Introduction :

In artificial intelligence, the problems may solved by searching algorithms, knowledge representations and evolutionary computations.

Problem solving using searching are composed of these steps :

1. Define the problem
2. Analyze the problem
3. Identification of possible solutions
4. Choosing the optimal solution
5. Implementation

And these properties must be taken in consideration :

- Completeness :
  The algorithm is said to be complete when it gives a solution for a given random input.

- Optimality:

  If it finds the best solution (lowest path cost) among all possible solutions .

- Time complexity:
  The time taken by an algorithm to complete its task .

- Space complexity:

  The maximum storage taken by the algorithm at any time while searching.

Also search algorithm can be classify as :

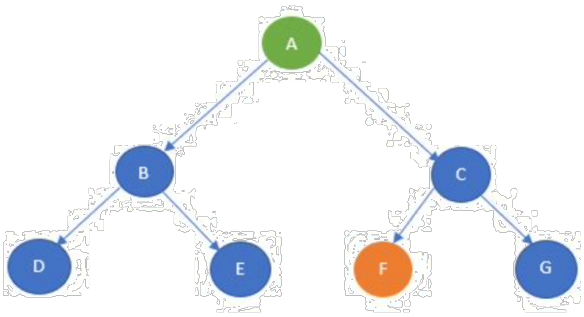
- Uninformed search :

  Which depends on the information about how to traverse the given tree and find the goal state and dose not have any extra domain knowledge

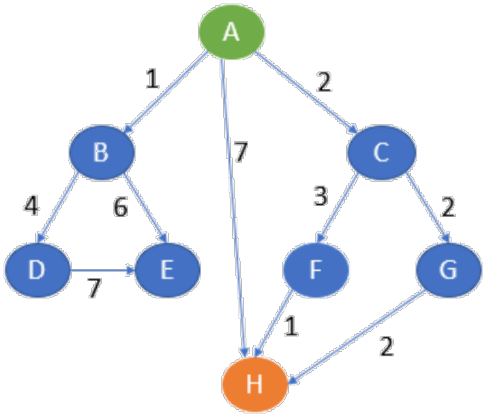
- Informed search :

  is also called heuristic search which require details such as distance to reach the goal (using heuristic value to make it more efficient)

Uninformed search tree



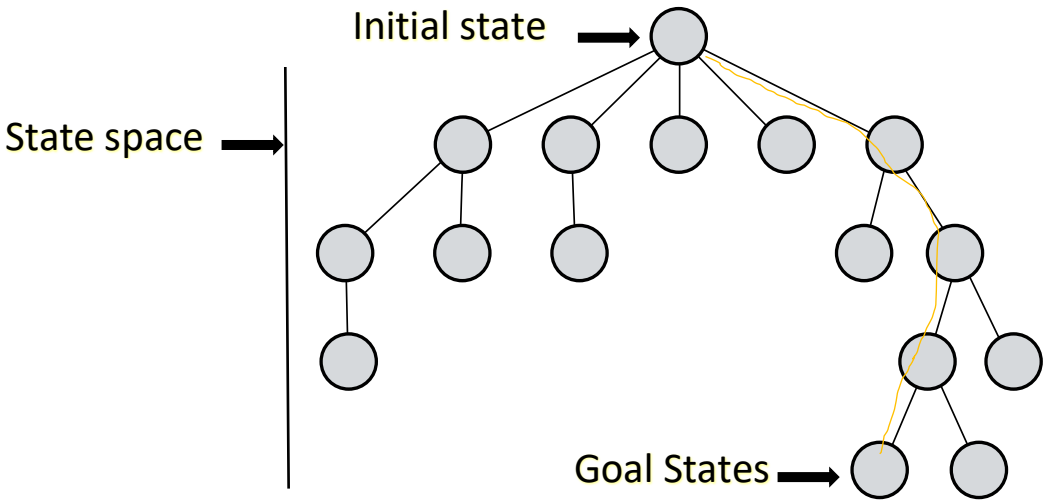
informed search tree



NODES	HEURISTICS
A	5
B	3
C	4
D	2
E	6
F	3
G	1
H	0

Using Adversarial search to solve games is a good way since it is possible to have the whole knowledge of all possibilities of moves and leading us to the optimal solution .
And the different states of the game can be represented as nodes in the game tree .
The nodes are arranged in levels that correspond to each players turns in the game .
The root node (initial state) is usually represent at the top of the diagram or the current state , it is the beginning position in the game , for instance in chess game this would be the board with its black and white pieces in its initial position .
Under root (current state) exactly on the next level of the tree there are the possible states that can result from the first players moves could be black or white chess pieces , these nodes are called the children of root node .

Each node on the next level would also have its children nodes (the space states that can be reached from it by the opposing players moves) .



This will continue level by level until reaching (goal state)where the game is over .
in chess this means that either one of the players gets to checkmate and win otherwise in other games if the board game gets full then it may occur of game ends in draw like tic tac toe or connect 4 games .

by using evaluation function it will help us to make a good decision by assigns real number scores to board positions and then comparing its values with each others which will drives us to choose the optimal solution or the path to win the games .

for example in tic tac toe the evaluation function can be determined by this formula
f(n) = (number of three lengths open for player one – number of 3 lengths player two) .

in brief, all the explanation above show us how to search in game tree and to make it in more efficient way is when we do not expand all nodes in the game tree ,so If you want to reduc the number of nodes explored in the Minimax strategy then apply Alpha-Beta cut on min-max strategy this will allow us to find the solution faster .

in ALPHA-BETA cutoff method we have two values which are alpha and beta

- Alpha value : a value never greater than the true score of its node .
- Beta value : a value never smaller than the true score of its node.

So by applying this method it will guaranteed to us that

- The score of a state will always be grater than the alpha value and less than the beta value of that node.
- As the algorithm evolves, the alpha and beta values of a node may change, but the alpha value will never decrease, and the beta value will never increase.
- When a node is visited last, its score is set to the alpha value of that node, if it is a MAX node, otherwise it is set to the beta value.

The reason of all these search algorithms is to get a high performance ,faster and reliable way to achieve our goal . the performance measure is depend on the quality of the evaluation function and the depth of search .

There is an video explanation of the difference between minmax strategy and Alpha-Bata strategy

https://youtu.be/l-hh51ncgDI

tic tac toe problem explanation and its formulation :

in our example which is tic-tac-toe the problem is each player try to get full three sequentially marks either by horizontally , vertically or diagonally .

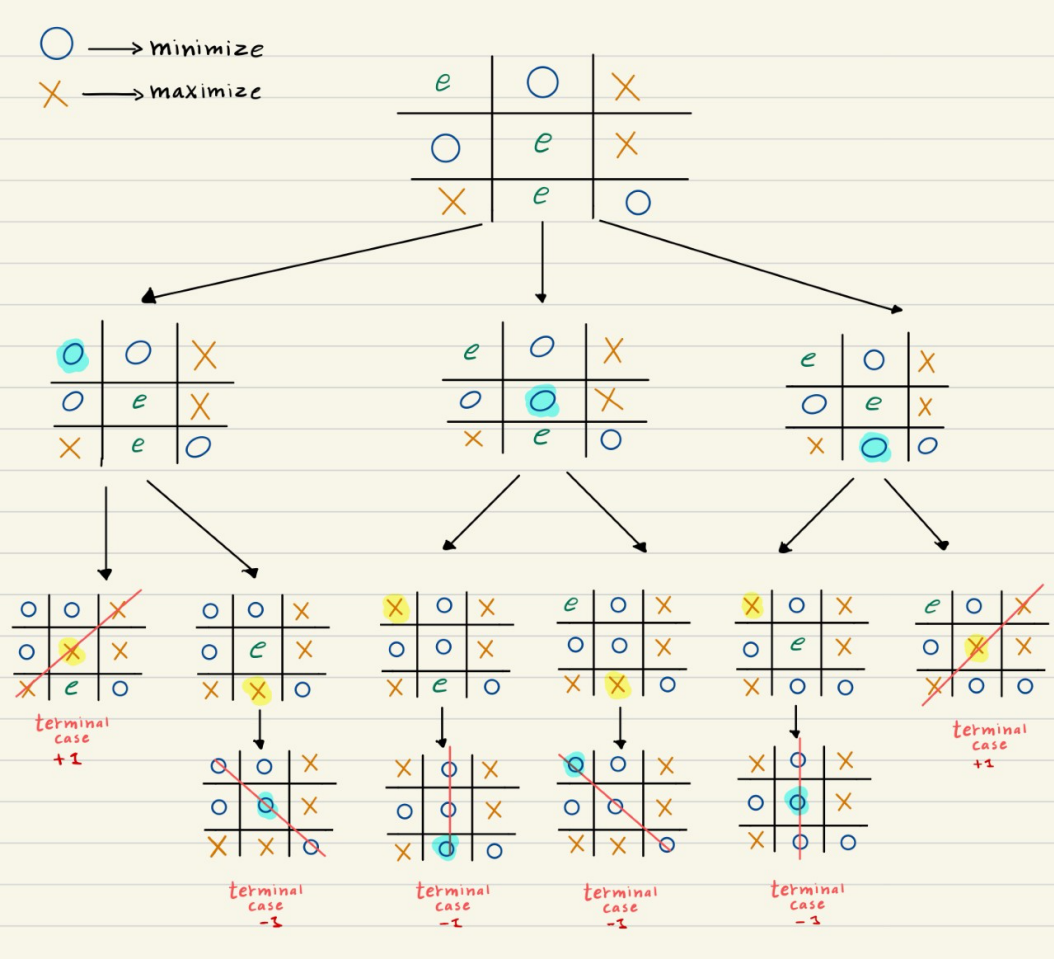
assume player one is O and player two is X so the player who gets three consecutive marks first will win the game .

before we start our example there are some concepts which are very important to know

1) Terminal case (base case)
    1) Player one win (return any negative number '+1')
    2) Player two win (return any positive number '-1 ')
    3) Draw (return 'zero')
2) Minimize player for example 'O'
3) Maximize player for example 'X'

First the game will check if the board is a terminal case if not then it will loop to the board and get all the empty squares .

Now it will tell the player that for each empty square try to mark it and reclusively call the function again until it reach a terminal case .

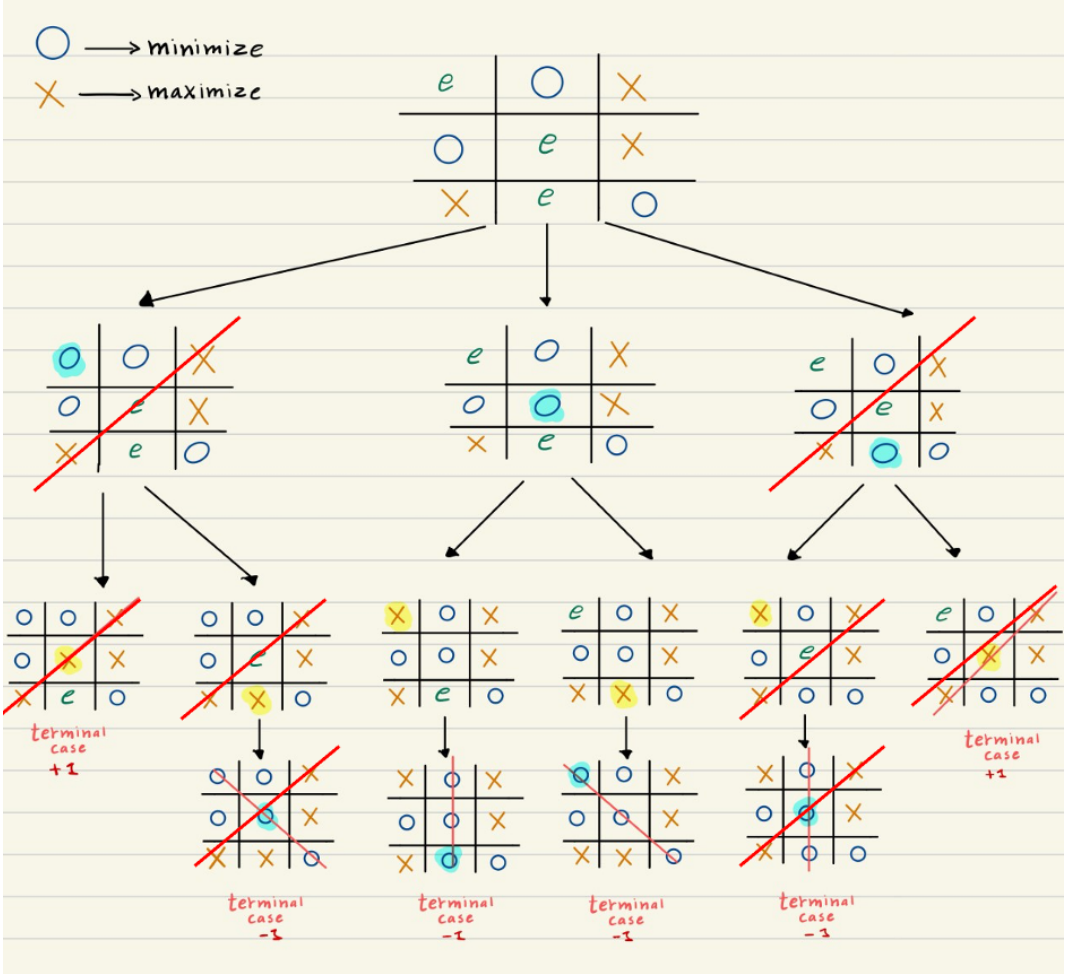


X will try to maximizing so if it comes to compare between -1 or 0 or +1 it will choose +1

0 will try to minimizing so if it comes to compare between -1 or 0 or +1 it will choose -1

If they can not either choose maximize to +1 or minimize to -1 it will end up to choosing 0 which is draw For O and X zero is -1 < 0 < +1

So now for example player O which is trying to minimize know what branch will follow and which not



Min-Max algorithm :

Basically in min max algorithm we will have these values

1) Maximum evaluation (which is initially = any negative number)
2) Minimum evaluation (which is initially = any positive number)
3) Best move ((which is initially = nothing)
4) Evaluation function for each state

It will start searching until reaching the terminal cases using resurgent call
And it will check some conditions to update these variables values

1) Check and assign evaluation value for each state by checking the this cases
   1- If case = player one then return 1 as evaluation function value and none for the move
   2- If case = player two then return -1 as evaluation function value and none for the move
   3- If the board is full then return 0 as evaluation function value and none for the move

2) In maximizing :
   check if evaluation function for this state is greater than the maximum evaluation if yes then the max evaluation value will equal to current state evaluation function and best move will equal its row and column then it will call the minmax method again .finally return these values to the main call method.

3) In minimizing :
   check if evaluation function for this state is less than the minimum evaluation if yes then the min evaluation value will equal to current state evaluation function and best move will equal its row and column then it will call the minmax method again. finally return these values to the main call method.

Alpha-Beta pruning :

Alpha is the best already explored option along path to the root for maximizer .
Beta is the best already explored option along path to the root for minimizer .
Basically in Alpha-beta algorithm we will have these values

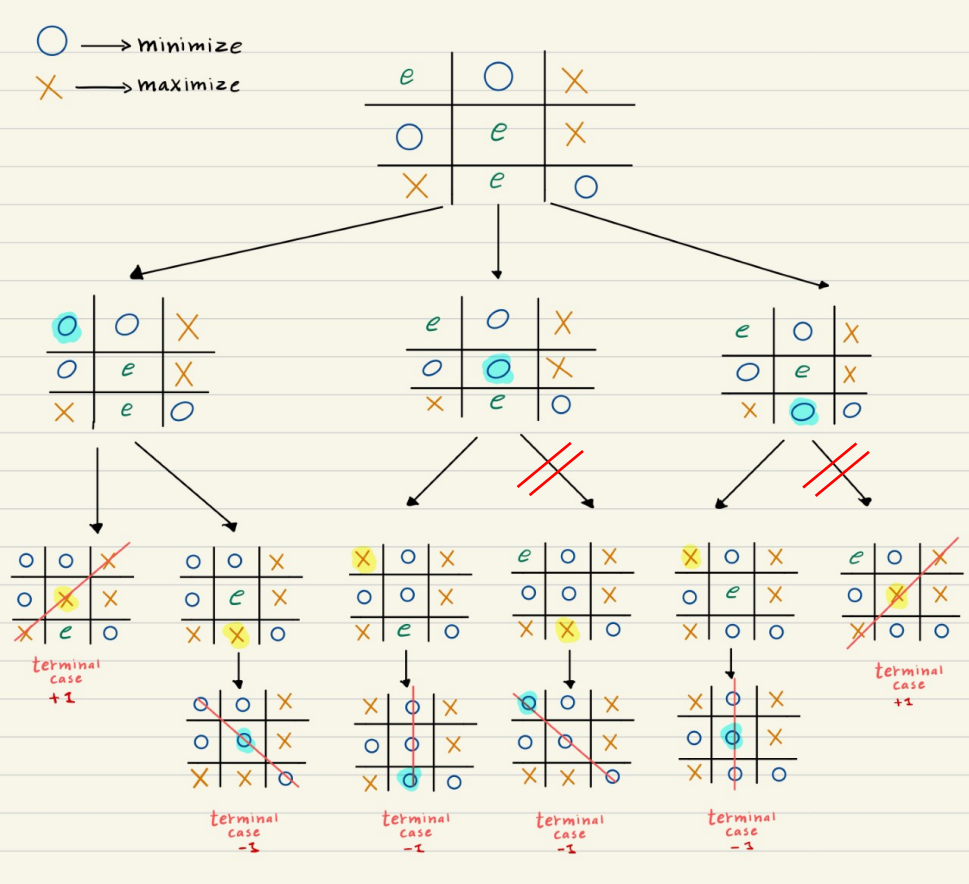
1) Maximum evaluation (which is initially = any negative number)
2) Minimum evaluation (which is initially = any positive number)
3) Best move ((which is initially = nothing)
4) Alpha value (which is initially = any negative number)
5) Beta value (which is initially = any positive number)
6) Evaluation function for each state

1) Check and assign evaluation value for each state by checking the this cases
    1) If case = player one then return 1 as evaluation function value and none for the move
    2) If case = player two then return -1 as evaluation function value and none for the move
    3) If the board is full then return 0 as evaluation function value and none for the move

2) In Maximizing :
   check if evaluation function for this state is greater than the maximum evaluation if yes then the max evaluation value will equal to current state evaluation function and best move will equal its row and column then check if max evaluation is greater than alpha then alpha will equal to this max evaluation and if alpha greater than or equal to beta then return maximum evaluation and best move so it will stop expanded the other state since it is not important any more.

3) In Minimizing :
   check if evaluation function for this state is less than the minimum evaluation if yes then the min evaluation value will equal to current state evaluation function and best move will equal its row and column then check if min evaluation is less than beta then beta will equal to this min evaluation and if beta less than or equal to beta then return minimum evaluation and best move so it will stop expanded the other state since it is not important any more.

   Like in our example before this is the result of alpha beta pruning



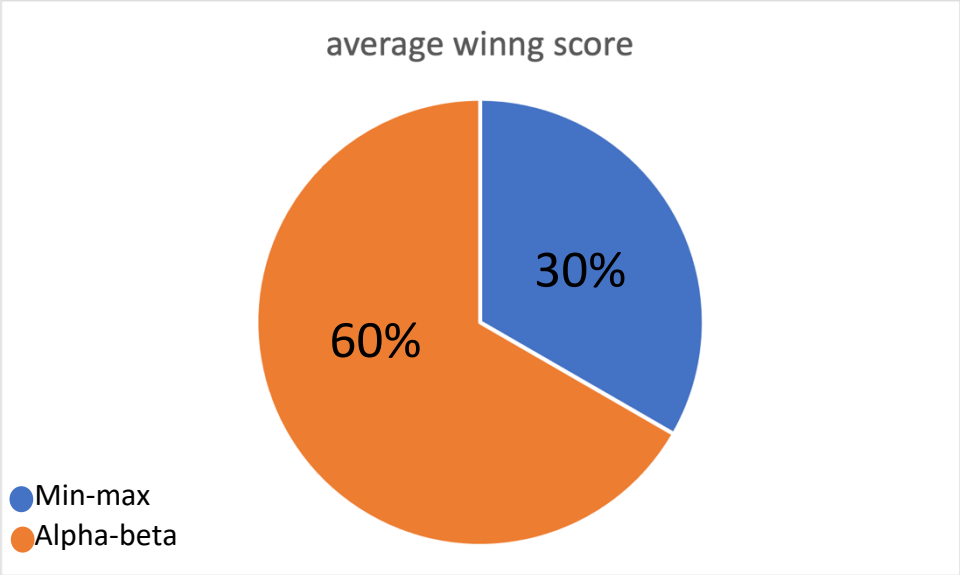
Testing :

MIN-MAX	Game result	time	Tree size	depth
1	draw	7.9144s	557487	9
2	AI win	8.1786s	558432	7
3	draw	8.0046s	557487	9
4	AI win	8.1240s	558432	7
5	draw	8.1501s	557487	9
6	AI win	8.3970s	558432	7
7	draw	8.5532s	557487	9
8	draw	8.3753s	557487	9
9	draw	8.3555s	557487	9
10	draw	8.1674s	557487	9

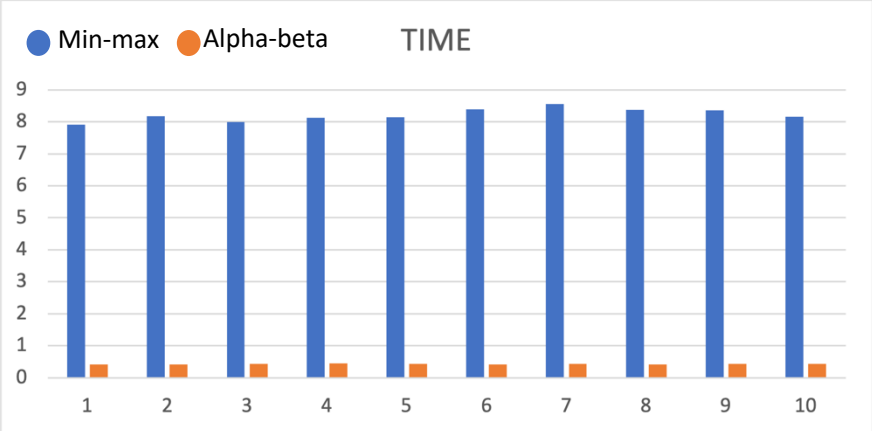
Alpha-Beta	Game result	time	Tree size	depth
1	draw	0.4155s	19212	9
2	AI win	0.4194s	18708	7
3	AI win	0.4345s	18711	7
4	draw	0.4384s	19212	9
5	AI win	0.4256s	18708	7
6	AI win	0.4161s	18840	7
7	draw	0.4307s	19212	9
8	draw	0.4161s	19212	9
9	AI win	0.4305s	18711	7
10	AI win	0.4318s	19140	7

Comparing between minmax and alpha beta:

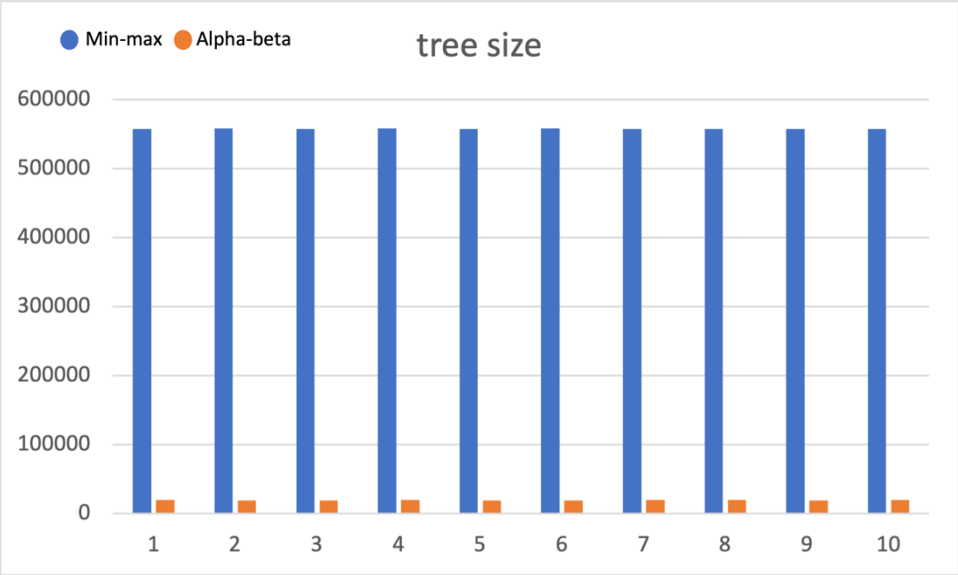
As we see In our resulting table we found that alpha-beta algorithm is more efficient than min-max algorithm since it can find the optimal solution to win more than min-max algorithm which ends up by draw.



Also the alpha-beta algorithm takes less time to expand and search the tree game rather than min-max algorithm which expands the whole tree game ,so alpha-beta algorithm is faster.



The tree size in alpha-beta algorithm is less than min-max algorithm since it did not expands all the states .



The depth is the same in both two algorithms which is starts from 0 to 9.

Reference:

- Course slides
- YouTube
- GitHub
- http://aimaterials.blogspot.com/p/blog-page_8.html
- https://cis.temple.edu/~ingargio/cis587/readings/alpha-beta.html
- https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-problem-solving-using-search-algorithms-for-beginners/
- https://course.elementsofai.com/2/3