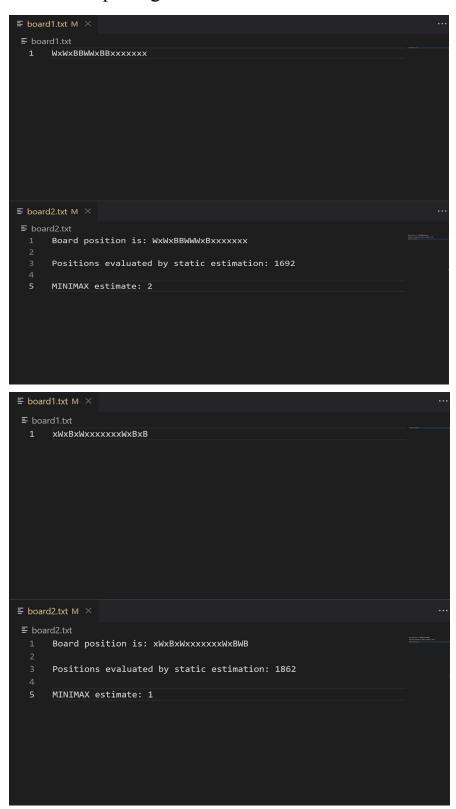
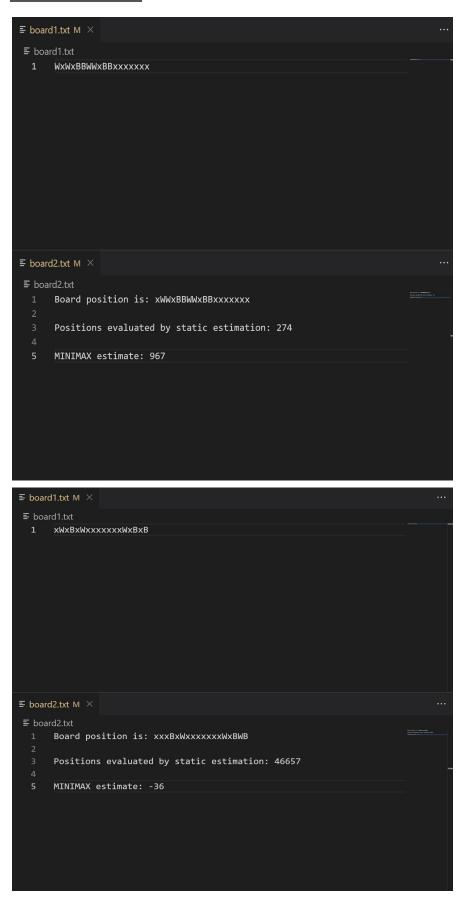
The depth for all the test cases is taken as 3.

MiniMaxOpening:



MiniMaxGame:





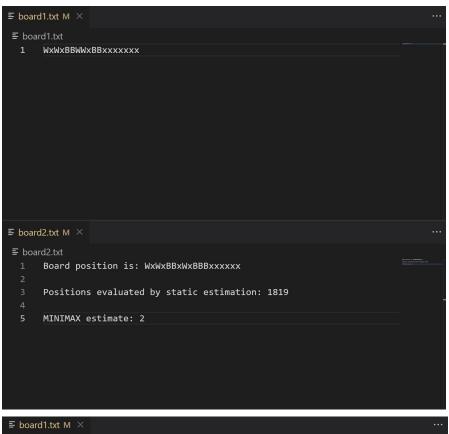
ABGame:

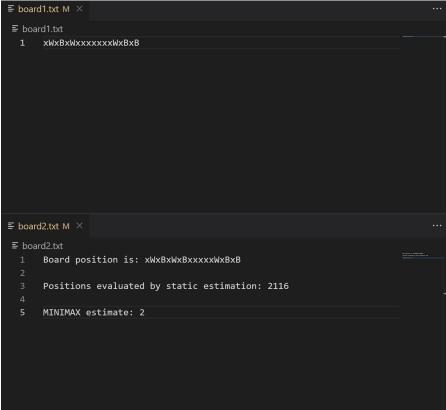


Positions evaluated by static estimation: 5432

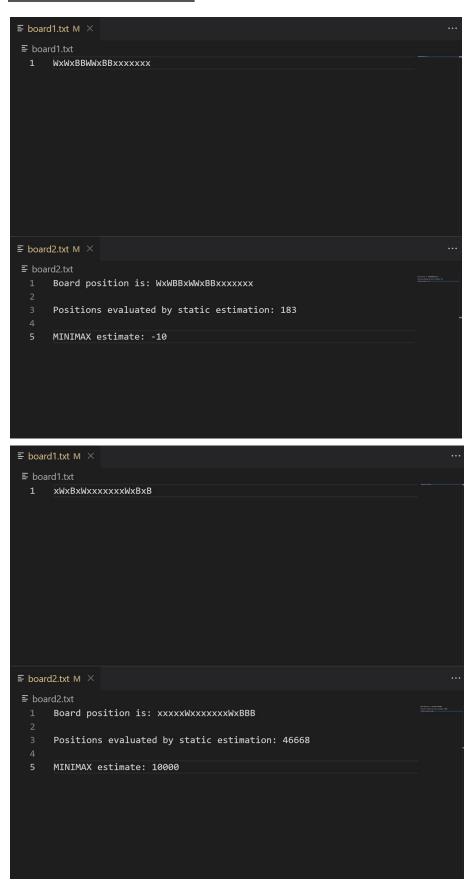
5 MINIMAX estimate: -36

MiniMaxOpeningBlack:

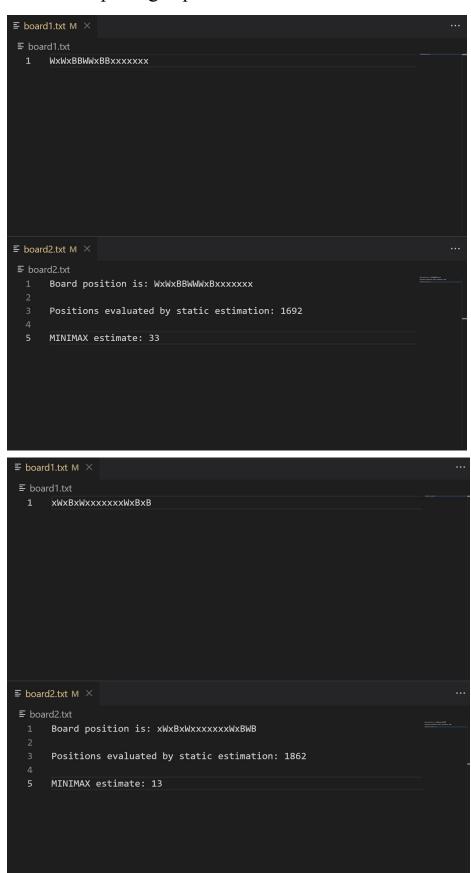




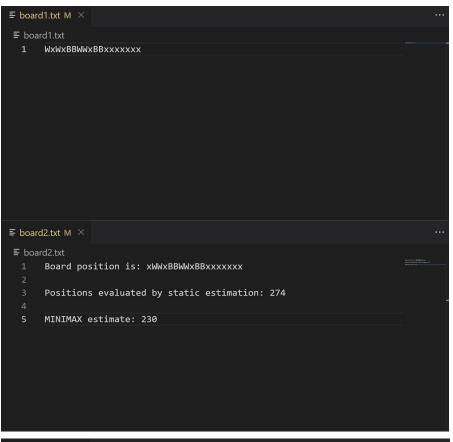
MiniMaxGameBlack:

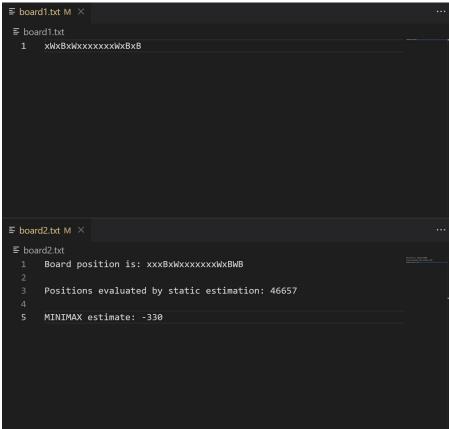


MiniMaxOpeningImproved:



MiniMaxGameImproved:





Cases where Alpha-Beta produces savings over MiniMax:

1. MiniMaxOpening vs ABOpening:

Case 1:

Input: WBWWBBWBxxxxxxxxx

Depth: 5

MinMaxOpening:

Case 2:

Input: xWxxxWxxBxxxxxBxxx

Depth: 6

MinMaxOpening:

Case 3:

Input: xWxBxWxBxWxBxWxBWB

Depth: 7

MiniMaxOpening:

2. MiniMaxGame vs ABGame:

Case 1:

Input: WBWWBBWBxxxxxxxxx

Depth: 5

MinMaxGame:

ABGame:

<u>Case 2:</u>

Input: xBBxxxWxxWxxBxBxxW

Depth: 4

MinMaxGame:

ABGame:

Case 3:

Input: xWxBxWxBxWxBxWxBWB

Depth: 5

MiniMaxGame:

ABGame:

In all the cases, the board positions and the MINIMAX estimate were the same for both MiniMax and AlphaBeta Pruning.

The difference is in the positions evaluated, and alpha beta reduces the positions to be evaluated significantly when compared to MiniMax and this savings improves much better as the depth increases.

For instance, in the case 3 of the MiniMaxOpening vs ABOpening the positions evaluated by minimax are 1717343, and the positions evaluated by alpha-beta are 36833. This case has a saving of approximately 98%.

Examples where improved evaluation function produced different moves than the standard evaluation function:

1. MiniMaxOpening vs MiniMaxOpeningImproved:

Case 1:

Input: xWBxWWBBxxBWxxWWBx

Depth: 5

MinMaxOpening:

MiniMaxOpeningImproved:

Case 2:

Input: WBxBBWxBxWxxxWWWBB

Depth: 5

MinMaxOpening:

MiniMaxOpeningImproved:

2. MiniMaxGameImproved:

Case 1:

Input: BWBxWWWBBxWBBBWBWW

Depth: 5

MinMaxGame:

MiniMaxGameImproved:

Case 2:

Input: BWBxxWxxWxBxxxBxxB

Depth: 3

MinMaxGame:

MiniMaxGameImproved:

Explanation for the improvement for the evaluation function:

We can see that the standard evaluation function is very basic and naive as it checks only the difference between the number of white pieces and black pieces in the opening and additionally the number of black moves in the midgame/endgame. I think there are a very vast number of parameters to consider to make the algorithm perform better. I think my algorithm will perform better than the standard evaluation function that was given to us as I have considered additional parameters like

- 1. The difference between the number of mills of our's and the opponent's.
- 2. If a board position made a mill in its last move.
- 3. The number of 2 piece configurations of our pieces as it can possibly result in a mill in the future.
- 4. The number of blocked pieces of the opponent as this will make that piece not to move in the midgame.

I have considered these parameters in addition to the parameter in the standard evaluation function. So, this will make my evaluation function perform at least as good as the standard evaluation function in the worst case.