Xtreme Tic Tac Toe - Assignment 1

Team 6 (HeuristicApproach) - Rishabh Singhal (20171213), Pratyush Kumar (20171168)

REPORT

1. Approach Used :

Minimax Algorithm (with alpha-beta pruning) :

Alpha-beta pruning is an optimization of minimax algorithm, used to eliminate branches of search tree (which is not required for search), using two values namely - alpha and beta. This way, the search time can be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the branch and bound class of algorithms.

- Ofcourse, it's better than minimax search algorithm from basic definition.
- It is better than breadth first search because it uses lesser memory than it.
- It is better than depth first search because it takes lesser time because it does not search a depth completely if pruning condition occurs unlike depth first search.
- Why better than **monte carlo** in this case ???
  - Alphabeta/minimax is an exact calculation. It's been proven that if a player adopts a minimax strategy, the opponent does not have any strategy that will perform better than also adopting minimax. The reason minimax engines don't play perfectly is due to having to approximate the full tree by doing evaluation estimates at non-terminal positions which are inaccurate, and pruning that is done to increase depths reached, which increases the accuracy of some nodes at the expense of others.
  - Monte Carlo depends on average experience. Inaccuracies in the evaluation get averaged out and good paths tend to get explored and played. But since it

depends on the average of a lot of playouts, it has a harder time with lines that require very precise play.

Implementation:

→ Here, we will apply minimax algorithm (alpha-beta pruning), until certain depth (to be decided by hit and trial)and then apply heuristics (described below) to account for **24 s** time limit.

We will code standard minimax algorithm with alpha-beta, and a function for calculating heuristic value for a given board state.

- Using **init** and **move** function with the same signature as given in the evaluator code.

2. Heuristic Used :

Heuristic A

This heuristic employs 4 overall *strategies*:

"Favouring" one factor over another implies that the favoured factor yields a higher utility value.

- Favour a higher total count of the number of wins across all subgames.
- Favouring -> Corners: 4 points • Center: 3 points • Remaining SmallBoards: 6 points (weighted heuristic, giving more importance to small boards having more points)
- Aim to *complete* a row, column, or diagonal.
- Ensure a zero-sum game by multiplying the utility value by a factor of -1 when computing the opponent's utility. This also enables proper use of alpha-beta pruning.

Heuristic B

Identical to Heuristic A, with the exception that 3rd point is removed in favour of higher performance (and thus greater depth limit can be used).