Name: Samuel Pun Chi Ho	UdaCity A.I. Nanodegree Term 1

Research Summary of Heuristic Score for Isolation Game Agent

While exhaustive search could give a definitive answer on which action a rational player should take, it is computationally expensive, if not impossible to search all the options given the number of searches (branch ^{depth}) needed to conduct. The goal of this research is to find an optimal heuristic evaluation score that could effectively and efficiently evaluate the likelihood of winning the game from the perspective of an active player.

A good choice of heuristic factor should fulfill the following requirements:

- easy to compute (without taking too much resource)
- winning rate
- able to produce consistent predictive ability across different depth in the tree
- easily configurable (tuning)

From the lecture we know that there a few good potential candidates for this winning approximation tasks:

1) No of moves (active player) vs no of moves (opponent)
2) The ability to take over center position.

Hence we assumed that the final evaluation score would be a linear function of these 2 factors.

No of Moves:

The no of moves is relatively easy to compute. In this particular assignment, the game object also provides a method to return the no of moves given a player, hence the computation is relatively straightforward.

As regarding the winning rate, with some simple tests, we found that for both Minimax VS Minimax player and Alphabeta vs Alphabeta player, No of moves as a score of direction leads to over 50% of winning rate (figure 1). Here we used this formula for our winning rate resting:

```
mov_scores = noMyMoves - noOppMoves*weight
```

Where the weight could alter the "aggressiveness" of the player. Here we tested a few different options we concluded a magnitude of 100 seem to be the most optimal and logical result:

Match #	Opponent	MMmove1		MMmove100		MMmove1K		ABmove1		ABmove100		ABmove1K	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	9	1	10	0	7	3	1	9	0	10	0	10
2	MM_Open	6	4	5	5	2	8	0	10	0	10	0	10
3	MM_Center	8	2	9	1	3	7	0	10	0	10	0	10
4	MM_Improved	4	6	4	6	0	10	0	10	0	10	0	10
5	AB_Open	10	0	10	0	10	0	6	4	8	2	4	6
6	AB_Center	10	0	10	0	10	0	7	3	6	4	7	3
7	AB_Improved	10	0	10	0	9	1	3	7	4	6	5	5
	Win Rate:	81.4%		82.9%		58.6%		24.3%		25.7%		22.9%	

(figure 1)

However, we also find that no of moves works well only if the search is over certain level. In order to test this out, we changed the game setting to 5x5 (fewer combinations for search) and tested the recommended optimal first move across different search level and found that the

recommendation tend to vary a bit in the first few levels and remain relatively stable thereafter (figure 2). Hence if the board is big and we do not have time to search too deep, no of moves might not be a good choice.

```
array([[ -6.,  -6.,  -6.,  -6.,  -8.,  -6.,  -6.,  -6.,  -6.],
       [  0.,   0.,   0.,   0., -inf,   0.,   0.,   0.,   0.],
       [ -1.,  -1.,  -1.,  -1.,  inf,  -1.,  -1.,  -1.,  -1.],
       [-inf, -inf, -inf, -inf,  inf, -inf, -inf, -inf, -inf],
       [-inf, -inf, -inf, -inf,  inf, -inf, -inf, -inf, -inf],
       [-inf, -inf, -inf, -inf,  inf, -inf, -inf, -inf, -inf]])
```

(figure 2)

Center first:

From the lecture, we know that being able to take over the center position could significantly increase the chance of winning. In the 5 x5 test above in figure 2, we also confirm that taking the center position seem to offer a higher chance of winning. Also calculation of the center position is relatively easy. It seems that Center first is a good strategy to include in our score calculation.

However, we need a function that returns a greater positive value when the move is exactly (or very close) to center and have apparently no effect on other positions (because we do not know if other positions have any preferential advantage). Here we tried to use the heuristic function of Guassian (normal) kernel so that a maximal value would be returned as the move is closest to the center position:

```
pos_scores = math.exp(-1.*(((h - y)**2 + (w - x)**2)/(2*sigma)**2))
```

Where sigma describes the "smoothness" of the bell distribution. We tried a few different values and here is the result(figure 3):

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

Match #   Opponent    MMcenter.2   MMcenter.5   MMcenter1   ABcenter.2   ABcenter.5   ABcenter1   ABcenter2
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     9  |  1     7  |  3     10 |  0     0  | 10     0  | 10     0  | 10     0  | 10
    2       MM_Open    7  |  3     1  |  9     5  |  5     0  | 10     0  | 10     0  | 10     0  | 10
    3      MM_Center   7  |  3     5  |  5     9  |  1     0  | 10     0  | 10     0  | 10     0  | 10
    4     MM_Improved  0  | 10     2  |  8     5  |  5     0  | 10     0  | 10     0  | 10     0  | 10
    5       AB_Open   10  |  0    10  |  0    10  |  0     6  |  4     4  |  6     4  |  6     3  |  7
    6      AB_Center  10  |  0    10  |  0    10  |  0     5  |  5     5  |  5     5  |  5     7  |  3
    7     AB_Improved 10  |  0    10  |  0    10  |  0     5  |  5     5  |  5     3  |  7     5  |  5
-------------------------------------------------------------------------------------------------------
          Win Rate:      75.7%        64.3%        84.3%        22.9%        20.0%        17.1%        21.4%
```

(figure 3)

It is quite obvious that a smaller sigma (sharper) bell, would result in the best winning result. After a few rounds of testing, we decided to use 0.05 as the value of sigma which seems to work best for both Alphabeta and MiniMax players. As a matter of fact, with this implementation, the Center first factor has almost zero effect when it comes to other position than Center which is good as it will not introduce noise to other factors.

Combined Result

After doing the evaluating both the No of Moves and Center First, it is time to put them together. The following table summarize the considerations of these 2 factors against the criteria we mentioned earlier:

Features	No of Moves	Center First
Easy to compute?	Relatively easy	yes
Winning rate of same players	>50%	>50%
Consistency across different depth	For the first few level it is unstable	Relatively stable across different level
Easy to configure	yes	yes

Base on the analysis above, we believed that it is safe to include these two factors as the core contributing factors as the approximation of the board utility.

The final step here is to determine the weight between these 2 factors which is the ratio of A VS B. Here we tested a few different combination and found that the best rough range of A:B would be around 1:100 which produces the highest winning rate (figure 4)

total_score = A*pos_scores + B*mov_scores

Match #	Opponent	MM1K		MM100		MM10		AB1K		AB100		AB1	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	10	0	9	1	8	2	0	10	0	10	0	10
2	MM_Open	6	4	5	5	4	6	0	10	0	10	0	10
3	MM_Center	6	4	8	2	9	1	0	10	0	10	0	10
4	MM_Improved	7	3	4	6	5	5	0	10	0	10	0	10
5	AB_Open	10	0	10	0	10	0	9	1	8	2	3	7
6	AB_Center	10	0	10	0	10	0	6	4	6	4	7	3
7	AB_Improved	10	0	10	0	10	0	7	3	5	5	4	6
	Win Rate:	84.3%		80.0%		80.0%		31.4%		27.1%		20.0%	

(figure 4)

Hence we updated the final recommendation of the heuristic formula to:

total_score = pos_scores + 100*mov_scores

And shall submit this as the “best” function in custom_score function.