ISOLATION GAME

HEURISTIC REPORT

AARON BALSON CAROLTIN
ARTIFICIAL INTELLIGENCE ND
08-March-2017

INTRODUCTION

The objective of the project is to design and implement an AI agent which is capable of playing a turn based strategy game called Isolation against other AI agents (as well against humans) and consistently beat them within the given time limit. We developed many heuristic functions to be used with the following search techniques in the program.

- MINIMAX
- ALPHA-BETA PRUNING
- ITERATIVE / PROGRESSIVE DEEPENING

HEURISTIC ALGORITHMS

In order to use these search techniques effectively, we ideated and experimented with numerous evaluation functions to measure the goodness of the board at the desired depth for our player and managed to implement four of them in this program.

(i) custom_seek_sum_movements

The improved score (from lecture) returns the difference in legal moves left for each player at the desired depth. Our evaluation takes one more step, it returns the difference in sum of all available branching options pursuable from every legal move left at the desired depth for both players.

Ex: If our player has three legal moves left at the leaf node – LM1, LM2, LM3 and taking each of them has 2, 3, 1 branching options, the function returns 2 + 3 + 1 = 6

There is a higher possibility of the result being skewed based on few entries, but that entry may never be pursued in next turn. It is observed the performance is of acceptable level and consistently reached the desired depth without timeout resignations. Because of simplicity, it can achieve deeper levels if required. It also scored the highest of all other heuristics *[79.29% wins of 140 games]*.

```
************************
 Evaluating: ID_Improved
************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null     Result: 15 to 5
  Match 3: ID_Improved vs   MM_Open     Result: 13 to 7
  Match 4: ID_Improved vs MM_Improved   Result: 14 to 6
  Match 5: ID_Improved vs   AB_Null     Result: 17 to 3
  Match 6: ID_Improved vs   AB_Open     Result: 15 to 5
  Match 7: ID_Improved vs AB_Improved   Result: 13 to 7


Results:
----------
ID_Improved         75.00%

************************
   Evaluating: Student
************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 16 to 4
  Match 2:   Student   vs   MM_Null     Result: 16 to 4
  Match 3:   Student   vs   MM_Open     Result: 13 to 7
  Match 4:   Student   vs MM_Improved   Result: 17 to 3
  Match 5:   Student   vs   AB_Null     Result: 15 to 5
  Match 6:   Student   vs   AB_Open     Result: 17 to 3
  Match 7:   Student   vs AB_Improved   Result: 17 to 3


Results:
----------
Student             79.29%
PS C:\Users\AARONW10\Desktop\STUDY\UDACITY\AIND\AIND-Isolation>
```

(ii) custom_seek_average_movements

This one is built on top of earlier one, with instead of returning the difference of sum of all pursuable movements, it returns the sum of average movements pursuable from each legal move left.

Ex: Using the earlier case, the function returns (2 + 3 + 1) / 3 = 2

It is observed the performance is of acceptable level and consistently reached the desired depth without timeout resignations. However we realized, instead of average, logarithm might be of better choice (for future iteration).

```
************************
 Evaluating: ID_Improved
************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 17 to 3
  Match 2: ID_Improved vs   MM_Null     Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open     Result: 14 to 6
  Match 4: ID_Improved vs MM_Improved   Result: 12 to 8
  Match 5: ID_Improved vs   AB_Null     Result: 14 to 6
  Match 6: ID_Improved vs   AB_Open     Result: 12 to 8
  Match 7: ID_Improved vs AB_Improved   Result: 13 to 7


Results:
----------
ID_Improved         70.00%

************************
   Evaluating: Student
************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 15 to 5
  Match 2:   Student   vs   MM_Null     Result: 13 to 7
  Match 3:   Student   vs   MM_Open     Result: 12 to 8
  Match 4:   Student   vs MM_Improved   Result: 13 to 7
  Match 5:   Student   vs   AB_Null     Result: 14 to 6
  Match 6:   Student   vs   AB_Open     Result: 11 to 9
  Match 7:   Student   vs AB_Improved   Result: 13 to 7


Results:
----------
Student             65.00%
PS C:\Users\AARONW10\Desktop\STUDY\UDACITY\AIND\AIND-Isolation>
```

(iii) custom_seek_center_position

This one is seeking positional advantage of the board for our player. We implemented a basic version, thereby determining who is closest to the center of the board under the belief, that player may have more degree of freedom to expand across the board and able to partition the board to restrict the opponent to edges.

It is observed the performance is of acceptable level and consistently reached the desired depth without timeout resignations. This can be more effective in a

constrained 7x7 board with KNIGHT movement, rather than bigger board with QUEEN movement. Unlike most games, center position is occupied only once, so we need to take partitions taken into account [future iteration].

```
************************
 Evaluating: ID_Improved
************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 16 to 4
  Match 2: ID_Improved vs   MM_Null     Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open     Result: 12 to 8
  Match 4: ID_Improved vs MM_Improved   Result: 15 to 5
  Match 5: ID_Improved vs   AB_Null     Result: 16 to 4
  Match 6: ID_Improved vs   AB_Open     Result: 12 to 8
  Match 7: ID_Improved vs AB_Improved   Result: 14 to 6


Results:
----------
ID_Improved         72.14%

************************
   Evaluating: Student
************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 13 to 7
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 9 to 11
  Match 4:   Student   vs MM_Improved   Result: 11 to 9
  Match 5:   Student   vs   AB_Null     Result: 18 to 2
  Match 6:   Student   vs   AB_Open     Result: 12 to 8
  Match 7:   Student   vs AB_Improved   Result: 11 to 9


Results:
----------
Student             63.57%
PS C:\Users\AARONW10\Desktop\STUDY\UDACITY\AIND\AIND-Isolation>
```

(iv) custom_seek_movements_positions

This combines (i) and (iii) in an attempt to move towards better positions which hopefully has more freedom of movement. Like (iii), this also needs refinement by considering partitions on board.

It is observed the performance is of acceptable level and consistently reached the desired depth without timeout resignations.

```
 Evaluating: ID_Improved
************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 16 to 4
  Match 2: ID_Improved vs   MM_Null     Result: 14 to 6
  Match 3: ID_Improved vs   MM_Open     Result: 10 to 10
  Match 4: ID_Improved vs MM_Improved   Result: 14 to 6
  Match 5: ID_Improved vs   AB_Null     Result: 16 to 4
  Match 6: ID_Improved vs   AB_Open     Result: 11 to 9
  Match 7: ID_Improved vs AB_Improved   Result: 14 to 6


Results:
----------
ID_Improved         67.86%

************************
   Evaluating: Student
************************

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 17 to 3
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 11 to 9
  Match 4:   Student   vs MM_Improved   Result: 12 to 8
  Match 5:   Student   vs   AB_Null     Result: 17 to 3
  Match 6:   Student   vs   AB_Open     Result: 13 to 7
  Match 7:   Student   vs AB_Improved   Result: 13 to 7


Results:
----------
Student             70.00%
PS C:\Users\AARONW10\Desktop\STUDY\UDACITY\AIND\AIND-Isolation>
```

CONCLUSION

We explored other heuristics like endgame and transposition tables and tried advanced positional algorithm with partition identifier but couldn't implement them for this project. Based on pure win% we recommend using custom_seek_sum_movements in our agents but we wish to develop custom_seek_movements_positions with partition identifier for future iterations.