CSC384 - Assignment 2 - Design

Morgan Tran - tranmorg - 1006331159

1. How does your program calculate the utility of each terminal state? Describe briefly.

For our situation, we are stating that terminal states are states in which a player has won the game. This means that either there are no more opponent moves, or all opponent pieces have been captured. My implementation searched for any opponent pieces remaining and/or if they were could they move - aka do a jump or a standard move for that piece and made the utility +-INF (1000000000) depending on who won in that state. Positive for red, negative for black.

2. How does your program estimate the utility of each non-terminal state? Describe your evaluation function in a few sentences.

Now if it wasn't a terminal state it would do an evaluation on the board. I followed a similar eval of the board from multiple papers, mainly this one: http://www.cs.columbia.edu/~devans/TIC/AB.html from the university of Columbia. Essentially I had 5 factors at play.
int(1000000*count + 10000*closer + 100*amount_left + (edges+centre))
Negative for black.
The count was a multiplying score for how many pieces are on the board, 3 for man 5 for king
The closer metric was taken from the above link and essentially, if a man is closer to the promotion square that is worth more, amount left is a metric to calculate a score based on who is winning currently based on the # of pieces. If red has more pieces, trades are preferred and so a higher score indicates that, and if black has more trades favour them.
Edges and centres are the score for man pieces near the edge are safer, while kings should take space and favour the centre and the edges.

The eval heuristic function is very much not optimal and will change your solutions. Reading from the papers. I used a similar method of having the metrics be weighted in such a way by digits - so the first two digits are for the count, etc

3. Does your program perform other optimizations, such as node ordering or state caching? If so, describe each optimization in a few sentences.

The program performs node ordering and that is really just it. Now to optimize for the assignment, I added in a clock timer to just submit a solution if there is one before the 2 min mark before entering and if there is no solution continue on until 4 mins. - node ordering is based on the evals search max for reds turn, other way around for min turn

To be honest, the implementation of the code is quite slow just due to the fact that I don't cache, Im not doing what everyone is doing and brute forcing it and doing an "iterative" depth searcher where they are caching things. Nor did I optimize indexing for the pieces, as I just had 2 for loops and looked through everything aka all 64 squares but checkers is only played on 32 of them.