MiniMax Algorithm for Game Artificial Intelligence

LaTeX template adapted from:
European Conference on Artificial Intelligence

Bahadir Erkam Bakoglu[1]

Other group members:
Sergiu Macsim[2]**, Siddhant Sashital**[3]

Abstract. The games that use artificial intelligence have created an alternative design category named as player versus computer (PVC). The games that are designed as PVC consist of at least one human player and one artificial intelligence algorithm. For instance, the game of chess is a solid example to PVC game design. One of the most efficient designs for the artificial intelligence algorithm created for playing chess is by considering all the possible steps to decide the best outcome by minimizing the gain of the opponent and maximizing the chance of success in a certain situation during the game. This algorithm is called MiniMax algorithm. This paper will be focusing on the MiniMax algorithm's explanation, chess game applications and structural design followed by the experiment results and discussion.

1 Introduction

During the last two decades, a large amount of progress has been made in the development and improvement of technological components in the artificial intelligence and automated decision making in various settings. These developments have affected the designs of many technological sectors, such as game development and risk management. The new design ideas have created a new and practical category which combined artificial intelligence with the existing projects to increase the efficiency and accuracy of the outputs. The artificial intelligence algorithms added to the programs are aimed to make better decisions in a certain given scenario considering the limitations and advantages. One of the most important algorithms is the MiniMax algorithm which is originally a backtracking algorithm used in game theory for optimal decision making [1]. The MiniMax algorithm uses a search tree embedded with min and max functions on its every layer to get the best potential outcome in each payoff setting. The min and max functions resemble the two players in a game. The aim of the max is to get the higher score while min focuses on getting the lowest score [7]. The algorithm is widely used in player versus computer (PVC) games such as chess and tic-tack-toe because of its accurate decision making and convenient application. However, the games like chess are more complicated and many moves need to be taken in account at every turn of the game. This causes run-time problems and makes it impossible to consider every move [5].

[1] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: bb0948y@gre.ac.uk
[2] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: sm8265g@gre.ac.uk
[3] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: ss0739q@gre.ac.uk

This article will be focusing on the applications of the MiniMax algorithm in the game of chess and demonstrate the structural design. Then, explaining the conducted experiments and their results on the algorithm. Finally, discussing the overall efficiency followed by general wrap up.

2 Background

The MiniMax, also known as MM or saddle point, is a decision-making strategy used in artificial intelligence, game theory and statistics for minimizing the loss in a worst-case scenario. The idea of the MiniMax is to offer the best move to get the largest payoff by surveying all the possible advantageous responses of the opponent. The MiniMax is an optimal solution in combinatorial games, sequential games where players take turns and has perfect information. The game of chess can be given as an example to combinatorial games. All the positions on the chess board have a value to indicate the payoff in every situation. In the game the first player aims to maximize the evaluation of the position. As for the second player, the focus is to minimize the payoff of the first player's action while getting to a more advantageous position in the game.

3 Experiments and results

Chess is one of the traditional strategy games which requires practical and instantaneous thinking. The main purpose of the players in the game is not only to make the best moves but also cornering their opponents and decrease their chase of success. This ideology of the game is perfectly suited for the MiniMax algorithm [4]. The MiniMax algorithm's purpose is to minimize the gain of the other players without knowing their strategies or maximize the gain of the player when the strategies of the other players is known. Based on these functioning protocol definitions, it is possible to form a mathematical expression to state the algorithm more evidently. The algorithm requires a player *i* and a chosen strategy *s* along with the other players selected strategies *s-i*. The value of the move of the player *i* as the output, *u*, of the algorithm. Based on these definitions we can express the MiniMax algorithm as the following:

$$\bar{u}_i = min_{s-i} max_{si} u_i(s_i, s_{-i}) \tag{1}$$

This is the recognized definition of the MiniMax algorithm in game theory and statistics. However, for this algorithm to form the backbone of the artificial intelligence in a complicated game like chess where there are thousands of possible moves to consider at

any turn of the game, the algorithm needs to be limited to a certain search strategy so the program does not cause run-time errors and successively decide the most beneficial action in a given point [2].

Since the MiniMax algorithm decides the most advantageous move according to the other players strategies, the chess board needs to be indicative of the better moves to the artificial intelligence to understand what is better for itself and what is worse for the other players. To set the game board as indicative of the value of the positions, every tile of the chess board needs to have a different value for the artificial intelligence algorithm to decide, along with the values of all the different pieces in the game [8]. The values on the board are either negative or positive. The negatively valued, *-1*, positions are for losing or non-beneficial positions and positively valued, *1*, positions are for winning or beneficial positions. However, the algorithm needs to possibly avoid all the negatively valued positions and choose one most beneficial move. For the artificial intelligence to decide the best action to execute, it needs to create a search tree with all the possible actions listed in the final nodes at the very bottom of the tree. Then, to select which move to execute, the algorithm applies *min* and *max* functions to every layer from bottom to top to every branch of the tree. The value gained after the applications is considered as the best move in case of MiniMax algorithm. The figure of the search tree may be considered as the following:

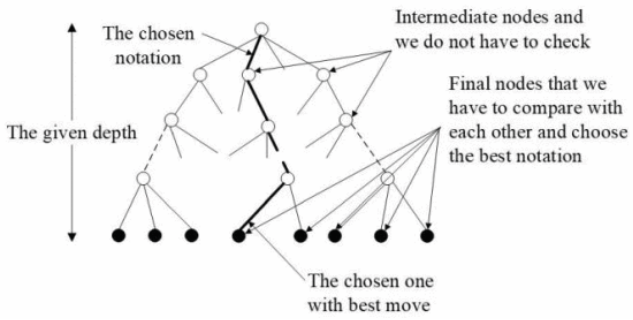

Figure 1. The MiniMax algorithm for the AI [6]

The search tree implemented in our project succeeds in case of creating the search tree and choosing the best probable option in the given circumstances of the game. Nevertheless, the MiniMax algorithm have a critical drawback, which is the immense use of the computer's memory. For a regular computer with a standard memory ad RAM capabilities to handle the search tree, the tree needs to be limited to a certain depth, so the application does not run out of memory. The implemented search tree in our project has a maximum depth of four which means that the algorithm can differ winning and losing moves and the actions taken by the artificial intelligence are may be evaluated as the best possible options considering the short response time of the artificial intelligence algorithm. However, the artificial intelligence still does not evaluate all the possible moves and this causes the algorithm to overlook some of the possible actions it may take.

To determine the efficiency and the response time of the artificial intelligence, the application have been set to different search tree depths and the time required for the reaction time against the player is recorded. To create a stable run-time graph, the tested depths are recorded from one to four respectively. The results acquired from the efficiency tests are used to form the values of the graph to demonstrate efficiency of each search depth. The created graph is as the following:

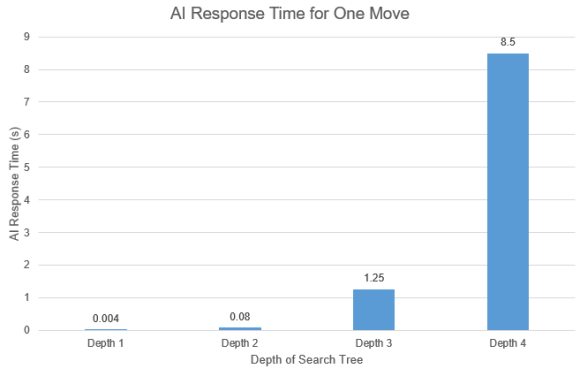

Figure 2. Response time of the AI for one move in the game

Despite the various depths, the accuracy of the algorithm has noted as satisfactory. The artificial intelligence was able to successfully respond various situations in the game. The reason for the increasing run-time of the application is the increasing number of positions to consider by the algorithm as the created search tree grows. This causes the artificial intelligence to spend more time and memory into deciding the next action. Because of this, it is ideal for standard systems to run the algorithm at the search depth of three to be most efficient.

4 Discussion

The MiniMax algorithm is one of the best artificial intelligence algorithms to play the game of chess because of its reliability and quick response time on smaller scale search trees. The algorithm implemented succeeds in considering all the listed available possibilities and evaluating the winning or losing moves to decide the limit the winning chance of the opponent and to increase the payoff from the move. The values of the pieces and the board tiles work as intended and the artificial intelligence can clearly differentiate and make its move considering the various pieces such as a valuable piece, a queen, and a common piece, a pawn. Although, the algorithm needs a larger search tree to be more accurate on the decisions, this only increases the memory use of the application and causes run-time issues. Therefore, we can say that the MiniMax algorithm is not totally accurate with the shorter search trees, but it is very efficient considering the instantaneous response time of the artificial intelligence with using a search tree of depth three. The search three of depth three is enough to satisfy the needs of the players in a casual chess game with standard setups [3].

On the other hand, if we compare the MiniMax algorithm other other similar wide-spread artificial intelligence algorithms, such as Monte Carlo Tree Search and Negamax, it is evident that the MiniMax algorithm is better defined and simpler algorithm than Monte Carlo. However, the difference between Negamax and MiniMax is very subtle. We can define the Negamax algorithm as the more concise application of the MiniMax algorithm. In MiniMax there are two subroutines for min and max, but in Negamax there is only one which combines min and max in a single subroutine. Considering both of the algorithms' running time is very similar, the more explicit structure of the MiniMax is a better application.

5 Conclusion and future work

In conclusion, The MiniMax algorithm is an optimal artificial intelligence algorithm for playing chess. The aim of the algorithm is ti minimize the payoff of the opponents by making the most beneficial move to itself. The algorithm uses a search tree to determine its response to the player by considering all the possible moves in a certain situation. Consequently, the depth of the search tree is related to the accuracy of the output. As the search tree grows, it can list more possibilities. However, in a complicated game like chess, where there are numerous possibilities to evaluate, it is inconvenient for the artificial intelligence to calculate the output if the search tree is not limited to a certain depth. Because of this, an artificial intelligence which uses MiniMax algorithm to play chess on an average system needs to limit the depth of the search tree to three to get the maximum efficiency from the program.

For the implemented MiniMax algorithm in the project to improve, the current artificial intelligence may be enhanced with the Alpha-beta algorithm to decrease the memory use of the artificial intelligence. This can lead to the use of more accurate, higher search trees.

ACKNOWLEDGEMENTS

I would like to thank my teammates for their hard work on our research. Their contributions assisted me to establish ideas on different artificial intelligence algorithms' advantages and disadvantages. I also appreciate their support to form my ideas on my research report. I must also thank my family members for their encouragement to help me get through my research project.

REFERENCES

[1] Akshay L. Aradhya. Minimax algorithm in game theory — set 1 (introduction), 2021. Available at: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/.

[2] Vikrant Chloe and Vijay Gadicha, 'A review towards human intuition based chess playing system using ai & ml', *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, **11**(2), 792–797, (2020). Available at: https://turcomat.org/index.php/turkbilmat/article/view/9826/7510.

[3] Fabrizio Ginesi, 'What to know in order to build a chess playing ai', *ScienceOpen Preprints*, (2021). DOI: 10.14293/S2199-1006.1.SOR-.PPPAZQN.v1.

[4] Yang Hu and Guoyu Zuo, 'Expectation minimax algorithm for the two-player military chess game', in *2019 Chinese Control And Decision Conference (CCDC)*, pp. 6357–6362, (2019). DOI: 10.1109/CCDC.2019.8833085.

[5] Alexander. Katz and Eli. Ross. Minimax, 2021. Available at: https://brilliant.org/wiki/minimax/.

[6] Ngo Luong Thanh Tra, Phung Tri Cong, and Nguyen Duy Anh, 'Design a chess movement algorithm and detect the movement by images classification using support vector machine classifier¡sup¿*¡/sup¿', in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, pp. 335–340, (2018). DOI: 10.1109/GTSD.2018.8595604.

[7] Nilam Upasani, Ansh Gaikwad, Arshad Patel, Nisha Modani, Prashanth Bijamwar, and Sarvesh Patil, 'Dev-zero: A chess engine', in *2021 International Conference on Communication information and Computing Technology (ICCICT)*, pp. 1–6, (2021). DOI: 10.1109/ICCICT50803.2021.9510148.

[8] Putra Verda Buana and Lukman Heryawan, 'Applying alpha-beta algorithm in a chess engine', *Jurnal Teknosains UGM*, **6**(1), 37–43, (2016). Available at: https://www.neliti.com/publications/180237/applying-alpha-beta-algorithm-in-a-chess-engine.