Heuristic analysis of custom scoring function in ‘Isolation’ game

Alexey Denisov

12/26/2017

In the development of the Isolation game playing agent, I have used the following approach:

First, I identified as many as possible features which could influence the final score:

- #own_moves
- #opponent_moves
- own_position_coordinates
- opponent’s_position_coordinates
- game phase (like opening, middle and endgame)

The last three features may be transformed into a set of

- relative_distance (like my_position_to_opponent’s_position_distance, my_position_relative_center_distance, my_position_relative_corner_distance etc.)

Second, I constructed several functions which used these features with different weights and experimented with the functions and weights.

I quickly found out that any sophisticated score functions (like weights which depend on game phase etc.) may require additional computation time. It may in turn reduce the depth of the alpha-beta pruning algorithm. The reduction of the depth is a very serious negative factor for the score function.

So, I had to come up with computationally simple but explicit score functions.

After all the experiments I have developed 3 scoring functions. They are based on three features: #my_moves, #opponent’s_moves and my_distance_to_center but with different computational methods and weights.

1) **custom_score - “Logarithm”**: The key function is the logarithm of the #my_moves/#opponent’s_moves slightly corrected to prefer more central moves. It helps to better differentiate the endgame positions but requires more time to compute.

```
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)

distance = float((abs(h - y) + abs(w - x))/2)

return float(math.log((own_moves + .1)/ (opp_moves + .1)) - 0.001 * distance )
```

2) **custom_score_2 - “Fraction”**: Here I have a computationally simple function which helps to find a better difference between numbers of my moves versus opponent’s ones weighted by the sum of our moves. It helps in my view to better differentiate among positions in the middle game.

```
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

return float((own_moves - 2 * opp_moves)/(own_moves + opp_moves + .1))
```

3) **custom_score_3 - “Improved+Center”**: This heuristic function is based on the improved_score but gives a bonus to our agent if the agent is within the center region of the board.

```
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)

distance = float((abs(h - y) + abs(w - x))/2)
return float(own_moves - 2 * opp_moves - 0.2 * distance)
```

Initially in the tournament .py, the num_matches was 5. But in my opinion, this was too small a sample size. So I made the num_matches to 50. This was the result I got:

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

 Match #    Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                         Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1        Random      96  |   4    98  |   2    96  |   4    98  |   2
    2        MM_Open     81  |  19    83  |  17    86  |  14    82  |  18
    3       MM_Center    93  |   7    95  |   5    97  |   3    97  |   3
    4      MM_Improved   80  |  20    83  |  17    78  |  22    84  |  16
    5        AB_Open     53  |  47    55  |  45    54  |  46    61  |  39
    6       AB_Center    58  |  42    63  |  37    60  |  40    59  |  41
    7      AB_Improved   46  |  54    53  |  47    55  |  45    52  |  48
--------------------------------------------------------------------------

            Win Rate:      72.4%        75.7%        75.1%        76.1%
```

We see that all the heuristic functions, perform better than the AB_Improved. The second one – the “Fraction” - though is better in the face-to-face battle against AB_Improved with the 55-45 chance to win.

In my opinion, the third function - “Improved+Central” - is the most promising algorithm between these 3 because:

A) it is computationally simple which helps to keep the depth at the good levels.
B) the central bias looks strategically the right thing to do.