Major mistakes I made

In the middle, after trying out a few different evaluation functions, I was wondering why the number of moves taken to beat PlayerNaive was about the same. There had to be something wrong with my implementation. I spent a really long time debugging and printing error messages for each function, and after hours of debugging and seeing the forum for help, I realized it is because I forgot to invert the board when coming up with successor states for the white moves (to calculate min_value). So I added the inverting of the board when coming up for successor states for white moves. Indeed, this saved me a lot, and now my bot is able to beat PlayerNaive in much fewer number of moves, and there were differences in the different evaluation functions used.

Improving efficiency of code

1. Modifying alpha-beta pruning

Initially, without using alpha-beta pruning, for the starting board, each of the black pawns in the second row is able to move to 3 possible positions. This gives us 3x6= 18 possible new states. With an initial branching factor of 18, which will increase as the first row of black pawns is able to move as well, the order in which alpha-beta pruning is implemented is important. The order in which the tree is traversed affects the number of branches that can be pruned, and hence the time spent in traversing the entire tree can be reduced significantly if we prune the tree in the most optimal direction.

At first, I implemented alpha-beta pruning from left to right (starting from the top leftmost corner to bottom rightmost corner). I ran the code a few times, and the average number of moves for each run was 25. I also took the average time taken for 1 move by using the time package in python, the average time per move was about 2.2 seconds. (I limited the depth to 4 for this)

I decided to try alpha-beta pruning from another direction instead (starting from the rightmost pawn closest to the goal row). For this, the average number of moves for each run was 19, average time per move was about 1.5 seconds.

Clearly, alpha-beta pruning from the other direction seems to give a better performance since the number of moves taken to win the game is lesser, and time taken for each move given the same limited depth of 4 was lesser.

2. Using a transposition table

After researching more about similar chess AI heuristics, I realized that using a transposition table can improve the performance of my bot by avoiding redundant computations. When searching the game tree, it is common to encounter the same board state multiple times, especially in the case of deeper searches. By storing the results of previously evaluated positions in a transposition table, one for maximum and another one for minimum values of each board, my bot's performance improved significantly as less time is used to recompute previously encountered states.

Another thing I realized: If time/depth is the limiting factor, then alpha-beta will improve performance since it is perhaps possible to increase the depth of the search. However, the problem might lie in the evaluation function. If the evaluation function is really bad, then alpha-beta might not improve the agent's performance at all. It is also important to note that alpha-beta does not actually change the outcome, only improves the speed.

Evaluation function

Testing which evaluation function is better:

To test which evaluation function works better, I implemented another PlayerAI which will use the other heuristic that I am testing the current heuristic against. Everything, including alpha-beta pruning and data structures used, was kept constant, only changing the heuristic. By pitting the two players against each other, I was able to see which evaluation function was "better" by seeing which won more times.

1. Number of black pawns left on the board
    a. I started off with the simplest heuristic I could think of. Given that a player wins as long as 1 of its pawns reaches the end row, I thought that a board with greater number of black pawns is more advantageous than a board with lesser number of black pawns.

2. Number of black pawns - number of white pawns
    a. I tried to improve the original heuristic by accounting for the number of white pawns as well. This means that we are also being "defensive" by trying to eat as many white pawns as we can, and hence a board with a lesser number of white pawns would be more advantageous.

3. Minimizing the distance between the closest black pawn to the end goal row
    a. As we are trying to get 1 black pawn to the end goal row, I tried to use the minimal distance between the closest black pawn to the end goal row. I thought that this would be a better evaluation function, but turns out not, and even lost to white.

4. Minimizing the total distance between all the black pawns to the end goal row
    a. I realized that the previous heuristic only considered 1 black pawn, and in a scenario where all the other black pawns are in the starting position and we only care about advancing the closest black pawn, it is a very "myopic" approach as we are banking on this pawn reaching the end goal. However, this does not account for very likely cases of the pawn being eaten by the whites, as there would be more whites as we approach the end goal. Hence, I tried to minimize the distance of all the black pawns to the end goal to overcome this drawback.

5. Giving different weights to each position on the board

a. I used the row number as the values. E.g. All squares in row 0 will have a weight of 0, row 1 will have a weight of 1, row 2 will have weight of 2 etc…Hence, the bot will pick the move that results in the largest total sum. After doing a deeper research into chess AI heuristics, I realised that there were many possible ways to give different positions different weights, depending on what the end goal was. For breakthrough, each pawn only has 3 possible moves for the inner columns, but only 2 possible moves for the columns 0 and last column. Hence, a pawn on the inner column would be more advantageous, and hence I gave greater weights to inner columns.
b. I also applied the same logic to account for the white pawns, so I subtracted the value of the whites from the blacks.