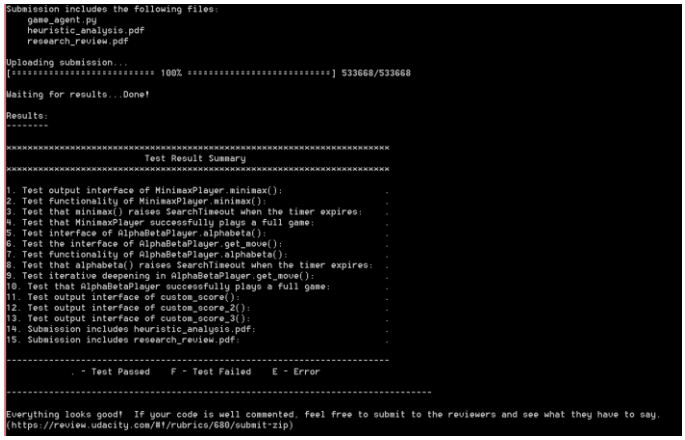
Author: Rafał Biegacz, October 2017

Isolation Game – Research Report

Introduction

The correctness of the implementation of all the functions within “Isolation Game” project was proven in 3 ways:

a) Tests executed by Udacity (command: **`udacity submit isolation`**)



b) By playing tournament (command: **`python tournament.py`**)

c) Unit tests that were implemented in agent_test.py (available on GitHub)

Configuration of timeout Parameter

Timeout parameter in IsolationPlayer class (inherited by MiniMaxPlayer and AlphaBetaPlayer classes) was set to 40ms. This timeout lets avoid situations where there was not enough time for MiniMaxPlayer and AlphaBetaPlayer to finish their calculations (as it was presented below).

```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                        ××××××××××××××××××××××××××
                             Playing Matches
                        ××××××××××××××××××××××××××

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       8  |   2      8  |   2      9  |   1      7  |   3
    2       MM_Open      7  |   3      7  |   3      5  |   5      7  |   3
    3      MM_Center     9  |   1      9  |   1      8  |   2      8  |   2
    4     MM_Improved    8  |   2      5  |   5      4  |   6      4  |   6
    5       AB_Open      6  |   4      7  |   3      4  |   6      7  |   3
    6      AB_Center     6  |   4      7  |   3      5  |   5      6  |   4
    7     AB_Improved    4  |   6      7  |   3      4  |   6      4  |   6
--------------------------------------------------------------------------
          Win Rate:      68.6%        71.4%        55.7%        61.4%

Your ID search forfeited 248.0 games while there were still legal moves available to play.
```

Implementations of Evaluation Functions/Heuristics

3 evaluation functions/heuristics were implemented in the form of the following functions.

Heuristic #1 (custom_score) implementation

custom_score function implements the simplest possible heuristics – this heuristics gives a difference between the number of possible moves for player #1 and player #2. The more moves player #1 has compared to number of possible options of player #2 the better score.

Player #1 and player #2 can have up to 8 moves available so the difference between their number of moves could be in the range of [-8, 8]. custom_score was normalized and it delivers values from the range [-1, 2].

Normally, the return values are within the range between -1 and 1. There is one special case, though. If the player #1 position is central point of the board we promote this move by adding 1 to the calculated score.

Heuristic #2 (custom_score_2) implementation

custom_score2 calculates Manhattan distance of player #1 and player #2 from the center of the board and the returns the difference. The return values are normalized so the function returns from the range from -1 to 2.

Normally, the return values are within the range between -1 and 1. There is one special case, though. If the player #1 position is central point of the board we promote this move by adding 1 to the calculated score.

The idea is that the farther you are from the center of the board the less options for next moves you have.

Heuristic #3 (custom_score_3) implementation

custom_score3 function is similar to custom_score – this heuristics gives a difference between the number of possible moves for player #1 and **doubled** number of moves for player #2. This evaluation function promotes games which give big number of moves for player #1 and penalizes games where player #2 has way bigger number of moves. The custom_score3 values are normalized and the function returns values between -1 and 2.

Normally, the values are within the range between -1 and 1. There is one special case, though. If the player #1 position is central point of the board we promote this move by adding 1 to the calculated score.

Conclusion

```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                    ************************
                         Playing Matches
                    ************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       8  |  2     10  |  0     9  |  1     9  |  1
    2       MM_Open      8  |  2      5  |  5     5  |  5     3  |  7
    3     MM_Center      9  |  1      6  |  4     6  |  4     9  |  1
    4    MM_Improved     6  |  4      9  |  1     7  |  3     7  |  3
    5      AB_Open       6  |  4      6  |  4     6  |  4     5  |  5
    6     AB_Center      6  |  4      8  |  2     9  |  1     5  |  5
    7    AB_Improved     7  |  3      4  |  6     3  |  7     8  |  2
----------------------------------------------------------------------
          Win Rate:      71.4%        68.6%        64.3%        65.7%

There were 1.0 timeouts during the tournament -- make sure your agent handles search timeout correctly, and consider increasing the timeout margin for your agent.


Your ID search forfeited 243.0 games while there were still legal moves available to play.
```

Comparing the results achieved with custom_score, custom_score2 and custom_score3 evaluation functions the best results were achieved using custom_score – which means that the best moves are those which limit the number of moves of the opponent and the smaller number of potential moves of the opponent is the better.

Here are three reasons why custom_score is the best based on this report:

a) custom_score gives the best results in tournament.py competition; the scores delivered by this evaluation function are on a similar level as “AB_Improved” scheme.
b) custom_score is simple from the computational point of view which means that get_move function will finish quicker and one will be able to make more calls of get_move function in the same period of time;
c) custom_score is very simple to implement
d) because custom_score is simple from computational point of view it could be combined with other evaluation functions to create more efficient evaluation functions