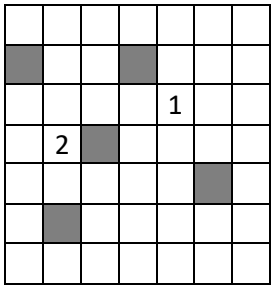
Heuristic Analysis

The aim of this report is to assess the performance of the three heuristics implemented for project completion. They basically make use of one of two major function which get distances from any location to every place in the board and get the maximum reachable depth from any location in board respectively.

Distances

Bearing in mind the particular move pattern of knights, which is an L shape in all directions, calculating distances is not as straight as getting the Euclidean or Manhattan distance. Positions seeming to be close to the starting point are in fact further away. In order to meet this problem, breadth-first search was implemented, it starts looking for closest reachable locations and continues through the tree's levels until no more locations are available or reachable.



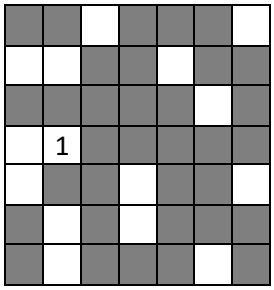
Current state

2	3	4	1	2	1	∞
∞	2	1	∞	3	2	1
2	3	2	3	0	3	2
3	∞	∞	2	3	2	1
4	3	4	1	2	∞	4
3	∞	3	4	3	2	3
4	5	2	3	2	3	4

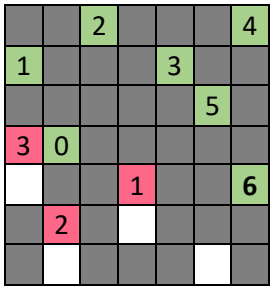
Distances from player 1 location

Maximum depth

Same consideration is taken into account when maximum depth is calculated, in this case though, board is explored using depth-first search, keeping track of all possible knight's moves. On grounds of efficiency, the search is stopped once a depth of 6 is reached, otherwise it could take a long time to finish.



Current state



Maximum reachable depth from 1 is

Heuristics

- **Custom Score:** This function uses distances from each player location and returns the difference between the numbers of first-reachable positions of each player.

```
score = 0
# Count how many positions are closer to each player
for i, own_dist in enumerate(own_distances):
    opp_dist = opp_distances[i]
    if own_dist < opp_dist:
        score += 1
    elif own_dist > opp_dist:
        score -= 1
return float (score)
```

- **Custom Score 2:** This function sums up the maximum reachable depth from each legal move of each player and returns the difference.

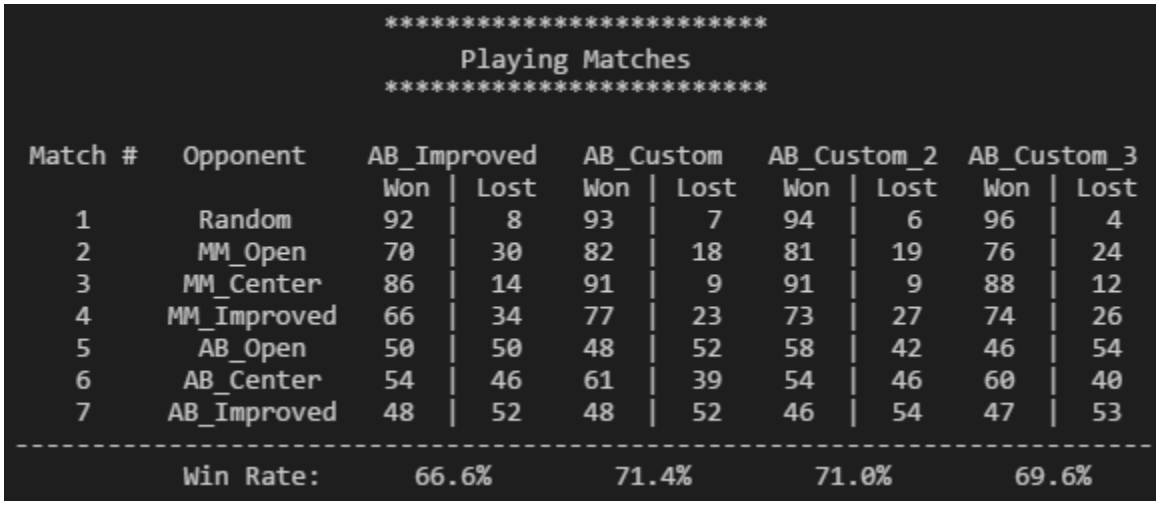
```
accum = 0
# Get max reachable depth from each player's legal move and sum up
for move in own_moves:
    accum += (get_max_depth(game, move)) + 1

# Get max reachable depth from each opponents's
# legal move and substrac from player' sum
for move in opp_moves:
    accum -= (get_max_depth(game, move)) + 1

return float (accum)
```

- **Custom Score 3:** Same as before, it sums up the maximum reachable depth from each legal move of each player but before calculating difference, it divides the sum by the number of legal moves of each player (arithmetic mean).

```
# Sum up max reachable depth from each player's legal move
own_accum = 0
for move in own_moves:
    own_accum += get_max_depth(game, move) + 1

# Sum up max reachable depth from each opponent's legal move
opp_accum = 0
for move in opp_moves:
    opp_accum += get_max_depth(game, move) + 1

# Calculate the max reachable depth mean for each player
own_mean = own_accum / len (own_moves) if len(own_moves) else 0
opp_mean = opp_accum / len (opp_moves) if len(opp_moves) else 0

return float (own_mean - opp_mean)
```

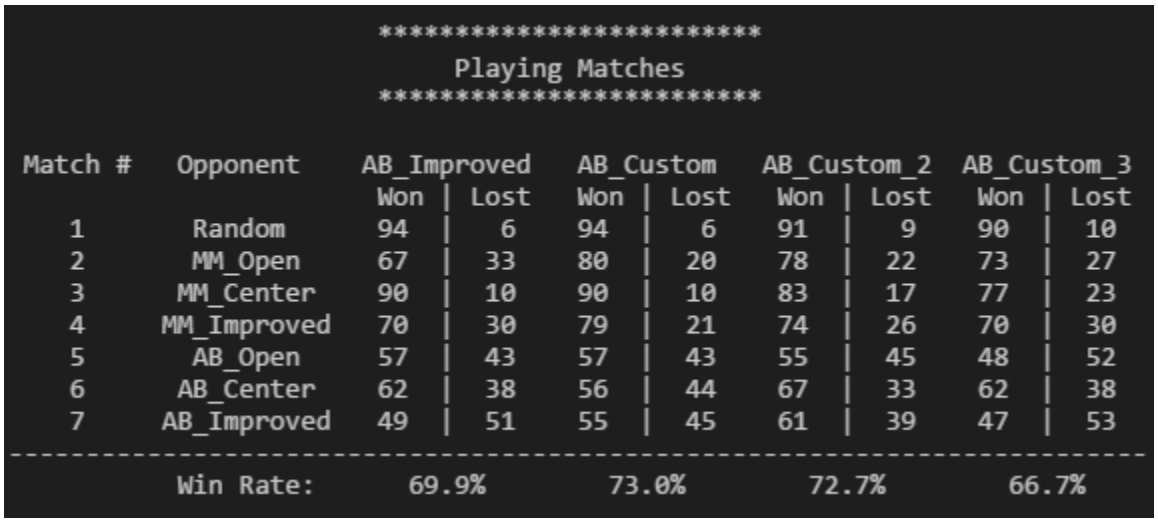
Results

200 games were played between agents in order to determine which one is better overall, games were played in two rounds just as images below show.



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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     92 |   8    93 |   7    94 |   6    96 |   4
    2       MM_Open    70 |  30    82 |  18    81 |  19    76 |  24
    3      MM_Center   86 |  14    91 |   9    91 |   9    88 |  12
    4     MM_Improved  66 |  34    77 |  23    73 |  27    74 |  26
    5       AB_Open    50 |  50    48 |  52    58 |  42    46 |  54
    6      AB_Center   54 |  46    61 |  39    54 |  46    60 |  40
    7     AB_Improved  48 |  52    48 |  52    46 |  54    47 |  53
----------------------------------------------------------------------
          Win Rate:      66.6%        71.4%        71.0%        69.6%
```

First round of games



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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     94 |   6    94 |   6    91 |   9    90 |  10
    2       MM_Open    67 |  33    80 |  20    78 |  22    73 |  27
    3      MM_Center   90 |  10    90 |  10    83 |  17    77 |  23
    4     MM_Improved  70 |  30    79 |  21    74 |  26    70 |  30
    5       AB_Open    57 |  43    57 |  43    55 |  45    48 |  52
    6      AB_Center   62 |  38    56 |  44    67 |  33    62 |  38
    7     AB_Improved  49 |  51    55 |  45    61 |  39    47 |  53
----------------------------------------------------------------------
          Win Rate:      69.9%        73.0%        72.7%        66.7%
```

Second round of games

Based on games played, first heuristic (AB_Custom) turns out to be the best, however it's not clearly decisive when it comes to play against Alpha-Beta players especially against AB_Open and AB_Improved where its performance is roughly the same. It could be said that second heuristic performs a little better against these agents however it is not totally clear, more games would have to be played to assess better its success. A combination between both heuristics may be good, though, especial attention would have to be put since it could drastically affect the overall performance.