Alpha-Beta Pruning in Mini-Max Algorithm –An Optimized Approach for a Connect-4 Game

Simran Rawat
Department of computer applications
Graphic era hill university
Dehradun , uttrakhand, India
simisingh20031812@gmail.com

Kaushal Joshi
Department of computer applications
Graphic era hill university
Dehradun , uttrakhand, India
jkaushal578@gmail.com

ABSTRACT

More than six decades after the term Dz Artificial Intelligence dz was coined by John Mc Carthy to describe brilliantly behavior shown by machines, at long last the technology empowered the world with robotization in each perspective of human life is getting to be a reality. Counterfeit insights (AI) isused to portray the insights shown by machines, which defers from the insights shown by people and other animals regularly alluded to as Normal Insights (NI). Right from the initiation of AI, amusement playing has been an range of research. One of the primary cases of AI being utilized in game playing was the computer amusement of Nim which was developed in 1951 and distributed in 1952. After that AI has been utilized in numerous computer diversions like Chess, Tic-tac- toe, and Connect-4. The diversion of Connect-4 was to begin with unraveled by James Dow Allen in 1988. Connect-4 Gaming Entrance using Adversarial Look (Counterfeit Insights), gives an online interface to the zero entirety, idealize data technique game of Interface 4. A client will be able to log in to the framework where he / she can challenge different levels of AI with an increasing degree of trouble. The moves of the AI will be optimized by using mini-max calculation and alpha beta pruning.

Key Words: Adversarial Search, Zero Sum, Perfect Information, Mini-max Algorithm, **Alpha Beta Pruning**

INTRODUCTION

Connect-4 is a popular two-player game where players take turns dropping colored discs into a grid with the goal of creating a line of four of their own colored discs either horizontally, vertically or diagonally. This game is considered a solved game, meaning that optimal moves can guarantee a win for a player. Over the years, various technologies such as data mining and computational intelligence have been used to improve the game's efficiency and interactivity. Artificial Intelligence (AI) has also been used to simulate the game play of Connect-4 and create challenging opponents. In this context, the concept of adversarial search, where an agent must plan ahead in a world where other agents are playing against it, is explored. The game is classified as adversarial and zero-sum, meaning that any advantage for one player is a disadvantage for the other. The online Connect-4 gaming interface provides users with the option to compete against bots with varying levels of difficulty, ranging from random AI to eight AI players that utilize the Mini-max algorithm. This paper discusses the two algorithms used in the interface for generating game states- Mini-max and Mini-max with Alpha Beta Pruning. Section 2 explores Connect-4 using the Mini-max algorithm, section 3 discusses Connect-4 using Mini-max with Alpha Beta Pruning, and section 4 compares the two algorithms in terms of computation time and number of iterations performed. Section 5 discusses the heuristic function used to calculate the value of the game states. The paper concludes with a summary of the findings and references. Overall, the Connect-4 game provides an interesting platform for exploring adversarial search and the application of AI in gaming. The different levels of difficulty offered by the online interface make it a great tool for testing and comparing different AI algorithms.

CONNECT4 USING MIN MAX ALGORITM

The Minimax algorithm is a widely used algorithm in game theory and decision-making that is often used to determine the best move to make in two-player games. The algorithm works by searching the game tree for all possible moves and their outcomes, and then selecting the move that maximizes the minimum outcome for the current player. Connect4 is a game that is well-suited for the Minimax algorithm, as it is a two-player game where each player takes turns placing pieces on a vertical board with 6 rows and 7 columns. The objective of the game is to connect four pieces of the same color in a row, column, or diagonal before the opponent does. To implement the Minimax algorithm for Connect4, we first need to define a game tree. At the root of the tree, we have the current state of the game, which includes the positions of all the pieces on the board and whose turn it is. Each level of the tree represents a different turn, with the root node representing the current player's turn. At each level, there are nodes for all possible moves that the current player can make. The leaf nodes of the tree represent the outcomes of the game (i.e., whether the current player wins, loses, or draws). To determine the best move to make, the Minimax algorithm recursively evaluates the tree from the bottom up. The algorithm assigns a score to each leaf node, based on whether the current player wins, loses, or draws the game. The score is then propagated up the tree to the root node, alternating between maximizing and minimizing the score, depending on which player's turn it is. When the algorithm reaches the root node, it selects the move that leads to the highest score. This move is considered the optimal move for the current player, assuming that the opponent is also playing optimally. One issue with the Minimax algorithm is that it can be computationally expensive, especially for games with a large number of possible moves like Connect4. To address this issue, various techniques can be used, such as alpha-beta pruning and iterative deepening search, which can improve the algorithm's performance without sacrificing its accuracy.
Consider the perfect binary tree in Fig-1. It has four final states and these four leaves can be reached from the root through various paths. The first player to make the move is at the root. Considering that the maximizing (or minimizing) player makes the first move, which means it is at root, and opponent at next level. Interesting thing to observe here is that which move maximizing (or minimizing) player makes considering that opponent also plays optimally.

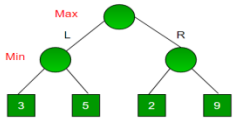

Fig-1: Initial Game Tree in case of Mini-Max

It belongs to the category of backtracking algorithm therefore the decision is made by backtracking after trying all the possible moves. Initially maximizer has the option to go left or right. Assuming it goes left after which it is the minimizerȝs turn. The minimizer has a choice to make between 2 and 4. The aim of minimizer is to minimize its value therefore it chooses the least value among both, which is equal to 2. Next maximizer goes right after which it is the minimizerȝs turn. The minimizer has to choose a value between 1 and 8. It will definitely choose 1 over 8. Now it is maximizerȝs turn to choose between ´ and Ť. Maximizerȝs aim is to get the maximum value possible therefore it chooses the larger value of the both that is 2. So finally the maximizer gets the optimal value of 2 and its best interest lies in going left. The game tree used in the example is very small and is just used for the purpose of describing the concepts clearly and easily but in reality Connect-4 game results in a game tree with a large number of game states. To find and compute each of the game state is virtually impossible. The depth of the tree may be six with a branching factor of seven. Therefore a large number of game states are computed. Fig-2: Final Game Tree in case of Mini-max There are two possible scores based on the left and right moves for the maximizer as can be seen from the above tree. The minimizer will never pick the value 8 from the right sub tree because it is assumed that the opponent always plays optimally.

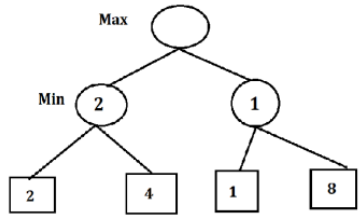

Fig-2: Final Game Tree in case of Mini-max

There are only two choices for a player in the above example but generally, there are many choices. In that case maximum/minimum is found out by recurring all the possible moves. The scores stored by leaves of the game tree for the specified example have been assumed randomly but for a typical game, these values are derived [6]. In the study carried out it is observed that when user plays the game in single player mode and chooses minimax as the operating algorithm then AI takes 2799 iterations to generate the game state and computation time is 33.00 milliseconds for a difficulty of depth 4.

CONNECT4 USING ALPHA BETA PRUNING

The technique that can be used to optimize mini-max algorithm is the application of alpha-beta pruning. When mini-max with alpha beta pruning is used instead of simple mini-max algorithm then less number of nodes is evaluated in the game tree. Adversarial search algorithm is often used in two-player computer games and this algorithm also falls under this category [6]. Alpha-Beta pruning is not altogether a different algorithm than mini-max; rather it is an optimized version of the mini-max algorithm discussed in the previous section. It optimizes the mini-max algorithm in various aspects like bringing down the computation time by performing less iteration than mini-max. In this algorithm two extra parameters are passed in the mini-max function, namely alpha and beta and that is why it is known as AlphaBeta pruning. The introduction of two additional parameters make searching much faster and game tree can be searched to much more depth. Game tree has many branches leading to different game states and benefit of using alpha-beta pruning is that if there already exists a better move then other branches need not be searched and can be pruned saving computation time. At any node in the game tree the maximizing player is assured of a maximum score which is stored by alpha and similarly the minimizing player is assured of a minimum value which is stored by beta. These two values namely alpha and beta are updated and maintained by the algorithm. Initially both players begin with worst possible values they can score and therefore alpha is allotted a value of minus infinity and beta is allotted a value of plus infinity. It is possible that when a certain branch of a certain node is chosen the minimum possible score that the minimizing player can get becomes less than the maximum score that the maximizing player is assured of getting i.e. beta is less than or equal to alpha. In this case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, there is no need to explore other branches of that node [7].Alpha is the best value that can be guaranteed by maximizer at that level or above whereas beta is the best value that can be guaranteed by minimizer currently at that level or above.

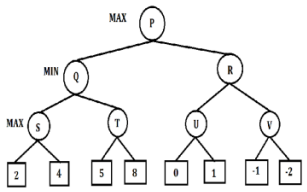

Fig-3: Initial Game Tree in case of Alpha-Beta Pruning

The initial condition of the game tree is shown in Fig-3. Initially the value of alpha is minus infinity and the value of beta is plus infinity. These values are passed down to nodes Q and R in the tree. P is the first node to be called. At P the aim of maximizer is to choose maximum of Q and R, so Q is called first by P but it is not necessary as R can also be called first. At Q the aim of minimizer is to choose minimum of S and T and hence calls S first. At S, left child which is a leaf node returns a value of 2. Now the updated value of alpha at S is maximum of minus infinity and 2 which is 2. To decide whether it should look at its right node or not, it checks the condition beta is less than or equal to alpha. This is condition does not hold true since beta is equal to plus infinity and alpha is equal to 2. So, the game tree is searched on. S now looks at its right child which returns a value of 4. At S, alpha is equal to maximum of 2 and 4 which is 4. Therefore, finally the value of node S is 4. S returns this value of 4 to its parent node Q. At Q, beta is equal to minimum of plus infinity and 4 which is 4. The minimizer is now guaranteed a value of 4 or lesser. Q will now call its right child T to check if it is possible to get a value lower than 4. At T the values of alpha and beta is not minus infinity and plus infinity but instead minus infinity and 4 respectively, because the value of beta was changed at Q and that is passed down by Q to T. Here T checks its left child which is 5. At T, alpha is equal to maximum of minus infinity and 5 which is 5. Here the value of beta is 4 and value of alpha is 5 therefore the condition beta is less than or equal to alpha holds true. Hence T does not need to check for its right child even though it stores a larger value than 5. Here right branch of node T is pruned and T returns the value of 5 to Q. Here no matter what the value of TǯŽs right child is, it is not explored. Even if the value is plus infinity or minus infinity, it still wouldnǯt make any difference. It is not necessary to even look at it because the minimizer is guaranteed a value of 4 or lesser. The basic idea behind this logic is that as soon as the maximizing player got the value of 5 it knew the minimizing player can get a lesser value of 4 on the left side of

node Q and if it comes to right side of Q it will get a value of 5 or more which in any case in greater than 4 and hence will never come to the right side and therefore there is no need to waste computation time by exploring T̓s right child which stores a value of 8. This way this algorithm works faster than simple mini-max algorithm. Now T returns a value of 5 to Q. At Q, beta is equal to minimum of 4 and 5 which is equal to 4. Therefore, node Q stores a value of 4.

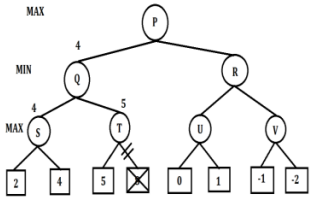

Fig-4: Intermediate Game Tree in case of Alpha-Beta

So far this is the condition of game tree. The right branch of node T which stores the value of 8 is crossed out because it was never computed. Node Q returns the value of 4 to its parent node P. At node P, alpha is equal to maximum of minus infinity and 4 which is equal to 4. Here the interesting thing to note is that the maximizer is guaranteed a value of 4 or greater. P now calls its right child R to check if it is possible to get a value higher than 4. At R, alpha is equal to 4 and beta is equal to plus infinity. Here R calls its left child U. At node U, alpha is equal to 4 and beta is equal to plus infinity. Node U looks at its left child which stores the value of 0 therefore the updated value of alpha is equal to maximum of 4 and 0 which is still 4. Right child of U stores a value of 1. Hence the best value this node can achieve is 1 but alpha still remains 4. U returns a value of 1 to R. At R, beta is equal to minimum of plus infinity and 1. The condition beta is less than or equal to alpha becomes false as beta is equal to 1 and alpha is equal to 4. So, the further search for deeper levels breaks and there is no need to compute the entire subtree of V. The intuition behind pruning of entire sub-tree of V is that, at node R the minimizer is assured a value of 1 or lesser but before that maximizer was already guaranteed a value of 4 if it goes to its left and chooses node Q. So, it does not make any sense for maximizer to go to its right and choose node R and get a value less than 1. Again, it does not matter what values were stored by both children of node V. Hence it can be seen how alpha-beta pruning algorithm saves a lot of computation time by skipping an entire sub tree. Now node R returns a value of 1 to its parent node P. Therefore, the best value at P is maximum of 4 and 1 which is equal to 4. Hence the optimal value that the maximizer can get is 4. The final game tree is shown in Fig-5. The entire subtree V has been crossed out as it was never compute

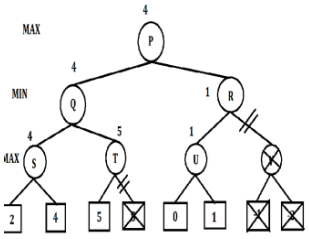

Fig-5: Final Game Tree in case of Alpha-Beta Pruning

User plays the game in single player mode and chooses minimax with alpha-beta pruning as the operating algorithm then AI takes only 477 iterations to generate the game state and computation time is only 6.00 milliseconds for a difficulty of depth 4.

4. COMPARISION BETWEEN MINI-MAX AND ALPHA-BETA PRUNING

As discussed in previous sections it is quite evident that mini-max and alpha-beta pruning are not entirely different algorithms rather alpha-beta is just an optimized version of simple min-max algorithm.

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

Table-1: Comparison between Two Algorithms

In the study carried out it is observed that when the user plays the game in single player mode and chooses minimax as the operating algorithm then AI takes 7 iterations to generate the game state and computation time is 0.00 milliseconds for a difficulty of depth 1 whereas for the same difficulty level alpha-beta pruning also takes 7 iterations and 0.00 milliseconds to compute. The difference between the two algorithms becomes starker when difficulty level is increased as it is clear from the observation that for a difficulty of depth 4 AI takes 2799 iterations to generate the game state and computation time is 33.00 milliseconds in case of mini-max whereas for the same difficulty level alphabeta pruning takes only 477 iterations and 6.00 milliseconds to compute. Further it is observed that when difficulty level is increased to 8 which is the toughest level AI takes 5847005 iterations to generate the game state and computation time is 55441.00 milliseconds in case of minimax algorithm whereas for the same difficulty level alphabeta pruning takes only 71773 iterations and 1009.00 milliseconds to compute. The result has been summarized in table-1.

HEURISTIC FUNCTION

The application of heuristic function is to coordinate robot agents in adversarial environment [4]. The AI will exhaustively evaluate the tree to a certain depth (the number of moves to Dzthink aheaddz) and then evaluation of the board is done using heuristic function. Eventually, as the end of the game draws near, the AI finds winning or losing states confined to the specified depth and plays perfectly. The whole idea here is that AI can be guided to a winnable position by heuristic function. A function is implemented that calculates the value of the board depending on the placement of pieces on the board. This function is often referred to as Evaluation Function and sometimes also called Heuristic Function [3]. The basic idea behind the heuristic function is to allot a high value for a board if maximiserʒs turn is being played or a low value for the board if minimizerʒs turn is being played.

In the actual algorithm it is generateComputerDecision()'s responsibility to calculate the most optimal move for the AI to play. It is to be kept in mind that the AI is the maximizer and human is the minimizer. The aforementioned function calls upon the maximizePlay() function with current board and the search depth as parameters. The two other parameters are alpha and beta. Alpha is the best value that the maximizer currently can guarantee at that level or above. Beta is the best value that the minimizer currently can guarantee at that level or above [6]. The first thing that is done is the calculation of the score (value) of the present board. Next check if the current state is a terminal state or not. The next step is to create multiple game states for the next iteration or stage. For each game state created call the minimizer function. The minimizer function is structurally same as the maximizer function. The board score is computed first. Then it is checked if the game state is a terminal state or not. A loop is run to generate the next iteration of game states. For each game state generated the maximizing move for the AI is calculated. The principle of alpha beta pruning is applied to reduce the number of game states that needs to be checked. This function returns the minimum score of the human player.

The heuristic function calculates two parameters, namely human points and computer points. Next, two arrays are initialized whose job is to save the winning position of the respective player. The next step is to determine through the amount of available chips. It is done through incrementing the human or computer points whenever the particular field of the game state is empty or not. The final stage is to check if the current state is a winning condition or not.

The actual function that calculates the value of game state calculates five parameters, namely vertical points, horizontal points, diagonal one point, diagonal two points and the final points. Next a nested loop is run for each column, where we rate the column according to the aforementioned function and the points to the respective column. The same procedure is carried out in

calculating the horizontal points and the two diagonal points. The final points are nothing but the summation of the former four points.

EVALUTION OR SYSTEM ANALYSIS

Connect 4 is a two-player strategy game that has been popular for many years. One of the most significant challenges in designing a Connect 4 game is developing an AI that can play the game effectively. One of the most common approaches to building a Connect 4 AI is to use the alpha-beta pruning algorithm, which is a search algorithm that is commonly used in game-playing AI. In this evaluation, we will analyze the Connect 4 game that uses the alpha-beta pruning algorithm and evaluate its strengths and weaknesses from a system analysis perspective.

Usability: Connect 4 that uses the alpha-beta pruning algorithm is easy to use for players. Players do not need to have any special skills or knowledge to play the game. The gameplay remains the same as the regular Connect 4 game, and players can choose to play against the AI or another player. The AI opponent is challenging but not too difficult, making the game enjoyable and not too frustrating.

Functionality: The alpha-beta pruning algorithm is designed to search through possible moves in the game tree and identify the best possible move. The Connect 4 game that uses this algorithm is designed to take advantage of this search algorithm, allowing the AI to make better decisions than a human player. The game functions as expected, with the AI opponent making smart and challenging moves. The game is designed to run efficiently, with no noticeable lag or delay between moves.

Scalability: Connect 4 that uses the alpha-beta pruning algorithm is scalable to some extent. The game is designed to allow two players to play against each other, with one of the players being an AI opponent. The game can be played on different platforms, making it accessible to players worldwide. However, the game's scalability is limited in terms of the number of players that can play simultaneously.

Security: Connect 4 that uses the alpha-beta pruning algorithm is secure since it does not require players to share any personal information or data. The game's AI opponent is not capable of accessing any sensitive information or data. The game's developers must ensure that the game is secure and that no vulnerabilities exist that could be exploited by malicious actors.

Reliability: The alpha-beta pruning algorithm is a reliable and well-known algorithm that has been used in game-playing AI for many years. Connect 4 that uses this algorithm is a reliable and trustworthy game that players can enjoy without any concerns. The game's rules are simple and easy to understand, and the game functions as expected. The game's AI opponent is challenging but not too difficult, making the game enjoyable and not too frustrating.

Performance: Connect 4 that uses the alpha-beta pruning algorithm performs well in terms of its speed and accuracy. The game is designed to make use of the algorithm's search capabilities, allowing the AI to identify the best possible move. The game is designed to run efficiently, with no noticeable lag or delay between moves. However, the game's performance can be affected by the hardware or software capabilities of the device on which it is played

FUTURE SCOPE

Connect4 is a classic two-player strategy game that has been popular for several decades, and it continues to be a popular game for both casual players and serious gamers. Here are some potential future scopes of Connect4:

1. Online Multiplayer: One of the most significant trends in gaming is the growth of online multiplayer games. While there are several online versions of Connect4 available, there is still room for improvement. A more immersive and engaging online multiplayer version of Connect4 could attract a new generation of players to the game.

2. Augmented Reality: Augmented reality (AR) is a rapidly developing technology that is finding its way into many applications, including gaming. An AR version of Connect4 could bring a new level of immersion to the game, allowing players to see the board and pieces in 3D and interact with them in new and exciting ways.

3. Machine Learning: Machine learning is a rapidly developing field that is already finding its way into many applications, including gaming. By training machine learning algorithms on large datasets of Connect4 games, we could develop new strategies and tactics that could improve the gameplay experience for players.

4. Competitive Gaming: Competitive gaming, or esports, is a rapidly growing industry that offers opportunities for players to compete at a high level and earn significant prizes. Connect4 has already been played competitively in some tournaments, but there is still room for growth in this area.

5. Education: Connect4 is a game that can help teach important skills, such as strategic thinking, decision-making, and planning. By incorporating Connect4 into educational curriculums, we could help students develop these important skills in a fun and engaging way.

Overall, Connect4 is a game with a rich history and a bright future. With the right combination of innovation and creativity, it has the potential to remain a popular game for generations to come.

CONCLUSION

In this paper the game of Connect-4 has been implemented using two algorithms and comparison between them is studied. First algorithm is mini-max algorithm and second one is mini-max with alpha beta pruning which is an optimized version of mini-max algorithm. The study revealed that for the same level of difficulty the two algorithms behave very differently in terms of number of iterations performed and time taken with alpha beta pruning taking much less time and performing very few iterations than mini-max to generate the game state.

REFERENCES

[1] Ribeiro, A. C. ,Rios, L. M., Gomes, R. M. 2017. “data mining in adversarial search-players movement prediction in connect,” *4 games*, 12th Iberian Conference on Information Systems and Technologies (CISTI).

[2] Bagheri, Bagheri, S., Thill, M., Koch, P., & Konen, W. DzOnline adaptable learning rates for the game Connect-4dz. *IEEE Transactions on Computational Intelligence and AI in Games*.

[3] Lucas, S. M., & Runarsson, T. P. (2015). DzOn imitating Connect-4 game trajectories using an approximate n-tuple evaluation functiondz IEEE Conference on Computational Intelligence and Games (CIG).

[4] Levner, I., Kovarsky, A., & Hong, Z. (2006). *DzHeuristic search for coordinating robot agents in adversarial domainsdz*. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. ICRA 2006.

[5] http://www.geeksforgeeks.org/category/algorithm/ga me-theory/.

[6] https://en.wikipedia.org/wiki/Alpha-beta_pruning