© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 13 Issue: 03 | NCASIT-2022 | April 18th 2022
Paper is available at http://www.jrps.in | Email : info@jrps.in
Refereed & Peer Reviewed

Implementation and analysis of Monte Carlo Tree Search and Minimax algorithm for computer to play Dots and Boxes

Mr.Charan Pote ,Professor
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Mr.Sagar Singh
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Mr.Dhruv Shende
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Mr.Sahil Bhonde
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Mr.Ritik Satokar
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Mr.Tanmay Tarte
of Computer Technology
Priyadarshini College of Engineering
Nagpur, India

Abstract— **Implementation and analysis of Monte Carlo Tree Search and Minimax algorithm for computers to play Dots and Boxes is the online version of the pen and paper game, Dots and Boxes which is the most widely played game. We have made the Dots and Boxes game Convenient, Fast to Play, & of course pleasing to the eye (with our UI). The Rules of the game are simple: Players take turns joining two horizontally or vertically adjacent dots by a line. A player/Computer that completes the fourth side of a square (a box) colours that box and must play again. When all boxes have been colored, the game ends, and the player/Computer who has colored more boxes wins. Easy Right? Well not that easy when you play, it challenges your critical thinking as the game progresses. Keeping that in mind we also have scores counted by computer and shown as the boxes get colored to make sure the player knows. We have used Artificial Intelligence Algorithms Monte Carlo Tree Search for searching the game tree. Monte-Carlo Tree Search (MCTS) and other algorithms like MiniMax or Alpha-Beta pruning are used to play against human players and also against themselves, The process of Monte Carlo Tree Search can be broken down into four distinct steps, i.e. selection, expansion, simulation, and backpropagation. Minimax algorithm plays a critical role in selecting moves that will try to find moves that give fewer points to the opponent and more to the computer making it tough on an opponent.**

Keywords - Monte-Carlo Tree Search, Mini-Max Algorithm, Game Tree, Game.

I. INTRODUCTION

Dots and Boxes is a fun and simple classic pen-and-paper game for 2 or more players. But what if it is not constrained to pen and paper? Well, in that case, it surely saves more paper, and what if we make it a bit more interesting and pleasing to the eye, that's what we thought & and that's exactly what we did. An elegant UI enhances the game experience and makes it comfortable and exciting at the same time. The game starts with an empty grid of dots. Usually, a coin is flipped or Rock-Paper-Scissors is played to see who goes first, but in Dots and boxes, the Player/Computer will be selected randomly. Players/Computers take turns connecting 2 unjoined horizontally or vertically adjacent dots. A player who completes the fourth side of a 1x1 box earns one point and must take another turn. The game ends when all lines are drawn and boxes are claimed. The player/Computer with the most points wins. If more than one player/Computer has the same high score, the game is a tie. Now, you have 2 options, Play with Algorithms or Watch algorithms play. We have used the Monte Carlo tree search algorithm and Minimax algorithm for the computer to select the upcoming moves and let them learn as they play, Player to Computer and Computer to Computer both. After certain moves, we have enough parameters to analyze the two algorithms and differentiate between them. Also, we have tried to make an elegant UI, Convenient as it only requires a Browser (Chrome or another), which is Challenging as the Game progresses.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 13 Issue: 03 | NCASIT-2022 | April 18th 2022
Paper is available at http://www.jrps.in | Email : info@jrps.in
Refereed & Peer Reviewed

II. LITERATURE SURVEY

The paper describes the idea and the execution of Implementation and analysis of Monte Carlo Tree Search and Minimax algorithm for computers to play Dots and Boxes, The biggest problem arrived is the parameters for differentiating Monte-Carlo algorithm and Mini-max algorithm, the only way we could arrive at the conclusion is the make Algorithms play against each other at least 100 - 200 times to have enough data to allocate parameters for differentiating between them. The goal of this project is to develop an artificial intelligence player for dots and boxes (a simple children's game) which improves upon standard methods. Dots and boxes have proven more difficult to work with than other simple games.

Even games whose rules are much more complicated – chess, for instance – have seen great success with the standard methods, such as minimax and alpha-beta. However, these approaches have not worked well with dots and boxes due to the difficulty of evaluating a given board and a large number of possible moves.

To overcome these problems, a relatively new method of guiding gameplay is used: Monte Carlo tree search (MCTS). MCTS has recently been successful in Go players, which previously had been extremely weak MCTS was used to overcome the inherent difficulties that arise in Go because of the massive number of possible moves and board configurations in a standard game. Because the two games share the features which make other approaches unsuccessful and because of its success in Go, MCTS seems to be the ideal candidate for playing dots and boxes. This project applies MCTS to dots and boxes and offers two potential improvements to the standard MCTS.

The advantage of testing such methods in a game setting is that one can evaluate the performance in closed systems with simple rules that are easier to write algorithms for and where the optimal result can usually be calculated using known methods for comparison.

III. OBJECTIVE

The overarching objective of this project is to analyze the game of Dots and Boxes using Monte-Carlo Tree Search and MiniMax Algorithm and to analyze various game-playing strategies made by Both the Algorithm. In which we will be differentiating both the algorithms against each other and find out which Algorithm has the most chances to win the game. As both algorithms are used in playing games or to make game moves, The final goal is to find which algorithm is best to make moves in Dot and Boxes.

The first phase of this project is to create a working and reliable implementation of the game Dots and Boxes that can be played between two players, human or Algorithm. The second phase of this project is to analyse both the game itself and various strategies used to play the game. This will be achieved using the implementation of the game as a testbed. The specified game-playing strategies to be implemented are: making moves randomly, making moves in a predetermined order, making moves based on a Minimax evaluation of the game tree, and making moves based on a Monte Carlo Tree Search (MCTS) evaluation of the game tree.

IV. METHODOLOGY

The algorithms chosen and implemented have been picked to help analyse the game itself, and also to demonstrate how effective different strategies can be in the game of dots and boxes.

The approach being taken to tackle this problem is a practical one. A significant proportion of the work being done for this project is in the implementation. Creating the game and creating the players for the game will take some time; the rest of the time will be spent analysing the performance of the players and writing the report. Results will be gathered through repeated testing of each game playing strategy. Each player type will have a strategy they use. To compare them they will be matched against every other player as well as themselves, in order to produce a ranking from best to worst and quantify their relative strength. All of the planned players, with the exception of the random player, will have parameters that can be tweaked to improve their performance. There will also be repeated testing of these players with modified parameters to determine what the best configuration is for each player. Once the best parameters for each player are discovered, the players can be trialled against each other. The players will be trialled on multiple sizes of the game board. Standard trials will take place on 3x3 and 4x4 boards as these boards have previously been solved, allowing comparison between the results achieved and known results. Trials will also take place on a 5x5 board, as this board has not yet been solved and it will be interesting to analyse the results on this board. Facing every player against every other is a good way to analyse the relative strengths of each player and is also a good way to analyse the game itself. The data obtained from players playing against copies of themselves can show if a particular board starting position is stronger. If the player who starts the match wins significantly more games than the player who goes second, it will show that player 1 is in an intrinsically stronger position than player 2.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 13 Issue: 03 | NCASIT-2022 | April 18th 2022
Paper is available at http://www.jrps.in | Email : info@jrps.in
Refereed & Peer Reviewed

A. Monte Carlo Tree Search

The principle operation of MCTS is to analyse the best moves that are available to the player. A state tree is constructed with nodes corresponding to potential moves that can be made. These moves are discovered by the selection process and analysed by random playout. The selection of moves is made using the UCB formula for exploitation and exploration. The formula for calculating
UCB for each individual move is:

$$\frac{W_i}{n_i} + c * \sqrt{\frac{\ln N_i}{n_i}}$$

W_i = win score for this node,
n_i = number of times this node has been visited,
N_i = number of times the parent of this node has been visited,
C = exploration coefficient, which is tuned experimentally.

This calculates a value for UCB with two parameters. The first parameter is the exploitation
parameter, which is simply a ratio of how many winning states have been discovered after this node and how many times this node has been visited. The second parameter is the exploration parameter, which factors in how many times this node has been visited in total compared to its parent node.
The exploration parameter is scaled by *c*, which allows the algorithm to be tuned to explore more or explore less depending on the use case.

One iteration of the MCTS algorithm includes four steps:

1. **Selection:** The selection will start at the root node and select the child of this node with the highest UCB value. This process continues, selecting the child of the current node with the highest value until a leaf node is reached. A leaf node is a node that has no children.

2. **Expansion:** Once a leaf node has been found and selected, create children from this node corresponding to all of the possible moves that could be made from this game state.

3. **Simulation:** Choose one of these node children and perform 'rollout'.This is a simulation of the game playing out until the end. The basic implementation of this is a random playout.

4. **Backpropagation**: Take the result of the simulation and backpropagate the score up the tree, all the way to the root node.

The implementation of Monte Carlo Tree Search (MCTS) was created following guides found online. The Player class MonteCarloPlayer contains an instance of MonteCarloTree. This instance of MonteCarloTree is populated with multiple MonteCarloNode instances. The methods that make up the MCTS algorithm are contained in MonteCarloTree and in MonteCarloNode.

The main loop for MCTS controls the logic and methods like it controls the 'Selection' step of the algorithm and can initiate the 'Simulation' and 'Backpropagation' steps. Each pass of the loop will first check if the current node has not been visited, or if the game in that state is finished. If these conditions are not true then the best child of this node will become the new current node. This method, choose child, also controls the 'Expansion' step of the algorithm. If the node has no children when its choose child method is called then it will create children for itself and return one of these. If the current node has not been visited before (n == 0), then the algorithm will perform a rollout from this node. A rollout consists of making a copy of the game, getting the list of all moves left to be made, shuffling this list, and then playing all of the moves. At the end of the rollout method, the node will call its own backpropagate method. This will increment its visit counter (n) and will update its win counter (t). The backpropagate method then calls the backpropagate method of its parent node. This will ensure the values are back propagated up the entire tree all the way to the root. Once this is complete, the main loop will set the current node back to the root node and start again. This is performed until the time limit is reached, at which point the child of the root node with the highest UCB value is returned.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 13 Issue: 03 | NCASIT-2022 | April 18th 2022
Paper is available at http://www.jrps.in | Email : info@jrps.in
Refereed & Peer Reviewed

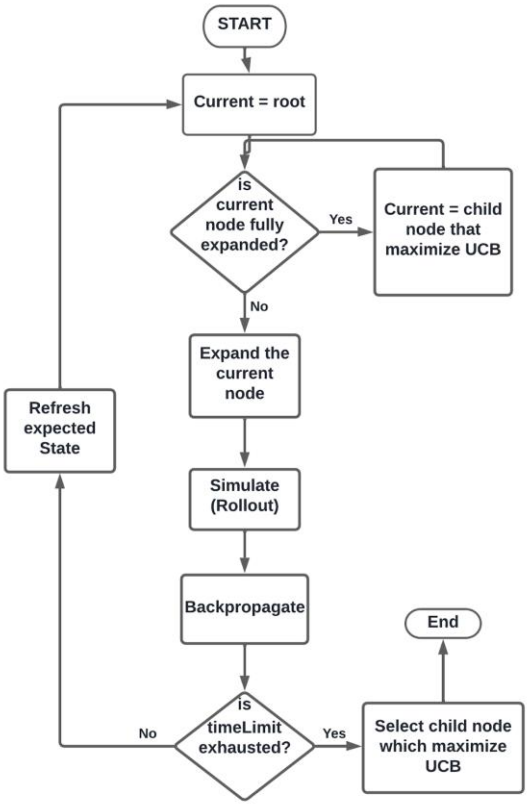

Figure 1. Monte Carlo Tree Search

B. Minimax Algorithm

The minimax algorithm is a decision-making algorithm used to determine the most optimal move of a player in a two-player game against their opponent. The minimax algorithm is simply a recursive backtracking algorithm that uses the depth-first search to explore the entire tree while selecting the most optimal path for a player.

Minimax was originally designed for n-player zero-sum games with perfect information and has more recently been extended to complex games and general decision making in the existence of uncertainty. As Dots and Boxes is an n-player zero sum game with perfect information, minimax is a very good fit for this scenario.

Here, the player (maximizer) aims to get the maximised value from the algorithm while the opponent/other player (minimizer) aims to get the minimised value from the algorithm.The initial values of the maximizer & minimizer are set to their worst case values of -infinity & +infinity. The maximizer selects the maximum possible value from its children while the minimizer selects the minimum possible value from its children.

Minimax is mainly implemented using three main functions; Maximise, Minimise and Evaluate.The MaxMove function receives a game state. If the game has terminated in this game state it will return a static evaluation of the game using the EvalGameState function. This returns a score for the state the game is in, using specific game knowledge. If the game has not terminated then all possible moves from this position are generated and iterated through. For each move that can be made a new game state is generated, simulating the player making this move. This new game state is then passed to the MinMove function, which will perform the same operations with one key difference. When the bottom of the tree is reached and the game states are evaluated, the scores are returned and passed back to these functions. The MaxMove function will save the move that returned the highest score, whereas the MinMove function will save the move that returned the lowest score. These represent opposing players making moves. The MaxMove function represents the player making the best move they can possibly make and the MinMove function represents their opponent making the best move they could possibly make. In the case of a zero-sum game the best move for an opponent will correspond to the worst move for the player.
Minimax that exploits the property:

$$Max(a, b) = -Min(-a, -b)$$

This leads to the NegaMax algorithm, which simply negates the value returned from the recursive call to NegaMax. In the case of a game in which players strictly make alternate moves this produces the exact same result as Minimax. This simplifies the algorithm as it removes the need for separate MinMove and MaxMove functions. This algorithm also introduces the concept of a depth variable. This allows the calling function to control the depth of the search in the game tree. When NegaMax is called it will be passed a positive integer, when NegaMax is called again, the depth is lowered by 1. When the depth value reaches 0 an evaluation of the game state is made and returned. This is the version of the algorithm that will be adapted for use in this project.

© INTERNATIONAL JOURNAL FOR RESEARCH PUBLICATION & SEMINAR
ISSN: 2278-6848 | Volume: 13 Issue: 03 | NCASIT-2022 | April 18th 2022
Paper is available at http://www.jrps.in | Email : info@jrps.in
Refereed & Peer Reviewed

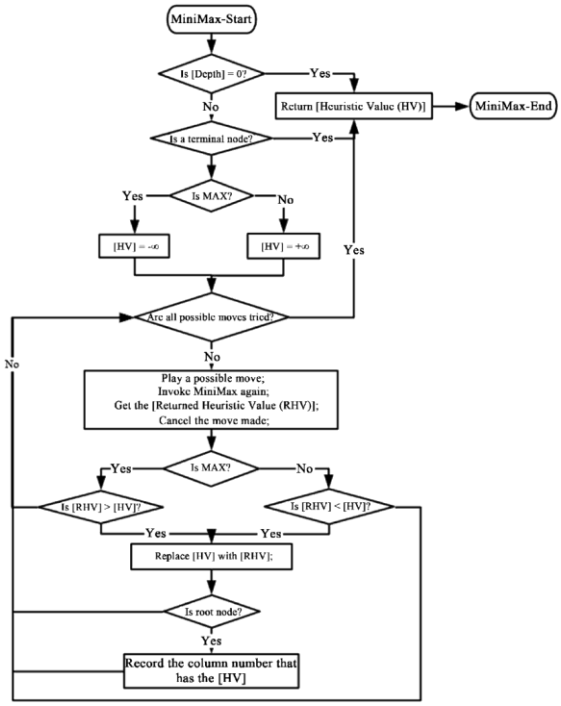

Figure 2. Minimax Algorithm

V. CONCLUSION

The aims of this project that were outlined in the important conclusions section included creating an implementation of the game, creating working computer players and analysing game playing strategies. Most of these objectives have been successfully achieved, with the exception of determining which player position is stronger. A working and reliable implementation of the game Dots and Boxes has been created. The time spent designing and implementing the game in a modular way paid off; the game is reliable and the player system is very loosely coupled to the game system.

A working GUI has also been created to make playing the game enjoyable and easy. The GUI has been designed so that users cannot enter data that would cause the game to crash. It has also been designed so that humans can play against each other, against the computer or can watch two computer players play against each other.

The computer players created for the game are successful in their implementation. They play reliably and are consistent in their strategies. Data was collected by simulating many games with each of the players, on multiple different board sizes. This data was analysed and a ranking was produced, ordering the players from strongest to weakest. This ranking is Minimax, Monte Carlo Tree Search, Ordered then Random. Multiple potential improvements to the players that could be made have been identified after analysis of the data obtained and further research. These are described in the future work section of the report.

The only aim that has not been achieved is determining if either player 1 or player 2 is in a stronger position on any board size. This was attempted and data was gathered, but the data was inconclusive and no definitive answer was found. This is likely because the computer players that have been implemented are not strong enough to exploit the strength inherent in either position. To determine this in future, stronger players would need to be produced and more trials would need to be performed.

VI. REFERENCES

[1] Solving Dots and Boxes. Barker, Joseph K, and Korf, Richard E. 2012, Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, Vol. 26, pp. 420-426.

[2] Chapter 16: Dots and Boxes. Berlekamp, Elwyn R, Conway, John H and Guy, Richard K. 2, s.l. :Academic Press, 1982, Winning ways for your Mathematical plays, Volume 3, Vol. 3, pp. 507-550.

[3] Improving Monte Carlo Tree Search With Artificial Neural Networks without Heuristics. Cotarelo,Alba, et al. 5, 2021, Applied Sciences, Vol. 11, p. 2056.

[4] Haran, Brady and Berlekamp, Elwyn. How to always win at Dots and Boxes -Numberphile. YouTube. [Online] 12 January 2015. https://www.youtube.com/watch?v=KboGyIilP6k.

[5] Wilson, David. Dots-And-Boxes Analysis Results. [Online] [Cited: 01 04 2021.] https://wilson.engr.wisc.edu/boxes/results.shtml.

[6] Mastering the game of Go with deep neural networks and tree search. Silver, David, et al. 2016, Nature, Vol. 529, pp. 484-489.
.

[7] Champandard, Alex J. AiGameDev. [Online] 12 August 2014. [Cited: 14 April 2021.] https://web.archive.org/web/20170313041719/http://aigamedev.com/open/coverage/mcts-romeii/.

[8] Monte Carlo Methods. Johansen, A.M. 2012, International Encyclopaedia of Education (Third Edition), pp. 296-303.

[9] Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. Coulom, Rémi. Turin, Italy : Springer,

2006, Computers and Games, 5th International Conference, pp. 29-31.

[10] Choudhary, Ankit. Analytics Vidhya. [Online] 2019. [Cited: 20 03 2021.] https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithmdeepmind-alphago/.

[11] Riverbank Computing Limited. PyQt5==5.15.2. Dorchester : Riverbank Computing Limited, 2020.

[12] Geeks for Geeks. GeeksforGeeks. [Online] 2019. https://www.geeksforgeeks.org/minimaxalgorithm-in-game-theory-set-4-alpha-beta-pruning/.

[13] GeeksforGeeks. [Online] 2019. https://www.geeksforgeeks.org/ml-monte-carlo-tree-searchmcts/.

[14] Baeldung. Baeldung. [Online] 2020. https://www.baeldung.com/java-monte-carlo-tree-search.

[15] Parallel Monte-Carlo Tree Search. Guillaume, M.J-B., Chaslot, Mark H.M. and Winands, Jaap van den Herik. Beijing : Springer, 2008, Computers and Games, 6th International Conference, pp. 60-71. 978-3-540-87607-6.

[16] Wikimedia. [Online] 21 11 2011. [Cited: 01 04 2021.] https://commons.wikimedia.org/wiki/File:Dots-and-boxes.svg.

[17] Wikimedia. [Online] 3 April 2020. [Cited: 20 04 2021.] https://en.wikipedia.org/wiki/File:MCTS-steps.svg