Connect-4 using Alpha-Beta pruning with minimax algorithm

Brenda Lim Geok San
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
TP055961@mail.apu.edu.my

Yap Jia Xin
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
TP055804@mail.apu.edu.my

Zailan Arabee bin Abdul Salam
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

Chanpreet Kaur Dhanoa
School of Computing
Asia Pacific University of Technology & Innovation (APU)
Kuala Lumpur, Malaysia
TP055952@mail.apu.edu.my

***Abstract* — The classic game of Connect-4 is fading from the face of society, especially with the younger generations due to the addictive instance of digital games, available anywhere, hence measures to preserve the continuity of this game lies in the recreation of the game in the virtual environment. In this paper, the formulation of a classic Connect-4 games utilizing the implementation of Alpha-Beta Pruning with Minimax algorithm is carried out with the objective of modifying its parameters to determine its influence on the execution of the game. The parameters involved in the modification process includes the depth of search and size of board as a small change of constant can lead to a drastic difference. Minimax algorithm serves the purpose of enabling the computer (AI) to place its piece strategically whereas Alpha-Beta Pruning is incorporated to reduce the size of its search tree. All results pertaining the changes made have been recorded accordingly and the optimal constant of parameters are identified, leading to an ideal execution of the game.**

Keywords — Artificial Intelligence (AI), Alpha-Beta Pruning, Minimax, Connect-4, Optimization

I. INTRODUCTION

With the constant evolution of technology, it comes as no surprise that what seemed like a brilliant source of entertainment back then is slowly fading into the background with a shift towards digital games. The emergence of computer and mobile games, offering a variety of gaming genres have taken over the need for physical board games such as Monopoly, UNO, Connect-4, Scrabbles, etc. Though some games managed to stay relevant in this day and age, namely the ones with multiple players option, others fade to grey. Classic physical games such as Connect-4 and Chess are slowly losing their place in the society, hence it is necessary to computerize the core elements of the games to recreate them in a virtual environment. As such, this would serve as an effort to preserve the continuity of these games in the future and hopefully resuscitate the fun and joy it once bought to the society.

Connect-4 is a classic two-player board game, each represented by yellow and red pieces respectively on a board matrix of seven rows and six columns. Players are free to place their piece alternately at any available positions, constricted to the six columns by dropping it into their desired compartment. Due to the nature of gravity, the pieces will always fill in the baseline of the board, eventually building up to all seven rows. The goal is for the players to successfully line up four consecutive pieces either vertically, horizontally or diagonally, hence the game can lead to either a victory upon accomplishing the task or a draw if none manages to put forth consecutive pieces. Connect-4 is a solved game whereby players with the primary move can guarantee a 100% chance of winning rate if played accordingly to the winning strategy despite whatever move placement the opponent lays out [1].

In this paper, we will be exploring how a change in the parameters of the algorithm, Alpha-Beta Pruning with Minimax can impact the time taken for a decision to be made on the placement of pieces, the number of lines analyzed and its winning rate. The purpose for the implementation of Minimax is because of its ability to proceed with a decision-making process whereas Alpha-Beta Pruning is incorporated to reduce the size of a search tree from unnecessary exploration [1]. Both algorithms play an essential role respectively to ensure a successful experiment result.

The following paper is segregated into several sections and sub-sections whereby section I provides an introduction to the research topic and a general overview of this paper, and the aspects of related literature reviews. On the other hand, section II focuses on the details of materials and methods used for the execution of the experiment carried out and II describes the implementation of the algorithms mentioned priorly. Section III presents the results obtained and its discussion, explaining our findings for each modified parameter and lastly, section IV summarizes a conclusion for the entire paper.

A lot of researchers had conducted experiments to analyze the performance of the Alpha Beta Pruning algorithm in Connect Four Prototype. Several algorithms such as MTD(f) and Scout algorithm were involved to make comparison. The researchers have also compared the behavior of the algorithm in parallel and sequential implementation.

Heuristic algorithm with influence mapping was implemented to study how to play Connect Four Game with artificial intelligence [2]. The Connect Four application was

run on Windows platform and the language used was C++. The improvements made to the game where the timer can be applied optionally depends on the player. Secondly, the new version of Connect Four in this paper allows the user to select either one or two players. A timer was applied to increase the difficulty of the game. The algorithm was able to fulfil the system requirements, but it did not return an optimal move as a greedy algorithm.

Sarsa and Q-Learning trained the AI agents on how to play Connect Four Game with optimal strategy [3]. Docker was utilized to run the application by putting agents of the same or different algorithm to play the game simultaneously, which contributed to the speedup of the investigation process. TensorFlow was utilized to develop the learning models for Connect Four Game. The researchers aimed to develop the influence of exploration rate and rewards models on the performance of both algorithms. Similar winning rates between the agents of the same algorithms and against the opposing algorithm were found, proving that the two algorithms do not have a crucial difference.

An investigation on several implementations of Alpha-Beta pruning algorithm was conducted to find out which algorithm was suitable for parallelism [4]. The authors stated that Alpha-Beta pruning was beneficial in enhancing the performance of Minimax algorithm within the sequential form. The authors applied beam search optimization to carry out the parallel alpha-beta pruning in both of the mesh architecture called Compute Unified Device Architecture (CUDA) and a shared memory model called Open Multi-Processing (OpenMP). The speedup for the algorithm using CUDA was 2 times faster than using OpenMP. In conclusion, the combination of beam search optimization in mesh architecture is the most optimal for Alpha-Beta pruning algorithm.

An experiment was carried out to compare the Alpha-Beta pruning and Memory-enhanced Test Driver (f) (MTD(f)) to find out which algorithm contributes to the highest optimality and the speed [5]. The experiments were executed by running the application on computers with 12 sets of conditions with changes in the search depth and which computer is the first player. As a result, MTD(f) evaluated the moves at a faster pace than Alpha-Beta pruning, the computational time was reduced due to the lesser amount of leaf nodes to be evaluated. The win percentage of MTF(f) was 45.83%, the time taken for its execution was 35.19% faster than Alpha-Beta pruning in the search depth 8. The evaluated leaf nodes for MTD(f) were 56.27% fewer than Alpha-Beta pruning. The increment in search depth did not cause the execution time of the MTD(f) to be slowed down. The limitation of the research was no human player's involvement in the experiment, only computer versus computer whereby the same depth was applied to both computers.

The efficiency of the mini-max algorithm and its after combining with Alpha-Beta pruning was investigated [6]. The number of nodes that were evaluated in a search tree decreased radically. Alpha Beta pruning falls under the adversarial search in which the agents are placed in a competitive environment. The concept of minimizer and maximiser was implemented in both algorithms, alpha and beta were extra parameters that helped the Mini-Max Algorithm to prune away the unnecessary nodes to be evaluated. There was a decrement in time taken for producing an optimal move with the implementation of Alpha-Beta pruning in the same depth. Alpha-Beta pruning was helpful for achieving the optimal objective of the game, which was producing the most optimal move in a short period of time.

The authors stated that the gaming application in the past was not as efficient as nowadays due to lack of computer memory space poor tree algorithm [7]. Parallelism had been introduced to speed up the evaluation process. Young Brothers Win Concept was one of the parallelism concepts that was applied to evaluate the sibling nodes parallelly. The researchers used tic tac toe as an example to study the difference in the computational speed and efficiency among Sequential and Parallel Alpha-Beta Pruning techniques. The conclusion made by the researchers was Alpha Beta Pruning which runs parallelly using OpenMP is cost and time efficient. It consumed lesser computational time to generate the next optimal move.

Hernandez et al. suggested that it is necessary to implement a neural network to assign the weight to the factors of the heuristic search [8]. The heuristic search can then be trained through unsupervised learning, it will be able to analyse the elements in the database to reinforce the weights. Therefore, the algorithm can execute movements based on the analysis.

II. MATERIALS AND METHODS

A. Hardware

When the experiment is conducted for observation, it was conducted on a HP laptop whereby the technical specification of the laptop will be listed in Table I.

TABLE I. SPECIFICATION OF HP LAPTOP

Specification	***Description***
Model Name	HP Laptop 14s-cf1xxx
Processor	Intel® Core™ i5-8265U CPU @ 1.60GHz
Topology	1 Processor, 4 Cores, 8 Thread
Storage	512GB Solid State Drive Capacity (SSD)
Memory	8.00 GB -1MHz
Operating System	Microsoft Windows 10 Home Single Language

B. Programming Language

The language that the been used to code the Connect-4 game is in Java. This is due to the fact that the Java language is object-oriented which would thus allow modularity where the code can be reusable and does not need to be repeated. Java is a robust language since it uses strong memory management and allows exceptions to be handled so that the program does not crash in the mid-game when any error occurs. Java is also a simple language as it is free from pointers and this would make the execution time shorter. It is a portable language and it can run on various operating systems and processors unlike other languages such as C and C++. Hence the Connect 4 game can be played on any system that has the proper IDE installed.

C. Software

In order to run the program, the IDE that has been used is NetBeans 8.2 which uses the JDK version of 1.8.0_301. This

can be obtained from the Oracle website and properly configured to the system such as setting the Java Home path. NetBeans 8.2 has been chosen as it supports various operating systems and even runs on macOS. The IDE allows any application to be developed in a set of modular software which is often known as modules. This is a user-friendly IDE as it is not complicated to understand how to write code on it and it has user interface management such as menu and toolbars which would navigate and help the user throughout the whole development process. Individuals who would like to run the Connect-4 game could just download the source code and click on "Import Project" to import the whole code into the IDE, click on the main Java Class and finally click on Run to run the game.

D. Algorithm

The algorithm that has been implemented into the Connect-4 game is classical Alpha-Beta pruning with Mini-Max algorithm. Mini-Max is known as a backtracking algorithm where it will predict the next move and get the optimal move [6]. In a multiplayer game such as Connect-4, there are two players where each will be known as the maximizer and the opponent will be known as the minimizer. The maximiser will try to get a higher score as they could and the minimizer will do the exact opposite and obtain the lowest mark possible. As the depth increasing there will be more branches to be explored to get the optimal move which results to longer time taken. Hence Alpha-Beta pruning has been implemented into the Mini-Max Algorithm to optimize the algorithm. Alpha Beta pruning is the powerful version of Mini-Max algorithm where two more extra parameters will be added to the code which is known as Alpha and Beta [7]. The implementation of Alpha Beta will help to significantly drop the searching time to get an optimal as not all branches will be explored. Pruning of subbranches of a particular node will be done when a better path has been explored [7]. Which mean that the path that has been pruned would not be explored as there is no need for it since it would not change the top node result.

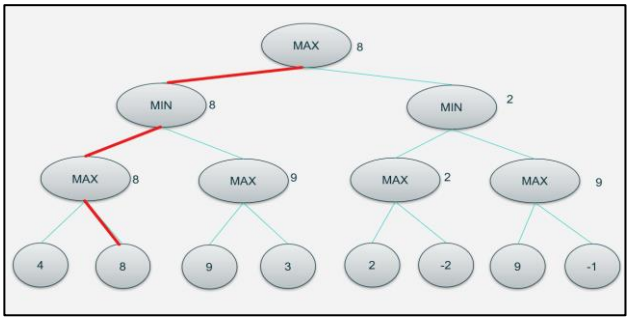

Fig. 1. Mini Max algorithm

Fig 1. Shows that when the Mini Max Algorithm has been used each level top node will be given a label either it is a Maximizer or a Minimizer. It goes sequentially where if the top node is a Maximizer the below node will be the opposite and act as a Minimizer and it goes on. The Maximiser will try to obtain the highest value and Minimizer will obtain the lowest value. It will perform a depth-first algorithm and reach the final node to get value. When it performs the depth-first search the first value that will be captured is 4. Then the next value would be 8 and these two values will be compared to see which has the higher value which in this case is 8. Thus 8 will be passed and the max node. This will go on until the final node value has been explored and compared. Based on the values then a path will be formed wherein Fig 2. is the highlighted red path.

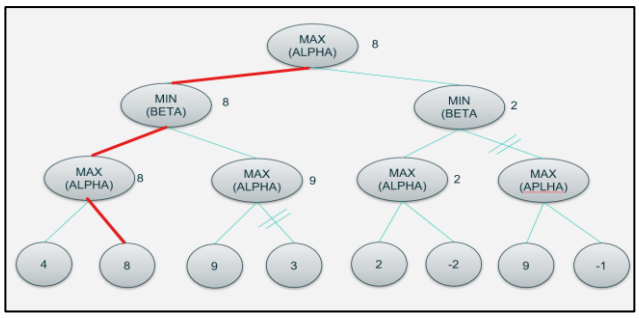

Fig. 2. Alpha-Beta Pruning with Mini Max

The Alpha Beta pruning does exactly the same as Mini-Max Algorithm where each node will be categorised either as a maximizer or minimizer. The only difference is in Alpha-Beta Pruning, not all nodes will be explored where since there is an extra parameter that has been passed which is Alpha and Beta. Alpha will store the highest value and beta will be storing the minimum value. As shown in Fig 2., there has been pruning occurred in a sub-branch. The pruning occurs as when 9 has been passed to the Alpha parameter it will be compared with the existing Beta parameter value which is 8 at the top node. Hence, when the Alpha value is higher than the Beta value there will be no need to explore the branch anymore as that path would not even be needed as it would not lead to the optimal move. Which result in no need for further checking of the nodes value and the branch will be prune. This will save a lot of time as right now the depth is not even deep enough so it is easier to visualise but in the actual game the search tree will be huge and long as the depth gets deeper making it to have more branches that need to be explored. When pruning occurs it would not need the system to check through the branch anymore. The highlighted red path is the optimal move which is the same as Fig 1. that uses the Mini Max algorithm where the same result is obtained but in a faster time which makes Alpha Beta Pruning much more powerful to be used in this game.

E. Parameter Changes

As explained above Alpha Beta pruning does significantly improve the search time rather than the usage of a simple Mini-Max algorithm. In order to test the efficiency, this experiment has been conducted where there will be three parameter changes to check if it affects the search time to get an optimal move, winning possibly and line analyzed. Every time when one parameter is changing the other two is kept constant in order to make it more systematic and easier for analysis purposes.

1. Depth

During this experiment, the depth will be changed from 0 up to 10 and the observation which is line analyzed, time taken to make a move and the AI winning probability will be taken into account.

2. Width

This parameter is the width of the Connect-4 game It is one of the important components of the connect 4 as different width sizes would make the placing of token either loose or

packed. Different width has been used to check on the line analyzed and time taken for the AI move.

3. Height

When it comes to height, the parameter was changed to various heights to check if it would affect the line analyzed and the time taken for the AI move.

F. Source Code to Change Parameter

```
public static final int HEIGHT = 8;
public static final int WIDTH = 9;
public static final int DEPTH_LIMIT = 8;
```

Fig. 3. Game parameter change

Fig 3. shows a snippet of the source code for Connect 4 game. If any parameter needs to be changed, it can be done by clicking on the "Game" java class and make the changes that are needed. For example, if the depth needs to be changed, the 8 should be erased and an integer value should be given.

```
public void drawEmptyBoard() {
    for (int i = 0; i<8; i++) {
        int tempI = i;
        for (int j = 0; j<9; j++) {
            int tempJ = j;
            Tile tile = new Tile(null);
            tile.setOnMouseClicked((event) -> {

                if (game.isMoveLegal(tempI, tempJ, game.getTextBoard()) && !game.isOver()) {
                    game.userMove(tempI, tempJ);
                    updateBoard(game.getTextBoard());

                    Thread newThread = new Thread(game);
                    newThread.start();
                }
            });
            grid.add(tile, j, i);
        }
    }
}
```

Fig. 4. Board parameter change

In case, in Fig 4. the height and width have been made changes, then another java class which is the "Board" class should have changes made also. The height and width that has been changed in the game parameter should be corresponding to the one in the Board class. For example, if the height has been changed to 10, then the i loop should have changes where now it becomes i < 10. The same goes with width whereby this time the j loop will have modification.

III. RESULT AND DISCUSSION

There are three parameters that will be changed which is depth, width and height. All this is divided into its own category and the result will be discussed accordingly as different parameters bring different results.

A. Depth Change

Below is the result when different depth has been used. Throughout the whole experiment, there are two parameters that are kept constant which is the height and width. The height that is kept throughout the whole experiment is 8 meanwhile the width is 9. The width is calculated based on equation (1).

$$Width = Height + 1 \quad (1)$$

TABLE II. DEPTH RESULT OF 0 – 5

DEPTH	*MOVE*	*TIME TAKEN(S)*	*LINE ANALYSED*
0	1	0.12	9
	2	0.15	9
	3	0.12	9
	4	0.13	9
	5	0.15	9
1	1	0.15	90
	2	0.14	90
	3	0.15	90
	4	0.14	90
	5	0.15	90
2	1	0.17	331
	2	0.11	430
	3	0.13	459
	4	0.14	489
	5	0.13	468
3	1	0.14	2209
	2	0.13	3322
	3	0.12	1904
	4	0.13	2417
	5	0.14	2594
4	1	0.15	10339
	2	0.14	16461
	3	0.16	13803
	4	0.15	14546
	5	0.14	15868
5	1	0.14	63276
	2	0.18	48957
	3	0.19	47771
	4	0.19	32432
	5	0.18	65182

Table II shows the time taken and the line analysed for the first five moves of the depth from 0 to 1. It can be observed that the time taken is less than 1s which is extremely quick and almost instant. Although the line analysed at depth 5 is significantly higher compared to depth 1 but the time taken is almost the same. The line analysed when the depth is set at 0 is equivalent to the width size. Whereby if the width is set as 8 the line analysed will be 8 also. In this case, the width has been set to 9 thus 9 lines has been analysed. When the depth is increased to 1 depth level, the line analysed is time ten of the set width referring to equation (2). Although depth that is lower than 3 takes lesser time to make a move it is not making the most optimal move. This is because the winning percentage of humans is higher which is 95% which is not the aim of the experiment. Depth 4 and 5 makes much more optimal move and also gives the move in a short period of time. Not only that but it has a higher AI winning percentage which is 98%.

$$Depth\ 1\ Line\ Anlaysed\ = Width\ Length * 10 \quad (2)$$

TABLE III. DEPTH RESULT OF 6 – 10

DEPTH	*MOVE*	*TIME TAKEN(S)*	*LINE ANALYSED*
6	1	1.25	307994
	2	2.14	491721
	3	3.95	315744
	4	4.49	443107
	5	3.86	368837
7	1	63.09	2927702
	2	125.56	6783007
	3	85.87	2810740
	4	88.15	1379037
	5	86.32	14536958
8	1	63.41	16602524
	2	126.43	7374165
	3	105.43	4389983
	4	90.13	13112089
	5	110.43	27432773
9	1	83.42	45102424
	2	184.06	32354071
	3	129.12	43289321
	4	188.12	22791022
	5	110,43	43531913
10	1	290.54	55764660
	2	309.40	57754960
	3	244.13	71661621
	4	491.21	69162372
	5	197.18	71271139

As the depth starting from 6 it can be observed that the system starts taking seconds to get the next move. This is because there are more lines being analysed and it is taking a longer time to return an optimal move. More nodes have to be explored resulting in a longer search time. As can be seen when the depth is 10 the machine took up to 491.21s to make a move which is almost up to 8 minutes. Although the moves made are optimal but in such an online game no one would want to wait that long for a machine to make a move and would rather just leave the game.

TABLE IV. SUMMARY OF DEPTH 0 – 10

Depth	*Time Average (s)*	*Average Line Analysed*
0	0.134	9
1	0.146	90
2	0.136	435
3	0.132	2489
4	0.148	14203
5	0.176	51524
6	3.138	385141
7	89.798	5587506
8	99.166	13782307
9	139.03	37413750
10	306.492	65122950

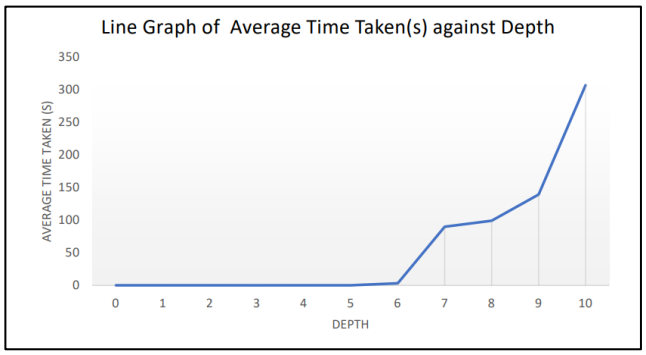

Fig. 5. Graph of average time against depth

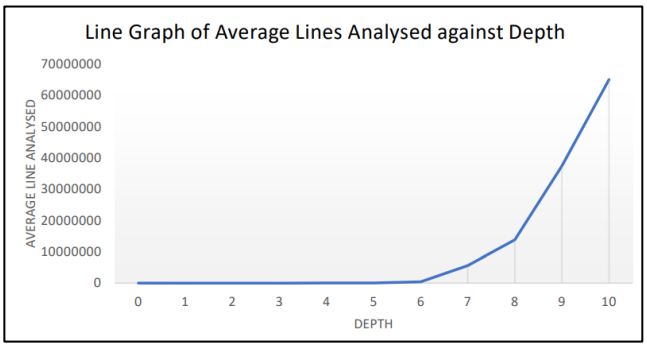

Fig. 6. Graph average line analysed against depth

Different depths have been used to check on the time analysed and also line analysed. This is the summary and line chart that has been produced based on the depth with the time taken average and average line analysed. It will give a better visualisation of how the depth impact the average line analysed and average time taken to make an optimal move. As clearly shown that the when the depth increases the average time taken and average lines analysed increase. Alpha Beta Pruning with Mini Max algorithm is suitable to be implemented for games that need a depth search ranging from 4 to 6. If a game requires a much deeper search, then another algorithm shall be applied rather than Alpha-Beta Pruning with Mini Max algorithm.

B. Width Change

Below is the result when different width has been used. Throughout the whole experiment, there are two parameters that are kept constant which is the depth and height. The depth that is kept throughout the whole experiment is 4 meanwhile the height is 8. Higher width value resulted in more columns in the board, with the increment of the action space to be evaluated, the computational time for the algorithm to construct a move increased. Below is the equation for calculating action space.

Action Space = Column Width - Number of Full Columns at Current State (3)

TABLE V. WIDTH RESULT OF 8 – 16

WIDTH	*MOVE*	*TIME TAKEN(S)*	*LINE ANALYSED*
8	1	0.35	7878
	2	0.49	8211
	3	0.32	5689
	4	0.28	8869
	5	0.16	9435
10	1	0.48	17670
	2	0.63	27286
	3	0.55	28480
	4	2.75	21847
	5	2.69	20532
12	1	0.53	35178
	2	0 48	36862
	3	0.86	22077
	4	1.32	2096
	5	5.82	47063
14	1	0.64	44718
	2	0.98	58130
	3	1.23	55576
	4	1.36	51923
	5	1.76	45156
16	1	0.45	44883
	2	0.21	55311
	3	3.73	82733
	4	2.5	48456
	5	1.8	63695

TABLE VI. SUMMARY OF WIDTH 8 – 16

Width	*Time Time Average (s)*	*Average Line Analysed*
8	0.32	8016
10	1.42	23163
12	1.802	32430
14	1.194	51101
16	1.738	59016

With more width added to the board, the computer had to analyse more possible actions, therefore the average of time taken for it to make a new move is increased. The smallest width value of 8 was used as the application crashed whenever the width was smaller than 8. A depth value of 4 was selected as it is the most optimal.

C. Height Change

Table VII shows a summary result of when different board heights have been used. Throughout the whole experiment there are two parameters kept constant; the width and search depth. The width was kept at 8 whereas the search depth was a constant of 4.

TABLE VII. SUMMARY OF HEIGHT 1-10

Height Range	*Time Taken*	*Line Analysed*	*Crashed*
1-2	-	-	Yes
3-6	-	-	Yes
7	Very efficient	Half	Half
8-10	Very efficient	Yes	No

Table VII showcases how the height of the board impacts the running of the Connect-4 game. Within the height range of 1 to 2, the program does not run at all whereby when players click on any empty spaces, nothing occurs. Even though the program crashes with a height range of 3 to 6, players are able to click on the board and receive a response from the program, placing its pieces randomly up to a point of no response given after several clicks. As for the height of 7, the game runs efficiently and effectively during the first half of the game before crashing. Upon filling up more spaces on the board, the program would not respond to any player interaction, thus manual cancellation of the game has to be initiated. Lastly, for the height range of 8 to 10, the program runs efficiently and effectively throughout the entire gaming experience.

TABLE VIII. HEIGHT RESULT OF 8 – 10

HEIGHT	*MOVE*	*TIME TAKEN(S)*	*LINE ANALYSED*
8	1	0.37	9190
	2	0.53	16148
	3	0.43	11017
	4	0.25	6173
	5	0.41	10339
9	1	0.37	9190
	2	0.42	10773
	3	0.44	12241
	4	0.37	9812
	5	0.54	17462
10	1	0.37	9190
	2	0.36	8213
	3	0.43	11581
	4	0.32	7748
	5	0.45	11984

Table VIII shows the time taken and lines analyzed within the first 5 moves for the board height ranging from 8 to 10. As shown, the observed parameters for each height constant remain almost similar whereby there is no distinct or drastic difference measured. This is because, during the first few moves of the game, all pieces will be filling in the board baseline. With the implementation of Alpha-Beta Pruning which reduces the size of a search tree from unnecessary exploration, the number of lines analyzed and the time taken remains almost similar considering with an increase of height, it would lead to more unnecessary expansion of search branch unless the next move requires the extra height row. The height of the board is only taken into consideration for the number of lines analyzed when the pieces start filling in the upper quadrant of the board. In conclusion, if the algorithm used does not utilize Alpha-Beta Pruning, the height of board could possibly make a huge impact on the obtained result.

TABLE IX. SUMMARY OF HEIGHT 8 – 10

Height	*Time Taken Average (s)*	*Average Line Analysed*
8	0.398	10573
9	0.428	11896
10	0.386	9743

Table IX summarizes the average time taken and lines analysed throughout the first five play movements. It can be derived that the height of the board does not influence the outcome of the game, considering hypothetically, as the height increases, the average time taken, and lines analysed should increase as well. This is possible because in the game of Connect-4, each piece placed will fill the baseline of the board due to the nature of gravity, hence the height of the board does not necessarily impact the result of the game rather it allows for the prolonging of each gaming session. With a shorter height, it would result in a shorter game if both players were to fill the entire board, nonetheless, the winning percentage for each player remains the same.

IV. CONCLUSION

In conclusion, this paper contributes to seeking the optimal parameters for an efficient running of the Connect-4 game using Alpha-Beta Pruning with Minimax to achieve the highest winning rate. Modifications were made to the depth of search and size of the board to study how these changes can lead to an impact on the algorithm's overall performance in different criteria. Each parameter was changed respectively while keeping constant of other parameters to keep track of the progression of result, however, there could be a possibility that if all parameters were changed concurrently, better optimization can be achieved.

Based on our findings, the algorithm, Alpha-Beta Pruning with Minimax is best implemented with a constant search depth of either 4 or 5 as it results in an optimal time taken for decision-making and is proven to have a great winning percentage. Despite higher search depth leads to a better winning course, in terms of the evaluation of other criteria such as the execution time, it did not achieve efficiency. For every online game, the fundamental concern is to ensure it is both functional and quick to respond to any user interaction, hence other aspects are of secondary priority. The same logical sense is to be applied in this instance because Connect-4 is a turned-based game, thus achieving optimal response time should be the main concern followed by its winning percentage.

REFERENCES

[1] V. K. BC, N. Jashank, and M. S. Nadiger, "Alpha-Beta Pruning–A streamline approach for perceptive game playing," International Research Journal of Modernization in Engineering Technology and Science, 2(6), pp. 1306–1318, June 2020.

[2] A. M. Sarhan, A. Shaout, and M. Shock, "Real-time Connect 4 game using artificial intelligence," Journal of Computer Science, 5(4), pp. 283-289, 2009.

[3] E. Alderton, E. Wopat, and J. Koffman, "Reinforcement learning for Connect Four," January 2019.

[4] S. P. Singhal, and M. Sridevi, "Comparative study of performance of parallel Alpha Beta Pruning for different architectures," In 2019 IEEE 9th International Conference on Advanced Computing (IACC), pp. 115–119, December 2019.

[5] 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, vol. 190, 2017.

[6] R. Nasa, R. Didwania, S. Maji, and V. Kumar, "Alpha-Beta Pruning in Mini-Max algorithm – An optimized approach for a Connect-4 game," International Research Journal of Engineering and Technology, 5(4), pp. 1637–1641, April 2018.

[7] S. Mandadi, B. Tejashwini, and S. Vijayakumar, "Implementation of sequential and parallel Alpha-Beta Pruning algorithm," International Journal of Innovations in Engineering Research and Technology, 7(8), pp. 98–104, August 2020.

[8] J. Hernandez, K. Daza, and H. Florez, "Alpha-Beta vs Scout algorithms for the Othello game," In CEUR Workshops Proceedings, vol. 2846, pp. 65–79, 2019.