Team Well Forked

Team Members:

1) Pratyanshu Pandey

2) Vedansh Mittal

3) Bhaskar Joshi

4) Prince Singh Tomar

5) Utkarsh Upadhyay

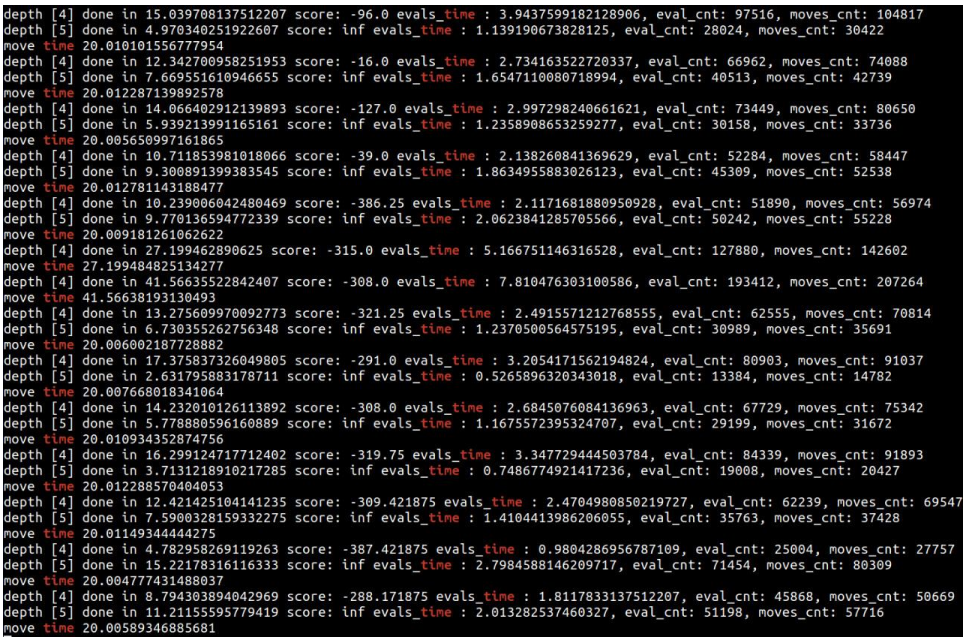
6) Pranjai Srivastava

Basic algorithm used:

Minimax algorithm in which we make a decision tree using backtracking and recursive function. The tree works like the first depth of tree picks the move which is supposed to be generating the best possible state of the board in the future for the player. The state is determined by an evaluation function which returns a high positive number if AI is winning else negative number if AI loses. The future state is determined by backtracking. At even depths the turn is of human and he will try to minimize the score of evaluation function in future states. Because chess has a large branching factor of the decision tree, we cannot evaluate tree till the last possible moves and hence we stop the tree growth at a feasible predetermined depth. Currently explained algorithm is the base of out AI but there are a lot of optimizations which help in improving the AI.

Alpha beta:

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. In minimax search algorithm the number of game states it must examine is exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning.



With alpha-beta pruning AI took about 14-15s to calculate till depth 4 in starting moves.

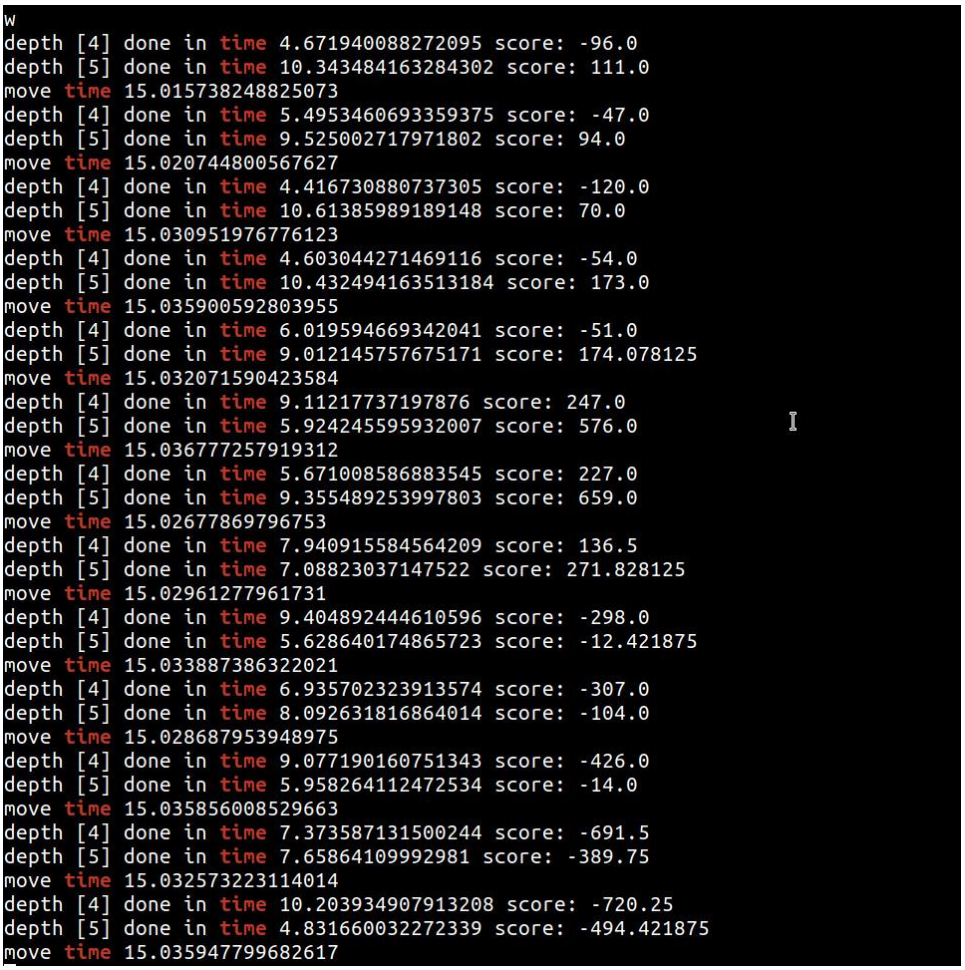
Iterative deepening:

There is a way to combine the space efficiency of depth-first search with the optimality of breadth-first methods which is iterative deepening. The idea is to perform depth limited DFS repeatedly, with an increasing depth limit, until a solution is found. Intuitively, this is a dubious idea because each repetition of depth limited DFS will duplicate uselessly all the work done by previous repetitions. But this useless duplication is not significant because a branching factor $b > 1$ implies that the number of nodes at depth k *exactly* is much greater than the total number of nodes at all depths k-1 and less. For a problem with branching factor b where the first solution is at depth k, the time complexity of iterative deepening is $O(b^k)$, and its space complexity is $O(bk)$. This means that iterative deepening simulates breadth-first search, but with only linear space complexity.

With iterative deepening we can occasionally have moves predicted till extra depths.

Multi-processing:

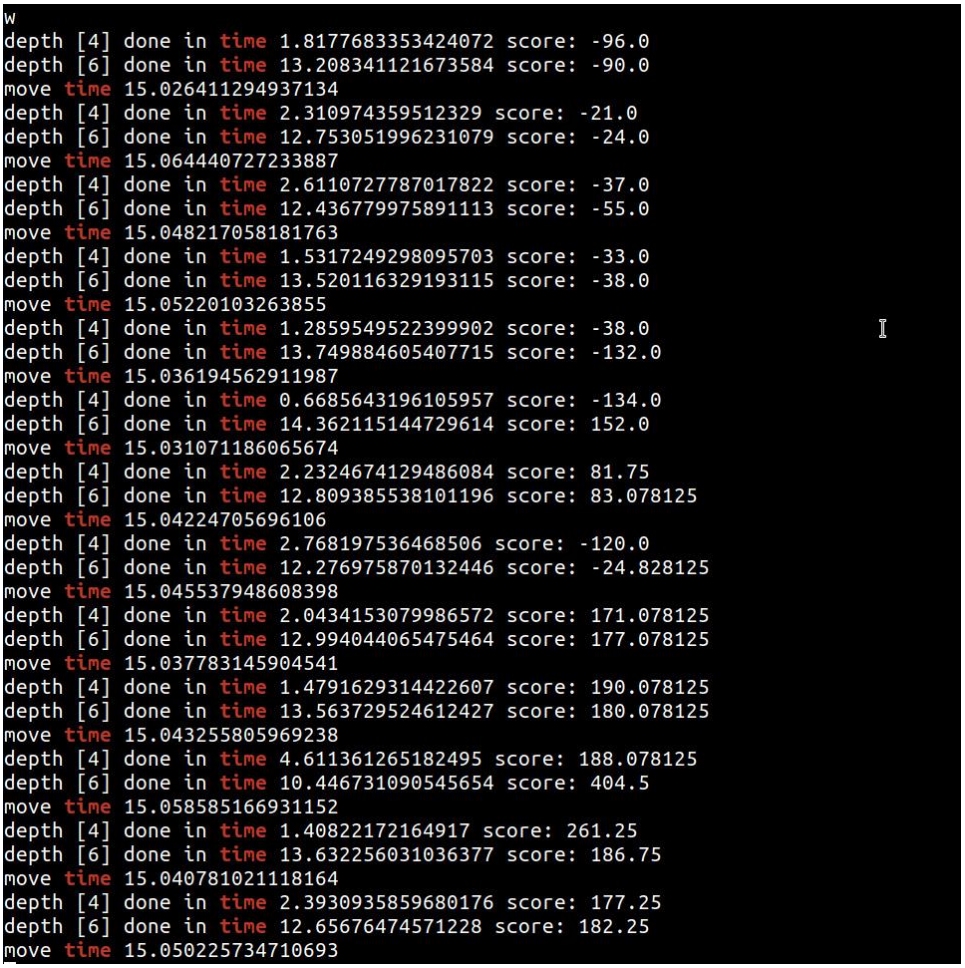
Since Python is a slow language so the time take to go for 4-6 depths was too much, hence we used multi-processing to overcome that issue. We are using 6 processes to calculate the next move. We divided the decision tree into 6 parts and calculated them and later picked the best out of the 6 returned moves.



As can be seen in the screenshot, the performance grew remarkably with depth 4 being calculated in just 4s.

Move ordering:

For the alpha-beta algorithm to perform well, the best moves need to be searched first so that we can eliminate most of the not so good moves. Hence, we approximated which moves can be best some hints like most valuable piece moved, or most valuable piece captured by least valuable piece.



After move-ordering, the speed of depth calculation grew almost 2 times.

Evaluation function:

This function is used to estimate the value or goodness of a position in the game tree. The tree is part of the search paradigm which returns a particular node and its evaluation because of alternately selecting the most favorable move for the side on move at each ply of the game

tree. The value is presumed to represent the relative probability of winning if the game tree were expanded from that node to the end of the game.

Positive points:

The AI can play from basic to intermediate chess moves. It is aware of central control and positional player strategies. It can think up to 8 depths in the end game and will search the tree till at least 4 depths.

It can play the chess960 version too as good as normal chess.

Some optimizations that could be made:

We can also implement the 50-move rule.

The current iterative deepening recreates the entire game tree instead of using previously evaluated data. This can be improved by implementing an actual game tree data structure rather than recursive function with backtracking.

An opening book can be created. This is the area where AI seems to have most of the problems.

The value of piece sqaure tables can be fine-tuned. We can also extend the evaluation to include more chess strategies.

If we could find a way that odd depths will evaluate correct score, then we can include these depths in iterative deepening too and this will improve the performance.