Taylor Del Matto
Heuristic Analysis

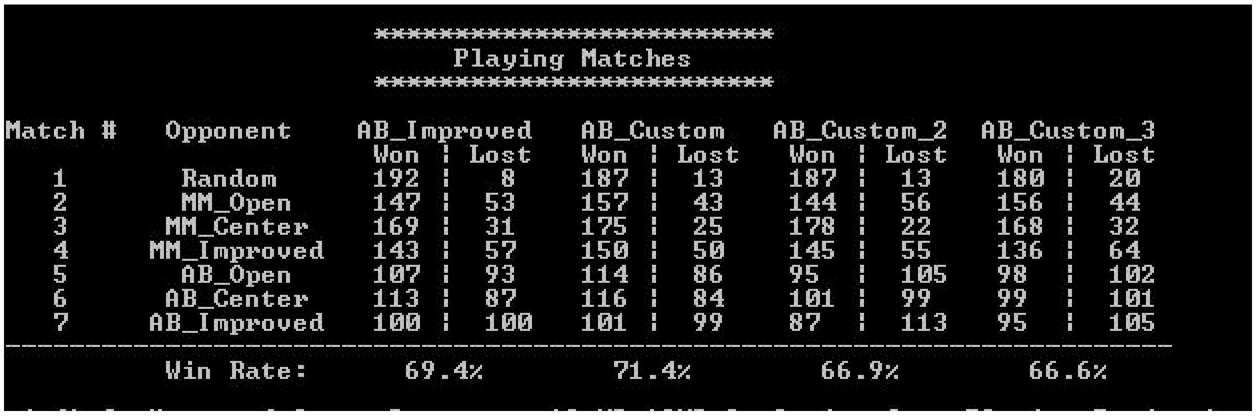
Overview

For my heuristic functions, I wasn't entirely sure where to start, because I thought improved was a very good heuristic. I knew there were a couple things that could be added to improved, but ultimately, I think that it is hard to beat improved, but rather you could add onto it, and improve it. Here's why: I think number of available moves is a great heuristic for knight isolation. The ultimate goal is winning, but number of moves just gives you more options and a less likely chance of losing. I don't see many other advanced heuristics that work much better, because the simplicity of improved also works for it, since its low computational overhead allows it to calculate further down the tree than a more sophisticated heuristic. Essentially the improved ends up giving you as many more moves possible relative to your opponent as far down into the game as it can see - that's pretty good, and partitioning the board doesn't work as well as a heuristic in knight isolation since you would need two rows of occupied squares to effectively do this. I would think the number of scenarios where the board is partitioned in such a way would be lesser in this game because of the odd fashion in which a knight moves. It's hard to get a knight to a square directly above or to the right of it. (requires multiple moves.)

It would be easy to add to improved to improve it. You could search a limited number of opening moves using the symmetry of the board, and you could add some weight to staying in the center. Although I also think that the heuristic is not as important in this game as actually working on the pruning/alphabeta. In the chess alphabeta that beat gary kasparov, they did some more complex things than just traditional alphabeta. For example, you could expand the tree of moves for your best two moves at the end of the tree at essentially no cost. Once you get to a certain level of iterative deepening, you know you will not complete the next level. Since it will take several times longer than the previous level of the tree (grows exponentially). So after a certain point, we won't be able to finish the following level of the tree….maybe after >60% of the time has been used up. If we don't finish the level of the tree, that computation is wasted. So we are left with the results from the previous level of the tree in iterative deepening. We could easily calculate further possibilities from say the best two or three choices from the previous level. Since each of these is its own new tree we could quickly iterate deeply on these moves and make much more informed choice between those moves than wasting computation time on a new level of the tree that will not be completed. It could only improve our decision, since we would be making a less informed decision on one of those moves anyways.
So it's almost a greedy alphabeta continuation between the best move options of the last level of the tree we can calculate in the allotted time.

Taylor Del Matto
Heuristic Analysis

Data



```
                         *************************
                              Playing Matches
                         *************************

Match #   Opponent    AB_Improved   AB_Custom    AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
   1       Random       192 |   8    187 |  13    187 |  13    180 |  20
   2       MM_Open      147 |  53    157 |  43    144 |  56    156 |  44
   3      MM_Center     169 |  31    175 |  25    178 |  22    168 |  32
   4     MM_Improved    143 |  57    150 |  50    145 |  55    136 |  64
   5       AB_Open      107 |  93    114 |  86    95  |  105   98  |  102
   6      AB_Center     113 |  87    116 |  84    101 |  99    99  |  101
   7     AB_Improved    100 |  100   101 |  99    87  |  113   95  |  105
--------------------------------------------------------------------------
          Win Rate:       69.4%        71.4%        66.9%        66.6%
```

My heuristics

Custom_score added onto improved by adding a weight from -.33 to .33 based on how close our player was to the center, and then it subtracted the same weight for the opponent. So it would prefer positions where player was closer to the center and opponent was further away. Choosing a max weight of +/- .66 was also important because we do not want the center score to become too important. The max influence is just over 1, so in general it would help us differentiate between two moves that were within 1 or otherwise the same. And we still add +/- infinity for a win or a loss, so theoretically this wouldn't impact a critical endgame outcome either. This was actually the last heuristic I developed.

Custom_score_2 has a score which values moving towards the center, when the opponents are close together. In other words it is trying to trap the opponent on the outside, (or at least that is the idea.) This heuristic essentially lost all games until I added the if win, inf if lose, -inf statements, and then performed at around 66% wins.

Custom_score_3 gives a score of 10 to all moves which allow the player to stay in the middle 9 squares. If not it returns the length of legal moves for the player. This heuristic has the win/lose if statements incorporated in also.

Analysis/Observations

A couple funny things that happened. Moving the win/loss check to the top of my functions significantly improved performance. Since winning is the ultimate goal, I think including this is important. It literally improved custom_score_2 from 2% to >60%. For running my own unit tests, I had set iterative deepening to not go past 10 levels deep so that I could get the program running. When I ran it like this, my custom_score heuristic demolished the improved score heuristic 9 to 1! Then upon running itunes, this went back to 6 to 4, possibly because it decreased processing power per time. Then after increasing the max depth to 25 moves, custom_score and improved went roughly even the rest of the time. With custom_score having a slight advantage.

But this is important because it shows how much processing power, and limitations like move depth and computational power can impact the effectiveness of these heuristics. Also without the if win/loss +/- inf statements many of the evaluation functions would perform much worse, because they would prioritize a move with a higher heuristic score rather than one that actually wins the game. #of total moves eventually implies win/loss so I suppose this is an exception. But since my heuristics all played off of the fact that center position is advantageous in some way, if they did not have if win or # of total moves incorporated in some fashion they would perform terribly as did my original custom_score_2.

Results/Recommendation

For the final test data, num_matches was set to 100. Custom_score had the highest win percentage of 71.4% and it beat every other algorithm it played including AB_Improved. It beat AB_Improved by 2% win percentage and it won 1 more game against it head to head. Also in previous iterations of testing where search depth was decreased, this algorithm ran much better. My guess is the center heuristic generalizes well when the search depth is not as great. So in other scenarios, this algorithm could outperform AB_Improved even more significantly. It had a higher win percentage than AB_Custom_2 (custom_score_2) and AB_Custom_3 (custom_score_3) by about 5%. AB_custom_2 and 3 had 66.9% and 66.6% win percentages respectively. Since it outperformed the other algorithms, I would reccomend custom_score which adds a center score for each more (how close it is to the center.) Custom_score_2 did win 3 more games against MM_center, but still won less against the superior AB_center than Custom_score. AB_Improved won 5 more games than custom_score against random, but lost less against every other algorithm.

Final Recommendation: custom_score with AlphaBeta.