PROJECT II: BUILD AN ADVERSARIAL SEARCH AGENT

Analysis of custom evaluation functions

Contents

1 Methodology

We have run three separate runs of match playing between the various agents, with a number of matches N=5,10,25,50. In all the cases, the timeout margin of the gaming-agents was 20ms.

2 Definitions for the evaluation functions

We recall the python implementations of each of the three evaluation functions.

2.1 custom_score

```
def custom_score(game, player):
    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 - 0.5*opp_moves)
```

2.2 custom_score_2

```
def custom_score_2(game, player):
    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(0.5*own_moves -opp_moves
```

2.3 custom_score_3

```
def custom_score_3(game, player):
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")
    blank_spaces_n=len(game.get_blank_spaces())
    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
    w_own_moves=own_moves/(opp_moves+ own_moves)
    if blank_spaces_n >= 32:
        return float((1-w_own_moves)*own_moves – w_own_moves*opp_moves)
    elif 16<= blank_spaces_n <32:
```

```
            return                          float(0.5*own_moves*w_own_moves-(1-
w_own_moves)*opp_moves)
        else:
            return float(own_moves -0.5* opp_moves)
```

3 Background on the selection of the functions

We separate the three proposed custom evaluation functions in two categories:

i. Functions custom_score (CS) and custom_score_2 (CS2)
ii. Function custom_score_3 (CS3)

First, for category i, the intention was to design two types of static weight agents: One, more "conservative", so as to be less concerned in restricting the moves of the opposite player, and more in maximizing its own. In practice, this was implemented by attributing half of the weight of "opp_moves" to "own_moves" (as can be seen in the definition of CS). In addition, and to obtain an opposite style of player, we looked to design a more aggressive player, in the sense that it actively looks to minimize the number of moves of the opposite player. This was achieved in CS2 function by reversing the weighting scheme in CS function, and attributing the smaller weight to "own_moves".

In designing the CS3 function in ii, the objective was to allow dynamic weighting in the evaluation function. The intuition behind this goal was the intention to implement a game-agent which would start conservative, and would become increasingly aggressive after the middle of the game. In the end, however, we implemented an agent that is aggressive or conservative according to its position during the first third of the game, is aggressive during the second third of the game, and becomes conservative on the last third.

Indeed, by looking at the definition of CS3, we can see that in the first third of the game (blank_spaces>32) if we fix the quantity "own_moves + opp_moves", the heuristic function will make the player "chase" the opponent as long as its own number of legal moves is larger than those of the opponent. On the contrary, if the ratio of legal moves of the player is smaller than their opponent's, they will prefer to increase their own moves, at the expense of reducing their opponent's. We believe that since the game is still in the beginning, there is still time left to correct bad moves that may have occurred while the player tires to chase a winning streak, during these first part of the game. Now, for the second third of the game (16<=blank_spaces<32), we attributed a weighting scheme similar to function CS2, to the extent that it leads to an aggressive game-playing style. Although we have allowed the weights to be dynamic and proportionate to the number of moves, which could cause conservative or neutral game-playing, the factor ½ in own_moves guarantees that the heuristic function will always penalize opp_moves more than own_moves. Finally, for the last third of the game, we have decided for a conservative and static-weighted game-playing style, where the evaluation function is the same as CS

4 Results and discussion

We first look at the results for N=5,10.

```
                    *************************
                        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     9  |  1     8  |  2     10 |  0
    2       MM_Open      8  |  2     7  |  3     8  |  2     8  |  2
    3      MM_Center    10  |  0    10  |  0     9  |  1     7  |  3
    4     MM_Improved    7  |  3     7  |  3     7  |  3     7  |  3
    5       AB_Open      5  |  5     5  |  5     8  |  2     4  |  6
    6      AB_Center     6  |  4     6  |  4     6  |  4     6  |  4
    7     AB_Improved    5  |  5     5  |  5     3  |  7     4  |  6
--------------------------------------------------------------------------
-
          Win Rate:      70.0%        70.0%        70.0%        65.7%

Match #    Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      17  |  3    18  |  2    20  |  0    20  |  0
    2       MM_Open     16  |  4    16  |  4    15  |  5    15  |  5
    3      MM_Center    18  |  2    18  |  2    19  |  1    17  |  3
    4     MM_Improved   17  |  3    13  |  7    17  |  3    16  |  4
    5       AB_Open     11  |  9    11  |  9     9  |  11    9  |  11
    6      AB_Center     8  |  12   13  |  7    11  |  9    14  |  6
    7     AB_Improved    7  |  13   13  |  7    11  |  9    10  |  10
--------------------------------------------------------------------------
-
          Win Rate:      67.1%        72.9%        72.9%        72.1%
```

From here, we can see that the functions seem to have difference performance statistics according to N. In order to obtain a more robust comparison we have then decided to increase N. We thus obtained the following tables for N=25,50.

```
Match #    Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      47  |  3    45  |  5    44  |  6    50  |  0
    2       MM_Open     41  |  9    40  |  10   38  |  12   35  |  15
    3      MM_Center    42  |  8    47  |  3    46  |  4    45  |  5
    4     MM_Improved   40  |  10   39  |  11   39  |  11   41  |  9
    5       AB_Open     26  |  24   27  |  23   26  |  24   29  |  21
    6      AB_Center    28  |  22   27  |  23   30  |  20   30  |  20
    7     AB_Improved   28  |  22   26  |  24   26  |  24   26  |  24
--------------------------------------------------------------------------
-
          Win Rate:      72.0%        71.7%        71.1%        73.1%

Match #    Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      97  |  3    95  |  5    96  |  4    95  |  5
    2       MM_Open     74  |  26   74  |  26   78  |  22   75  |  25
    3      MM_Center    89  |  11   90  |  10   88  |  12   89  |  11
    4     MM_Improved   77  |  23   70  |  30   81  |  19   76  |  24
    5       AB_Open     55  |  45   61  |  39   51  |  49   53  |  47
    6      AB_Center    47  |  53   56  |  44   57  |  43   49  |  51
    7     AB_Improved   54  |  46   47  |  53   51  |  49   55  |  45
--------------------------------------------------------------------------
-
          Win Rate:      70.4%        70.4%        71.7%        70.3%
```

Taking the average weight of the results of these last tables we obtained, respectively for AB_improved, AB_Custom, AB_Custom_2 and AB_Custom_3, a win ratio of 71.2, 71.1, 71.4 and 71.7. With the information offered, and taking into consideration the fact that AB_Custom_3 won to AB_Improved in both cases, with a margin of 10 when N=50, we elect custom_score_3 as our own preferred evaluation function. The three reasons, supported by the data, for my choice are: The average overall win rate between N=25 and N=50 is the highest; it beats AB_Improved, which was one an important objectives of the heuristic design, by the highest margin when N=50; The average number of wins (37) for N=25 is higher, although slightly, than the others (36) (Although when N=50 in that measure custom_3 is not the best, that fact that it beats AB_improved by 4 more games than custom_2 is a reason for my choice as a candidate for a beat-AB_improved heuristic).