Heuristic Analysis

Michael Richardson

Three very basic heuristics were implemented and tested in the isolation game playing agent.

Heuristic 1

The first heuristic came from research online. The common moves among the two players were removed and then the uniques moves left were compared.

```
opp = game.get_opponent(player)
player_moves = game.get_legal_moves()
opponent_moves = game.get_legal_moves(opp)

common_moves = player_moves and opponent_moves

unique_player_moves = list(set(player_moves)^set(common_moves))
unique_opponent_moves = list(set(opponent_moves)^set(common_moves))

n_own_moves_left = len(unique_player_moves)
n_opp_moves_left = len(unique_opponent_moves)

return float(n_own_moves_left - n_opp_moves_left)
```

Heuristic 2

The second heuristic came from my own version of the above one. Instead of just comparing the unique moves for each player. The opponent's moves were scaled with respect to the inverse of the number of moves so far.

```
opp = game.get_opponent(player)
player_moves = game.get_legal_moves()
opponent_moves = game.get_legal_moves(opp)

common_moves = player_moves and opponent_moves

factor = 1 / (game.move_count + 1)

unique_player_moves = list(set(player_moves)^set(common_moves))
unique_opponent_moves = list(set(opponent_moves)^set(common_moves))

n_own_moves_left = len(unique_player_moves)
n_opp_moves_left = len(unique_opponent_moves)
```

```
return float(n_own_moves_left - factor * n_opp_moves_left)
```

Heuristic 3

The third heuristic is very similar to the second. The only difference is that the players unique moves are also scale as can be seen below

```
opp = game.get_opponent(player)
player_moves = game.get_legal_moves()
opponent_moves = game.get_legal_moves(opp)

common_moves = player_moves and opponent_moves

factor = 1 / (game.move_count + 1)

unique_player_moves = list(set(player_moves)^set(common_moves))
unique_opponent_moves = list(set(opponent_moves)^set(common_moves))

n_own_moves_left = len(unique_player_moves)
n_opp_moves_left = len(unique_opponent_moves)

return float((1 - factor) * n_own_moves_left - factor * n_opp_moves_left)
```

Performance

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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     8  |  2     10  |  0     10  |  0     9  |  1
    2       MM_Open    6  |  4     10  |  0      8  |  2     8  |  2
    3      MM_Center   8  |  2      8  |  2     10  |  0     10 |  0
    4     MM_Improved  8  |  2      8  |  2      8  |  2     7  |  3
    5       AB_Open    4  |  6      2  |  8      4  |  6     4  |  6
    6      AB_Center   6  |  4      4  |  6      3  |  7     5  |  5
    7     AB_Improved  4  |  6      3  |  7      4  |  6     3  |  7
--------------------------------------------------------------------------
          Win Rate:      62.9%        64.3%        67.1%        65.7%
```

All of the custom heuristics have higher win rates than AB_Improved.

Conclusion

From the results presented above it is clear that heuristic AB_Custom_2 should be used as

- It has the highest win rate of 67.1%.
- No history of previous moves is required as it uses the current state. Therefore no extra memory requirements.
- The time taken to compute is negligible as it has very few calculations.
- It is straightforward to implement.
- It emphasises the fact that the player with more unique moves available will have a greater chance of winning.