American University of Armenia

College of Science and Engineering

CS 246: Artificial Intelligence Project

Efficient Ways to Play Connect-4

Team: Anna Asatryan, Armine Babajanyan,

Ruzanna Karamyan, Nane Hayrapetyan

Fall 2024

Abstract

Connect-4 is a two-player strategy game where various artificial intelligence techniques can be explored. The study evaluates multiple AI algorithms and heuristics to identify strategies for solving Connect-4. It explores the Monte Carlo Tree Search (MCTS), Minimax algorithm variations, and heuristic approaches, such as feature-based and board-based methods. Through testing, the project assesses different agents' computation time, memory usage, and win rates. Results highlight the first-player advantage, effectiveness of Minimax with the Alpha-Beta Pruning algorithm, and MCTS's high computational cost.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview of Connect-4

Connect-4 (Connect Four) is a two-player game played on a 6-row, 7-column grid with the aim of, as its name suggests, connecting four discs of a chosen color horizontally, vertically, or diagonally. It is like an enhanced version of Tic-Tac-Toe, which also incorporates a vertical grid, similar to Tetris, with discs falling to the lowest available position in a column due to gravity. At the beginning of the game, each player chooses a color and, on each turn, places a disc of that color onto the board. The nature of the game can be described as a zero-sum, deterministic, non-dynamic, multiplayer environment, aligning closely with topics covered in our Artificial Intelligence course, such as heuristic evaluation functions, adversarial search techniques, and game trees. The goal of this project is to compare different game tree strategies, such as Alpha-Beta Pruning and Monte Carlo Tree Search, and to explore different heuristics used to win Connect-4 [All88].

1.2 History of the problem

The origins of the game are debated, with some claiming it dates back to the voyages of James Cook, who allegedly spent so much time playing it that it became known as *The Captain's Mistress*. However, according to official sources, the game was created by

Howard Wexler and first released under the *Connect Four* trademark by Milton Bradley in 1974. Connect-4's large state space and perfect information structure make it ideal for testing AI algorithms. The game was first solved by James D. Allen in 1988 and independently by Victor Allis two weeks later during the same year. Their solutions demonstrated that the first player can always win with perfect play [MIT10].

1.3 The Structure of the Project Paper

The project draft consists of an Introduction, Literature Review, and Current Progress and Plan sections, which will later be replaced by the Methodology and Conclusion chapters. The Introduction discusses the rules and history of Connect-4, while the Literature Review examines various AI solutions for the game. The Current Progress and Plan sections talk about our future steps and ideas for the final draft. Finally, a Bibliography will be provided.

Chapter 2

Literature Review

2.1 Problem Setting and Description

Connect-4 is a two-player game where each of the players chooses a color (either red or yellow) and takes turns by dropping their colored discs into any unfilled columns on a 7x6 game board. The board is vertically oriented, meaning when a player drops a disc into a column, it takes the lowest possible place of the column. The aim for each of the players is to line up four discs in a row horizontally, vertically, or diagonally.

The board consists of 6 rows and 7 columns. Each position on the grid can be empty or occupied by either red or yellow discs. Usually, a player with red discs starts the game.

The state space of the game represents the number of all the configurations of the board at any time point during the game. We can imagine a single state of the board as a matrix with 42 elements (possible positions for discs), each with the possibility of taking 3 values (red, yellow, empty). So, the number of possible configurations is

$$3^{42} \approx 1.0941899 \times 10^{20}$$

At any given state, there are at most 7 possible actions for players, as there are 7 columns to drop a disc. However, if a column has reached its maximum height of 6 discs, the possible actions decrease.

The transition model represents how the state of the board changes after each action. After each move, the chosen column of the board is filled with one disc, which takes place at the lowest possible position. Other columns stay unchanged.

The player reaches a win state when he/she has 4 consecutive discs in the same row, column, or diagonal. The game ends when either a win state is reached by one of the players or the board is filled, and no player wins, resulting in a draw. Thus, we have 2 terminal states: win and draw.

At the beginning of the game, the root node of the game tree represents the empty board. Later on, each branch of the tree represents a state as a result of a move made by the player. The number of branches increases exponentially at each depth since the number of possible states increases with each move. The game tree potentially can have a depth of 42 since there are 6 rows and seven columns on the board, and each of the cells might be filled with a disc. The tree has a branching factor of 7 since there are 7 columns the player can choose from. Still, the branching factor decreases as the game goes on since the columns fill up, and less space is left for discs.

2.2 Existing Algorithms for Connect-4

2.2.1 Monte Carlo Tree Search Algorithm

Monte Carlo methods are widely used in science. They are based on repeating random simulations to get numerical results [Fu18]. In our case, random samples are the random moves of players, and we will estimate the probability of winning with each move. The idea is to simulate a large number of board configurations from a given state which allows us to know which states will lead to a better outcome. So, the algorithm allows us to choose better moves because we already know that their result would be better.

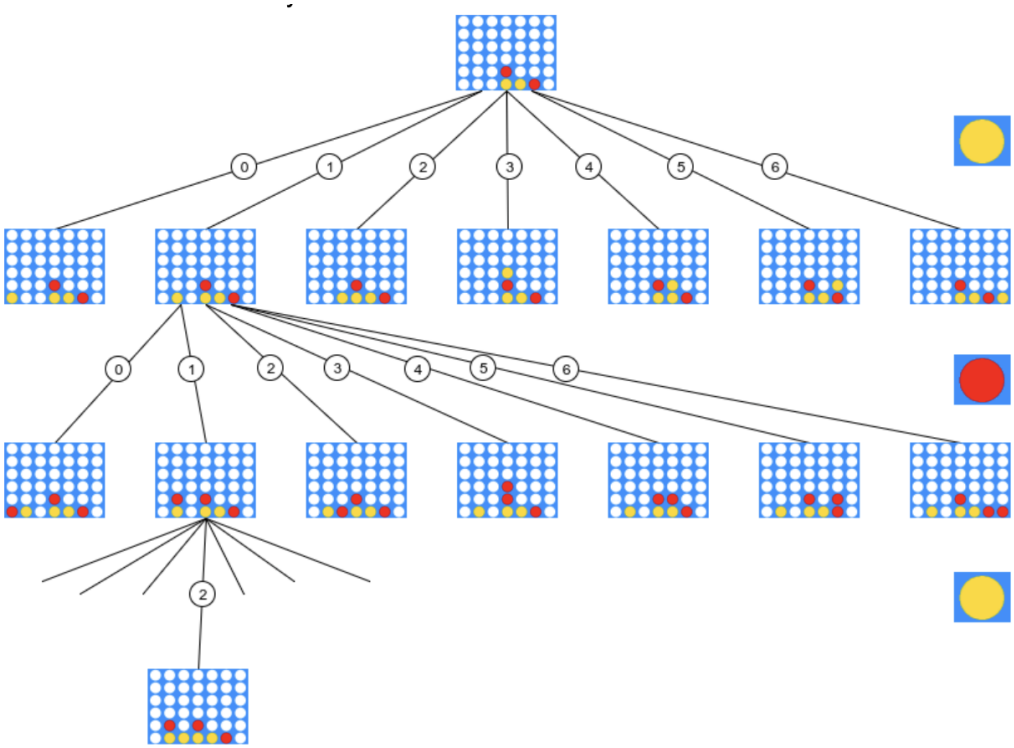

Figure 2.1: Illustration of the Connect-4 Game Tree with Move Sequences

We can divide the Monte Carlo tree search algorithm into 4 stages:

1. selection
2. expansion
3. simulation (sometimes called Playout)
4. backpropagation

[SDD+22]

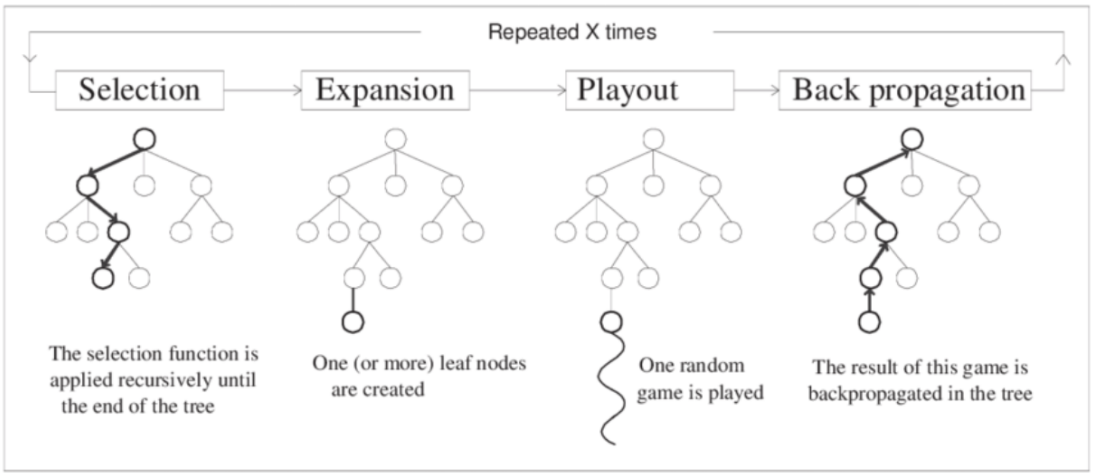

Figure 2.2: Monte Carlo Tree Search Workflow for Connect-4

Selection phase

This is the first phase of our algorithm where we select a node to then expand and simulate a game from. The tree consists of nodes that represent different board states. Initially, we have a root node (initial state) that has no children yet. Our aim is to select a leaf node for simulating the outcome. MCTS then traverses the tree by selecting the child nodes based on a specific policy. As our policy while selecting a node, we will use an Upper Confidence Bound applied to Trees to calculate the value of the node.

$$UCT(n_i) = \frac{Q}{N} + c_i\sqrt{\frac{logNp}{N}}$$

where n_i is our current node, N is the number of simulations in that given state, Q is the number of wins from those simulations, c_i represents the exploration rate of an agent, and it ranges from 0 to 1. c_i N_P is the number of simulations that the parent of the current node has.

The first term of this equation is called exploitation which represents the pure average win rate while choosing that particular node, and the second term is called exploration, representing the exploration of nodes with fewer simulations. Hence, the exploration term is higher for the nodes with fewer simulations. The main difficulty while choosing

a node is to balance between those two terms, as it can not only increase the performance of the algorithm but also give us confidence about the computed winning probabilities.

The higher the UCT value of a node, the more likely we will select it. From the given equation we can see that the probability of selecting a node is higher when it yields a high chance of winning and is less explored compared to its parent.

Expansion phase

After we select a node, we add all possible actions as children of that node. This will result in creating new nodes for all possible actions.

Simulations phase

After expanding all the child nodes, we then select one of them randomly and simulate a game from it where each player plays randomly until we reach a terminal state and the score is calculated. We will have either a win (+1), a draw (0), or a loss (-1).

Backpropagation phase

After the simulation process, we need to update the tree with the simulation results. We start with a terminal state when the game could be over in a win, loss, or draw. Each time we visit a node during a simulation phase, we keep track of how many times that node was visited. We start from a terminal node, and we increment the "visit count" of that node by 1. Then, we change the score of a node. If the player whose turn was at the time of node selection wins, we add a positive score to that node. If the current player loses, we decrease the score, and if the game ends in a draw, the score of the node remains unchanged. After updating the terminal node, we move back to the parent of that node and then perform the same actions that we did in the case of the terminal node. We do these actions on ancestor nodes repeatedly until we reach a root node.

We have a time limit while performing our algorithm, and when it is reached, the algorithm returns us the move, resulting in a child of a root node that has the largest number of visits. This means that the returned move has been chosen the most while performing simulations, because it led to the winning states with more probability than the other nodes in the first layer.

2.2.2 Advantages and Disadvantages of Monte Carlo Method

Advantages

- **Understanding the probabilities:** By performing many simulations Monte Carlo method helps us understand probabilities of winning. Thus, by knowing the probability of winning for each move in connect-4, we can take the one which will make us closer to winning the game.
- **Risk assessment:** Monte Carlo method helps us understand the risks with respect to different situations(states). For example, in connect-4, when we hesitate which move to take, Monte Carlo simulations help us by providing a value for each node so we can evaluate risks associated with each move.
- **Handling Uncertainty:** Monte Carlo methods can model random variables. For example, in connect-4, there are two players and the future states depend on the opponents' moves. Monte Carlo method simulates different possible outcomes. Hence, Monte Carlo method can be useful even when we have some uncertainties in our problem.

Disadvantages

- **Computational cost:** Monte Carlo method's accuracy rises when we increase the number of simulations. However, that increases the computational cost. Taking into consideration that our branching factor is 7, the problem might require a lot of time and space.
- **Limited precision:** Although in connect-4 we have estimated values for our nodes in a tree beforehand, the values are not 100% precise. There exists some degree of statistical variability. The provided values for nodes will be more accurate in case of larger numbers of simulations. However there will always be some degree of uncertainty.
- **Exploration vs Exploitation tradeoff:** In connect-4, during the selection phase we use the UCT formula to balance between exploration of nodes with fewer simulations and average winning rates of nodes. However, early in the game,

when most of the nodes are less developed, the method might favor exploration over exploitation, which will lead to moves that are not necessarily optimal.

2.2.3 Minimax Algorithm

The Minimax algorithm is an adversarial search method that builds a game search tree for the agent in determining the optimal action by predicting future game states up to a specified depth d. It evaluates the best move in a game by calculating the minimax value of each state under the assumption that both players, MAX and MIN, play optimally. In the game tree, each level changes between the turns of the two players. Each node represents a possible game state, and each edge corresponds to a potential move [RN21].

When the algorithm reaches a node at the specified depth d or a terminal (leaf) node, it performs a static evaluation to estimate the value of that state. The terminal nodes show the utility values for either the MAX or Min player, depending on which player just took a turn.

For example, when d = 3, the MINIMAX algorithm searches all game states where player 1 makes a move, followed by player 2 making a move, and then again by player 1 making a move. Setting the depth to the maximum possible number of moves in Connect-4, d= 42, allows the algorithm to play a perfect game, but this increases computation time. For a better approach, heuristic functions are often put in use to efficiently evaluate intermediate game states, as discussed in later sections of the paper.

In a related study [TS24], the authors used a payoff function, which serves as a heuristic, to evaluate game states. This payoff function assigns positive numeric values to game states based on the MAX player's perspective and negative values for the MIN player. The evaluation assigns infinite points to winning moves and also gives scores to states with one, two, or three pieces aligned in a row. Higher weights are also assigned to discs in certain columns for their strategic importance. When multiple conditions apply simultaneously, for example forming two in a row in a central column, the values are summed to compute the final evaluation of the state.

2.2.4 Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique that enhances the performance of the Minimax search algorithm. The term *pruning* refers to the process of cutting off branches and leaves. The value of α is initially set to $-\infty$ and is updated the highest value whenever it is the MAX player's turn. Similarly, the value of β is initially set to ∞ and is updated to the lowest value whenever it is the MIN player's turn.

A branch or subtree is pruned when the condition $\alpha \geq \beta$ is met. This occurs because the MIN player would already have identified a move providing a better outcome for itself, making the MAX player's subtree irrelevant. The same logic applies when it is the MIN player's turn. The author compares the MiniMax algorithm and the MiniMax Alpha-Beta Pruning algorithm across three different difficulty levels, focusing on two metrics: the number of iterations performed and the computation time. [NDMK18]

Algorithm Used	**Mini Max**		**Alpha-Beta Pruning**	
Difficulty Level	**No. of Iterations performed**	**Computation Time (ms)**	**No. of Iterations performed**	**Computation Time (ms)**
Depth 1 (Easiest)	7	0.00	7	0.00
Depth 4 (Normal)	2799	33.00	477	6.00
Depth 8 (Hardest)	5847005	55441.00	71773	1009.00

Figure 2.3: Comparison of MiniMax and Alpha-Beta Pruning Algorithms at Different Depths

The study by Abdoul Wahab Touré tested various grid dimensions. [Tou23] An agent using the Minimax with Alpha-Beta pruning played against a greedy agent. Results demonstrated high winning rates even when grid sizes were increased. The standard 7x6

board achieved a perfect win rate, while larger grids had slightly decreased performance. The research showed that while search depth decreased with larger grids, the algorithm still remained consistently effective. A downside to this experiment was that the Minimax search algorithm was tested only against a greedy agent; however, it was able to show that the algorithm is scalable. [Tou23]

Grid size	Winrate % as first player	Winrate % as second player	Overall win rate %	Average search depth	Highest search depth reached
7×6	100%	100%	100%	2.69	21
9×8	100%	98%	99%	2.02	6
11×10	96%	93%	94.5%	1.81	3
13×12	94%	92%	93%	1.50	2

Figure 2.4: Winrate Comparison for MiniMax with Alpha-Beta Pruning Across Grid Sizes

2.3 Heuristic Approach

The Connect-4 game is constructed on a game tree with a depth of 42 and a branching factor of 7. Therefore, the whole game tree would have

$$7^{42} \approx 3.1197348 \times 10^{35}$$

nodes, so running the general minimax algorithm and calculating the utility value for each node would take too much time. Even though alpha-beta pruning allows for a significant decrease in this time, the algorithm is still not feasible and takes too much computational power. Therefore, a heuristic approach in combination with alpha-beta pruning is preferable in this case. A heuristic function assigns a score to non-terminal states, allowing the AI to estimate how favorable a position is for the player or opponent. This way, decisions can be made when the terminal state is not reached. An efficient way to implement heuristic evaluation is by limiting the depth of the game tree and evaluating the leaf nodes with the heuristic. So, combining alpha-beta pruning and

heuristic evaluation will make the algorithm more efficient and less complex [Sia24].

It is highly important to design reasonable heuristics that contribute to the efficient flow of the game. During our research, we found two main types of heuristics used in the Connect-4 game: a feature-based heuristic and a heuristic evaluating a square's location.

2.3.1 Feature-Based Heuristic

The feature-based heuristic is one of the most common heuristics for the game Connect-4 and appears regularly in literature. In his article, Xiyu Kang (2019) brings an example of the feature-based heuristic, where he selected four specific features the heuristic should find in the states and assigned corresponding values to each feature. The selected features are described below [KWH+19].

Feature 1: Four discs are connected vertically, horizontally, or diagonally. The feature immediately gets the value of infinity since it already is a win state.

Feature 2: Three discs are connected vertically, horizontally, or diagonally. If a move can be made on either of the immediately adjacent columns, the situation is evaluated as infinity since the players will surely win. If the move can be made only on one adjacent column or the same colored disk can be found a square away from two adjacent discs, the feature gets a score of 900,000 since there is a high chance of winning, but the opponent still might stop it. And if no move can be made on adjacent columns, the feature gets 50,000.

Feature 3: Three discs are connected vertically, horizontally, or diagonally. If a move can be made only on one adjacent column, the feature gets a score of 40,000, 30,000, 20,000, or 10,000 depending on the number of squares available along the direction of the discs correspondingly 5, 4, 3, 2.

Feature 4: A disc is not connected to any other disc in any direction. This feature is evaluated based on the column in which the disc is in. The highest score of 200 gets the central column. The two neighboring to the central columns get 120 while the ones neighboring them get 70. The two columns on the border get the value of 40.

So, the heuristic evaluates the state based on the likelihood of winning from that state. It favors the states where there are already several discs connected, and if that is not applicable, the heuristic evaluates the state based on the columns where the central columns are more favorable, since they provide more opportunities for winning.

A similar heuristic was used by Amos Cao in his research on Connect-4 [Cao]. The author again brings up four features to look for in each state.

1. Four adjacent squares contain three discs of the same color and one empty square. - 10 points
2. Three adjacent squares contain two same-color discs and one empty square. - 5 points
3. A disc is in the corner - 1 point
4. One of the two central squares is controlled - 3 points

This heuristic is also based on the likelihood of winning from the current state and might be quite useful when implemented. Still, the first heuristic function we discussed seems to be more accurate as it takes into account more states and is generally more detailed regarding the scores given to each feature based on favorability.

2.3.2 Board-Based Heuristic

Unlike the Feature-Based Heuristic, the Board-Based Heuristic by Xiyu Kang (2019) evaluates each square on the board rather than a specific feature. The main idea behind it is that the positions closer to the middle column and row are more favorable because they have more options to win; therefore, they should be assigned a higher value. The values of each square can be seen in the following matrix [KWH+19].

$$\begin{bmatrix} 3 & 4 & 5 & 7 & 5 & 4 & 3 \\ 4 & 6 & 8 & 10 & 8 & 6 & 4 \\ 5 & 8 & 11 & 13 & 11 & 8 & 5 \\ 5 & 8 & 11 & 13 & 11 & 8 & 5 \\ 4 & 6 & 8 & 10 & 8 & 6 & 4 \\ 3 & 4 & 5 & 7 & 5 & 4 & 3 \end{bmatrix}$$

Looking at the matrix, we can see that the highest values are assigned to (d, 3) or (d, 4). By dropping a disc on either of the squares, a player can have the largest expansion space, meaning the potential to form a whole horizontal, vertical, or diagonal 4-length line. So, the larger the expansion space, the bigger the value. This heuristic gives a clear understanding of how useful each square on the board is. [Tou23]

2.4 Neural Networks

A more advanced level of solving the Connect-4 game is implemented using neural networks. Neural networks can be trained to play Connect-4 using reinforcement learning techniques.

One approach that is used in this case is discussed by Rob Dawson in his article [Codnd]. The author is implementing a design similar to DeepMind's Alpha Zero project. A technique similar to the Feature Based Heuristic is used for designing the neural network. The neural network acts as a value function approximator which returns a score based on the situation on the board. For learning purposes, the network plays against itself several times and learns which positions are the most favorable. This way the agent learns on its own without outside help. At the end of each training game, the network receives feedback on the effectiveness of its suggested moves. The scores for board positions recommended to the losing player are slightly decreased, while those recommended to the winning player are slightly increased.

Chapter 3

Methodology

To test the suggested algorithms, different methods for each algorithm will be implemented. The goal is to find which of the discussed algorithms is the most efficient and gives the best results.

3.1 Heuristics

The Board-Based Heuristic and Feature-Based Heuristic are tested against each other. The heuristics play 1000 plays and the win rate of each heuristic is measured. For the justice of the test, it is carried out 2 times: first time Feature-Based starts the game, second time Board-Based starts the game. Besides the win rate, time and memory usage for each of the heuristics is measured. The goal is to find the heuristic that has the higher win rate but at the same time consumes low memory and time.

3.2 Minimax Algorithm and Alpha-Beta Pruning

Different variations of minimax algorithm combined with the board based heuristic are explored and tested to find the most efficient one. Firstly, for a baseline purpose a minimax algorithm is tested against itself with different depth limits. The tested limits are: (1,2), (2,3), (3,4), (4,5), (2,6). Each pair plays 10 games against each other, with each player starting in 5 of the games.

Afterwards, a minimax algorithm is developed with a random component and a component of blocking the opponent. The blocking component plays a role in situations where the current player has a winning move, allowing it to prioritize that move instead of relying on the pure heuristic. It also blocks the opponent whenever the opponent has a winning move. Randomness comes into play when there is more than one best move, with the algorithm randomly choosing one of those moves. Without the randomness component, the algorithm always selects the first best move it encounters. The algorithm is against tested with them same limit pairs defined previously.

Finally, the same minimax algorithm with blocking and randomness is combined with Alpha-Beta Pruning. And it is again tested with the same depths.

3.3 Monte-Carlo Approach

The Monte-Carlo Algorithm is tested several ways to assess its performance in different situations. To have more reliable results. The algorithm is tested with different number of simulations: 100, 500, 1000 and 3000.

Firstly, it is tested against a random agent with different number of simulations and the win rate in 10 games is measured. This way we can see the general performance of the algorithm since if it is not efficient even against a random agent, it will not be able to beat another intelligent agent.

After that, MCTS is played against Minimax Algorithm with Alpha-Beta Pruning combined with Board-Based Heuristic. This way we can compare two of the most developed algorithms against each other. And the one with the largest win rate in 10 games is found out.

Chapter 4

Evaluation and Conclusion

4.1 Results

So far, we have investigated the applications of game tree strategies such as Monte Carlo Tree Search (MCTS) and Alpha-Beta Pruning, as well as two types of heuristics for Connect-4. Our goal is to identify ways of efficiently solving the game based on our reviewed literature and comparison across strategies.

4.1.1 Heuristics

The two heuristics played against each other 2000 times, of which one half was started by Feature-Based Heuristic, second half was started by Board-Based Heuristic. In the first 1000 games, where Feature-Based was starting, it beat Board-Based 704 times. Meanwhile, when Board-Base was the first player, it beat Feature-Based 860 times. So, in general, we can see that the one starting the game has the advantage. Still, Board-Based heuristic had slightly higher win rate. Besides that, it consumed less memory and time. The results can be seen in Figures 4.1 and 4.2. So, in general, Board-Based heuristic performs slightly better than Feature-Based one. This is why we have later chose it to combine with the Minimax algorithm.

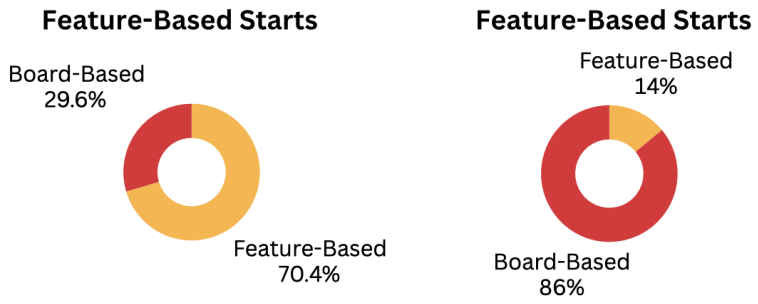

Figure 4.1: Win Rated of Feature-Based and Board-Based Heuristics

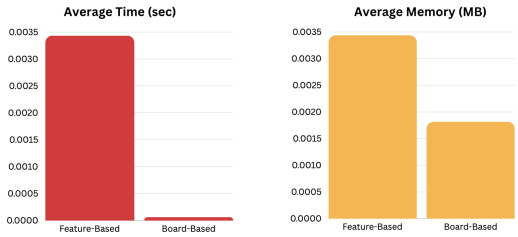

Figure 4.2: Average Time and Memory Usage of Feature-Based and Board-Based Heuristics

4.1.2 Minimax Algorithm Variations

At first, the MINIMAX algorithm without blocking and randomness factors was tested. With this deterministic approach, the algorithm consistently followed the same decision-making pattern. As a result, there were only two possible configurations for how the game went(depending on which player started). This resulted in the first player, who began winning, and the depth of the players did not play a role at all.

#Player 1	Player 2	Player 1 Wins	Player 2 Wins	Draws	Time (sec)	Memory(mb)
Depth 1	Depth 2	4	6	0	0.007101 0.042961	0.000259 0.000287
Depth 2	Depth 3	5	5	0	0.038847 0.255208	0.000336 0.000380
Depth 3	Depth 4	4	6	0	0.285646 1.827429	0.000432 0.000486
Depth 4	Depth 5	2	2	0	1.997734 13.620902	0.000543 0.002074
Depth 2	Depth 6					

Figure 4.3: Description of Results for the MINIMAX Algorithm with Randomness and Blocking Factor

The key metrics are Win Rates, the number of games won by each player; Average Move Time, the average computational time per move (in seconds) for each depth player in each pair; and Memory Usage, the memory consumed (in MB) by each depth player during the games. The computational time was too big to test out the depth of 2 and 6 pairs. Due to the same reason, pair (3,4) was tested four times.

Player 1	Player 2	Player 1 Wins	Player 2 Wins	Draws	Time(sec)	Memory(mb)
Depth 1	Depth 2	5	5	0	0.007882 0.023659	0.000277 0.000292
Depth 2	Depth 3	6	4	0	0.024941 0.113422	0.000316 0.000379
Depth 3	Depth 4	3	7	0	0.149828 0.524361	0.000460 0.000488
Depth 4	Depth 5	6	4	0	0.465016 2.301666	0.000542 0.000583
Depth 2	Depth 6	5	5	0	0.024883 5.340867	0.000324 0.000696

Figure 4.4: Description of Results for the MINIMAX Algorithm using alpha-beta pruning with Randomness and Blocking Factor

Introducing randomness and blocking eliminated the first-player advantage, as the second-player to start won in some games. The search depth for an agent did not significantly affect the winning rate. The algorithm ran smoothly up to a depth of 4.

The MINIMAX algorithm with alpha-beta pruning produced similar results regarding the winning rate, demonstrating that the agents' depth does not play a significant role. Including alpha-beta optimization significantly reduced the computational time of the MINIMAX algorithm. This allowed us to explore agents with depths of 2 and 6 against each other to examine whether the absolute depth difference influenced the winning rates. However, as shown in the results table, the absolute depth difference was not crucial, as agents with depths of 2 and 6 ended in a tie.

The graph in Figure 4.5 compares the average time per move for agents with different search depths using the MINIMAX algorithm and the MINIMAX algorithm with pruning. The x-axis represents the search depth of the agents, and the y-axis indicates the average time per move in seconds.

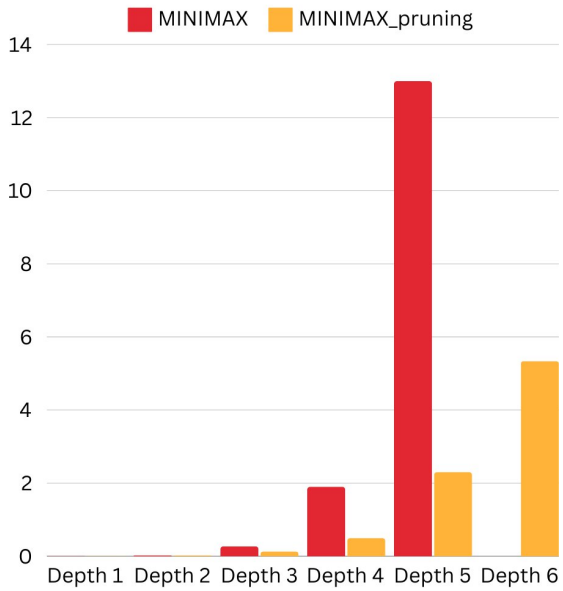

Figure 4.5: Comparison of Average Move Time for Minimax and Minimax Pruning Agents with Different Depths

4.1.3 Monte-Carlo Algorithm

Monte-Carlo Algorithm was tested against a random agent and against the minimax algorithm with alpha-beta pruning which evaluates based on the board-based heuristic. We have chosen to combine exactly that heuristic with the minimax, since it uses less memory and time than the feature-based one.

In the case of an agent playing based on a Monte Carlo Method versus the random agent, we observed that the win rate of an MCTS agent increases as we increase the number of simulations. The number of games played is set to 10. The results are shown in Figure 4.6.

In the case of an agent playing based on a Monte Carlo Method versus the agent playing with minimax with Alpha Beta pruning and board heuristic algorithm, we observed that the win rate of an MCTS agent was much less (30%). This, however, also depends on the number of simulations. In this case the number of games played is set to 10 and the number of simulations is 500. The tree depth for Minimax agent is set to 4. As we have seen before, when we increase the number of simulations, the win rate of an MCTS method also increases, so we would expect higher win rates for MCTS agent if we increase the number of simulations. The results are shown in Figure 4.7.

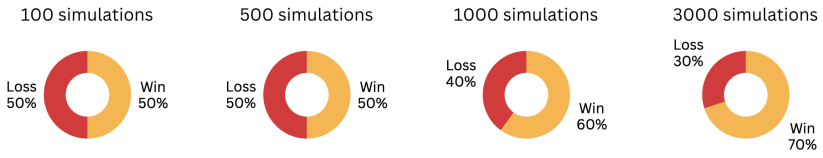

Figure 4.6: MCTS vs Random Agent

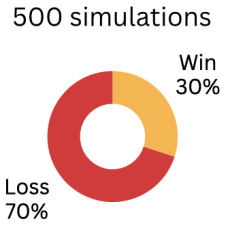

Figure 4.7: MCTS vs Minimax with Alpha-Beta Pruning and Board-Based Heuristic

4.2 Conclusion

In conclusion, we can see that in general, in the Connect-4 game, the first player has the bigger changes of winning. Turning to the discussed algorithms, it is evident that even though all the algorithms are performing well, some of them have more advantages. Particularly, the Monte-Carlo Algorithm is at disadvantage as it consumes too much time and memory while not giving high win rates. In case of the heuristics, both are performing quite good but the Board-Based one has a little more advantage. So, in combination with the Minimax Algorithm with Alpha-Beta Pruning, it becomes an even better version of a solution to the problem. Though, the minimax search should not be too deep, since it will become too much time consuming.

In the future work, it would be efficient to concentrate on deep learning and neural networks. These would provide more optimized solutions and be more human-like while the algorithms discussed here generally did not perform as humans would.

Bibliography

[All88] Louis Victor Allis. A knowledge-based approach of connect-four. *J. Int. Comput. Games Assoc.*, 11(4):165, 1988.

[Cao] Amos Cao. Connect-451: A real time connect 4 board processor.

[Codnd] CodeBox. Connect 4: The perfect game, n.d. Accessed: 2024-11-17.

[Fu18] Michael C Fu. Monte carlo tree search: A tutorial. In *2018 Winter Simulation Conference (WSC)*, pages 222–236. IEEE, 2018.

[KWH+19] Xiyu Kang, Yiqi Wang, Yanrui Hu, et al. Research on different heuristics for minimax algorithm insight from connect-4 game. *Journal of Intelligent Learning Systems and Applications*, 11(02):15, 2019.

[MIT10] MIT Special Topics. Connect four: Strategy and computation, 2010. Accessed: 2024-11-17.

[NDMK18] Rijul Nasa, Rishabh Didwania, Shubhranil Maji, and Vipul Kumar. Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. *Int. Res. J. Eng. Technol*, 5:1637–1641, 2018.

[RN21] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach, 4th us ed. *aima: . URL: https://aima. cs. berkeley. edu/(: 26.02. 2023)*, 2021.

[SDD+22] Kavita Sheoran, Geetika Dhand, Mayank Dabas, Nishthavan Dahiya, and Pratish Pushparaj. Solving connect 4 using optimized minimax and monte carlo tree search. *Mili Publications. Advances and Applications in Mathematical Sciences*, 21:3303–3313, 2022.

[Sia24] Siam Mandalay. Connect 4: Mathematical strategy, April 2024. Accessed: 2024-11-17.

[Tou23] Abdoul Wahab Touré. Evaluation of the use of minimax search in connect-4—how does the minimax search algorithm perform in connect-4 with increasing grid sizes? *Applied Mathematics*, 14(6):419–427, 2023.

[TS24] Henry Taylor and Leonardo Stella. An evolutionary framework for connect-4 as test-bed for comparison of advanced minimax, q-learning and mcts. *arXiv preprint arXiv:2405.16595*, 2024.