Heuristic Analysis

By: Jordan L. Carson

Date: 2018-02-03

Custom Score

Analysis

The first custom_score functions goal is to obtain the heuristic value as a difference of number of legal moves available for player and its opponent. The intuition behind this is to subtract the number of moves between each player.

The code can be seen below:

```
# getting opponent
opponent = game.get_opponent(player)
# obtaining locations
playerMoves  = game.get_legal_moves(player)
opponentMoves = game.get_legal_moves(opponent)

# returning heuristic
return float(len(playerMoves) - len(opponentMoves))
```

Custom Score 2

The second custom score is very similar to the first heuristic calculation, however this is set to calculate the heuristic value as a difference of number of legal moves available for the player and its opponent plus add an incentive to take central location.

See the code below:

```
# Calculating center position of the game board
mid_w, mid_h = game.height // 2 + 1, game.width // 2 + 1
center_location = (mid_w, mid_h)

# getting players location
player_location = game.get_player_location(player)
# checking if player is the center location # returning heuristic1 with incentive
if center_location == player_location:
    return custom_score(game, player)+100
else:# returning heuristic1
    return custom_score(game, player)
```

Custom Score 3

The third custom heuristic function is based on the assumption that during later stages of the game, local properties become more important. Thus, a conditional is written that explicitly calculates the size of the board and the number of blank spaces if this is greater than 30% of the board size we then filter out moves that are within the local area

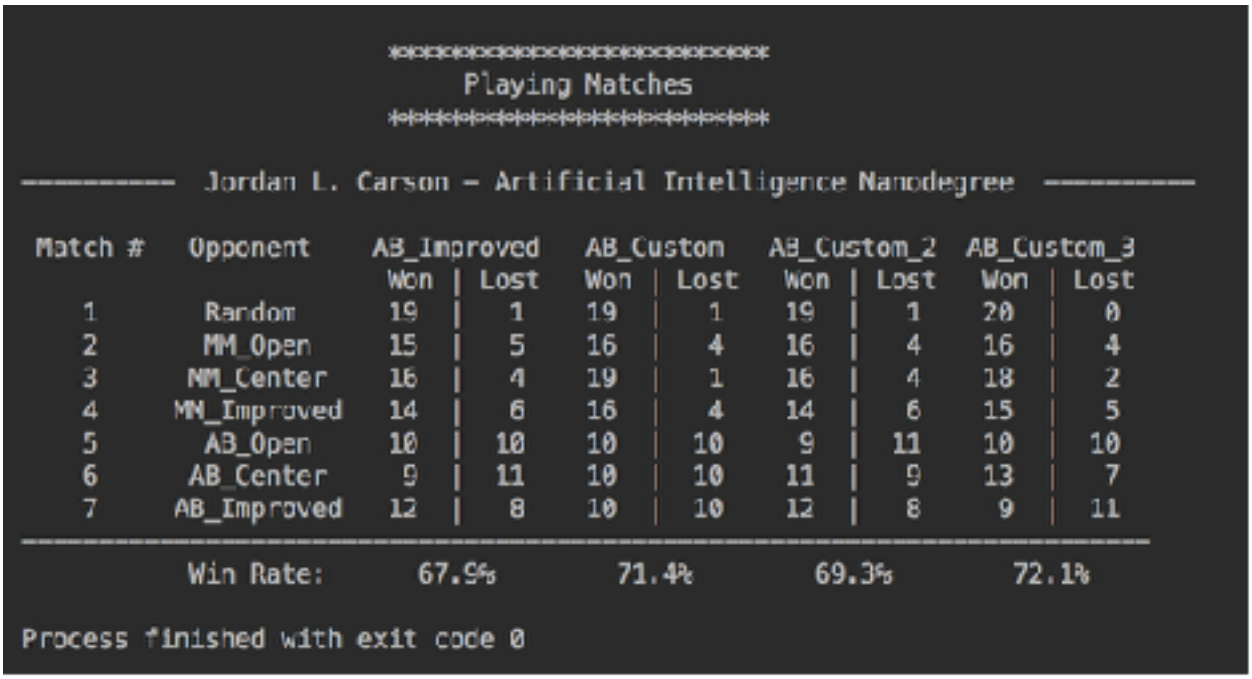
See the code below.

Proximity Helper Function

```
def proximity(location1, location2):
    """Function return extra score as function of proximity between two positions...."""
    return abs(location1[0]-location2[0])+abs(location1[1]-location2[1])
```

```
    opponent = game.get_opponent(player)
    player_location = game.get_player_location(player)
    opponent_location = game.get_player_location(opponent)
    playerMoves = game.get_legal_moves(player)
    opponentMoves = game.get_legal_moves(opponent)
    blank_spaces = game.get_blank_spaces()

    board_size = game.width * game.height

    # size of local area
    localArea = (game.width + game.height)/4

    # condition that corresponds to later stages of the game
    if board_size - len(blank_spaces) > float(0.3 * board_size):
        # filtering out moves that are within local area
        playerMoves = [move for move in playerMoves if proximity(player_location, move) <= localArea]
        opponentMoves = [move for move in opponentMoves if proximity(opponent_location, move) <= localArea]

    return float(len(playerMoves) - len(opponentMoves))
```

Custom Heuristic Results



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

---------- Jordan L. Carson - Artificial Intelligence Nanodegree ----------

Match #   Opponent    AB_Improved     AB_Custon     AB_Custom_2   AB_Custom_3
                      Won | Lost    Won | Lost    Won | Lost    Won | Lost
    1       Random     19 |    1     19 |    1     19 |    1     20 |    0
    2      MM_Open     15 |    5     16 |    4     16 |    4     16 |    4
    3    MM_Center     16 |    4     19 |    1     16 |    4     18 |    2
    4   MM_Improved    14 |    6     16 |    4     14 |    6     15 |    5
    5      AB_Open     10 |   10     10 |   10      9 |   11     10 |   10
    6    AB_Center      9 |   11     10 |   10     11 |    9     13 |    7
    7   AB_Improved    12 |    8     10 |   10     12 |    8      9 |   11
--------------------------------------------------------------------------
          Win Rate:      67.9%        71.4%        69.3%        72.1%

Process finished with exit code 0
```

Conclusion

In conclusion, we can see from the test results that the third custom heuristic function (custom_3) works best, however the first heuristic has relatively the same results. This is due to if the player has more available moves than his opponent, the chances to win are higher. In the third heuristic, we can see that in later stages of the game, the moves are more important thus we use the same function as the first, however we only grant the results in the later portion of the game. Thus it can be seen that the position on the game board is very important. Considering the results in the above screenshot, I would recommend using the AB_Custom_3 heuristic function because the winning rate is the highest, is easy to implement and understand. The second is the first custom heuristic, AB_Custom, as this is a over-simplified version of the third heuristic we selected as our selected heuristic model.