Heuristics Analysis

The following heuristics were mainly derived from what was discussed in the lectures in addition to my personal experience as a chess player. I do have more ideas about how to improve on these heuristics, but the time limitation didn't allow me to fully explore those ideas. There are many systematic ways of determining better heuristics; for example, we could look at the distributions of moves after playing many random games to get an idea of what characteristics do the good moves have in common.

Firstly, I increased the number of matches played with each opponent from 5 to 20. The goal was to reduce the observed variability in the results. This led to a more reliable performance comparison.

The following heuristics (ordered from the simplest to the most elaborate one) were tested:

Heuristic 1

This is somewhat similar to the original idea of looking at the differences between the numbers of available moves to our player (i.e. own_moves) compared to the opponent's. Here, I use the ratio of the number of own_moves to those of the opponents. In case the number of available opponent moves is zero, I just consider the number of available own_moves as the score. Here similar to the differences, the score is increased is this ratio is higher and is decreased if it's lower. This follows the same logic albeit with a different rate. The code is shown below:

```
if game.is_winner(player):
        return float("+inf")
    if game.is_loser(player):
        return float("-inf")

    my_legal_moves = game.get_legal_moves()
    opp_legal_moves = game.get_legal_moves(game.get_opponent(player))

    if len(opp_legal_moves)>0:
        score = float(len(my_legal_moves)/len(opp_legal_moves))
    else:
        score = float(len(my_legal_moves))

    return score
```

Heuristics 2

In this case, again, I am considering the difference between the lengths of own_moves and opponent moves, penalizing all corner moves. Basically, the idea is that we don't want to make corner moves, so I

don't count them in the number of moves available as an estimate of the score. The code is shown below:

```
if game.is_winner(player):
      return float("+inf")

   if game.is_loser(player):
      return float("-inf")

   # Difference in the number of available moves for both players + an
   # additional penalty for corner moves:
   my_legal_moves = game.get_legal_moves()
   opp_legal_moves = game.get_legal_moves(game.get_opponent(player))

   my_moves_not_corner = [move for move in my_legal_moves if move!=(0,0)
                                    and move!=(game.height-1,0)
                                    and move != (0,game.width-1)
                                    and move!= (game.height-1,game.width-1)]
   opp_moves_not_corner = [move for move in opp_legal_moves if move!=(0,0)
                                    and move!=(game.height-1,0)
                                    and move != (0,game.width-1)
                                    and move!= (game.height-1,game.width-1)]

   return float(len(my_moves_not_corner) - len(opp_moves_not_corner))
```

Heuristic 3

Finally, my best heuristic, I make a few additional refinements:

- The edge moves are penalized for the own_player. This is based on the intuition that in general we would like to play closer to the centre.
- The corner moves (which are included in the edge moves) are penalized with a larger weight of 2. Again this follows the same intuition as the last heuristic that the corner moves are not desirable.

The code is shown below: (Note that I probably could've selected the edge moves more efficiently, but I ran out of time).

```
if game.is_loser(player):
      return float("-inf")

   if game.is_winner(player):
      return float("inf")
```

```
    my_moves = game.get_legal_moves(player)
    opp_moves = game.get_legal_moves(game.get_opponent(player))
    edge_moves = []
    for i in range(game.height):
        for j in range(game.width):
            if i==0 or i==game.height-1:
                if j==0 or j==game.width-1:
                    edge_moves.append((i,j))

    my_edge_moves = [move for move in my_moves if move in edge_moves]
    opp_edge_moves = [move for move in opp_moves if move in edge_moves]

    my_corner_moves =  [move for move in my_moves if move in edge_moves and move[0]==move[1]]
    opp_corner_moves = [move for move in opp_moves if move in edge_moves and move[0]==move[1]]

    score = float(len(my_moves) - len(my_edge_moves) - 2*len(my_corner_moves)
                - len(opp_moves) + len(opp_edge_moves) + 2*len(opp_corner_moves))
    return score
```

Performance comparison

As mentioned before, the number of matches in tournament.py was increased from 5 to 20. Even though this significantly increased the runtime, I believe it was essential to get better estimates of the performances. The results are summarized in the following table:

```
game_agent.py.

                    **************************
                        Playing Matches
                    **************************

 Match #    Opponent    AB_Improved    AB_Custom    AB_Custom_2    AB_Custom_3
                         Won | Lost    Won | Lost    Won | Lost    Won | Lost
    1         Random      39  |   1     39  |   1     32  |   8     36  |   4
    2         MM_Open     33  |   7     33  |   7     24  |  16     25  |  15
    3        MM_Center    40  |   0     40  |   0     36  |   4     34  |   6
    4      MM_Improved    33  |   7     32  |   8     22  |  18     20  |  20
    5         AB_Open     20  |  20     21  |  19     12  |  28     14  |  26
    6        AB_Center    28  |  12     28  |  12     31  |   9     28  |  12
    7      AB_Improved    18  |  22     26  |  14     10  |  30     15  |  25
-----------------------------------------------------------------------------
            Win Rate:      75.4%        78.2%        59.6%        61.4%
```

Figure 1 - Performance comparison

The heuristics in the table are labelled as follows:

AB_Custom: Heursitic 3
AB_Custom2: Heuristic 1
AB_Custom3: Heuristic 2

We see that my best heuristic actually outperforms all others (including the AB_Improved). Heuristic 1 and 2 are comparable in performance and are not nearly as good as AB_Improved.
I found it strange that the random player was able to beat both the AB_Improved and AB_Custom once! There might be a bug related to the terminal state identification somewhere.

Heuristic recommendation

Heuristic 3 (i.e., AB_Custom) is recommended for the following reasons:

1. As you can see from the table statistics, it out-performs all other heuristics.
2. The increased level of complexity compared to the other heuristics is negligible.
3. Given that it essentially eliminates (by heavily penalizing) exploring corner moves, and also lowers the chances of considering edge moves, it will result in more relevant nodes being explored. Therefore, in general it could lead to a faster (and therefore deeper) tree search given the time constraints.