Game playing systems

In this project, we will implement tree search in order to learn to win (or at least not lose) at the game Tic-tac-toe on the classic 3x3 grid with a possibility to expand to 4x4 or more. We start with two choices of algorithms, first a classical tree search algorithm Minimax and second a Monte Carlo tree search algorithm.

Algorithm Choices

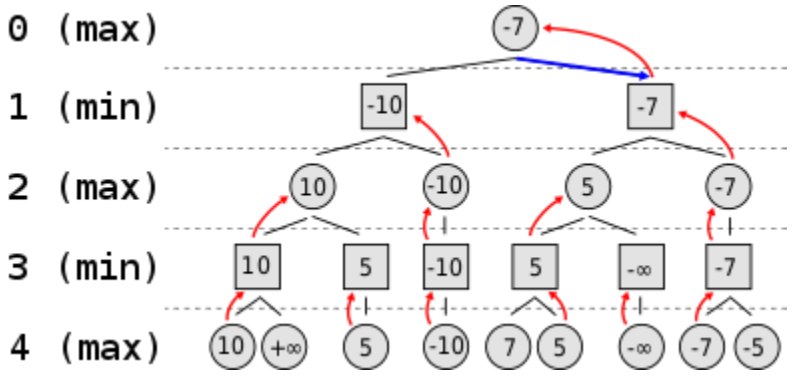
Minimax

Overview: Minimax is a backtracking algorithm which traverses the whole tree using recursion in order to find an optimal move of the state the current player is in. By using The maximin value is the highest value that the player can be sure to get without knowing the actions of the other players as a decision function.

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

Where:

- *i* is the index of the player of interest.
- $-i$ denotes all other players except player i.
- a_i is the action taken by player *i*.
- a_{-i} denotes the actions taken by all other players.
- v_i is the value function of player *i*.



A minimax tree example

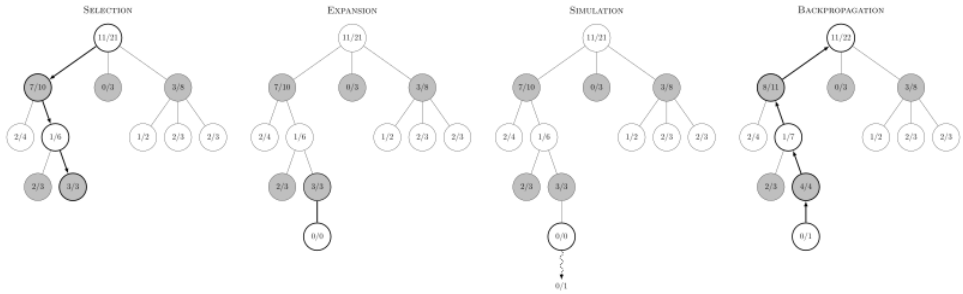
Source: https://en.wikipedia.org/wiki/Minimax

Policy assumption: The assumption of both party policy when using this algorithm is that both players in the game are playing optimally. And we will try to maximize our win with respect to the opponent's move.

Monte Carlo Tree Search (MCTS)

Overview: Monte Carlo Tree Search is a search algorithm which has been combined with neural network and used in multiple board games as well as in turn-based-strategy video games. The algorithm consists of four steps:

- Selection: Start from the current game state as the root, the algorithm will select a successive child node until a leaf node is reached. The leaf node is any node with a potential child yet simulated.
- Expansion: If the leaf node is not the terminal node (node that decided the result of the game) for either player, the algorithm will create one (or more) child nodes which is another valid move from the current game state.
- Simulation: After expansion, it will complete a random play out from the node until the game is decided.
- Backpropagation: The algorithm will update the result of the playout in all nodes on the path from the node that get expanded to the root node.



Step of Monte Carlo tree search

Source: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

The algorithm will balance exploration and exploitation using UCT , where the next action is the move which the expression has the highest value in this formula.

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$$

Where:

- w_i stands for the number of wins for the node considered after the *i*-th move
- n_i stands for the number of simulations for the node considered after the *i*-th move

- N_i stands for the total number of simulations after the *i*-th move run by the parent node of the one considered
- *c* is the exploration parameter—theoretically equal to √2; in practice usually chosen empirically

Policy assumptions: In this algorithm we have to make choices about the rollout policy of both party, in this project, for simplicity we will use a random rollout where each party will select a random move until the game is decided. Keep in mind that this method reduces the algorithm's performance, another alternative is to not naively select the move but evaluate the position first.

Comparison

In general, Minimax algorithm tends to be faster than Monte Carlo tree search (MCTS) in a smaller search space. This is because Minimax exhaustively searches the tree to find the optimal move, which is more efficient where the depth of the tree is manageable.

On the other hand, MCTS relies on random simulation and selection based on statistical outcomes, which require more iteration to converge to an optimal or near-optimal solution. In a smaller search space, MCTS may not be able to exploit its advantages fully because it does rely on randomly sampling many potential future states.

However, once the search space has been expanded, to 4x4 grid,5x5 grid or more, MCTS will perform better since it does not require the algorithm to search the whole tree to find an optimal or near-optimal solution.