Game Playing Agent Heuristic Evaluation

Jeremie Lecomte

```
                    *************************
                         Playing Matches
                    *************************
 Match #    Opponent    AB_Improved    AB_Custom    AB_Custom_2  AB_Custom_3
                         Won | Lost    Won | Lost    Won | Lost    Won | Lost
    1       Random        20  |   0     20  |   0     20  |   0     20  |   0
    2       MM_Open       18  |   2     15  |   5     17  |   3     18  |   2
    3      MM_Center      19  |   1     19  |   1     20  |   0     19  |   1
    4     MM_Improved     15  |   5     17  |   3     18  |   2     16  |   4
    5       AB_Open       11  |   9      8  |  12      7  |  13     13  |   7
    6      AB_Center      10  |  10     14  |   6     11  |   9      9  |  11
    7     AB_Improved      9  |  11     10  |  10      9  |  11      9  |  11
----------------------------------------------------------------------------
           Win Rate:      72.9%        73.6%        72.9%        74.3%
```

The main problem I had for this study was that the tournament took really long but at the same time 5 games per test weren't reliable to get proper confidence interval. The doing twice the same experiment gave up to 15% relative difference for the same agent.

AB_Custom == custom_score_2
Score ranged from 70.0% to 73.6%

The main idea here is that in the first part of the game you try to get as close to you opponent as possible and in the second part you do not computation in order to get some time to get deeper in the search tree since there should be way less possibility to scan for. And we may have an average better result if we increase our chance to reach the leaves (optimal in the Min_Max sense)

```
percent = game.move_count/(game.height * game.width)*100

if  percent < 60:
    y_opp, x_opp = game.get_player_location(game.get_opponent(player))
    y_own, x_own = game.get_player_location(player)
    return float(-float((y_own-y_opp) ** 2 + (x_own-x_opp) ** 2))
else:
    return 1
```

AB_Custom_2 == custom_score
The scores ranged from 72.9% to 82.9%

The main idea here is to be really aggressive by actively trying to select a position that give us some flexibility but moreover limiting the number of position that the opponent can play. The weight of 2.5 was empirically chosen after multiple set of tests.

```
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(own_moves - 2.5*opp_moves)
```

AB_Custom_3 == custom_score_3
The scores ranged from 71.4% to 74.3%

The main idea here is that in the first phase we do as for the previous tactic but then in the later phase of the game we try to predict two moves ahead the number of possible legal move for both player and adjust the score accordingly.

```
percent = game.move_count/(game.height * game.width)*100

if  percent < 50:
    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(own_moves - 2.5*opp_moves)
else:

    opp = game.get_opponent(player)
    own_score = opp_score = 0

    for first_move in game.get_legal_moves(player):
        next_game = game.forecast_move(first_move)
        for second_move in next_game.get_legal_moves(player):
            next_next_game = game.forecast_move(second_move)
            own_score += len(next_game.get_legal_moves(player))

    for first_move in game.get_legal_moves(opp):
        next_game = game.forecast_move(first_move)
        for second_move in next_game.get_legal_moves(opp):
            next_next_game = game.forecast_move(second_move)
            opp_score += len(next_game.get_legal_moves(opp))

    return float(own_score - opp_score)
```

Follow up:
I rerun the test with more games per test and got:

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	40	0	39	1	40	0	40	0
2	MM_Open	33	7	37	3	34	6	31	9
3	MM_Center	39	1	37	3	39	1	39	1
4	MM_Improved	36	4	30	10	35	5	35	5
5	AB_Open	23	17	16	24	22	18	21	19
6	AB_Center	25	15	21	19	21	19	22	18
7	AB_Improved	17	23	20	20	19	21	19	21
	Win Rate:	76.1%		71.4%		75.0%		73.9%	

Therefor I would recommend using my AB_Custom_2 method that is after all the simplest one and showed the best average performance. This method may be better because less computational effort and then we may have more time to look at some more layer in the tree. It is also base on active playing and not just trying to escape the opponent, this is usually a better game strategy. I spend some time tuning the weigh function and found out that 2.5 was a good compromise.