A Comparative Study of Artificial Intelligence Algorithms for Variants of Connect Four

Peter Holmes, Freya Nagel, Yonatan Paserman, Rebecca Wilhelmi

1 Introduction

Connect Four is a two-player strategy game that appears simple at first glance but grows exponentially in complexity with each move. In its classic form, players alternate turns dropping colored discs into a vertical grid with 7 columns and 6 rows. The objective is to align four of their discs consecutively in a vertical, horizontal, or diagonal direction. As a finite and solvable game, Connect Four serves as an ideal environment for exploring artificial intelligence algorithms. In this paper, we apply and evaluate three AI techniques to the game: Monte Carlo Tree Search (MCTS), Alpha-Beta Pruning (AB), and Alpha-Beta Pruning with an evaluation function (ABE). These algorithms were assessed based on their performance, measured by win rates, and their computational efficiency, measured by execution time. Hyperparameter tuning was performed on all three algorithms to maximize their effectiveness. To analyze strengths and limitations of each algorithm, we introduce novel variations to the game, including the addition of a third player, modifications to the board size, and randomly placed blocker stones. These variations provide a robust framework for understanding the capabilities and constraints of the algorithms while addressing gaps in the existing literature on Connect Four AI gameplay.

1.1 Review of Related Work

There exists a substantial amount of research on the design and evaluation of AI algorithms for playing Connect Four. Much of the reviewed literature focuses on classical performance comparisons between search algorithms or state-of-the-art methods, such as, Reinforcement Learning-based agents. For instance, Tommy et al. (2017) compared AB Pruning and MTD(f)[1] in terms of win percentage, execution time and the number of evaluated leaf nodes. They found that MTD(f) is as optimal as AB Pruning at Connect Four, but MTD(f) is on average faster and evaluates fewer leaf nodes than AB Pruning making it a more efficient solution. They further found that existing AI algorithms such as random, brute force and greedy algorithms are fast but often return suboptimal moves. On the contrary, Minimax variants provide optimal moves but have longer execution times, especially at deeper search depths. Sawwan (2021) conducted an evaluation of the Minimax algorithm, Alpha-Beta pruning, and a utility function heuristic evaluator playing Connect Four. The AIs were tested under varying maximum depths for game-tree search. Across all configurations, the AIs consistently defeated a random player. However, it was observed that the Minimax algo-

1. A modified alpha-beta game tree search algorithm that utilizes ”zero-window” initial search limits and incorporates memory structures to efficiently reuse intermediate results.

rithm without Alpha-Beta pruning required impractically long computation times at depths greater than 4, while the Alpha-Beta pruning algorithm exhibited similar inefficiencies at depths beyond 6. The study revealed two key findings: (1) AIs configured with greater search depths consistently outperformed those with lower depths, and (2) the first-moving AI always prevailed when competing against an identical AI model with the same search depth. Allis (1988) took a different approach by using a Shannon C-type strategy to play Connect Four, rather than evaluating common AIs like other researchers. Instead of relying on brute-force search, this strategy applied rules and principles based on expert knowledge of the game. The goal was to make the AI play like a skilled human by using patterns, tactics, and strategies rather than exhaustive calculations. The program was tested on the standard 7x6 board as well as smaller and larger board sizes. The results showed that while the complexity of securing a win increases with board size, the strategic framework worked well across different sizes. Leung (2025) investigates the use of two classical reinforcement learning algorithms, Q-Learning and Monte Carlo Policy Iteration (MC-PI), to develop an AI capable of playing Connect Four. He framed the game as a finite-horizon Markov Decision Process (MDP), where states represent board configurations and actions correspond to column selections. Due to computational limitations, experiments were performed on a smaller 5x4 board instead of the standard 7x6 size. Training data was generated through self-play, allowing the AI to learn strategies without relying on external datasets. Leung finds that Q-Learning surpasses MC-PI in terms of convergence speed and win rates, particularly when tested against stronger Monte Carlo Tree Search (MCTS)-based opponents. Sheoran et al. (2022) explored optimizing the Minimax algorithm to enable the agent to search deeper into the game tree. The goal was to enhance Minimax performance to surpass MCTS, and by incorporating dynamic programming, the Minimax algorithm was optimized to reach a depth of N = 5. The results showed that the optimized Minimax agent achieved a 79% win rate against MCTS, making it the best-performing bot in the study.

The review of existing literature shows that most papers aim to compare different algorithms with respect to win percentage, execution time and the number of evaluated leaf nodes when playing connect four on the standard 7x6 board. Not much research has been done on how the performance of the evaluated algorithms changes with respect to changes in board size or other modifications of the game. We aim to address this gap in research by evaluating the performance of different algorithms for different variations of the game, namely: smaller and larger board sizes; the introduction of a third random player as well as adding blocker stones.

2 Problem Formulation

2.1 Connect Four Game Definitions

Connect Four is a fully observable game as the complete board and move history are known at all times, and there is no hidden information. The initial state is a 7x6 grid with no discs placed. There are 42 total squares, and each square has 3 possible states - either it is empty, has a player 1 disc, or has a player 2 disc. Therefore, the total state space = 3^{42}, which is approximately 1.44 x 10^{20} states. Actions refer to the valid move a player can make in Connect Four. A player can drop a disc in one of the 7 columns giving the player 7 possible actions for each turn. The goal state is the final state that determines the end of the game. This can occur one of two ways, either one player has four tiles consecutively horizontally, vertically or diagonally, which means that player wins or all the board pieces are full, but no player meets the winning condition, thus it is considered a draw.

2.2 Additional Rules

While there has been extensive research on algorithms to optimize Connect Four, the aim of this paper is to compare and contrast several AI algorithms with game modifications. Specifically, we are changing three rules to see how the algorithms adapt and perform. The first change is to test different board sizes. The typical Connect 4 games have a 7x6 board, but we will explore how the algorithms behave with both smaller (4x4) and larger (10x10) board dimensions, in addition to a strange sized (2x15) board. This will change the state space from 3^{42} to 3^{h*w} with h representing the board height and w representing the board width. The next change we are implementing is adding a third player, instead of the traditional two. Now, each square has four possible states: empty, player 1 disc, player 2 disc, or player 3 disc. Then, the state space becomes 4^{42} when playing on a traditional game board. Finally, the last rule change we are testing is a 7x6 board with 8 randomly generated spaces that have a blocker. This means that no agent can place a disc on the square that has a blocker, but they can place a disc above and below it. By testing these rule changes, we aim to better understand how various AIs perform in altered environments.

2.3 Algorithm Definitions

Monte Carlo Tree Search (MCTS) is a heuristic search method that focuses on analyzing the most promising moves by expanding the search tree based on random sampling of the search space. Each round of MCTS consists of four steps: selection, expansion, simulation, and back propagation, and these rounds are repeated until the time allotted ends or the number of rounds is capped. MCTS ranks potential moves using the Upper Confidence bound applied to trees. The algorithm for this is

$$UCT(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

Where U(n) is the total utility of playouts through node n, N(n) is the number of playouts through n, parent(n) is the parent node of n, and C balances between exploration and exploitation. The first part of this algorithm accounts for exploitation, while the second part accounts for exploration (Sheoran et al. (2022)).

Alpha-Beta Cutoff Search (AB) is an optimization of the Minimax algorithm. The Minimax algorithm works by exploring the game tree, where nodes are the state and edges are possible moves, and it has a maximizing player that is trying to maximize their utility score at each step, and a minimizer player that is trying to minimize the maximizers utility score. This algorithm utilizes a heuristic evaluation function which returns an estimate of the expected utility of the game from a given position. AB utilizes this Minimax algorithm, but the cutoff or pruning allows the algorithm to ignore part of the search tree that makes no difference to the final choice. The algorithm uses two parameters, "alpha" and "beta." The agent aims to maximize the score, which is tracked by the alpha parameter, while the minimizer seeks to reduce the agent's score, represented by the beta parameter, Alpha starts with a value of 0 or negative infinity, and beta starts with a value of positive infinity. The algorithm keeps updating these parameters, and the pruning can occur when the agent cannot do better than the current parameter. This is a benefit because it significantly reduces computation time, but still returns the same results as the Minimax algorithm (Sawwan (2021)).

3 Proposed Solution

3.1 Initial Game Design

To compare and contrast the different algorithms, we first created a Connect Four Game class that defines the necessary game components such as actions, results, utilities, and the game board.[2]. This class encapsulates the core game logic, including the rules for valid moves, win conditions, and utility calculations.

3.2 AI Agents

Next, we implemented three different AI bots: Monte Carlo Tree Search and Alpha-Beta with and without an evaluation function. MCTS allows the user to define N - the number of simulations the algorithm performs before making a decision. Alpha-Beta Cutoff Search allows the user to specify the search depth, which is the maximum number of moves the bot will look ahead in the game, and whether to use an evaluation function to assess board state. For the evaluation function, we implemented a heuristic based on 4 space segments on the board. The evaluation function calculates scores for all 4-space sections on the board and returns the overall score based on the player's most favorable configuration.

3.3 Game Design with Additional Rules

To accommodate the introduced game variations, we implemented several modifications to our codebase. First, we enabled variable-sized game boards by allowing users to specify the height and width, which required updating the evaluation function used by the Alpha-Beta bot to handle boards of different dimensions. Second, we modified the Connect Four game class to support three players. This involved updating the utility function and introducing a new method to determine the next player in a cyclic order (Player 1 $\rightarrow$ Player 2 $\rightarrow$ Player 3 $\rightarrow$ Player 1). The evaluation function was also adapted to account for a third player. Finally, for the blocker stones variation, we added a list to store obstacle positions and updated the action method in the game class to account for blocked moves.

4 Numerical Experiments

The experiments can be split into two steps. At first, the algorithms are trained on the classical 6x7 board to identify their optimal hyperparameters. We then let the bots with the optimal hyperparameter configurations play against each other - 60 times for each variation

2. We built on top of the Game and C4 class adapted from the publicly available aima-python repository Norvig and Contributors (2025)

of the game that has been described above. The best bot is then identified based on its overall score of the evaluation metric explained below.

4.1 Hyperparameter Tuning

Both hyperparameters, namely the number of simulations N that the algorithm will perform before making a decision (MCTS) and the cutoff-depth d of the game-tree-search (AB) are correlated with the algorithm's execution time and win percentage. To consider the trade-off between win percentage and computation time, we have defined the following evaluation metric to identify the optimal hyperparameters:

$$\text{Evalmetric} = \frac{\text{Percentage of Wins}}{\sqrt{\text{Average Time per Move}}}$$

For MCTS, the set of N = 1000, 5000, 10000 was tested. For each N, 360 game simulations were played on the classic 6x7 board, more precisely 30 games against each Alpha-Beta Cutoff Search algorithm depth/evaluation type: AB with depth 3, depth 4, depth 5 for with and without a heuristic function. For Alpha-Beta Cutoff Search, the hyperparameter tuning was conducted in the same manner: each AB or ABE algorithm played 360 games against all other potential opponents.

Table 1: Performance Results of the Hyperparameter Tuning

Bot	**% Wins**	**Avg. Move Time (s)**	**Evaluation Metric**
AB_3	3.06%	0.05	13.06
AB_4	5.00%	0.13	**14.10**
AB_5	7.50%	0.74	8.73
ABE_3	67.50%	1.05	**65.81**
ABE_4	71.39%	4.05	35.48
ABE_5	74.72%	13.94	20.01
MCT_1000	56.94%	5.60	**24.05**
MCT_5000	74.17%	33.19	12.87
MCT_10000	80.56%	66.34	9.89

Table 1 shows the best hyperparameter configurations based on our evaluation metric. Notably, our hyperparameter tuning fully confirms the results found by Sawwan (2021): AIs with greater search depths consistently outperform those with lower search depths in terms of win rate at the cost of greater execution time. We chose to slightly prioritize win percentage over execution, indicated by taking the square root of the average time per move. This usually results in us selecting the bots with the lowest N or d, with the exception of AB_4 selected over AB_3 as they have relatively similar execution time. We choose AB_4, ABE_3 and MC_1000 as candidates for our bot tournament in the following sections.

4.2 Tournament

The tournament was set up as follows: Each of the chosen algorithms with the optimal hyperparameter determined in the previous chapter competed against each other in 60 games for each of our three variations: the different board sizes, the 3-player game and the blocker stones. Tables 2, 3, 4 and 5 show the results of the matches.

4.2.1 Standard Game

First, we test the MCTS and the AB-bots on a standard game of Connect Four, with no rule changes. The tournament consists of each bot playing 60 games. The results are seen in Table 2 and show that the ABE_3 bot outperforms the others and beats the AB_4 bot in every single game.[3]

Table 2: Simulation Results for Standard Game

Bot	Opponent	Bot Position	# of Bot Wins	% Wins	Avg. Move Time (s)
MCT_1000	AB_4	First Player	12	71.67%	0.1908
		Second Player	15		
	ABE_3	First Player	8		
		Second Player	8		
AB_4	MCT_1000	First Player	0	5.00%	0.0043
		Second Player	3		
	ABE_3	First Player	0		
		Second Player	0		
ABE_3	MCT_1000	First Player	7	73.33%	0.0291
		Second Player	7		
	AB_4	First Player	15		
		Second Player	15		

4.2.2 Variation of Board Sizes

The MCTS and the AB-bots competed on three different board sizes outside of the traditional 6x7 board: 4x4, 2x15, 10x10. Table 3 shows the results for these tournaments. As can be seen, the results differ dependent on the board size: First thing to note is that the smaller or the more out-of-shape the board, the higher the percentage of draws. The table shows that for the 4x4 and the 2x15 board, approximately 50% and 27% of the matches resulted in draws, respectively. Obviously, the average execution time was also decreased with the decreased board size. The ABE_3 bot did best on both the 10x10 board as well as on the 2x15 board, while its execution time is always below MCT_1000 but above AB_4. Note that the ABE_3 bot never loses to the AB_4 bot, except for on the smallest board (4x4), where the latter does best. Interestingly, all of AB_4 wins against ABE_3 in the 4x4 board come when AB_4 plays as the second player. MCT_1000 also rarely wins against the ABE_3 bot, and its execution time is always notably higher. Overall, the results show that the ABE_3 bot is best suited to handle variations of the environment.

4.2.3 Three Players

The MCTS and the AB-bots then compete with three players version of the game instead of the traditional two. The tournament was 60 games and each bot played in all the games.

3. Note: The % of wins was calculated by dividing the number of wins per bot by the number of games it played (60). The total number of wins doesn't add up to 180 as some games ended in a draw.

Table 3: Simulation Results for Different Board Sizes

Board Size	Bot	Wins per Opponent		% Wins	Avg. Move Time (s)
10x10	MCT_1000	AB_4 ABE_3	22 9	51.7%	0.4637
	AB_4	MCT_1000 ABE_3	8 0	13.3%	0.0278
	ABE_3	MCT_1000 AB_4	21 30	85.0%	0.4173
4x4	MCT_1000	AB_4 ABE_3	14 0	23.3%	0.0883
	AB_4	MCT_1000 ABE_3	3 15	30.0%	0.0006
	ABE_3	MCT_1000 AB_4	14 0	23.3%	0.0010
2x15	MCT_1000	AB_4 ABE_3	8 2	16.7%	0.2741
	AB_4	MCT_1000 ABE_3	16 0	26.7%	0.0156
	ABE_3	MCT_1000 AB_4	25 15	66.7%	0.0927

Each bot took turns being the first, second, or third player. Having three players caused the order of play to be important, and it was found that the first player won 9 times, the second player won 15 times, and the third player won 18 times, indicating that it was preferred to play as the latter player. The number of draws also increased in this variation as opposed to the traditional game, and about 30% of the time no one won. Table 4 shows the results for this tournament. As can be seen in Table 4, ABE_3 was the clear winner out of the three players.

Table 4: Simulation Results for Three Players

Bot	Wins as 1st	Wins as 2nd	Wins as 3rd	% Wins	Avg. Move Time (s)
MCT_1000	1	0	4	8.33%	0.1126
AB_4	1	5	2	13.33%	0.0011
ABE_3	7	10	12	48.33%	0.0137

4.2.4 Blocker Stones

The final rule variation is having the MCTS and the AB-bots play on a board that randomly has 8 spaces that are blocked off. No opponent can place a piece on a space that is blocked off, but the blocked off positions are fully observable. The tournament occurred by each players playing 60 games, and they each played on 15 boards with the blocker stones randomly generated. Table 5 shows the results for this tournament with the opponents and the order of players, in addition to the win percentage and average time per move for each player. There does not appear to be a strong advantage for playing first or second, and ABE_3 was again the clear winner, winning 68.33% of the time.

Table 5: Simulation Results for Blocker Stones

Bot	Opponent	Bot Position	# of Bot Wins	% Wins	Avg. Move Time (s)
MCT_1000	AB_4	First Player	13	48.33%	0.3852
		Second Player	11		
	ABE_3	First Player	4		
		Second Player	1		
AB_4	MCT_1000	First Player	2	6.67%	0.0073
		Second Player	1		
	ABE_3	First Player	1		
		Second Player	0		
ABE_3	MCT_1000	First Player	12	68.33%	0.0290
		Second Player	8		
	AB_4	First Player	11		
		Second Player	10		

5 Conclusion

In this paper, we evaluated three artificial intelligence techniques - Monte Carlo Tree Search, Alpha Beta Pruning, and Alpha Beta Pruning with an evaluation function - on the game Connect Four. The introduction of rule variations - different board sizes, a third player, and blocker stones - highlighted the algorithms' adaptability and performance across different game conditions. When we first tuned the hyperparameters, comparing percentage of wins and the square root of average times per move, the optimal bots were the ABE_3, AB_4, and MCT_1000. Since our evaluation metric places a higher emphasis on wins than computation time, we ended up selecting AB_4 over AB_3, as it has a greater win percentage, costing only slightly longer execution time. We then used these bots on a variety of tournaments, and found that in a standard game MCT_1000 and ABE_3 have similarly high percentages of winning, 71.66% and 73.33% respectively. However, once we made our rule variations and introduced more complicated games, we found that ABE_3 outperformed MCT_1000 in terms of win percentages on everything except for on a 4x4 board. This shows that the ABE_3 bot is the most flexible and able to adapt to exponentially more difficult situations. While MCTS demonstrated strong performance with large branching factors and uncertain game states, Alpha-Beta with an evaluation function excelled in providing faster decision-making in a more constrained search space, especially after tuning the hyperparameters. Our results suggest that while no single algorithm outperformed all others across every condition, a combination of MCTS for exploration and Alpha-Beta for exploitation provides a strong foundation for building efficient game-playing agents in complex environments. Areas for future research could include implementing additional bots to the traditional Connect Four game and to the rule modifications we made such as a reinforcement learning bot. In conclusion, the study highlights the practical applications of classical AI techniques in a game environment, while also emphasizing the challenges posed by non-traditional game variants.

References

L. V. Allis. A knowledge-based approach of connect-four. *Journal of the International Computer Games Association*, 11(4):165, 1988.

B. Leung. Creating a reinforcement learning ai to play connect four, 2025. URL `https://b7leung.github.io/files/Connect%20Four.pdf`. Accessed: Jan. 18, 2025.

Peter Norvig and Contributors. aima-python: Implementations of algorithms from artificial intelligence: A modern approach. `https://github.com/aimacode/aima-python`, 2025. Accessed: 2025-01-21.

A. Sawwan. Artificial intelligence-based connect 4 player using python. Technical report, AI Course Project Document, Temple University, May 2021. Online.

K. Sheoran, G. Dhand, M. Dabaszs, N. Dahiya, and P. Pushparaj. Solving connect 4 using optimized minimax and monte carlo tree search. *Advances in Applied Mathematics and Sciences*, 21(6):3303–3313, Apr. 2022.

L. Tommy, M. Hardjianto, and N. Agani. The analysis of alpha beta pruning and mtd (f) algorithm to determine the best algorithm to be implemented at connect four prototype. In *IOP Conference Series: Materials Science and Engineering*, volume 190, page 012044. IOP Publishing, Apr. 2017.