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) Number of player moves

```
def custom_score(game, player):
    return float(len(game.get_legal_moves(player)))
```

2) The difference between players moves

```
def custom_score_2(game, player):
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    return float(len(game.get_legal_moves(player)) - len(opponent_moves))
```

3) 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         9  |   1    10  |   0    10  |   0      9  |   1
    2       MM_Open        7  |   3     6  |   4     7  |   3      8  |   2
    3      MM_Center       9  |   1     9  |   1     9  |   1      9  |   1
    4     MM_Improved      5  |   5     9  |   1     6  |   4      5  |   5
    5       AB_Open        5  |   5     8  |   2     4  |   6      6  |   4
    6      AB_Center       8  |   2     5  |   5     6  |   4      5  |   5
    7     AB_Improved      4  |   6     5  |   5     6  |   4      8  |   2
--------------------------------------------------------------------------
           Win Rate:      67.1%        74.3%        68.6%         71.4%
```

Tab.1 winning rate

From ***Tab. 1*** we can see that our three custom heuristics show not very bad results especially *AB_Custom_1* (number of moves) heuristics gives us **74.3%** winning rate and more computationally complicated *AB_Custom_3* heuristics function gives **71.4%** winning rate. In addition, we can see that our more complicated custom heuristics struggles with in two cases *AlphaBetaPlayer and MM_Imporoved player* except more simple heuristics which show more descent results:

```
 Match #   Opponent    AB_Improved   AB_Custom    AB_Custom_2   AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost    Won | Lost
 4      MM_Improved      5  |   5     9  |   1     6  |   4      5  |   5
 6       AB_Center       8  |   2     5  |   5     6  |   4      5  |   5
```

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