American University of Armenia

Artificial Intelligence Project

Solving Connect 4

Team: Anna Aghaloyan, Elvina Nosrati Alamdari, Sergo Poghosyan

Fall 2023

Abstract

The following paper analyzes different algorithm performances on the Connect-4 game. Specifically, the paper will have Random Walk, Minimax, Minimax with Alpha-Beta pruning algorithms, Heuristic Minimax with depth limit and Monte Carlo Tree Search performance on the game. After which, the algorithm's performances will be compared and evaluated. The project's findings are analyzed to determine possible improvements that may be achieved through future developments to solve the game. The project was carried out by a group of AI students at the American University in Armenia in Fall 2023.

Keywords: AI, Connect-4, Algorithms, Project, Minimax Search, Alpha-Beta pruning, Heuristic Minimax with depth limit, Monte Carlo Tree Search

Contents

Chapter 1

1 Introduction

1.1 Problem Setting and Description

Connect-4 is a two-player game. The goal of Connect-4 is to place four of your own discs on the vertically suspended grid. In this game, the players start with a specific color (usually red and yellow) and then take turns placing their discs (each having 21 discs). The first person to create a horizontal, vertical, or diagonal line on their own discs of four is declared the winner of the game. [N.D.M.K18] (see Figure 1)

Each player drops their discs from the top on previously dropped discs or an empty (if the game just started). One disc is allowed to drop at each turn. This continues until one player declares a winner.

1.2 History of the Game

Captain James Cook would spend several hours in his cabin using a game similar to Connect 4. Cook's favorite players were botanist Daniel Solander and naturalist Joseph Banks. This was mentioned in the “More Board Game Education” book. Due to the amount of time Cook would spend playing Connect-4, it was referred to as the Captain's Mistress.

In 1968, the game was developed by Funtastic under the name Score Four. It was a wooden game that featured a 4x4 grid and a rack with vertical poles that were protruding from the board. The goal of the game was to place four beads on the grid to create a row of them. Very similar to Connect-4.

The modern version of Connect-4 was developed by Ned Strongin and Howard Wexler in 1973. In 1974, Milton Bradley trademarked the rights to the game and released the original version of Connect-4. [G21]

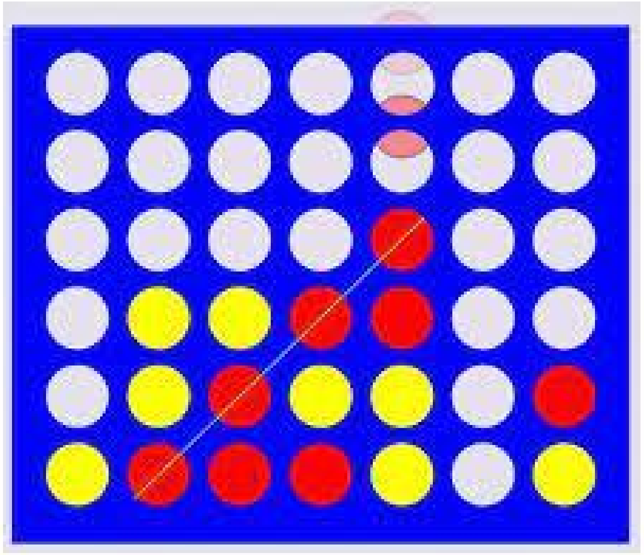


Figure 1: An example of a winning position. [R.J.12]

1.3 Known Solutions

The game Connect-4 is considered to be a zero-sum game since its advantage for one player is a disadvantage for the other player. Moreover, it is finite as the game has a finite number of states. The game is played on a seven vertical slots board, each having six positions. Its state space is 4.5 x 10 ^{12} board positions. [M.P.W.12]

Although players may not want to be the second player in Connect-4, it is still considered a perfect game to play. The red player cannot lose in any game of Connect-4 if, at the start of the game, the disc is placed in the center column. If the red player completes all the steps correctly, victory is guaranteed by 41 moves or less. If the red player drops the first disc in the two outer columns (A, B, F, or G), then the yellow can play to ensure that the other player loses. On the other hand, if the red player drops the first disc in the two adjacent columns (C or E), then the game will end in a draw.

Although it is presumed that the red player has an easy time winning the game, they still have to play perfectly to ensure that they do not lose. In practice, this can be challenging since Numberphile claims that there are more than four million ways to complete the grid. In 1988, Victor Allis was able to solve the issue of connecting four perfect plays with the help of a computer. [C2015]

1.4 The Structure of the Paper

The project paper is composed of an introduction, a literature review, and a method. In the first section, we talk about Connect-4's problem setting and description and the game's history. Then, we discuss the various algorithms that can be utilized to solve it. Although the methodology for the algorithm is still not finished, we have already discussed the reasons why each one can perform well and why not.

Chapter 2

2 Implemented Algorithms

2.1 Random Walk Algorithm

The first and probability the easiest algorithm that comes to mind to win the game is to use random moves. With the random walk algorithm, each player will place their colored disks randomly. Before placing a colored disk, it has to check whether the column is full or not; if it is not full, it can successfully place the disk; otherwise, it has to find empty space. Afterwards, we have to repeat this process until the game ends. It is important to note that this algorithm can be the worst one, as it just makes random moves even if there is a case when it can win the game by placing the right disks in the right place. Therefore, we expect this algorithm to perform not so well, as in some cases, it can even reach the number of maximum moves in the game.

2.2 Minimax Algorithm

The mini-max algorithm is a type of backtracking algorithm that can be used in various applications such as game theory and decision-making. It can help determine the optimal move for the user. The maximizer aims for the highest score possible, and the minimizer tries to get the lowest possible score.

Modern computers would be unable to perform a Minimax search with the estimated 4.5 trillion combinations in a standard Connect-4 set. Because of this, we need to limit our search depth. The theory is that having a greater search depth helps the algorithm find the best move. We need to find a search depth that is not too large or too small, but not too small that the algorithm doesn't have enough information to perform well. In our case, we will define the depth to five, meaning the algorithm will explore five possible moves ahead. [A.W.23] (see Figure 2.1)

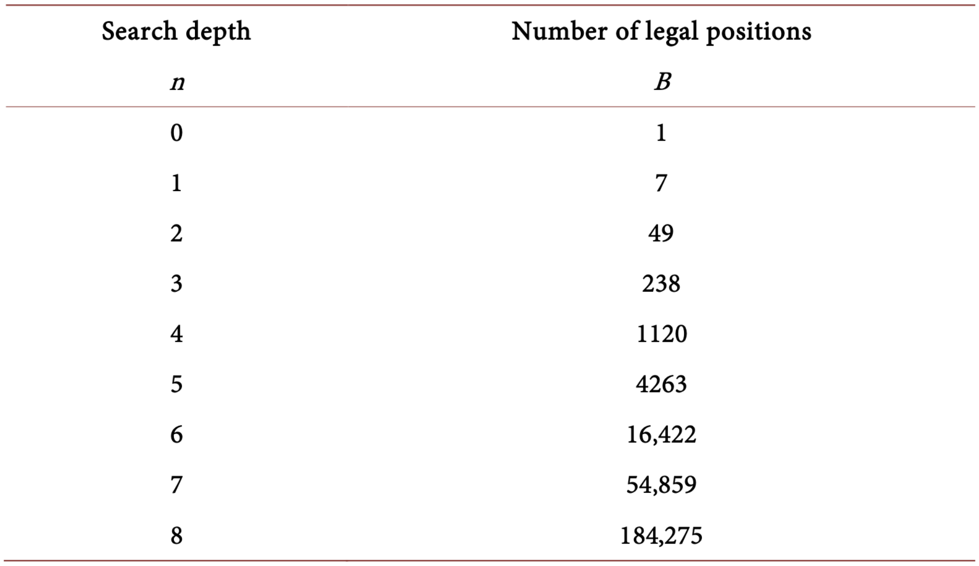
The game state of the tree is represented by its nodes and edges, which represent the different possible moves that one can make from a given state. There are two players in the tree, one of which is labeled as a maximizer and the other as a minimizer. The objective of the minimizer is to reduce the score of the agent, known as the maximizer. If we reach the leaf nodes or exhaust our chosen depth, a static evaluation is performed on our node.

For the game tree, each of the players has seven moves. After exhausting their depth, the evaluation is performed and the score is returned to the root node of the game. Although the agent is in charge of the game, he or she does not have total control over it.The score of the player on the left side can be -1 if they choose the left branch. On the other hand, if they choose the right branch, they get a score of 10, which is very probable. Because of this, the agent is more likely to choose a propitious move, and the opponents will try to reduce our agent’s score. [K.D.G.M.22]

2.3 Alpha Beta Pruning

The primary Minimax slows down as it sometimes explores some nodes that are unreasonable to explore at all. To solve this problem, we can apply alpha-beta pruning because it eliminates the unnecessary nodes that are going to be explored. The name “Alpha-Beta Pruning” comes from the use of two parameters: alpha and beta. Correspondingly, the agent gets a maximum score by storing the alpha parameter and the opponent desires to reduce the agent's score by storing the beta parameter. Initially, the alpha and beta are assigned with the worst possible scores for both the agent and the opponent. The algorithm keeps track of these parameters throughout the game. [K.D.G.M.22]

To successfully apply alpha-beta pruning, we have to deal with some computations of the node values. When α β ≥, a branch or subtree should be pruned because all higher values of α in the maximizing player's sub-tree will be meaningless because the minimizing player has already identified a move that would be more beneficial for him. It also works in the other way, all the lower values that β can have in the subtree of the minimizing player will be useless because the maximizing player has already found a move that would be more advantageous for him. It's crucial to remember that the Minimax search method employs the depth search to traverse the tree, meaning that the values of α and β are passed on throughout the backtracking [A.W.23]



Search depth *n*	Number of legal positions *B*
0	1
1	7
2	49
3	238
4	1120
5	4263
6	16,422
7	54,859
8	184,275

Figure 2.1: Number of legal Connect-4 positions after n depth levels. [A.W.23]

2.4 Heuristic Minimax with different depth limits

Considering minimax and minimax with alpha-beta pruning is a great idea because if we implement it efficiently, it will outsmart minimax by winning rate and minimax with alpha-beta pruning by average time performance. That is why it is crucial to try to implement different depth limits and find the best one. We will set starting depth limits to four and then increase it up to eight in order to find the optimal one. Now let's discuss how the implemented function in our will choose the heuristic value.

We have a function that evaluates the board and returns a score based on the position of the pieces. Its score can vary due to several factors. The first factor is about placing tiles in the center, as the agent that places tiles in the center has more advantages correspondingly, the tiles placed in the center return higher scores. Furthermore, it looks to create a winning position by creating possible horizontal, vertical, and diagonal alignments. Also, different scores are considered based on the number of pieces and empty spaces that can create a potential line. For example, if we have three tiles placed next to each other horizontally and other empty spaces, the algorithm will prefer placing a tile next to these three tiles and win the game rather than placing two empty tiles that will not guarantee the win.

2.5 Monte Carlo Tree Search

Monte Carlo Tree Search algorithm relies on randomness, which it is an important aspect of the Monte Carlo Tree search algorithm's approach. It allows it to explore the possibilities of a given game by performing simulations that are designed to mimic a real-life event. In the context of games like Connect-4 and other similar applications, simulation refers to the process used to create scenarios that are similar to real life. Simulations involve playing a game from a certain point in an artificial environment. Instead of trying to determine the best moves based on a comprehensive search, the game uses random simulations to evaluate possible moves. In our code, we will have 100 simulations, based on which the algorithm will choose the best simulation. The principle behind this algorithm is building a search tree, adding nodes based on the simulation playouts.

We are putting a constraint on the agent to choose the move. The purpose was to analyze our MCTS agent’s performance and relationship to other agents under the constraint. **[K.D.G.M.22]**

Chapter 3

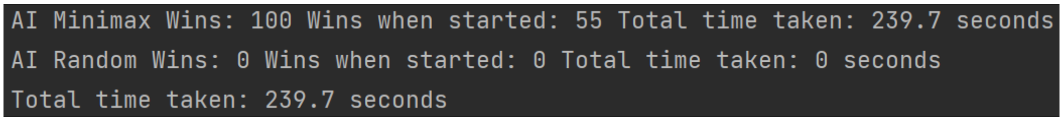
3 Algorithm Comparison

As this is a multi-agent game, it will be better to compare algorithms against each other. We created a code where two artificial intelligence agents use different algorithms while playing against each other. As mentioned before, our research focuses on algorithms such as random walk, minimax, minimax with alpha-beta pruning, heuristic minimax, and monte carlo tree search. We compared all possible combinations of algorithms and found out which one works the best, the most efficient, and the fastest. To get more accurate results, we performed 100 iterations for each algorithm. Algorithms start the game in a random order.

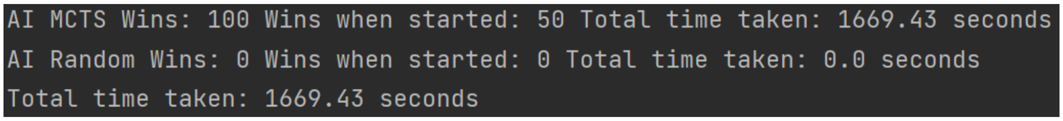
Comparisons discussed in one section of the algorithm will not be discussed in further sections in order to make the overall picture more understandable and avoid any confusion. In particular, there is no separate section for the h-minimax.

3.1 Random Walk Algorithm

Starting from the most straightforward algorithm, we compared the random walk algorithm with minimax and monte carlo tree search algorithms. Not surprisingly, after 100 iterations, the random walk did not win any game. Therefore, there was no reason to compare the random walk algorithm with h-minimax and minimax with alpha-beta pruning.



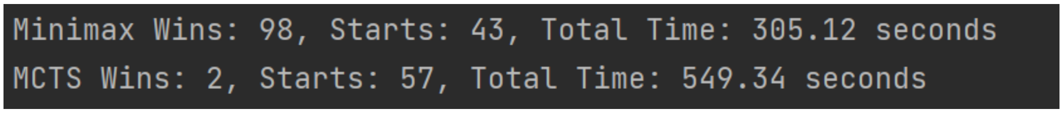
```
AI Minimax Wins: 100 Wins when started: 55 Total time taken: 239.7 seconds
AI Random Wins: 0 Wins when started: 0 Total time taken: 0 seconds
Total time taken: 239.7 seconds
```



```
AI MCTS Wins: 100 Wins when started: 50 Total time taken: 1669.43 seconds
AI Random Wins: 0 Wins when started: 0 Total time taken: 0.0 seconds
Total time taken: 1669.43 seconds
```

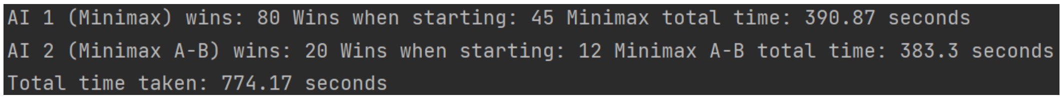
3.2 Minimax Algorithm

The Minimax algorithm is supposed to perform very well in this game, so let's find out about that. We compared it with monte carlo tree search and heuristic minimax with four and eight depth limits.



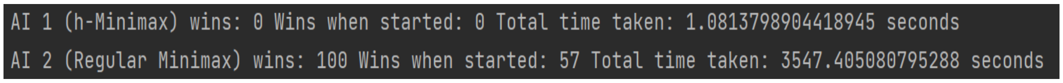
```
Minimax Wins: 98, Starts: 43, Total Time: 305.12 seconds
MCTS Wins: 2, Starts: 57, Total Time: 549.34 seconds
```

After 100 iterations, we see that minimax performs very well both in winning and timing aspects against MCTS. Therefore, MCTS is no competition for minimax.



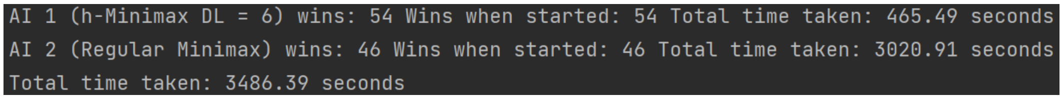
```
AI 1 (Minimax) wins: 80 Wins when starting: 45 Minimax total time: 390.87 seconds
AI 2 (Minimax A-B) wins: 20 Wins when starting: 12 Minimax A-B total time: 383.3 seconds
Total time taken: 774.17 seconds
```

Although they both have almost the same logic, we will still compare them. Here we can see that the regular minimax is four times more likely to win against its older brother. Nonetheless, it took them both almost the same time to perform.



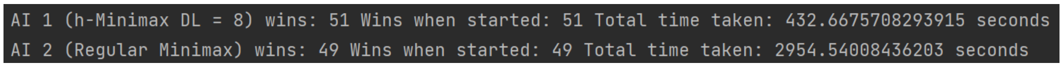
```
AI 1 (h-Minimax) wins: 0 Wins when started: 0 Total time taken: 1.0813798904418945 seconds
AI 2 (Regular Minimax) wins: 100 Wins when started: 57 Total time taken: 3547.405080795288 seconds
```

In this part, we compared the minimax against the heuristic minimax with a depth limit of four. As we can see, the h-minimax with a depth limit of four stands no chance against the regular minimax. However, it’s worth mentioning that while all actions of the regular minimax took an hour, h-minimax with a dl = 4 performed all actions in just one second. Therefore, let’s increase the depth limit to six and see what happens.



```
AI 1 (h-Minimax DL = 6) wins: 54 Wins when started: 54 Total time taken: 465.49 seconds
AI 2 (Regular Minimax) wins: 46 Wins when started: 46 Total time taken: 3020.91 seconds
Total time taken: 3486.39 seconds
```

Here, we can observe totally different results! The heuristic minimax with a depth limit of six performs almost as equally as the regular minimax and still needs several times less time to execute its actions. It's important to note that in this case matters only time performance because we can see that the algorithm that randomly starts the game always wins it



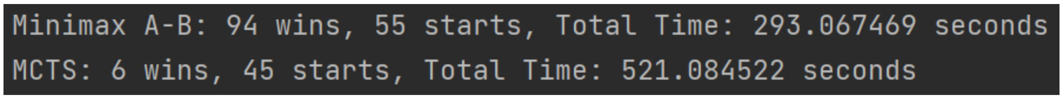
```
AI 1 (h-Minimax DL = 8) wins: 51 Wins when started: 51 Total time taken: 432.6675708293915 seconds
AI 2 (Regular Minimax) wins: 49 Wins when started: 49 Total time taken: 2954.54008436203 seconds
```

We can draw the same conclusion for this case, the algorithm that randomly starts wins the game. The time performance of the h-minimax with depth limit of six and eight is almost equal. Here we can see that it's even less, although it depends on some games that could have ended very soon.

It will be interesting to see how the h-minimax with different depth limits performs against the minimax with alpha-beta pruning, as the minimax has four to a winning rate against it. We will see that in the consequent experiments.

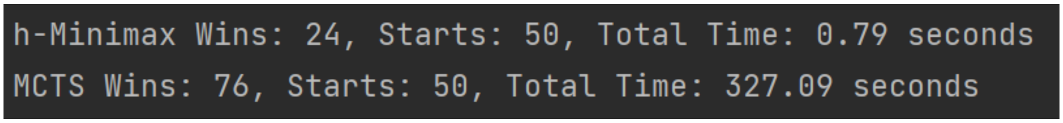
3.3 Monte Carlo Tree Search Algorithm

In this part, we will talk about comparisons between MCTS and h-heuristics with a depth limit of four, six, and eight, as well as the minimax with alpha-beta pruning.



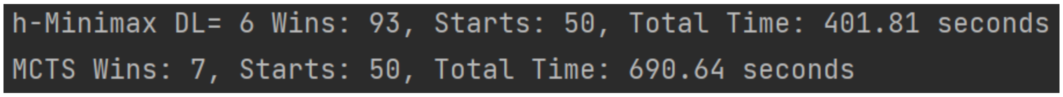
```
Minimax A-B: 94 wins, 55 starts, Total Time: 293.067469 seconds
MCTS: 6 wins, 45 starts, Total Time: 521.084522 seconds
```

As we can see, MCTS performs almost the same against the minimax with alpha-beta pruning as it performs against regular minimax, it has very few wins. This time the minimax with alpha-beta pruning performed a little bit faster.



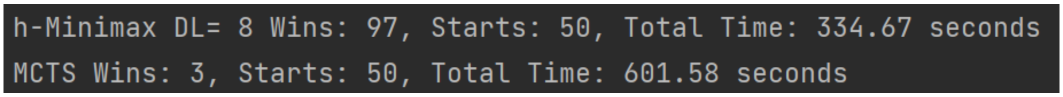
```
h-Minimax Wins: 24, Starts: 50, Total Time: 0.79 seconds
MCTS Wins: 76, Starts: 50, Total Time: 327.09 seconds
```

Here we can see the results that we have never seen, for the first time MCTS is winning against a solid algorithm: heuristic minimax with a depth limit of four. However, if we look at their time performance, we can see that they are incomparable. While h-minimax dl = 4 performed its actions in under a second, it took more than five minutes for MCTS. Therefore, let’s see what happens if we increase the depth limit to six.



```
h-Minimax DL= 6 Wins: 93, Starts: 50, Total Time: 401.81 seconds
MCTS Wins: 7, Starts: 50, Total Time: 690.64 seconds
```

By increasing the depth limit just by two, the h-minimax dominantly wins over MCTS. However, now it requires much more time, yet less than MCTS.

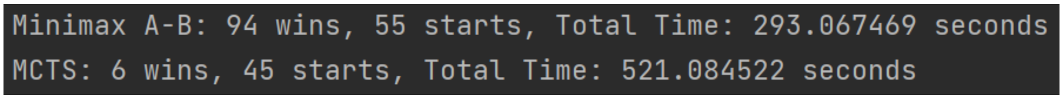


```
h-Minimax DL= 8 Wins: 97, Starts: 50, Total Time: 334.67 seconds
MCTS Wins: 3, Starts: 50, Total Time: 601.58 seconds
```

We see that h-minimax with a depth limit of eight leaves no chances for MCTS as well. It performs better than h-minimax with a depth limit of six, both in terms of winning ratio and time performance.

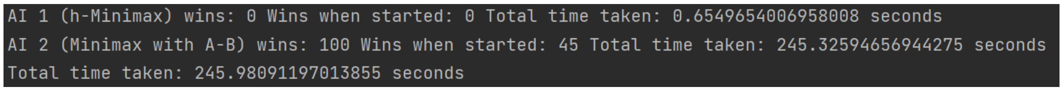
3.4 Minimax with Alpha-Beta Pruning Algorithm

In this section, we are left with the minimax alpha-beta pruning against h-minimax and MCTS.



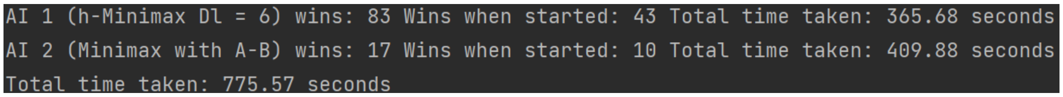
```
Minimax A-B: 94 wins, 55 starts, Total Time: 293.067469 seconds
MCTS: 6 wins, 45 starts, Total Time: 521.084522 seconds
```

After this many comparisons between different types of the minimax and MCTS, we can be sure that MCTS is not an opponent for the minimax. As in the previous cases, MCTS loses with a dominant score.



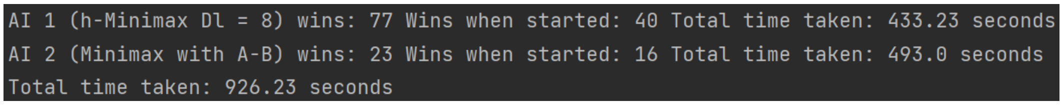
```
AI 1 (h-Minimax) wins: 0 Wins when started: 0 Total time taken: 0.6549654006958008 seconds
AI 2 (Minimax with A-B) wins: 100 Wins when started: 45 Total time taken: 245.32594656944275 seconds
Total time taken: 245.98091197013855 seconds
```

Comparing the h-minimax with a depth limit of four against the minimax with A-B pruning was not the best idea, so we increased the depth limit to six.



```
AI 1 (h-Minimax Dl = 6) wins: 83 Wins when started: 43 Total time taken: 365.68 seconds
AI 2 (Minimax with A-B) wins: 17 Wins when started: 10 Total time taken: 409.88 seconds
Total time taken: 775.57 seconds
```

Here, we can see a totally different picture. Now h-minimax with a depth limit of six has about four to one winning ratio over minimax with alpha-beta pruning.

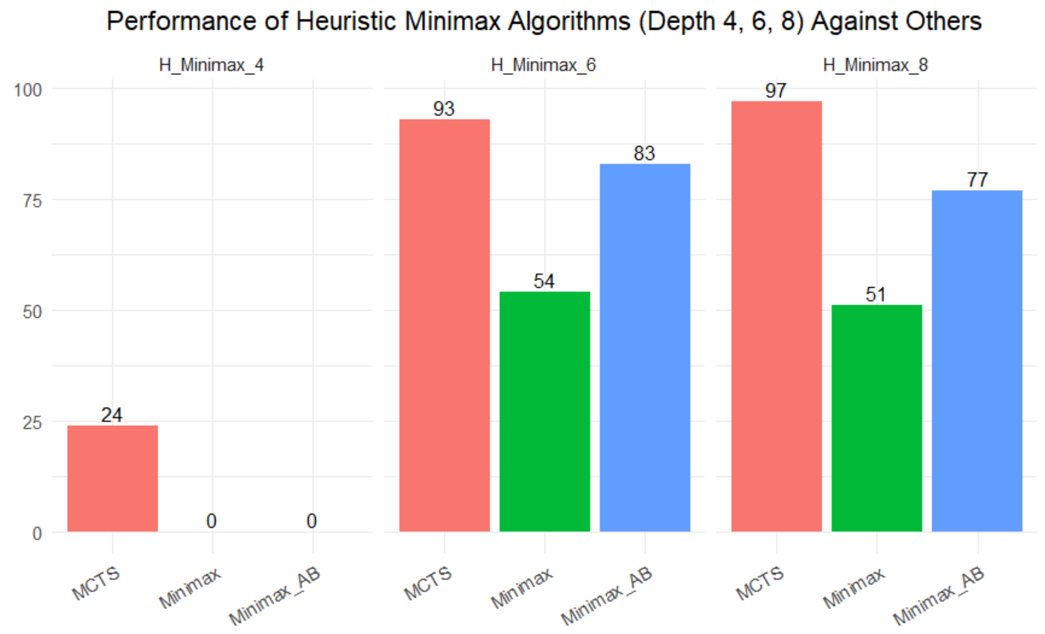


```
AI 1 (h-Minimax Dl = 8) wins: 77 Wins when started: 40 Total time taken: 433.23 seconds
AI 2 (Minimax with A-B) wins: 23 Wins when started: 16 Total time taken: 493.0 seconds
Total time taken: 926.23 seconds
```

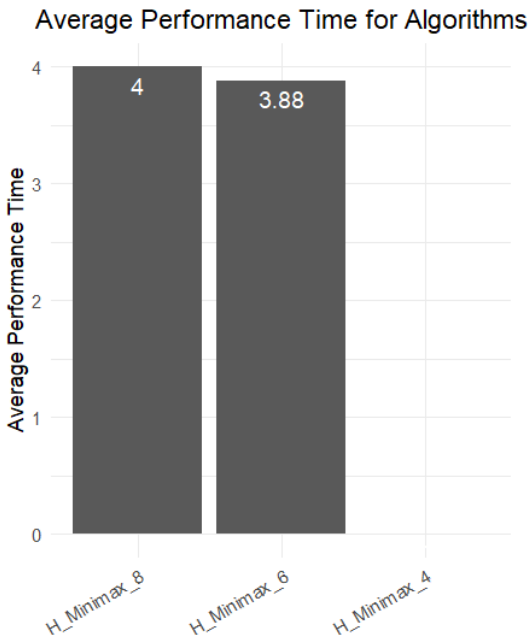
The depth limit of eight guaranteed the dominant win over the opponent, although it performed slightly worse than h-minimax with a depth limit of four. As discussed some iterations of algorithms can end very early, and some explore almost every possible move.

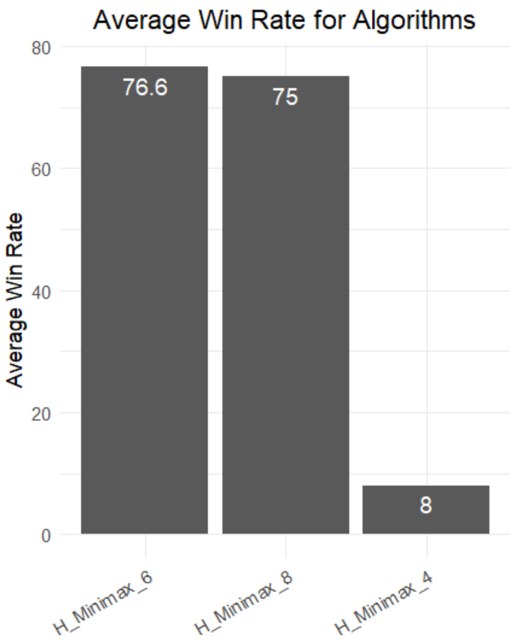
4 Conclusion

Before making our final decision about which algorithm is the best for this game, it is important to choose which depth limit we are going to choose for the heuristic minimax. Let's compare our results and decide.



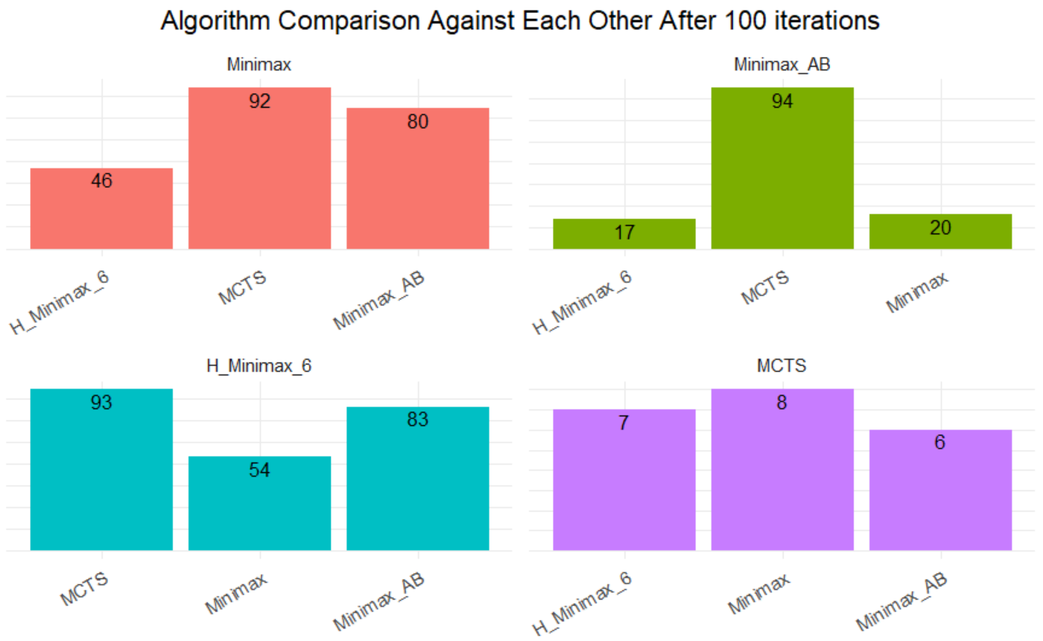
Here, we can clearly see that there is a competition between depth limits of six and eight, so let's focus on details to understand which one is better.



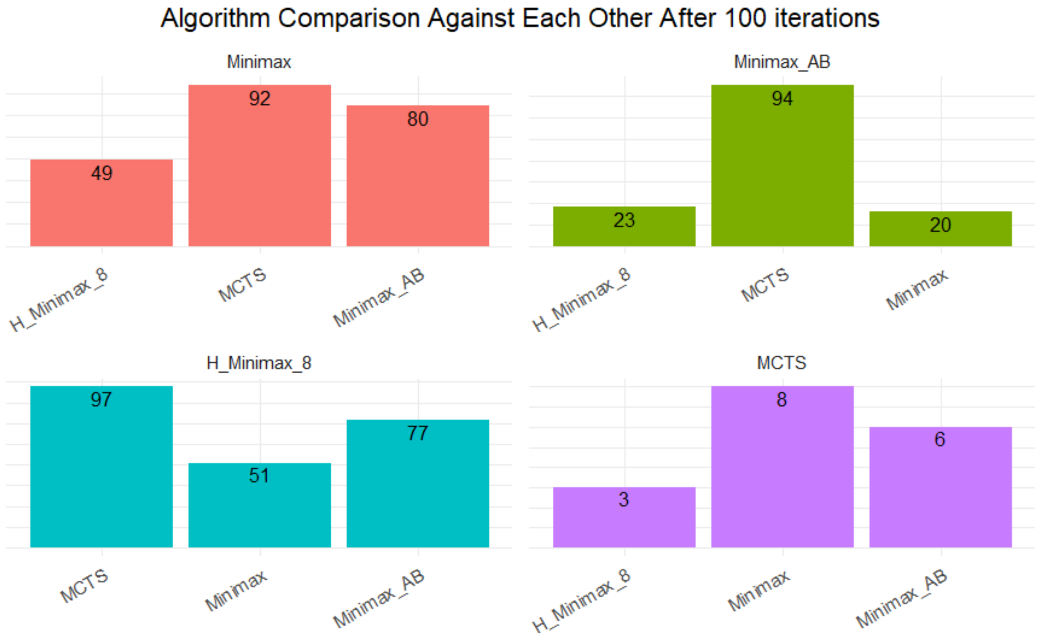


As we see heuristic minimax with a depth limit of four performs the best both in terms of context and the winning rate. Although, as the results are very close, just for the experiment we will consider both depth limits.

First, let's consider the results with h-minimax with a depth limit of six and other algorithms. Below we have a plot of the 100 iterations.

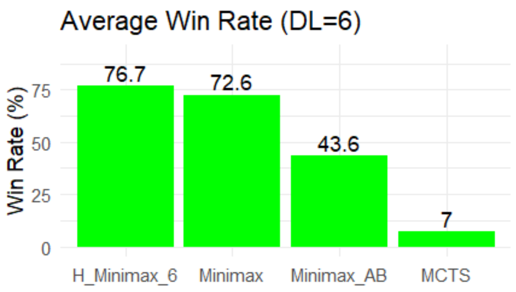


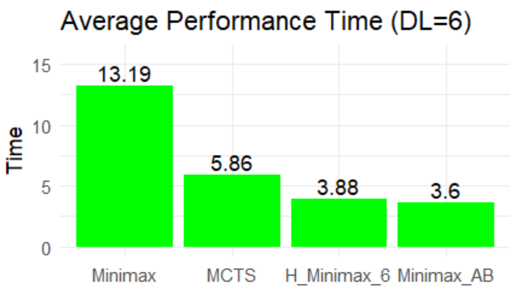
As we can see regular minimax and heuristic minimax with a depth limit of six have the best results. The weakest one is Monte Carlo Tree Search. Alpha-beta is doing fine. Let’s take a look at the results with h-minimax with dl = 8.

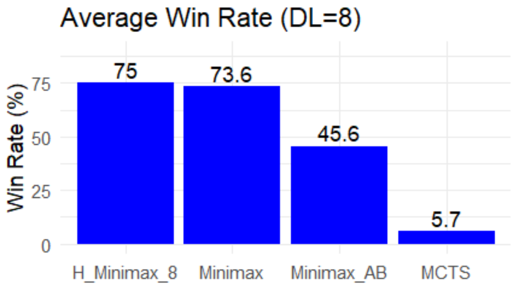


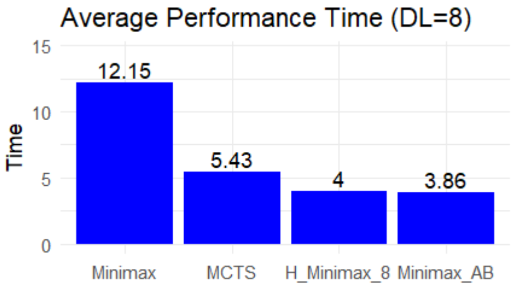
Almost the same situation is here. The first two places share minimax and h-minimax dl = 8. MCTS is the worst. Therefore, let's do some statistical analysis to determine our winner.

4.1 Final Results









After getting the final results of average win rate and performance time with all algorithms and heuristic minimax with depth limits of six and eight, we can indeed decide which algorithm is our winner. Clearly, heuristic minimax with depth limits of six and eight have the highest win rate: 76.7 and 75, respectively. Even though Minimax with alpha-beta pruning is slightly faster than heuristic minimax with dl (6,8), it has almost twice less win rate, therefore we will not consider it for the first place. It takes heuristic minimax with a depth limit of six on average 3.88 seconds per game to perform its all actions, against a depth limit of eight: 4. Therefore we choose it for our final results.

All in all, heuristic minimax with a depth limit of six has the best results with an average of 76.7 win rate and 3.88 seconds average performance time. In second place is regular minimax with an average of 72.6 win ratio and 13.19 seconds average performance time. In third place is Minimax with alpha-beta pruning with an average of 43.6 win ratio and 3.6 seconds average performance time. Next comes MCTS with an average 7 win ratio and 5.86 average performance time. In the last place is random walk, it was so bad that was not even worth putting in comparison section (0 WR, ~0s/g).

4.2 Rankings

1 Heurisitc minimax DL = 6 ~(76.7 WR, 3.88 s/g)

2 Regular Minimax ~(72.6 WR, 13.19 s/g)

3 Minimax with A-B pruning ~(43.6 WR, 3.6 s/g)

4 Monte-Carlo Tree Search ~(7 WR, 5.86 s/g)

5 Random Walk ~(0WR, 0 s/g)

5 Further Research

In this paper we performed analysis of five different algorithms (Random Walk, Minimax, Minimax with A-B pruning, Heuristic Minimax with different depth limits, Monte-Carlo Search Tree) on Connect four game. We compared algorithms with every possible combination and determined the winner (H-minimax DL = 6). In the future, it is possible to find algorithms that can perform better for this game. Alternatively, it is possible to select/create an algorithm and train it, so the more it plays the better it becomes in the future based on its experience.

References

[N.D.M.K18] Nasa, R., Didwania, R., Maji, S., & Kumar, V. (2018). Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. *Int. Res. J. Eng. Technol*, 1637-1641.

[M.P.W.12] Thill, M., Koch, P., & Konen, W. (2012). Reinforcement learning with n-tuples on the game Connect-4. In *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12* (pp. 184-194). Springer Berlin Heidelberg.

[G21] Team, G. (2021, July 19). *History of Monopoly (Game): Timeline (Magie, Darrow, Parker Brothers, and more)*. Gamesver. https://www.gamesver.com/history-of-monopoly-game-timeline-magie-darrow-parker-brothers-and-more/

[C2015] *Solving Connect Four with Game Theory : Networks Course blog for INFO 2040/CS 2850/Econ 2040/SOC 2090*. (2015, September 21). https://blogs.cornell.edu/info2040/2015/09/21/solving-connect-four-with-game-theory/

[A.W.23] Touré, A. W. (2023). Evaluation of the Use of Minimax Search in Connect-4&lt;/br&gt;—How Does the Minimax Search Algorithm

Perform in Connect-4 with Increasing Grid Sizes? *Applied Mathematics*, *14*(06), 419–427. https://doi.org/10.4236/am.2023.146025

[R.J.12] Hickey, R. J. (2012). Learning to Play Connect 4: A Study in. *AI and Cognitive Science'92: University of Limerick, 10–11 September 1992*, 157.

[K.D.G.M.22] Sheoran, K. A. V. I. T. A., DHAND, G., DABASZS, M., DAHIYA, N., & PUSHPARAJ, P. (2022). Solving Connect 4 Using Optimized Minimax and Monte Carlo Tree Search. Mili Publications. Advances and Applications in Mathematical Sciences, 21, 3303-3313