Build a Game-Playing Agent
Heuristic Analysis

Tibor Zahorecz
Artificial Intelligence Nanodegree

17th June 2017

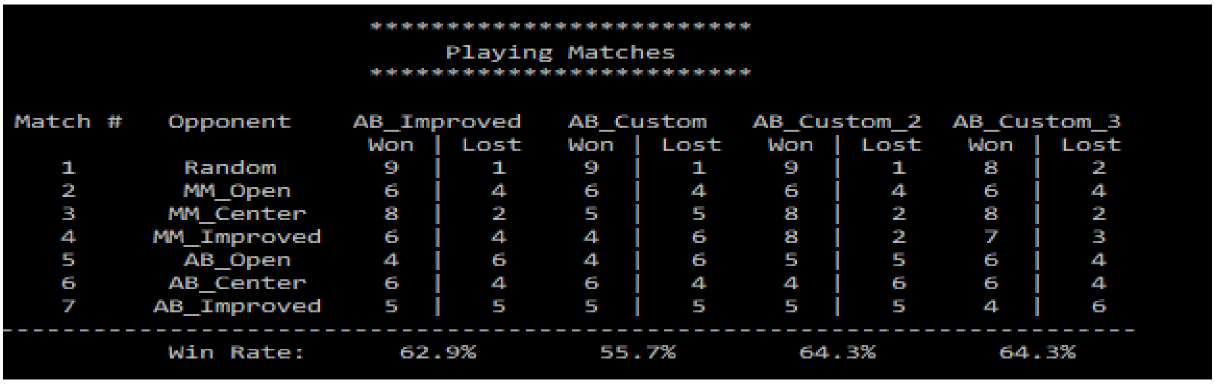
Heuristic 1

This heuristic function limiting the opponent’s moves and make my agent more aggressive. Any multiplier greater than 1 will have effect in this evaluation function. Use any multipler for opponent moves x > 1.

```
if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(own_moves - x * opp_moves)
```

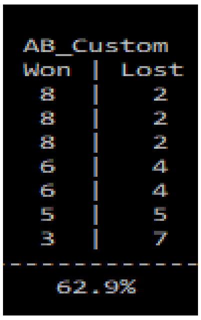
examples:

with x = 2 multipler the result is:



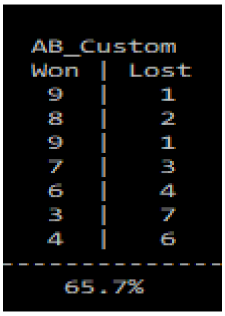
```
                              *************************
                                   Playing Matches
                              *************************

Match #    Opponent     AB_Improved    AB_Custom    AB_Custom_2  AB_Custom_3
                         Won | Lost   Won | Lost   Won | Lost   Won | Lost
   1        Random        9  |   1     9  |   1     9  |   1     8  |   2
   2        MM_Open       6  |   4     6  |   4     6  |   4     6  |   4
   3       MM_Center      8  |   2     5  |   5     8  |   2     8  |   2
   4      MM_Improved     6  |   4     4  |   6     8  |   2     7  |   3
   5        AB_Open       4  |   6     4  |   6     5  |   5     6  |   4
   6       AB_Center      6  |   4     6  |   4     4  |   6     6  |   4
   7      AB_Improved     5  |   5     5  |   5     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:       62.9%        55.7%        64.3%        64.3%
```

increased upto x= 2.5 multiplier has even better result:



```
 AB_Custom
 Won  |  Lost
  8   |    2
  8   |    2
  8   |    2
  6   |    4
  6   |    4
  5   |    5
  3   |    7
-------------
    62.9%
```

and increased further up to x= 3 multiplier, result:



```
 AB_Custom
 Won  |  Lost
  9   |    1
  8   |    2
  9   |    1
  7   |    3
  6   |    4
  3   |    7
  4   |    6
-------------
    65.7%
```

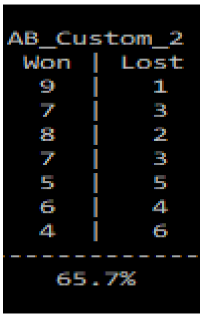
My revised decision is to stick x=3 multiplier.

Heuristic 2

The heuristic is based on the logic that player’s moves should be maximized. Any multiplier greater than 1 will have effect in this evaluation function. Use any multipler for own moves x > 1.

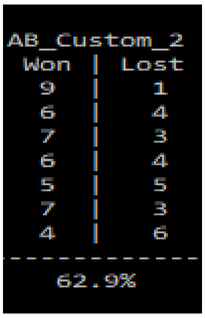
```
if game.is_loser(player):
    return float("-inf")
  if game.is_winner(player):
    return float("inf")
  own_moves = len(game.get_legal_moves(player))
  opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
  return float(x * own_moves - opp_moves)
```

selecting x=1.2 the result is:



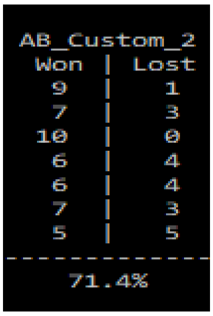
```
AB_Custom_2
 Won  |  Lost
  9   |    1
  7   |    3
  8   |    2
  7   |    3
  5   |    5
  6   |    4
  4   |    6
------------
   65.7%
```

selecting x=1.5 the result is:



```
AB_Custom_2
 Won  |  Lost
  9   |    1
  6   |    4
  7   |    3
  6   |    4
  5   |    5
  7   |    3
  4   |    6
------------
   62.9%
```

selecting x=2 the result is:



```
 AB_Custom_2
  Won  |  Lost
   9   |    1
   7   |    3
  10   |    0
   6   |    4
   6   |    4
   7   |    3
   5   |    5
------------
    71.4%
```

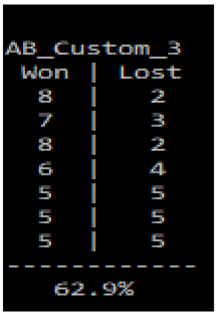
Selecting x=2 multiplier is the right solution of current experiments.

Heuristic 3:

This evaluation gives chance to explore more positions and evaluate more sub-trees. Use any multipler n for opponent moves as long as x < 1.

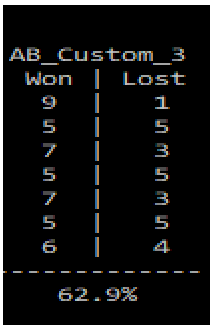
```
if game.is_loser(player):
    return float("-inf")
if game.is_winner(player):
    return float("inf")
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
return float(own_moves - x * opp_moves)
```

selecting x=0.5 the result is:



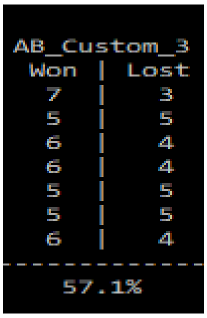
```
AB_Custom_3
 Won  |  Lost
  8   |    2
  7   |    3
  8   |    2
  6   |    4
  5   |    5
  5   |    5
  5   |    5
-------------
   62.9%
```

selecting x=0.25 the result is:



```
AB_Custom_3
 Won  |  Lost
  9   |    1
  5   |    5
  7   |    3
  5   |    5
  7   |    3
  5   |    5
  6   |    4
-------------
   62.9%
```

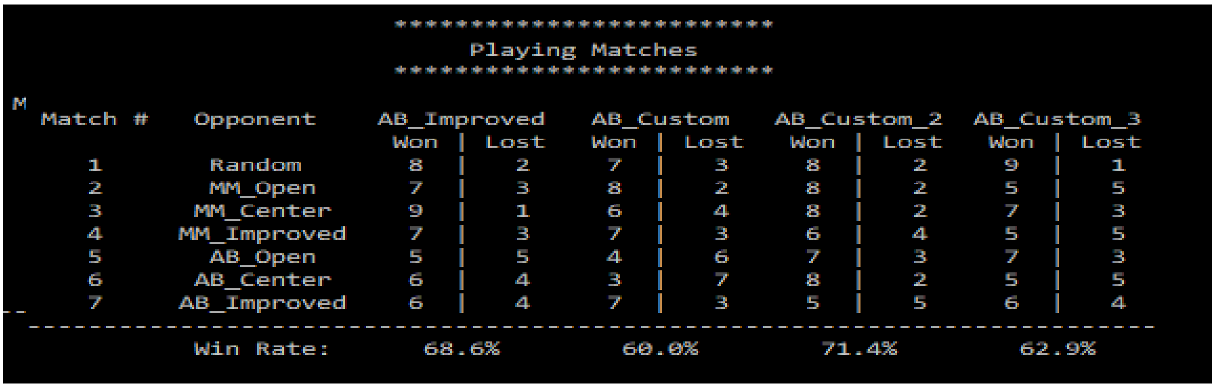
selecting x=0.75 the result is:



```
AB_Custom_3
 Won  |  Lost
  7   |    3
  5   |    5
  6   |    4
  6   |    4
  5   |    5
  5   |    5
  6   |    4
-------------
    57.1%
```

Selecting x=0.5 multiplier is the right solution of current experiments.

Heuristic Results



```
                                  *************************
                                       Playing Matches
                                  *************************
M
  Match #     Opponent     AB_Improved   AB_Custom    AB_Custom_2   AB_Custom_3
                            Won | Lost   Won | Lost   Won | Lost    Won | Lost
     1         Random        8  |  2      7  |  3      8  |  2       9  |  1
     2         MM_Open       7  |  3      8  |  2      8  |  2       5  |  5
     3        MM_Center      9  |  1      6  |  4      8  |  2       7  |  3
     4       MM_Improved     7  |  3      7  |  3      6  |  4       5  |  5
     5         AB_Open       5  |  5      4  |  6      7  |  3       7  |  3
     6        AB_Center      6  |  4      3  |  7      8  |  2       5  |  5
--   7       AB_Improved     6  |  4      7  |  3      5  |  5       6  |  4
 --------------------------------------------------------------------------
              Win Rate:        68.6%        60.0%        71.4%         62.9%
```

The results show AB_Custom_2 perform better than the AB_Improved .

In conclusion, it is recommended use AB_Custom_2 evaluation function:
* It performs better than other test agents
* Simple and quick to understand
* Quick to execute even on lighter CPU