Task 5 :- 3D Tic Tac Toe Agent

Srishty Gandhi, roll no. 20CS30052

***Abstract*— Tic Tac Toe is a simple game, with simple rules and simple strategy. We ca also teach a computer how to play Tic tac toe with help of Minimax algorithm.**

Minimax algorithm is kind of back tracking algorithm. Minimax is basically a recursive algorithm which does an exhaustive search of game space. The search space in 3X3 tic tac toe board is small, so a simple minimax algorithm is sufficient for that. But when we scale it to a 3X3X3 board, or a 3D board, the search space increases exponentially, which in turn increases the time taken by minimax exponentially, thus rendering it insufficient. To tackle this problem one can use alpha-beta pruning and heuristics to make the algorithm more efficient. Minimax algorithm is extensively used in decision making, game theory and artificial intelligence. It is used to minimize the possible loss for a worst case scenario.

I. INTRODUCTION

To make the tic tac toe agent i used a minimax algorithm and then further improved it by using alpha beta pruning and some heuristics

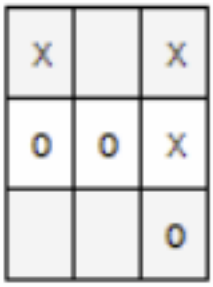
II. PROBLEM STATEMENT

The problem statement was to make an agent agent that will play tic tac toe in gym tic tac tie environment. (Since I faced some difficulties in setting up the environment, I tried to make a pseudo environment of my own)

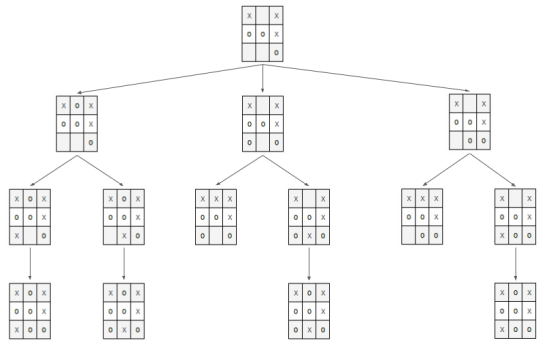
To make the agent I used a minimax algorithm. Minimax is a kind of backtracking algorithm that is used in decision making

Given a board state, we find the best move by simulating all the possible continuations from this position and choose the one that is best for us. We choose the best move by assuming that every move we make will result in maximizing our score and every move that our opponent make will result in maximizing their score and thus minimizing our score.

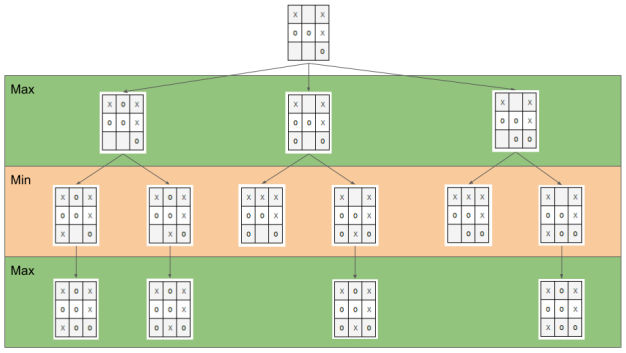
Lets take an example, to understand minimax algorithm Suppose this is board state at a given point of time and x has to move next.



The following continuations are possible:

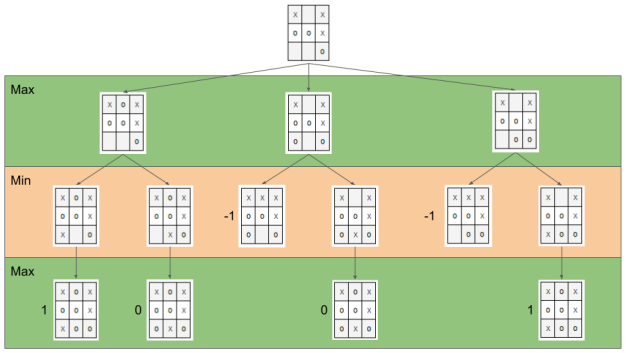


now, first player i.e. the player with an x tries to maximize the score, and the second player try to minimize it, in alternative moves



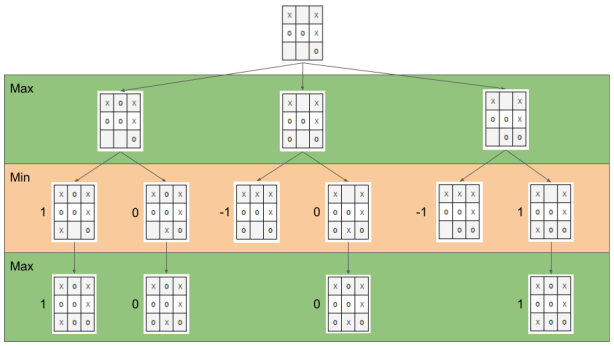
We label all the end game states as follows:

- 1 for a win
- -1 for a loss
- 0 for a draw

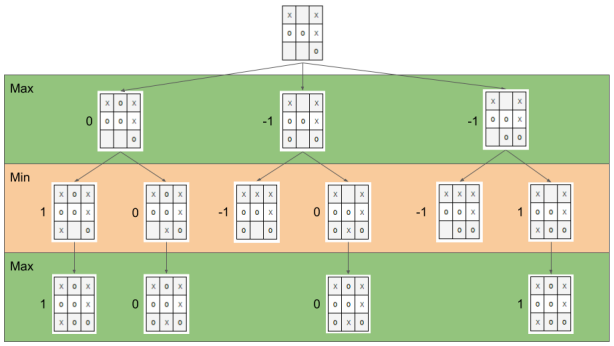


Now we move one layer up, Since in this case, as there is only one possible move, we just propagate that value to the

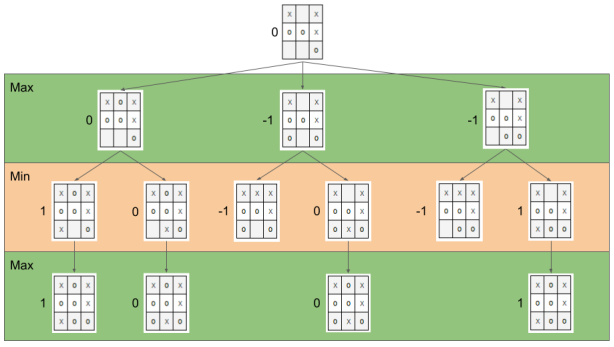
upper layer.



Now we propagate up again. This time it is the minimizing player's turn, so we propagate the smaller values for each possible move up.



Finally, we propagate one more layer up. This time it's the maximizing player again, so we chose the highest possible value of all moves for the position



So, that is how a minimax algorithm takes its decision, by maximizing and minimizing the score at alternate layers of the game tree.

III. Related Work

Another approach to make this agent, would be a machine learning approach. The machine learning approach that is used is called *reinforcement learning* , and the particular variant which is used is called *tabular Q*

IV. INITIAL ATTEMPTS

To make the board I used a dictionary(because a lot people find indexing it a little bit easier, as it resembles the keypad of a phone). This dictionary will store the positions of both the marks(X and O), throughout the game.

```
board = {1: ' ', 2: ' ', 3: ' ',
         4: ' ', 5: ' ', 6: ' ',
         7: ' ', 8: ' ', 9: ' '}
```

The environment that i created works in the following steps

Step1:- The environment asks the user to input a value between 1 and 9

Step 2:- It checks whether the position corresponding to the input is free or not, If the space is free it inserts the mark there, If not then it again asks the user for a valid input.

Step 3:- After a new position is marked, the algorithm checks for a draw. To check that it checks whether there is an empty spot in the board or not.

- if there is a draw , it prints 'Draw' and then exits from the program
- if there is an empty position then:-
  - it checks whether the last move was a winning move or not, if it was a winning move (which wont be possible in a 2d board if the agent gets the first move XD), it prints 'You win', and again exists from the program.
  - if it was not a winning move, then the agent tries to make its best possible move

Step 4:- The agent makes its best possible move(whether a certain move is best or not, is decided by the minimax algorithm as explained above)(p.s. minimax algorithm is like the computer is playing all the possible moves in its head and deciding which one is best)

- after the move is made, the possibility of draw is checked
  - if there is a draw , it prints 'Draw' and then exits from the program.
  - it checks whether the last move was a winning move or not, if it was a winning move, it prints 'Better luck next time'.
  - if none of the above condition is satisfied it goes back again to step one, and continues the game.

the condition for draw is checked before the conditions of winning move because

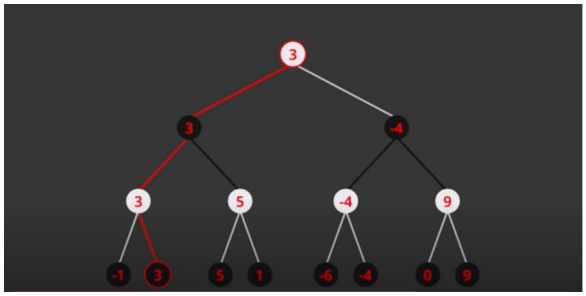
1) checking for draw takes less time
2) if both the players are moving optimally the last move is never a winning move

This approach works fine when the search space is less (2d tic tac toe), but it fails when the search space is increased (3d tic tac toe)

V. Final Approach

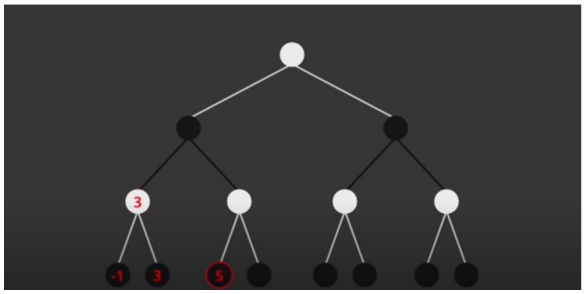
The final approach also follows the same procedure as initial approach but with a improvement in minimax algorithm. Now the minimax algorithm is equipped with alpha beta pruning too.

To understand alpha-beta pruning lets take an example. For simplicity lets assume, there are two possible move to choose between at any possible position of board.

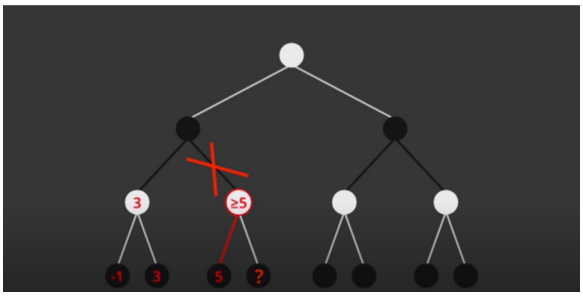


Lets assume the above position of the board and see how alpha beta pruning works.

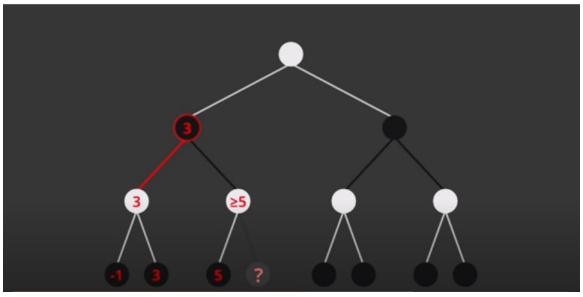
The first few steps are same as before, but consider the situation, after we have evaluated the +5 position



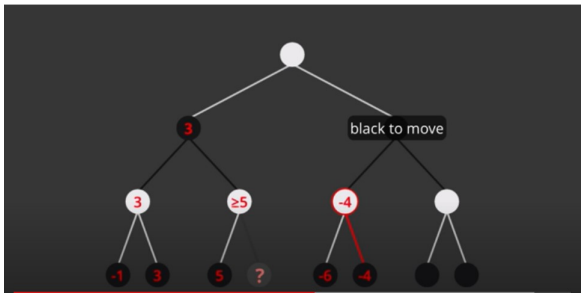
Now without evaluating the other position, we know that white can at least get a 5 from here, so we can mark this position as being greater than or equal to 5. The black wont go down this branch because he already has better option available.



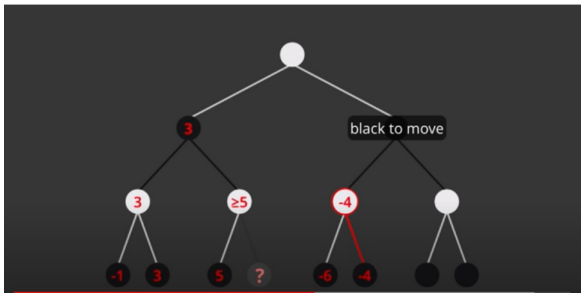
This observation means that we don't have to waste any computation on evaluating this final position, we can simply pretend it doesn't exists. In other words we have pruned it from the tree



Things are continued as normal again for a few steps, until we get here.



Black to play in this position will be choosing whichever of the two moves, leads to the lowest evaluation so we know the evaluation here is going to be less than or equal to - 4.



We can now be sure that white won't go down this branch because it already has a better option available to him and so we can prune other options

The steps of this approach is same as the previous approach till step 3

Step 4:- agent will make the best possible move according to improved minimax algorithm. and then checks for draw and winning conditions as explained above

Step 5:- if none of the above condition is met, then the value of depth is increased by one.

- if the depth has reached its max value then, the tree stops there and find the score of that position of board by using heuristics function
- if the depth has not reached its max value, then we again continue from step one

How to calculate heuristics:-

Step 1:- Fill the empty positions on the board with X (or the mark of the bot)

Step 2:- Now calculate all the possible winning combination of X = Xw

Step 3:- After that repeat the same for O = Ow

Step 4:- Heuristic = Xw - Ow

VI. Results and Observation

Depth v/s Time for 3d Tic Tac Toe

depth	time
3	5
4	27
5	207
6	25 minutes

VII. Future Work

The algorithm is still too slow, and moreover by using minimax algorithm i can make my computer to make intelligent decision, it still doesn't have the power to learn. To improvise the code further we can use machine learning algorithms.

CONCLUSION

Minimax algorithm is a way that allows a machine to make intelligent decision. Suppose we have to design a war drone then we can use this algorithm to make our drones capable to fight the enemy drone efficiently. This is one of the various examples where this method can be used in our drones

References

[1] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

[2] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-2-evaluation-function/?ref=rp

[3] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/?ref=rp