enhance the evaluation of game states. This heuristic considers factors such as piece positioning, potential winning combinations, and board control to provide a more nuanced assessment of the game state. By assessing these methods under various game scenarios, this project aims to provide details about the strengths, weaknesses, and computational efficiency of each approach for solving Connect 4.

1 Introduction

Connect 4 is a classic two-player game, which is played on a 6x7 grid, where the goal is to align four discs in a row vertically, horizontally, or diagonally. Although the game is simple and straightforward, its complexity arises due to the large number of possible moves at each stage, making it ideal as an AI problem. There are multiple algorithms that are used to tackle this challenge, such as the Minimax, Monte Carlo Tree Search (MCTS), and Negamax algorithms, which are among the most prominent. Previous works, such as those of Sarhan et al. (2009) [3] and Qiu et al. (2022) [1], provide insights into the efficiency and performance of these algorithms in similar strategic games.[6]

The Minimax algorithm uses a search method which is recursive to explore all the possible future moves by both the players, and selects the optimal move based on the assumption that both players play perfectly. This approach is computationally expensive as the search space is huge, especially when the search depth increases. To improve the minimax algorithm we can use alpha-beta pruning, which eliminates the unnecessary branches, therefore, improving the computational cost. However, MCTS is a flexible algorithm that uses random simulations to explore potential moves, making it suitable for games with a large decision space. Dynamically adjusts the search strategy based on previous outcomes, making sure there is a balance between exploration and exploitation. Negamax is a simpler version of Minimax, which uses a single evaluation function for both players, reducing the complexity of the decision-making.

A custom heuristic was implemented with alpha-beta pruning. This heuristic considers factors such as piece positioning and potential winning combinations to provide an evaluation of the game state. By incorporating this custom heuristic, the algorithm can make informed decisions, potentially leading to improved performance.

The goal of the project is to implement these algorithms, including the custom heuristic, and simulate games by pairing them in various combinations as competing players. Through these simulations, the performance of each algorithm will be evaluated based on win rates and time efficiency. This project

Figure 1: Traditional connect 4 Game

will help in understanding the strengths and weaknesses of these algorithms in decision-making, as well as the impact of the custom heuristic on overall performance.

The project allows for a comparison of different AI approaches to solving Connect 4. By implementing and testing these diverse algorithms, including the custom heuristic, we can gain insights into their relative effectiveness in strategic decision-making, computational efficiency, and adaptability to different game states. This knowledge can be valuable not only for game AI development but also for understanding how these algorithms might be applied to other complex decision-making scenarios in fields such as robotics, logistics, or resource management.

2 Literature Review

The integration of artificial intelligence (AI) into games like Connect 4 has been given significant attention due to its potential to enhance gameplay and provide insights into algorithmic efficiency. There are various techniques and algorithms used, some of which are discussed in this section.

The use of defensive, aggressive and random AI has been a common strategy. Random, as the name implies, randomly picks a play and is the easiest to beat. Defensive AI makes blocking a win a priority, while aggressive AI makes winning a priority. This experiment design also involved a time limit of 3-5 seconds within which a move had to be made. Both are harder to beat than the random AI. The use of MiniMax is generally very computationally inensive, which is why minimax with alpha beta pruning is encouraged to be used. The A* algorithm was also considered, but decided that it may not the best fit due to the time limitations the experiment has imposed. Another program called VICTOR, which is a combination of search table and depth-first search proved that white pawn will always be able to win on a standard 6*7 board. A very similar algorithm to VICTOR was Velena, where it would guarantee a win if the ai was the first player. Neural Networks with back propagation have also been a very common approach to implement game-solving AI.[2]

There have been experiments to find what agent works best in a game of Connect4. A few agents played against each other were Double DQN, Monte Carlo Tree Search (MCTS) and Minimax with alpha-beta pruning. MCTS beat both Double Dqn and Minimax, and further Double Dqn beat minimax. So, Minimax was the worst agent among the three, and MCTS [13] was the best. It also states that Neural Networks with enough given data and architecture, outperforms all the mentioned algorithms.[1]

3 Methodology

This section of the report aims to explain the approach and design of the solution proposed.

3.1 Approach

In our approach to solving the Connect 4 game, we leveraged the AIMA (Artificial Intelligence: A Modern Approach) code base as our foundation. This provided us with a framework for implementing and testing various AI algorithms. The AIMA codebase contains implementation of the algorithms the project intends to use. The Connect 4 game state in AIMA is represented as a two-dimensional array, where each cell contains either an empty space, a player's piece, or an opponent's piece. The empty space is represented by a hyphen, the player and opponent are represented by X's and O's. Minimax with alpha-beta pruning and Monte Carlo Tree Search (MCTS) were implemented through AIMA. A key enhancement to our implementation was the development of a custom heuristic function. This heuristic function was integrated into the alpha beta pruning mechanism. Negamax algorithm was separately implemented on top of the Connect4 class imported from AIMA. A function play_connect4 was developed to indicate which AI algorithm would be player 1 and player 2, handle the printing of the board and indicating the winner of the game. To visualize the game progress, we utilized AIMA's board printing function, which displays the current state of the Connect 4 grid after each move. It also denotes the winner of the game and the time function was used to denote the time it took to finish the game. The different algorithms we used were Minimax with alpha-beta pruning of depth 10, Minimax with modified alpha-beta pruning of depth 10, Negamax and Monte Carlo Tree Search. The following paragraphs describe the algorithms in detail.[5]

The Minimax algorithm was one of them used as an agent. The depth of the alpha-beta pruning was 10. The algorithm recursively evaluates game states, alternating between maximizing the AI's score and minimizing the opponent's score. Without alpha beta pruning, it played out the entire game and then takes a decision. The search space is too big in that case. Alpha-Beta pruning significantly enhances efficiency by eliminating branches of the game tree that cannot influence the final decision. At each level, the algorithm maintains two values: alpha (the best score for the maximizing player) and beta (the best score for the minimizing player). When beta becomes less than or equal to alpha, the algorithm prunes the remaining branches, as they cannot improve the current best move With a depth 10, it allows the AI to look ahead 10 moves, considering both its own and the opponent's potential actions. This pruning dramatically reduces the number of nodes evaluated, allowing for deeper searches within the same time constraints. The algorithm also uses a standard heuristic to evaluate the efficiency of the move.

The next algorithm used in the project was Minimax algorithm with alpha beta pruning with a custom heuristic for the evaluation of the move. There is a evaluation table defined which basically denotes a value to each cell of the game board. This value is defined based on the number of winning sequences this cell can be involved in. For example, the corner pieces of the table can always be maximally involved in 3 winning sequences, horizontal, vertical or diagonal. The evaluation function starts by handing a base initial value of 300 and identifies the opponent and the current player. It then starts by finding 4 connected or winning sequences for both players, if it finds the sequences for either of the player, it returns a very high value, and ends the search. If there is no winning sequence, the function proceeds to evaluate the board state by visiting each cell and adds the evaluation table value to the player while also subtracting the same value from the opponent, depending on whose pawn has been placed on the cell. It then counts sequences of three and two connected pieces for the current player, assigning them weights of 3 and 1 respectively, and adds these to the heuristic sum. Additionally, it counts sequences of two connected pieces for the opponent, assigning a weight of 2, and subtracts this from the heuristic sum. This approach shows that the AI has a mix of both defensive and aggressive strategy.

The next algorithm we used was Negamax. Negamax is similar to minimax in terms of building a search tree and iterating through every move possibility until the end of game. It is described to be a robust algorithm to be used under time constraints. It simplifies the minimax algorithm by leveraging the zero-sum nature of the game, where one player's gain is the other's loss. The algorithm recursively

evaluates game states, alternating between players by negating the score. By negating the score, both the players become maximizing players. When it's the opponent's turn, instead of minimizing their score, it maximizes the negation of their score. This simplifies the implementation as we need only one maximizing function and the negation of the score. In Connect 4, the algorithm evaluates positions to a chosen depth, typically assigning higher scores to states with more potential winning sequences. It uses a window defined by alpha and beta parameters to narrow the search space. If a move is found that's better than the current best move (beta), the algorithm can prune the remaining moves in that branch, as they won't affect the outcome. This allows for deeper searches and stronger play within time constraints. This has more of a defensive player strategy to it, in terms of making sure t minimizing the players score.[4]

The last algorithm we considered using was Monte Carlo Tree Search. The literature survey indicated that this was one of the best performing algorithms for Connect4. MCTS builds a game tree by iteratively performing four steps: selection, expansion, simulation, and backpropagation. During selection, the algorithm traverses the existing tree, choosing nodes based on their potential value using the Upper Confidence Bound formula. This ensures a balance between exploring new strategies and exploiting known strong moves. In the expansion phase, a new node is added to represent a possible game state. The simulation step then plays out a random game from this new state to its conclusion. Finally, backpropagation updates the statistics of all nodes in the path from the new node to the root, reflecting the outcome of the simulation. This process is repeated many times, gradually building a tree that focuses on the most promising moves. Essentially, it plays out a random set of games until the winner and updates the tree as it finishes each game. As the tree grows, MCTS becomes increasingly skilled at identifying strong positions and optimal plays in Connect 4, making it a formidable opponent even in complex game states.[3]

3.2 Design

The experiment was designed to check which of the players performed better. The agents were played against each other to see the performance in terms of win rate and the average time taken to complete the game. The agents were played against each other a few times one by one to see what interesting patterns would appear. The pattern observed between the tree search algorithms, Minimax with alpha beta pruning, Negamax and Minimax with alpha beta pruning with custom heuristic was that, the initial starting position would not change. Each of the algorithms when initiating the game, regardless of player 2 would choose the same starting position. The player 2 agent, if it was also a tree search agent, end up choosing the same second move. The game plays were identical, with very similar time taken to complete the game. We believe this is because there is no element of randomness or there is no change in the heuristics when the exact same moves are made. The results would be the same for one game or 50 games. The winner would be the same. The results are described in the table below. The same behaviour doesnt apply to MCTS beacuse of the randomness of the games played.

Starting State	**Player 1**	**Player 2**	**Winner**	**Total Time (seconds)**
1,4	Custom_abs	Alpha beta search	Tie	28.3
1,1	Alpha beta search	Custom_abs	Player 2	19.7

Table 1: Results of Game Simulations

The different game states are printed in the project in the following manner.

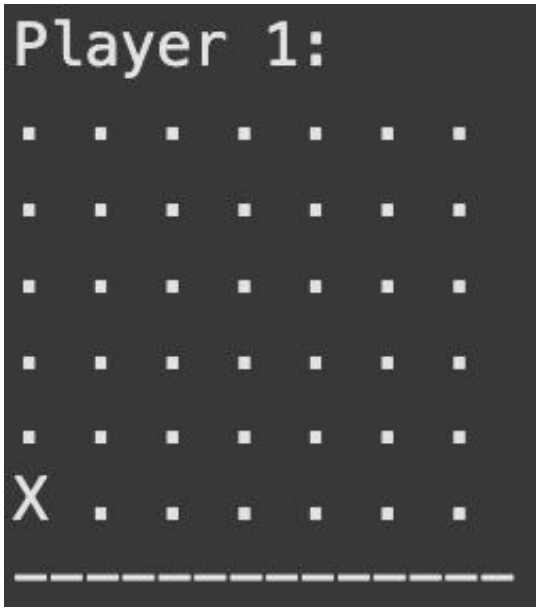

Figure 2: Initial game state after one move

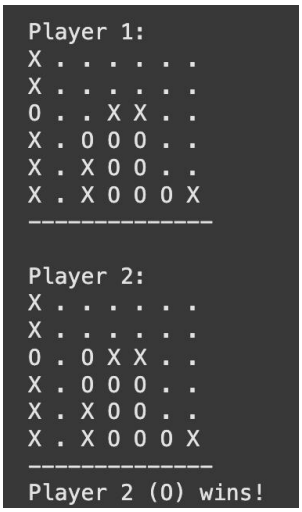

Figure 3: Final game state with a winner printed

4 Results

The objective of our project was to evaluate the performance of various search algorithms in the game-playing context. We conducted a series of match between different algorithm players to assess their efficiency and effectiveness. The key metrics recorded were the number of games played, the wins, and ties for each player and the total time taken to complete the games.

Our findings from these simulations indicate significant variation. The MCTS player was the best when played against both alpha beta search and custom heuristic players. In contrast, the custom heuristic player struggled against the MCTS but performed well against negamax. This could be because the search space depth for Negamax was limited to 5 as it was taking a long time to run 50 simulations. The Alpha-beta search algorithm showed competitive performance, especially against the MCTS. This was expected because the nature of both the algorithms is heuristic-based.

The results are summarized below and provide a detailed insight into the strengths and weakness of each algorithm.

Number of Games Played	**Player 1**	**Player 2**	**Winner**	**Total Time (seconds)**	**Average Time per Game (seconds)**
50	MCTS_player	Custom_abs	P1 = 48, P2 = 1, Tie = 1	5304.77	106.10
50	Custom_abs	MCTS_player	P1 = 3, P2 = 46, Tie = 1	6197.38	123.95
50	MCTS_player	Alpha_beta_search	P1 = 49, P2 = 0, Tie = 1	2078.70	41.57
50	Alpha_beta_search	MCTS_player	P1 = 0, P2 = 50, Tie = 0	1830.31	36.61
10	Negamax	MCTS_player	P1 = 0, P2 = 10, Tie = 0	482.53	48.25
10	Negamax	Custom_abs	P1 = 0, P2 = 10, Tie = 0	2213.86	221.39
50	Negamax	Alpha$_b$*etasearch*	P1 = 0, P2 = 0, Tie = 50	450.3133686	9.00

Table 2: Results of Game Simulations

5 Conclusion

In conclusion, we can say that the best performing algorithm is MCTS player as it has more number of wins against the custom heuristic, alpha beta search and Negamax players.

The worst performing algorithm is the Negamax. This could be because of the restrictive nature in our code. The search space depth was limited to 5 making it the worst performer. The future plan for the project would be to run more simulations for a larger dataset and get more data. Based on this data, the custom heuristic function could be modified to perform better and to have more features involved. There is scope for improvement in the time taken for decision making as well. Neural networks are a very important field to consider for game-solving algorithms

References

[1] A. M. Sarhan, A. Shaout and M. Shock, "Real-Time Connect 4 Game Using Artificial Intelligence," *Journal of Computer Science*, vol. 5, no. 4, pp. 283–289, 2009.

[2] K. Sheoran, G. Dhand, M. Dabaszs, N. Dahiya and P. Pushparaj, "Solving Connect 4 Using Optimized Minimax and Monte Carlo Tree Search," *Advances and Applications in Mathematical Sciences*, vol. 21, no. 6, 2022.

[3] C. Browne et al., "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[4] Megha Chakole1, Rohini Nawathe1, Sanjay Dorle2 ,Gargi Bhakre1, Nita Nimbarte, "Optimal Strategy Formulation for Tic-Tac-Toe Using Minimax Algorithm for Interactive Gaming" *Communications on Applied Nonlinear Analysis*, Vol 31 No. 2

[5] Santiago Videgaín a, Pablo García Sánchez, "Performance Study of Minimax and ReinforcementLearning Agents Playing the Turn-based Game Iwoki" *APPLIED ARTIFICIAL INTELLIGENCE*, 2021, VOL. 35, NO. 10, 717–744

[6] Game Theory and Algorithms: A Comprehensive Exploration, *Medium*, Jul 22, 2024