Adversarial Game Playing agent Heuristic analysis

By Artem Odintsov

In this paper I would like to present the results of analysis of heuristics for adversarial game playing agent. For solving a search problem for game agent we use two different algorithms such as Minimax algorithm and Alpha-beta pruning algorithms.

To illustrate the results of different heuristics we will use several agents to play Isolation game:

1) Random Agent - randomly chooses a move each turn
2) MM_Open: MinimaxPlayer agent using the open_move_score heuristic with search depth 3
3) MM_Center: MinimaxPlayer agent using the center_score heuristic with search depth 3
4) MM_Improved: MinimaxPlayer agent using the improved_score heuristic with search depth 3
5) AB_Open: AlphaBetaPlayer using iterative deepening alpha-beta search and the open_move_score heuristic
6) AB_Center: AlphaBetaPlayer using iterative deepening alpha-beta search and the center_score heuristic
7) AB_Improved: AlphaBetaPlayer using iterative deepening alpha-beta search and the improved_score heuristic

For the current experiment the following three heuristics have been chosen:

1) The first one is depends on the current players available moves and the distance from the center of the board

```
def custom_score(game, player):
    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    d = float((h - y) ** 2 + (w - x) ** 2)
    my_moves = len(game.get_legal_moves(player))
    return my_moves * math.fabs(1 - math.sqrt(d))
```

2) The second one relies on the number of moves of each player

```
def custom_score_2(game, player):
    opponent_moves = len(game.get_legal_moves(game.get_opponent(player)))
    my_moves = len(game.get_legal_moves(player))
    all_moves = my_moves + opponent_moves + 0.00001
    return float(0.3 * math.fabs(my_moves - opponent_moves) / all_moves)
```

3) A third one is a custom function with some dependencies of current position of the players

```
def custom_score_3(game, player):
    w, h = game.width / 2., game.height / 2.
    y, x = game.get_player_location(player)
    d = float((h - y) ** 2 + (w - x) ** 2)

    y_, x_ = game.get_player_location(game.get_opponent(player))
    d_ = float((h - y_) ** 2 + (w - x_) ** 2)
    return float(0.1 * (math.sqrt((math.fabs(d - d_)**2 / (d + d_)))))
```

```
                    *************************
                         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    10  |   0    10  |   0
    2        MM_Open      8  |   2     9  |   1     8  |   2     8  |   2
    3       MM_Center     8  |   2     9  |   1    10  |   0    10  |   0
    4      MM_Improved    8  |   2     8  |   2     6  |   4     6  |   4
    5        AB_Open      6  |   4     3  |   7     4  |   6     5  |   5
    6       AB_Center     5  |   5     4  |   6     6  |   4     5  |   5
    7      AB_Improved    4  |   6     5  |   5     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:      67.1%        68.6%        70.0%        68.6%
```

Tab.1 winning rate

From ***Tab. 1*** we can see that our three custom heuristics show not very bad results especially *AB_Custom_2* heuristic that gives us **70%** winning rate and other two *AB_Custom_3 and AB_Custom_1* heuristics give **68.6%** each.

In addition, comparing the heuristic with best performance *AB_Custom_2* and *AB_Improved* we can conclude that *AB_Custom_2* shows overall a better result and

gives us 70.0% winning rate over 67.1% for *AB_Improved.* Although AB_Custom_2 shows a poor performance for *AB_Open* player.

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
5	AB_Open	6	4	3	7	4	6	5	5

It showed the best results for *Random* and *MM_Center* players as well as AB_Custom_3 heuristic:

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	10	0	10	0
3	MM_Center	8	2	9	1	10	0	10	0

Recommendations

AB_Custom_2 showed the best performance over other heuristics and should be used.

- Overall it gives the higher winning rate **70%** over **67.1%** AB_Improved
- It's more flexible due to an introduced coefficient that can be tuned in order to improve overall performance
- It's uncomplicated owing to having a small number of simple arithmetic operations

To sum up, carefully selected heuristics can significantly improve chances of winning from the results above we can see that even very simple heuristics can show very descent results.