A

MINI PROJECT REPORT ON

"Tic Tac Toe using Minmax algoritham"

Prepared By

Mr. Chaudhari Shubham (21)

Mr. Chothave Sai (24)

Mr. Durdhawale Hemant (35)

(B.E. Computer)

SAVITRIBAI PHULE PUNE UNIVERSITY

In the academic year 2020-21

Department of Computer Engineering

Sanjivani Rural Education Society's

Sanjivani College of Engineering

Kopargaon – 423 603

ACKNOWLEDGEMENT

We would like to thank our family for their unconditional love and support and being there for us, despite all the troubles and problems that grow in life.

We express our sincere gratitude to Dr. D. B. Kshirsagar Head of Department (Computer) SRES COE for his unending support and encouragement during the years we have studied under his tutelage.

We are greatly indebted to project guide Prof. longani, Assistant Professor (Computer), for providing valuable guidance at every stage of this project work.

Our sincere thanks go to all the teachers and staff for their help and understanding.

Mr. Chaudhari Shubham (21)

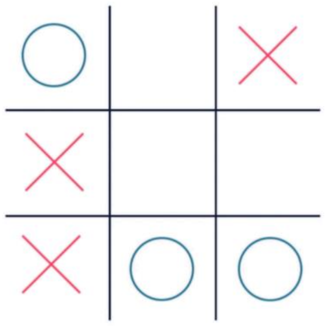
Mr. Chothave Sai (24)

Mr. Durdhawale Hemant (35)

Contents

Introduction

Tic Tac Toe (also known as noughts and crosses or Xs and Os) is a paper and pencil game for two players, X and O, who take turns marking the spaces in 3x3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal row wins the game.



Theory

Minimax Algorithm

Minimax is a backtracking algorithm that is used in decision making and game theory to find the optimal move of the player, assuming that your opponent also plays optimally. Minimax is a recursive algorithm and it is widely used in two player turn-based games.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Literature Survey

"Application of Combinatorial Techniques to the Ghanaian Board Game Zaminamina Draft- Elvis Kobina Donkoh, Rebecca Davis, Emmanuel D.J Owusu-Ansah, Emmanuel A. Antwi, Michael Mensah form Researchgate", Games happen to be a part of our contemporary culture and way of life. Often mathematical models of conflict and cooperation between intelligent rational decision-makers are studied in these games. Example is the African board game 'Zaminamina draft' which is often guided by combinatorial strategies and techniques for winning. In this paper they deduce an intelligent mathematical technique for playing a winning game. Two different starting strategies were formulated; center starting and edge or vertex starting. The results were distorted into a 3x3 matrix and elementary row operations were performed to establish all possible wins. MatLab was used to distort the matrix to determine the diagonal wins. A program was written using python in artificial intelligence (AI) to help in playing optimally.

"Evolution of No-loss Strategies for the Game of Tic-Tac-Toe-Anurag Bhatt,Pratul Varshney,kalyanmoy Deb from Researchgate", This paper is aimed at evolving a number of no-loss strategies using genetic algorithms and comparing them with existing methodologies. To efficiently evolve no-loss strategies, they have developed innovative ways of representing and evaluating a solution, initializing the GA population, developing GA operators including an elite preserving scheme. Interestingly, our GA implementation is able to find more than 72 thousand no-loss strategies for playing the game. Moreover, an analysis of these solutions has given us insights about how to play the game to not lose it. Based on this experience, they have developed specialized efficient strategies having a high win-to-draw ratio. The study and its results are interesting and can be encouraging for the techniques to be applied to other board games for finding efficient strategies.

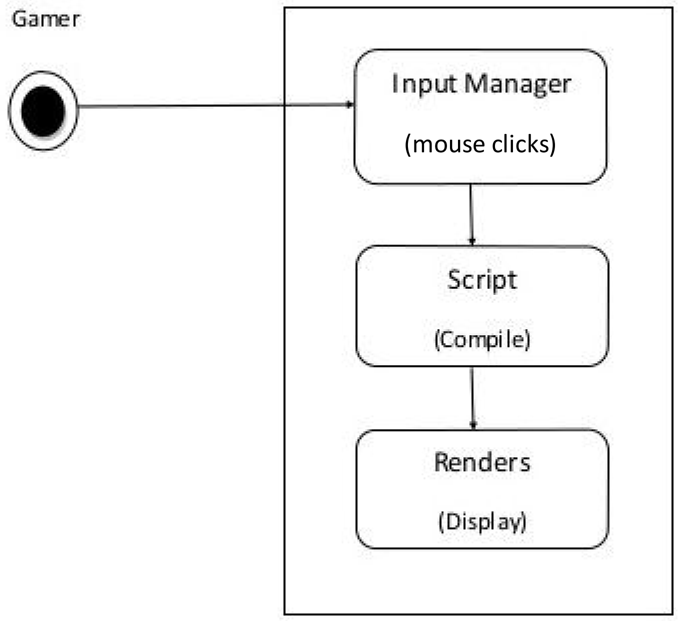
Scope

- Limited to 3*3 board

Objectives

- To play the best move using the minimax algorithm.

System Model



Technical Specifications

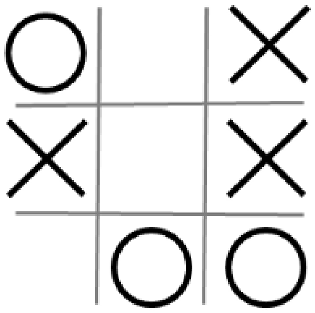
Software requirement

- Operating system (Linux/Windows)
- Browser (Chrome/Firefox)
- JavaScript
- CSS

Hardware requirements

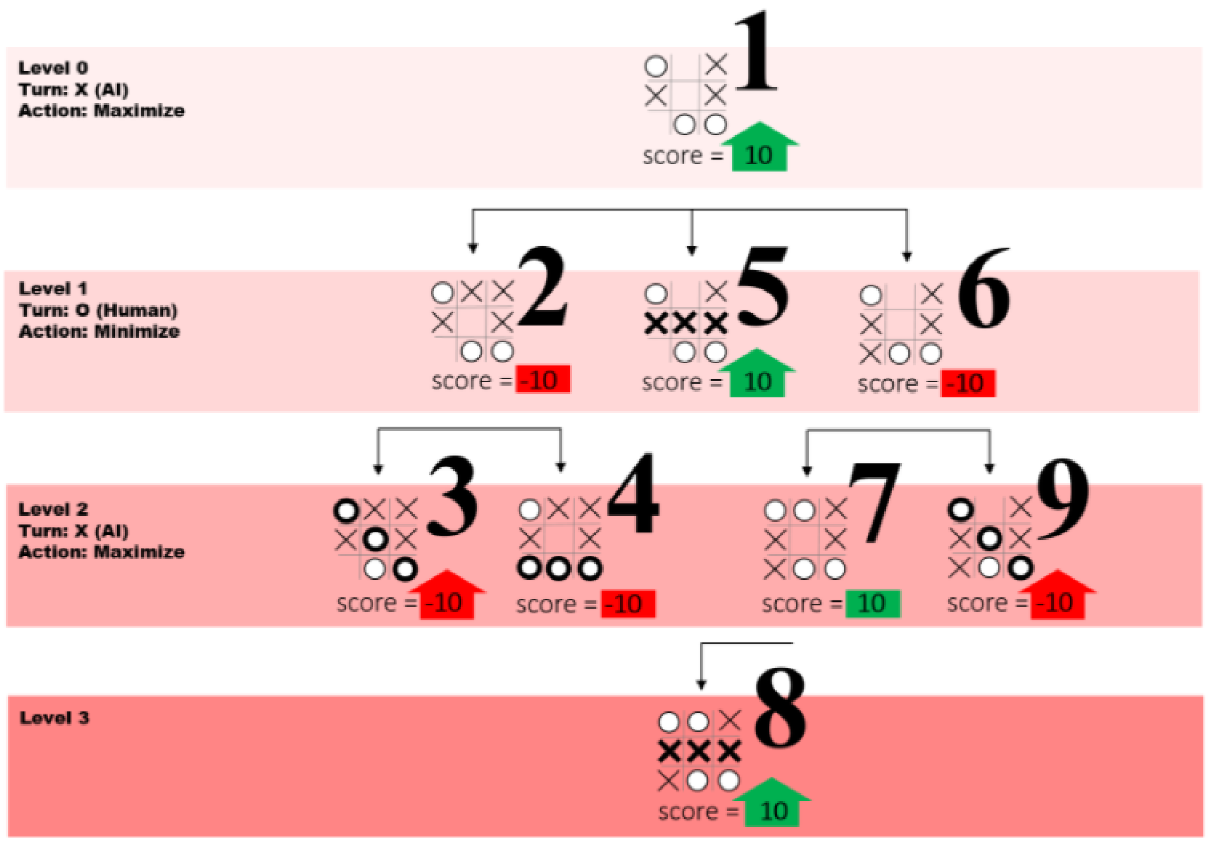
- Any processor
- Minimum 1 GB of RAM

Experimental Analysis



Assume the AI is X and the human player is O. By using the following figure, lets follow the algorithm's function calls (FC) one by one.

Note: In following figure, large numbers represent the each function call and levels refers to how many steps ahead of the game the algorithm is playing.



Variables that are used in following explanation

origBoard = refers to the state of board

aiPlayer = AI player

huPlayer = Human player

1. origBoard and aiPlayer is input to the algorithm. The algorithm makes a list of the three empty spots it finds, checks for terminal states, and loops through every empty spot starting from the first one. Then, it changes the newBoard by placing the aiPlayer in the first empty spot. After that, it calls itself with newBoard and the huPlayer and waits for the FC to return a value.
2. While the first FC is still running, the second one starts by making a list of the two empty spots it finds, checks for terminal states, and loops through the empty spot starting from the first one. Then, it changes the newBoard by placing the huPlayer in the first empty spot. After that it calls itself with newBoard and the aiPlayer and waits for the FC to return a value.
3. Finally the algorithm makes a list of the empty spots, and finds a win for the human player after checking for terminal states. Therefore, it returns an object with a score property and value of -10.

   Since the second FC listed two empty spots, Minimax changes the newBoard by placing huPlayer in the second empty spot. Then, it calls itself with the new board and the aiPlayer.
4. The algorithm makes a list of the empty spots, and finds a win for the human player after checking for terminal states. Therefore, it returns an object with a score property and value of -10.

   On the second FC, the algorithm collects the values coming from lower levels (3rd and 4th FC). Since huPlayer's turn resulted in the two values, the algorithm chooses the lowest of the two values. Because both of the values are similar, it chooses the first one and returns it up to the first FC.

   At this point the first FC has evaluated the score of moving aiPlayer in the first empty spot. Next, it changes the newBoard by placing aiPlayer in the second empty spot. Then, it calls itself with the newBoard and the huPlayer.
5. On the fifth FC, The algorithm makes a list of the empty spots, and finds a win for the human player after checking for terminal states. Therefore, it returns an object with a score property and value of +10.

After that, the first FC moves on by changing the newBoard and placing aiPlayer in the third empty spot. Then, it calls itself with the new board and the huPlayer.

6. The 6th FC starts by making a list of two empty spots it finds, checks for terminal states, and loops through the two empty spots starting from the first one. Then, it changes the newBoard by placing the huPlayer in the first empty spot. After that, it calls itself with newBoard and the aiPlayer and waits for the FC to return a score.
7. Now the algorithm is two level deep into the recursion. It makes a list of the one empty spot it finds, checks for terminal states, and changes the newBoard by placing the aiPlayer in the empty spot. After that, it calls itself with newBoard and the huPlayer and waits for the FC to return a score so it can evaluate it.
8. On the 8th FC, the algorithm makes an empty list of empty spots, and finds a win for the aiPlayer after checking for terminal states. Therefore, it returns an object with score property and value of +10 one level up (7th FC).

   The 7th FC only received one positive value from lower levels (8th FC). Because aiPlayer's turn resulted in that value, the algorithm needs to return the highest value it has received from lower levels. Therefore, it returns its only positive value (+10) one level up (6th FC).

   Since the 6th FC listed two empty spots, Minimax changes newBoard by placing huPlayer in the second empty spot. Then, calls itself with the new board and the aiPlayer.
9. Next, the algorithm makes a list of the empty spots, and finds a win for the aiPlayer after checking for terminal states. Therefore, it returns an object with score properties and value of +10.

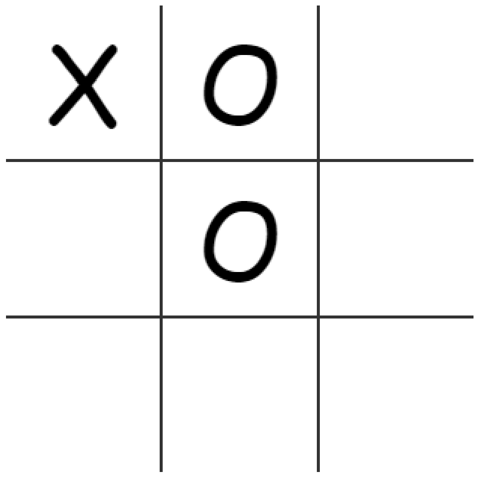
   At this point, the 6 FC has to choose between the score (+10)that was sent up from the 7th FC (returned originally from from the 8 FC) and the score (-10) returned from the 9th FC. Since huPlayer's turn resulted in those two returned values, the algorithm finds the minimum score (-10) and returns it upwards as an object containing score and index properties.

   Finally, all three branches of the first FC have been evaluated (-10, +10, -10). But because aiPlayer's turn resulted in those values, the algorithm returns an object containing the highest score (+10) and its index (4).

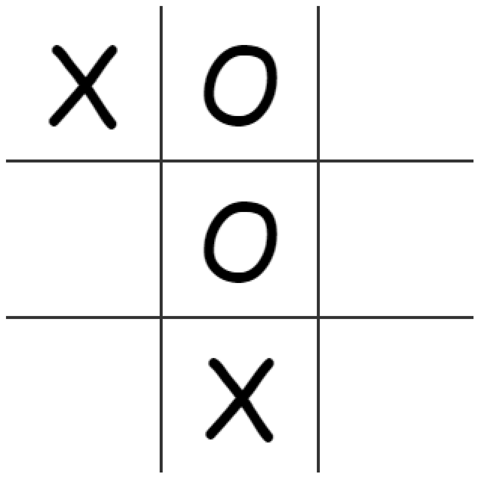
In the above scenario, Minimax concludes that moving the X to the middle of the board results in the best outcome.

Result Snapshots

Following the random board, next move is of an AI player



Here is the move made by an AI player



Conclusion

Future Scope

- This can be extended to larger board size for eg. NxN.

References

- https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
- https://www.freecodecamp.org/news/how-to-make-your-tic-tac-toe-game-unbeatable-by-using-the-minimax-algorithm-9d690bad4b37/