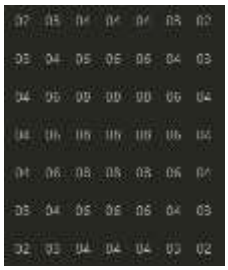
Heuristic Analysis - Knights Isolation

Krishna Kumar

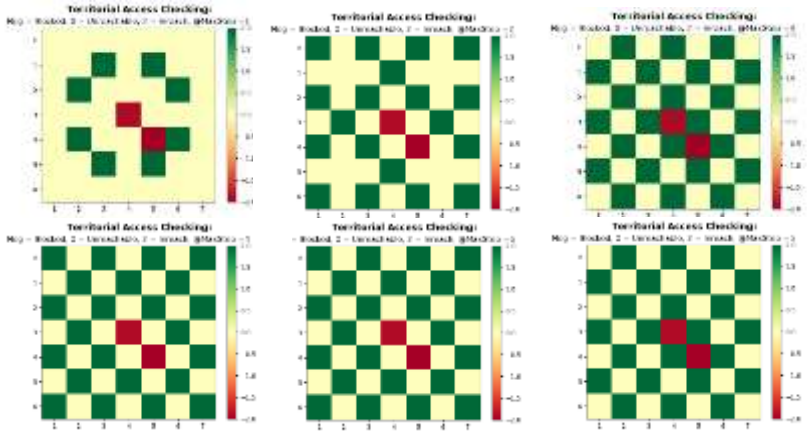
Board Visualizations.

Centrality

Max Possible Moves



Board Access and Board Parity



Formulated Heuristics

- Open Moves

```
openmoves = game.get_legal_moves(player)
```

- Open Moves Difference

```
ownmoves = game.get_legal_moves(player)
oppmoves = game.get_legal_moves(game.get_opponent(player))
score_om = len(ownmoves) - len(oppmoves)
```

- Centrality

```
ownloc = game.get_player_location(player)
opploc = game.get_player_location(game.get_opponent(player))
score_cent= distance(game, ownloc)
```

- Chase or Meander around the Opponent (18 for a quadrant)

```
score_chase = 1 / max(18, distance(game, ownloc, opploc))
```

- Open Moves Value

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

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

ownmoves = game.get_legal_moves(player)
oppmoves = game.get_legal_moves(game.get_opponent(player))

#memoizing move quality
#prioritizes picking good neighbourhoods over the openmoves difference
movesvalues = {(0, 0): 2, (1, 0): 3, (2, 0): 4, (3, 0): 4, (4, 0): 4,
               (5, 0): 3, (6, 0): 2, (0, 1): 3, (1, 1): 4, (2, 1): 6,
               (3, 1): 6, (4, 1): 6, (5, 1): 4, (6, 1): 3, (0, 2): 4,
               (1, 2): 6, (2, 2): 8, (3, 2): 8, (4, 2): 8, (5, 2): 6,
               (6, 2): 4, (0, 3): 4, (1, 3): 6, (2, 3): 8, (3, 3): 8,
               (4, 3): 8, (5, 3): 6, (6, 3): 4, (0, 4): 4, (1, 4): 6,
               (2, 4): 8, (3, 4): 8, (4, 4): 8, (5, 4): 6, (6, 4): 4,
               (0, 5): 3, (1, 5): 4, (2, 5): 6, (3, 5): 6, (4, 5): 6,
               (5, 5): 4, (6, 5): 3, (0, 6): 2, (1, 6): 3, (2, 6): 4,
               (3, 6): 4, (4, 6): 4, (5, 6): 3, (6, 6): 2
               }
#normalised move-quality
ownmovescore = float(sum([movesvalues[move] for move in ownmoves]) / (len(ownmoves) + 1))
score = len(ownmoves)*ownmovescore
return float(score)
```

- Open Moves Value Difference

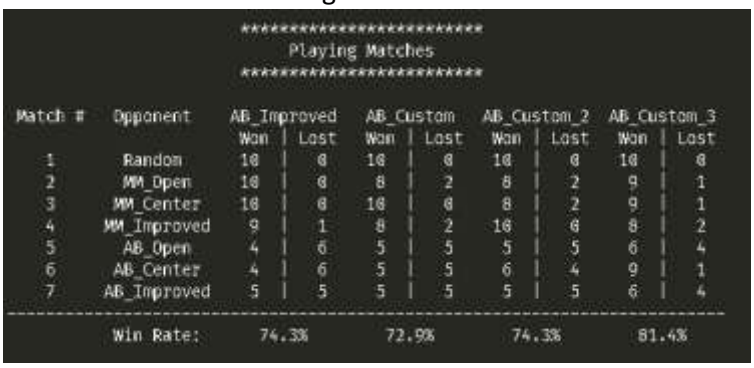
```
if game.is_loser(player):
    return float("-inf")

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

ownmoves = game.get_legal_moves(player)
oppmoves = game.get_legal_moves(game.get_opponent(player))

#memoizing move quality
#prioritizes picking good neighbourhoods over the openmoves difference
movesvalues = {(0, 0): 2, (1, 0): 3, (2, 0): 4, (3, 0): 4, (4, 0): 4,
           (5, 0): 3, (6, 0): 2, (0, 1): 3, (1, 1): 4, (2, 1): 6,
           (3, 1): 6, (4, 1): 6, (5, 1): 4, (6, 1): 3, (0, 2): 4,
           (1, 2): 6, (2, 2): 8, (3, 2): 8, (4, 2): 8, (5, 2): 6,
           (6, 2): 4, (0, 3): 4, (1, 3): 6, (2, 3): 8, (3, 3): 8,
           (4, 3): 8, (5, 3): 6, (6, 3): 4, (0, 4): 4, (1, 4): 6,
           (2, 4): 8, (3, 4): 8, (4, 4): 8, (5, 4): 6, (6, 4): 4,
           (0, 5): 3, (1, 5): 4, (2, 5): 6, (3, 5): 6, (4, 5): 6,
           (5, 5): 4, (6, 5): 3, (0, 6): 2, (1, 6): 3, (2, 6): 4,
           (3, 6): 4, (4, 6): 4, (5, 6): 3, (6, 6): 2
           }
#normalised move-quality
ownmovescore = float(sum([movesvalues[move] for move in ownmoves]) / (len(ownmoves) + 1))
oppmovescore = float(sum([movesvalues[move] for move in oppmoves]) / (len(oppmoves) + 1))
score = len(ownmoves)*ownmovescore - len(oppmoves)*oppmovescore
return float(score)
```

Analysis: I chose to keep all the heuristics lightweight, and simplistic, so as to allow our search to progress down the tree as far as it can, and avoid search timeouts that would result from using time expensive heuristic functions.

The heuristics chosen for the submission after some initial testing were:

- Custom score 1: openmoves_difference
- Custom score 2: opemoves_difference * score_cent()
- Custom score 3: openmoves_value_difference()

Results for the chosen configuration:



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

Match #   Opponent      AB_Improved   AB_Custom   AB_Custom_2   AB_Custom_3
                        Won | Lost    Won | Lost   Won | Lost   Won | Lost
    1       Random      10  |  0      10  |  0     10  |  0     10  |  0
    2       MM_Open     10  |  0       8  |  2      8  |  2      9  |  1
    3      MM_Center    10  |  0      10  |  0      8  |  2      9  |  1
    4     MM_Improved    9  |  1       8  |  2     10  |  0      8  |  2
    5       AB_Open      4  |  6       5  |  5      5  |  5      6  |  4
    6      AB_Center     4  |  6       5  |  5      6  |  4      9  |  1
    7     AB_Improved    5  |  5       5  |  5      5  |  5      6  |  4
--------------------------------------------------------------------------
          Win Rate:      74.3%        72.9%        74.3%        81.4%
```

Conclusion:

Out of all the simple / aggressive heuristics chosen, the custom_score_3 has the best performance, as it benefits from knowing the relative values of all tiles in the game, in addition to the open moves difference.