

# *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 our 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.

```

depth [4] done in 15.039708137512207 score: -96.0 evals_time : 3.9437599182128906, eval_cnt: 97516, moves_cnt: 104817
depth [5] done in 4.970340251922607 score: inf evals_time : 1.139190673828125, eval_cnt: 28024, moves_cnt: 30422
move time 20.010101556777954
depth [4] done in 12.342700958251953 score: -16.0 evals_time : 2.734163522720337, eval_cnt: 66962, moves_cnt: 74088
depth [5] done in 7.669551610946655 score: inf evals_time : 1.6547110080718994, eval_cnt: 40513, moves_cnt: 42739
move time 20.012287139892578
depth [4] done in 14.066402912139893 score: -127.0 evals_time : 2.997298240661621, eval_cnt: 73449, moves_cnt: 80650
depth [5] done in 5.939213991165161 score: inf evals_time : 1.2358908653259277, eval_cnt: 30158, moves_cnt: 33736
move time 20.005650997161865
depth [4] done in 10.711853981018066 score: -39.0 evals_time : 2.138260841369629, eval_cnt: 52284, moves_cnt: 58447
depth [5] done in 9.300891399383545 score: inf evals_time : 1.8634955883026123, eval_cnt: 45309, moves_cnt: 52538
move time 20.012781143188477
depth [4] done in 10.239006042480469 score: -386.25 evals_time : 2.1171681880950928, eval_cnt: 51890, moves_cnt: 56974
depth [5] done in 9.770136594772339 score: inf evals_time : 2.0623841285705566, eval_cnt: 50242, moves_cnt: 55228
move time 20.009181261062622
depth [4] done in 27.199462890625 score: -315.0 evals_time : 5.166751146316528, eval_cnt: 127880, moves_cnt: 142602
move time 27.199484825134277
depth [4] done in 41.56635522842407 score: -308.0 evals_time : 7.810476303100586, eval_cnt: 193412, moves_cnt: 207264
move time 41.56638193130493
depth [4] done in 13.275609970092773 score: -321.25 evals_time : 2.4915571212768555, eval_cnt: 62555, moves_cnt: 70814
depth [5] done in 6.730355262756348 score: inf evals_time : 1.2370500564575195, eval_cnt: 30989, moves_cnt: 35691
move time 20.006002187728882
depth [4] done in 17.375837326049805 score: -291.0 evals_time : 3.2054171562194824, eval_cnt: 80903, moves_cnt: 91037
depth [5] done in 2.631795883178711 score: inf evals_time : 0.5265896320343018, eval_cnt: 13384, moves_cnt: 14782
move time 20.007668018341064
depth [4] done in 14.232010126113892 score: -308.0 evals_time : 2.6845076084136963, eval_cnt: 67729, moves_cnt: 75342
depth [5] done in 5.778880596160889 score: inf evals_time : 1.1675572395324707, eval_cnt: 29199, moves_cnt: 31672
move time 20.010934352874756
depth [4] done in 16.299124747712402 score: -319.75 evals_time : 3.347729444503784, eval_cnt: 84339, moves_cnt: 91893
depth [5] done in 3.7131210910217285 score: inf evals_time : 0.7486774921417236, eval_cnt: 19008, moves_cnt: 20427
move time 20.012288570404053
depth [4] done in 12.421425104141235 score: -309.421875 evals_time : 2.4704980850219727, eval_cnt: 62239, moves_cnt: 69547
depth [5] done in 7.5900320159332275 score: inf evals_time : 1.4104413986206055, eval_cnt: 35763, moves_cnt: 37428
move time 20.01149344444275
depth [4] done in 4.782958269119263 score: -387.421875 evals_time : 0.9804286956787109, eval_cnt: 25004, moves_cnt: 27757
depth [5] done in 15.22178316116333 score: inf evals_time : 2.7984588146209717, eval_cnt: 71454, moves_cnt: 80309
move time 20.004777431488037
depth [4] done in 8.794303894042969 score: -288.171875 evals_time : 1.8117833137512207, eval_cnt: 45868, moves_cnt: 50669
depth [5] done in 11.21155595779419 score: inf evals_time : 2.013282537460327, eval_cnt: 51198, moves_cnt: 57716
move time 20.00589346885681

```

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.

```
W
depth [4] done in time 4.671940088272095 score: -96.0
depth [5] done in time 10.343484163284302 score: 111.0
move time 15.015738248825073
depth [4] done in time 5.4953460693359375 score: -47.0
depth [5] done in time 9.525002717971802 score: 94.0
move time 15.020744800567627
depth [4] done in time 4.416730880737305 score: -120.0
depth [5] done in time 10.61385989189148 score: 70.0
move time 15.030951976776123
depth [4] done in time 4.603044271469116 score: -54.0
depth [5] done in time 10.432494163513184 score: 173.0
move time 15.035900592803955
depth [4] done in time 6.019594669342041 score: -51.0
depth [5] done in time 9.012145757675171 score: 174.078125
move time 15.032071590423584
depth [4] done in time 9.11217737197876 score: 247.0
depth [5] done in time 5.924245595932007 score: 576.0
move time 15.036777257919312
depth [4] done in time 5.671008586883545 score: 227.0
depth [5] done in time 9.355489253997803 score: 659.0
move time 15.02677869796753
depth [4] done in time 7.940915584564209 score: 136.5
depth [5] done in time 7.08823037147522 score: 271.828125
move time 15.02961277961731
depth [4] done in time 9.404892444610596 score: -298.0
depth [5] done in time 5.628640174865723 score: -12.421875
move time 15.033887386322021
depth [4] done in time 6.935702323913574 score: -307.0
depth [5] done in time 8.092631816864014 score: -104.0
move time 15.028687953948975
depth [4] done in time 9.077190160751343 score: -426.0
depth [5] done in time 5.958264112472534 score: -14.0
move time 15.035856008529663
depth [4] done in time 7.373587131500244 score: -691.5
depth [5] done in time 7.65864109992981 score: -389.75
move time 15.032573223114014
depth [4] done in time 10.203934907913208 score: -720.25
depth [5] done in time 4.831660032272339 score: -494.421875
move time 15.035947799682617
```

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.

```
W
depth [4] done in time 1.8177683353424072 score: -96.0
depth [6] done in time 13.208341121673584 score: -90.0
move time 15.026411294937134
depth [4] done in time 2.310974359512329 score: -21.0
depth [6] done in time 12.753051996231079 score: -24.0
move time 15.064440727233887
depth [4] done in time 2.6110727787017822 score: -37.0
depth [6] done in time 12.436779975891113 score: -55.0
move time 15.048217058181763
depth [4] done in time 1.5317249298095703 score: -33.0
depth [6] done in time 13.520116329193115 score: -38.0
move time 15.05220103263855
depth [4] done in time 1.2859549522399902 score: -38.0
depth [6] done in time 13.749884605407715 score: -132.0
move time 15.036194562911987
depth [4] done in time 0.6685643196105957 score: -134.0
depth [6] done in time 14.362115144729614 score: 152.0
move time 15.031071186065674
depth [4] done in time 2.2324674129486084 score: 81.75
depth [6] done in time 12.809385538101196 score: 83.078125
move time 15.04224705696106
depth [4] done in time 2.768197536468506 score: -120.0
depth [6] done in time 12.276975870132446 score: -24.828125
move time 15.045537948608398
depth [4] done in time 2.0434153079986572 score: 171.078125
depth [6] done in time 12.994044065475464 score: 177.078125
move time 15.037783145904541
depth [4] done in time 1.4791629314422607 score: 190.078125
depth [6] done in time 13.563729524612427 score: 180.078125
move time 15.043255805969238
depth [4] done in time 4.611361265182495 score: 188.078125
depth [6] done in time 10.446731090545654 score: 404.5
move time 15.058585166931152
depth [4] done in time 1.40822172164917 score: 261.25
depth [6] done in time 13.632256031036377 score: 186.75
move time 15.040781021118164
depth [4] done in time 2.3930935859680176 score: 177.25
depth [6] done in time 12.65676474571228 score: 182.25
move time 15.050225734710693
```

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 square 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.