Minimax Algorithm:

* Minimax is a decision-making algorithm, typically used in a turn-based, two player games. The goal of the algorithm is to find the optimal next move.
* In the algorithm, one player is called the maximizer (usually the agent), and the other player is a minimizer (agent’s opponent).
* It is based on the zero-sum game theory concept. In a zero-sum game: “the total utility score is divided among the players. An increase in one player's score results into the decrease in another player's score”. [1]
* It is a recursive algorithm which provides an optimal move for the agent assuming that opponent is also playing optimally **(Important point about minimax)**.
* To do this, the agent looks ahead using depth-first search and evaluates the opponents potential move by exploring the complete game tree for the move it might make and evaluates the resultant score.
* It proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.
* In the case of Connect4 exploring the complete game tree is not feasible, so we take a different approach to backtrack the score, we use depth-limited search instead of depth-first search and apply a pruning method to limit the number of nodes examined by agent to looks ahead.
* In out case we look ahead by 5-6 moves and determine and intermediate score using the “evaluate” function (pseudo code for which can be found below **<If there is a code number like figure number insert here>)**.
* Maximizer (the agent) will try to get the Maximum possible score, and Minimizer (the opponent) will try to get the minimum possible score (which is an assumption).
* Properties of Minimax:
  + **Complete** - It will find a solution (if exist), in the finite search tree.
  + **Optimal** - optimal if opponent is playing optimally.
  + **Time complexity** - As it performs DLS for the game-tree the complexity is O(bm), where b is branching factor, and m is the depth.
  + **Space Complexity** - Similar to DLS which is O(bm).
* The pruning technique used is called “alpha-beta pruning” and can be applied to any depth in the tree, it not only prunes the leaves but also entire sub-tree making the algorithm faster.
* On applying alpha-beta pruning we still get the same optimal move as a standard minimax algorithm.

REFERENCE:

[1] <https://en.wikipedia.org/wiki/Zero-sum_game>

[2] <https://en.wikipedia.org/wiki/Minimax> (not really required, just used to refer stuff)

Expectimax Algorithm:

* It is a variant of minimax algorithm used in zero-sum games, in which the outcome depends on a probability element along with the player’s skill.
* In case of the minimax algorithm one player in the maximizer (the agent) and the other player (the opponent) is a minimizer, but here the other player takes the role of an player who’s move depends on a probability element (played by chance).
* Apart from this chance element everything else is similar to that of minimax, we can apply the techniques of depth-limited search and alpha-beta pruning to improve the performance of the algorithm.