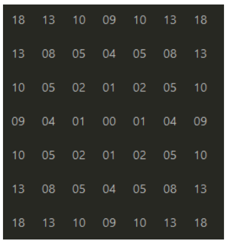
Heuristic Analysis - Knights Isolation

Krishna Kumar

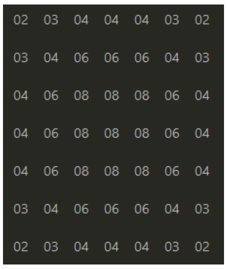
Board Visualizations.

Centrality



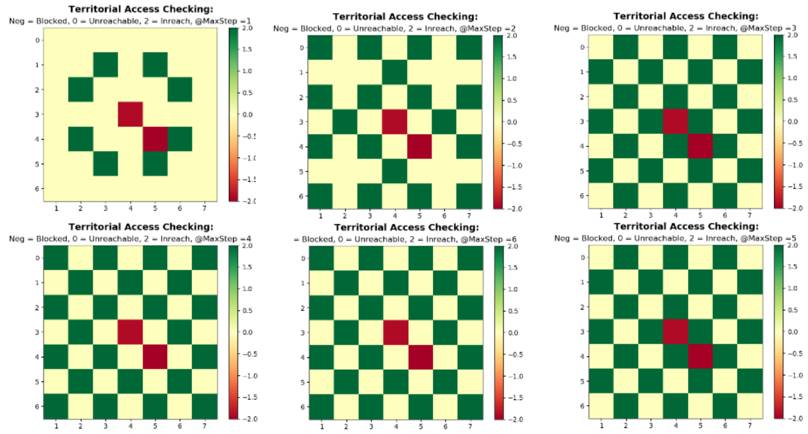
18	13	10	09	10	13	18
13	08	05	04	05	08	13
10	05	02	01	02	05	10
09	04	01	00	01	04	09
10	05	02	01	02	05	10
13	08	05	04	05	08	13
18	13	10	09	10	13	18

Max Possible Moves



02	03	04	04	04	03	02
03	04	06	06	06	04	03
04	06	08	08	08	06	04
04	06	08	08	08	06	04
04	06	08	08	08	06	04
03	04	06	06	06	04	03
02	03	04	04	04	03	02

Board Access / 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 = min(18, distance(game, ownloc, opploc))/18 #penalty
```

- 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:

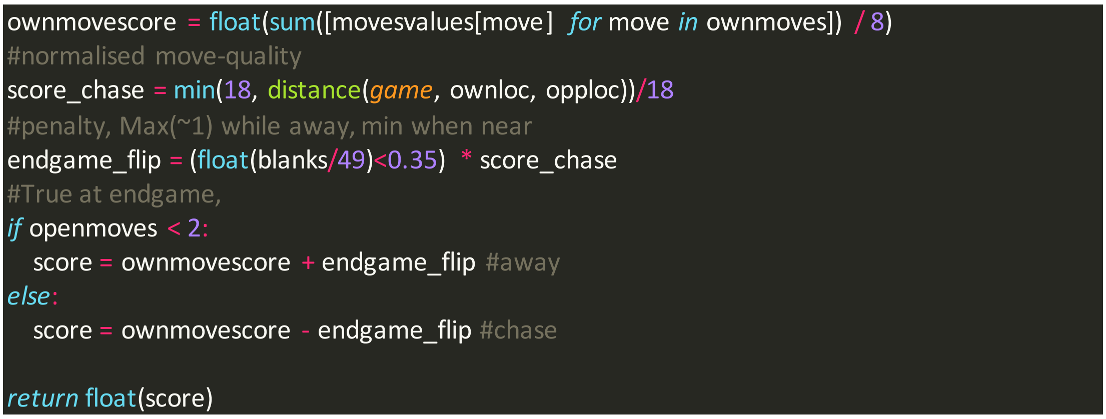
- <u>**Custom score 1**</u>: ***openmoves_difference***, *like AB_improved, this is our control to account for the randomness in the opening moves and the actual test run.*
- <u>**Custom score 2:**</u> **opemoves_difference * score_cent(),** *improves on custom_1 by adding a centrality score multiplier to the score in custom_score_1.*
- <u>**Custom score 3**</u>: **openmoves_value_difference**, *uses a dictionary, with priority values assigned to positions based on their open-ness, a higher value (e.g. 8 for the centre v/s 2 for a corner)*

Extra Heuristics:

<u>**Custom_Score 4:**</u> **openmoves_value with score_chase_:** *like custom_score_3 but multiplied with a score modifier that tracks for endgame (65%) and closeness to opponent.*

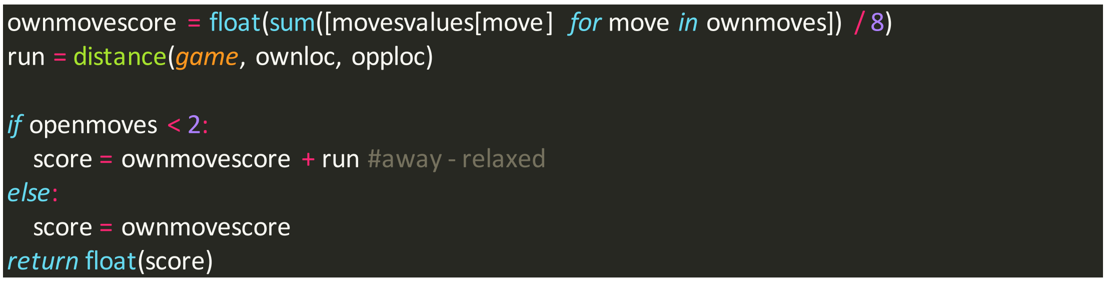
```
ownmovescore = float(sum([movesvalues[move] for move in ownmoves]) / 8) #normalised
move-quality
score_chase = min(18, distance(game, ownloc, opploc))/18
#penalty, Max(~1) while away, min when near
endgame_flip = (float(blanks/49)<0.35) * score_chase
#True at endgame,
score = ownmovescore - endgame_flip #stay close
return float(score)
```

Custom_Score 5: openmoves_value with -score_chase_: *like custom_score_3 but multiplied with a score modifier that conditionally tracks for endgame (65%) and distance to opponent.*



```
ownmovescore = float(sum([movesvalues[move]  for move in ownmoves])  / 8)
#normalised move-quality
score_chase = min(18, distance(game, ownloc, opploc))/18
#penalty, Max(~1) while away, min when near
endgame_flip = (float(blanks/49)<0.35)  * score_chase
#True at endgame,
if openmoves < 2:
   score = ownmovescore + endgame_flip #away
else:
   score = ownmovescore - endgame_flip #chase

return float(score)
```

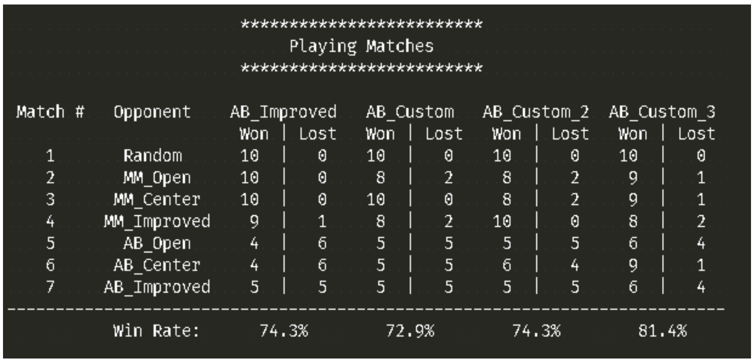
Custom_Score 6: openmoves_value with -conditional_distance_check_: *like custom_score_3 but with evasion at smaller openmoves.*



```
ownmovescore = float(sum([movesvalues[move]  for move in ownmoves])  / 8)
run = distance(game, ownloc, opploc)

if openmoves < 2:
   score = ownmovescore + run #away - relaxed
else:
   score = ownmovescore
return float(score)
```

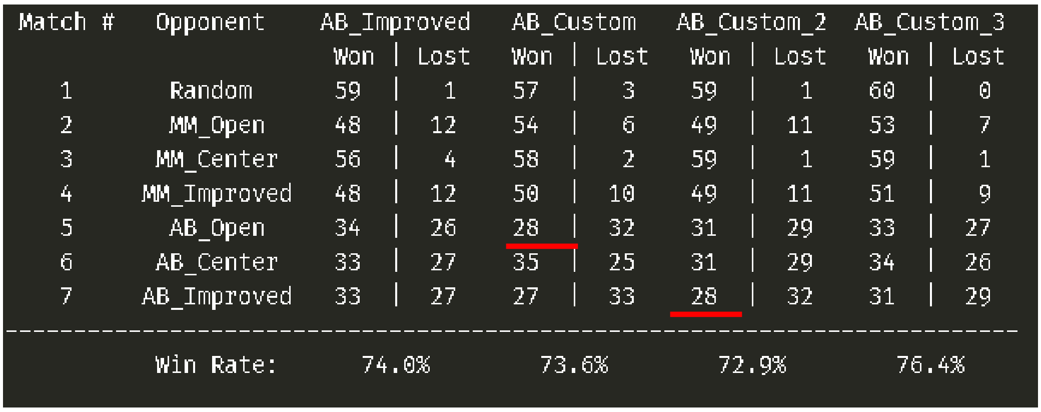
Expecting perfect-play or better in all cases (>=50 %)

Results for the chosen Original Set (1,2,3):



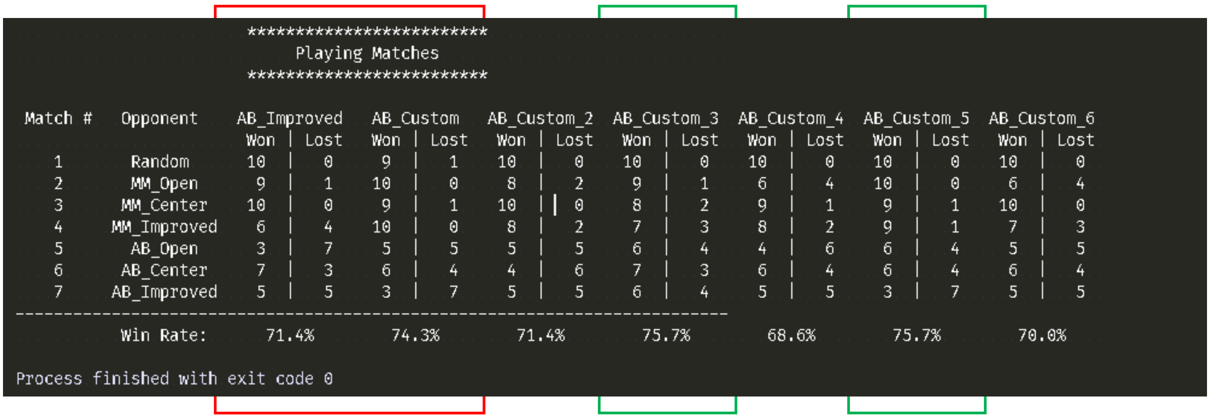
```
                         *************************
                              Playing Matches
                         *************************

 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%
```

Extended results: Original Set (Num_Matches = 30, numproc = 8, timeout=150ms.)



```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      59  |   1    57  |   3    59  |   1    60  |   0
    2       MM_Open     48  |  12    54  |   6    49  |  11    53  |   7
    3      MM_Center    56  |   4    58  |   2    59  |   1    59  |   1
    4     MM_Improved   48  |  12    50  |  10    49  |  11    51  |   9
    5       AB_Open     34  |  26    28  |  32    31  |  29    33  |  27
    6      AB_Center    33  |  27    35  |  25    31  |  29    34  |  26
    7     AB_Improved   33  |  27    27  |  33    28  |  32    31  |  29
--------------------------------------------------------------------------
           Win Rate:      74.0%        73.6%        72.9%        76.4%
```

Round 2: With Extra Heuristics



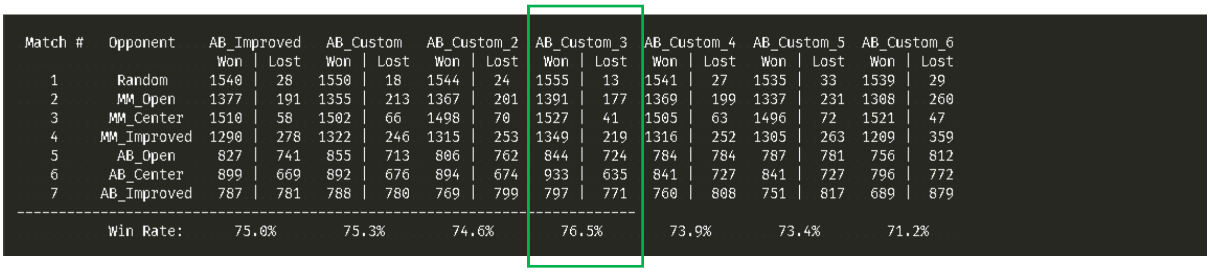
```
                          *************************
                               Playing Matches
                          *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4  AB_Custom_5  AB_Custom_6
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      10  |   0     9  |   1    10  |   0    10  |   0    10  |   0    10  |   0    10  |   0
    2       MM_Open      9  |   1    10  |   0     8  |   2     9  |   1     6  |   4    10  |   0     6  |   4
    3      MM_Center    10  |   0     9  |   1    10  |   0     8  |   2     9  |   1     9  |   1    10  |   0
    4     MM_Improved    6  |   4    10  |   0     8  |   2     7  |   3     8  |   2     9  |   1     7  |   3
    5       AB_Open      3  |   7     5  |   5     5  |   5     6  |   4     4  |   6     6  |   4     5  |   5
    6      AB_Center     7  |   3     6  |   4     4  |   6     7  |   3     6  |   4     6  |   4     6  |   4
    7     AB_Improved    5  |   5     3  |   7     5  |   5     6  |   4     5  |   5     3  |   7     5  |   5
--------------------------------------------------------------------------
           Win Rate:      71.4%        74.3%        71.4%        75.7%        68.6%        75.7%        70.0%

Process finished with exit code 0
```

Control

*

*

Extended Tests: Round 2: (Num_Matches = 1568, numproc = 8, timeout=150ms.)



```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4  AB_Custom_5  AB_Custom_6
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     1540 |  28   1550 |  18   1544 |  24   1555 |  13   1541 |  27   1535 |  33   1539 |  29
    2       MM_Open    1377 |  191  1355 |  213  1367 |  201  1391 |  177  1369 |  199  1337 |  231  1308 |  260
    3      MM_Center   1510 |  58   1502 |  66   1498 |  70   1527 |  41   1505 |  63   1496 |  72   1521 |  47
    4     MM_Improved  1290 |  278  1322 |  246  1315 |  253  1349 |  219  1316 |  252  1305 |  263  1209 |  359
    5       AB_Open     827 |  741   855 |  713   806 |  762   844 |  724   784 |  784   787 |  781   756 |  812
    6      AB_Center    899 |  669   892 |  676   894 |  674   933 |  635   841 |  727   841 |  727   796 |  772
    7     AB_Improved   787 |  781   788 |  780   769 |  799   797 |  771   760 |  808   751 |  817   689 |  879
-----------------------------------------------------------------------------
           Win Rate:      75.0%        75.3%        74.6%        76.5%        73.9%        73.4%        71.2%
```

Conclusion:

Out of all the heuristics that were implemented, custom_score_3 heuristic has the most consistent performance, as it benefits from knowing the relative values of all tiles in the game along with the difference in the number of available moves it has v/s it’s opponent, and since the values are stored in a dictionary and are not computed every time the evaluation is run, the search isn’t slowed down as much.