**Report on the Minimax-Algorithm employing the alpha beta pruning technique:**

* Heuristic: The evaluation function for a state has been considered with two variations. When the algorithm needs to evaluate the utility of a non-terminal state I have used a heuristic which counts the minimum number of stones using which the opponent player of that state can connect his pieces and win after the current player’s move. For this the common search to the other side has been implemented whose theory has been described in the redblobgames website. Here I simply employed a standard greedy search algorithm to find the shortest paths to either side. However the evaluation function is not pattern based identifying the patterns early into the game was misleading as well as time consuming. I tried to implement the dead-end detection but found that it was not improving the time utilization of the program in an effective way because of two reasons: 1. to detect time bound early in the game caused a lot of computation to find bridge connections for the opponent and 2. Towards the end game the dead end calculations are again redundant since the possible moves near the end become less and the branching factor decreases. So the alpha beta algorithm is able to perform better state utility faster than using calculations for finding dead ends. The heuristic for calculating the shortest paths also helped prevent inferior moves because during the game play the alpha beta algorithm tend to play aggressively towards forming a connection as fast as possible without playing into dead regions.
* Search Depth: The initial threshold of the search depth was 2. But since the program was running slower on a 2 level depth search, I reduced it to a 1 level depth. This caused the search to speed up quite a lot but the performance was not hampered. No, I did not employ any selective search expansion.
* Nodes Expanded: The minimax algorithm employing alpha-beta search expanded 236000 nodes on average on a depth 2 search. It expanded 18000 nodes on average on a depth 1 search. The size of the search tree was of depth 2 in a 2 ply deep search and of depth 1 on a 1 ply deep search. The maximum number of nodes added to the search tree was 18548 for 1 ply search and 243153 for 2 ply search.
* Game Statistics: The game I played with the algorithm lost on 3 occasions out of 5 on the 8\*8 board when I played using bridge formation efficiently. However when I played randomly it beat me every time. I could conclude that for a 2 ply deep search, the algorithm is strong enough. It also won every time against the standard MCTS game. I played over 7 games for that purpose. (2 on 1 ply deep and 5 on 2 ply deep.) It may be so that the MCTS game was not able to perform as nicely as the minimax algorithm employing alpha beta pruning.

**Report on the MCTS-Algorithm and RAVE-MCTS Algorithm:**

* Heuristic: The standard MCTS algorithm did not employ any form of knowledge domain application either in the backup stage or in the simulation stage. It only utilized the UCB Tree Algorithm while utilizing the tree policy because it helped select the most visited node with the highest win probability which gradually approached the actual game theoretic win value near the end of the tree.   
    
  On the other hand the RAVE MCTS algorithm was implemented following the Rapid Action Value Estimate paper given in blackboard. However I did not employ the pool RAVE modification as it was a little ambiguous to interpret. I just implemented the RAVE modification which has shown improvements in the games of GO and HAVANNAH. In this algorithm, apart from the usual statistics values, the RAVE win probability and the RAVE visit count were initialized at each RAVE-MCTS search. The former used to keep track of the win rate using a parent child complex equation and the later was used to implement. Thus the RAVE algorithm was implemented as is from the paper. It should also be noted that the RAVE algorithm implemented the notion of All Moves As First notion in the RAVE win probability which has shown to cause improvement as mentioned in the survey paper given in blackboard. Moreover this parameter has modified the UCT Bandit formula to provide the required form of the RAVE MCTS.
* Parameters:   
    
  The parameters of interest in the game was the parameter deciding how much we explore vs. how much we exploit at a given level. This was determined by the parameter ‘C’ which was multiplied with the exploration term. What I observed was that the variation of the parameter between 1 and (as mentioned in the survey paper) did not vary the performance too much. The only observation on varying the C too much was that the MCTS leaned on playing random moves which the opponent AI algorithm capitalized on to quickly build a bridge in the initial stages. So in the later stages when the MCTS could play fairly efficiently, the MCTS loses because the AI player has built a *winning connection* by that time.

The other parameter was the alpha for the RAVE MCTS. It was initially set at 300 but gradually decreased to 0 with the number of simulations. Initially it was low to leverage the variance to the advantage of the RAVE-MCTS player. When the number of simulation becomes high it is reduced to avoid the bias.

* Threshold for expansion depth: I did not give any threshold for search depth as the RAVE MCTS was fast enough to terminate to the end state through random simulation within 1 minute for a sufficient number of game simulations.
* Simulation method employed: The standard simulation method was employed for the game to terminate. As mentioned earlier I have not implemented any special techniques in the simulation side as a simple simulation enabled a better understanding of the board game in least time. Coupled with the RAVE back up policy and RAVE tree policy it proved to be a quite good tree algorithm instead of the standard MCTS.
* Evaluation function used for selection steps: The tree policy used the RAVE MCTS selection policy for selecting the best nodes as mentioned in details in the heuristics discussion.
* Expansion Policy: No, I did not expand multiple nodes at the same iteration. Hence there was not issue of memory constraints in the problem.
* Tree Policy: Here I picked the nodes with the highest overall value determined by the MCT algorithm. So the notion of most promising in terms of simulations, win statistics is a little vague to me as asked in the assignment pdf. However as asked in the assignment I used the different variations to see the results. The results were as follows: For nodes with highest win rates, the overall win rate was unsurprisingly highest. However there was not much improvement apart from against baseline players like a random move generator. Similarly a node selection with highest simulations and wins was not significantly creating a strong player apart from games against random moves hex player.
* Simulations performed per move: Here I would like to mention that the number of simulations per move was really preferred that the time bound because the time bound was kind of misleading depending on the machine I ran my code on. For example, on my laptop of 2.5GB RAM, the average time per move for a simulation stage of 3000 took 40 seconds. But when I ran MCTS on a fast 16GB RAM desktop, the time for the same simulations was pretty low and RAVE MCTS required only a little more time to run.

STATISTICS:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GAMETYPE | TIME PER MOVE | MEMORY | SEARCHDEPTH | WINS OUT OF 5 MATCHES | REMARKS |  |
| HUMAN VS ALPHABETA | 10 BYALPHA BETA | N/A | 1 PLY | HUMAN WINS 5 OUT OF 5 | AB PLAYS GREEDILY |  |
| HUMAN VS ALPHA BETA | 2 MIN 30 SECS ON AVERAGE | N/A | 2 PLY | HUMAN WINS 4 OUT OF 5 | AB PLAYS GREEDILY |  |
| ALPHA BETA VS MCTS | 2 MIN 30 SECS ON AVERAGE | N/A | 2 PLY FOR ALPHA BETA | EVENLY BOTH WINNER | AB PLAYS GREEDILY AND MCTS PLAYS RANDOMLY |  |
| HUMAN VS RAVE | HUMAN 1~2 MIN ONM AVERAGE AND  RAVE 60 SECS | N/A | NO PLY DEPTH ASSUMED | HUMAN WINS 3  RAVE WINS 2 | HUMAN MAKES BRIDGE CONNECTION  RAVE IDENTIFIES IT ***BUT CANNOT BREAK STRONG BRIDGE CONNECTIONS*** |  |
| ALPHA BETA VS RAVE | 2MIN 30 SECS ON AVERAGE BY ALPHA BETA AND 60 SECS BY RAVE | N/A | ALPHA BETA 2 PLY DEEP | RAVE WINS ALL THE MATCHES. ALMOST UNBEATABLE BECAUSE IT FORMS BRIDGE AND PREVENTS BRIDGE FORMATION BY THE OTHER PALYER | VERY EFFICIENT GAMEPLAY BY RAVE UNDER LIMITED TIME AND STOPS WEAK BRIDGE CONNECTIONS. |  |