Udacity AIND Game Playing Agent Heuristics Review

May 15, 2017

1 Heuristics review

1.1 Summary

As the goal was to beat the AB_Improved heuristic, I based mine on the same idea, of player moves vs opponent moves. I've built a total of 4 custom score functions. First played 3 of them against all CPU opponents, and then, after noting that MiniMax didn't stand a chance (due to low depth level =3), I played all 4 against the AB heuristics defined in sample_players.py.

The heuristics and the results are described below.

Please note that the custom score function in the first round were called differently from the second round. Due to time constrains it was left like that in report. Apologies

```
In [1]: %run tournament.py
```

```
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      20  |   0    20  |   0    18  |   2    20  |   0
    2       MM_Open     19  |   1    17  |   3    17  |   3    17  |   3
    3      MM_Center    19  |   1    20  |   0    19  |   1    20  |   0
    4     MM_Improved   18  |   2    18  |   2    18  |   2    18  |   2
    5       AB_Open      9  |  11    11  |   9     8  |  12    10  |  10
    6      AB_Center    11  |   9    12  |   8    13  |   7    13  |   7
    7     AB_Improved    7  |  13     8  |  12    11  |   9     8  |  12
--------------------------------------------------------------------------
           Win Rate:      73.6%        75.7%        74.3%        75.7%
```

1.1.1 AB_Improved

Heuristic provided in sample_players.py, it defines the score as own legal moves - opponen legal moves. It's worth noting that when played against itself, the score was 7-13, which indicates high sensitivity to starting positions.

1.1.2 AB_Custom

It's the **AB_Own-2*Opp** described below

1.1.3 AB_Custom_2

It's the **AB_Weighted** described below

1.1.4 AB_Custom_3

It's the **AB_Apart** described below

1.1.5 Results

Alpha Beta pruning algorithms, with iterative deepening were much more successful than MiniMax with fixed depth of 3. Also, it looks like AB_Open, looking at just the legal moves of the player alone, seems to beat even the AB_Improved, which could indicate there that keeping more moves available for yourself is important.

```
In [3]: %run tournament_slim.py
```

```
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_Conflict AB_Own-2*Opp  AB_Weighted   AB_Apart
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       AB_Open     18  |  22    18  |  22    21  |  19    22  |  18
    2      AB_Center    27  |  13    24  |  16    27  |  13    25  |  15
    3     AB_Improved   23  |  17    19  |  21    21  |  19    20  |  20
--------------------------------------------------------------------------
           Win Rate:      56.7%        50.8%        57.5%        55.8%
```

1.1.6 AB_Conflict

This heuristic (under the name customer_score_4 in the game_agent.py), builds upn the AB_Improved. It defines the score as difference between number of own legal moves and number of opponent legal moves. It also subtracts the number of overlaping legal moves from the total score. This way the agent is still encouraged to block the opponent, but also move into non overlapping territory (run away from the opponent). That can be beneficial in the later game.

This agent performed worse than AB_Open, but did beat AB_Center and AB_Improved by the widest margin (Although, the sample was still very small, only 40 games).

1.1.7 AB_Own-2*Opp The simplest heuristics of the four. Defines the score as number of own legal moves - 2 times number of opponent legal moves. It favors the blocking moves (reducing number of opponent moves) rather than expanding it's own territory.

This heuristic performed the poorest in the second round. Resulting in a tie with AB_Improved.

1.1.8 AB_Weighted

This heuristic builds on top of **AB_Own-2*Opp**. Instead of having fixed multiplier next to number of legal opponent moves, it weights own moves by proportion of blank spaces left to the board size and opponent moves by (1-proportion).

This way at the beginning of the game, when most of the board is blank, it favours expanding it's own territory, while with time, it puts more weight on moves that block the opponent.

This heuristic performed the best in the second round, although by a small margin compared to AB_Improved. It was the strongest against AB_Center.

1.1.9 AB_Apart

This heuristic expands the AB_Improved one, by multiplying the difference in own moves and opponent moves by the squared distance between players. It still favours blocking the opponent (reducing the number of legal moves), but when expanding it's own territory, it'll chose moves that place the player away from the opponent.

It performed well against the AB_Open and AB_Center, but didn't produce significant advantage over AB_Improved.

1.2 Chosen heuristics

The chosen heuristic is AB_Weighted (custom_score_2 in game_agent.py), for the following reasons: - Strong performance against all CPU agents, best accross the board - Short execution time, uses only the len() function, on the legal moves and blank spaces array, no sqrt() or exp() - It uses information about both player and opponent (number of available moves) - It uses information about the state of the game and board (how many blank spaces are left as an approximation of how far into the game are we)

1.2.1 Improvements

The performance of these simple heuristics is not really satisfactory, and could definitely be improved by analysing the game further, speaking to expert players, coming up with opening moves book, adjusting the weights in more complex way, rather than simple proportion, doing a 2 or 3 step ahead look up of available moves (e.g. available moves from each position available now, to get better grasp of location, as moves closer to the corner may be worse; or weighting the location of available move by gausiann kernel), looking for separation (is middle line blocked etc) and so on.

This project will therefore be surely revisited in the near future.