Heuristic Analysis

June 17, 2017

1 Heuristic Analysis

I implemented three different heuristic functions and run them with given heuristic 'AB_IMPROVED' that is equal to the difference in the number of moves available to the two players. Following sections describe the three functions and show the python code.

1.1 Heuristic 1: Weighted improved score

This heuristic function calculates legal moves for active and opponent players. Then it calculates a limit amount L as the division of total board cells by 2. Given L and the blank spaces B the function generates two weights: The weight_1 is division of L by B, while weight_2 is division of B by L. With this calculus if B is greater than L then weight_2 is greater than weight_1, otherwise weight_1 is greater than weight_2. weight_1 is used as linear weight for the active player moves, while weight_2 is used as linear weight for opponent player moves. This mechanism introduces a behaviour that penalize opponent player when the upperbound limit is exceeded. The method code lines are shown below:

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")
    if game.is_winner(player):
        return float("inf")

    active_player_moves = len(game.get_legal_moves(player))
    opponent_player_moves = len(game.get_legal_moves(game.get_opponent(player)))

    if active_player_moves is 0:
        return float("-inf")

    if opponent_player_moves is 0:
        return float("inf")

    available_positions = len(game.get_blank_spaces())

    total_positions = game.height * game.width
    limit = float(total_positions / 2)
    weight_1 = float(limit / available_positions)
```

```
    weight_2 = float(available_positions / limit)

    return float(weight_1 * active_player_moves - weight_2 * opponent_player_moves)
```

1.2 Heuristic 2: Active versus opponent moves

This heuristic function calculates player and opponent moves and return a ratio as division of player moves by opponent moves.

```
def custom_score_2(game, player):

    player_moves = len(game.get_legal_moves(player))
    if game.is_loser(player) or player_moves is 0:
        return float("-inf")
    opponent_moves = len(game.get_legal_moves(game.get_opponent(player)))
    if game.is_winner(player) or opponent_moves is 0:
        return float("inf")

    return float(player_moves / opponent_moves)
```

1.3 Heuristic 3: Manhattan distance

This function calculates coordinates of active player and coordinates of center point of the board. Then it calculates the Manhattan distance between then and return this value as score. This is a sperimental test in order to try a non-Euclidean distance between points as score function. Code lines are shown below:

```
def custom_score_3(game, player):

    if game.is_loser(player):
        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    active_x, active_y = game.get_player_location(game.active_player)
    center_x, center_y = int(game.width / 2), int(game.height / 2)
    active_distance = abs(active_x - active_y) + abs(center_x - center_y)

    return float(active_distance)
```

1.4 Overall results

Running the tournament.py script with given score function AB_Improved and new implemented heuristic functions i get the following output result:

```
In [ ]:                          ************************
                                      Playing Matches
                                 ************************
```

Match #	Opponent	AB_Improved		heuristic1		heuristic2		heuristic3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	6	4	10	0	9	1	9	1
2	MM_Open	8	2	6	4	4	6	6	4
3	MM_Center	7	3	8	2	10	0	6	4
4	MM_Improved	4	6	6	4	6	4	5	5
5	AB_Open	6	4	5	5	6	4	4	6
6	AB_Center	5	5	4	6	8	2	4	6
7	AB_Improved	3	7	5	5	7	3	5	5
	Win Rate:	55.7%		62.9%		71.4%		55.7%	

It seems a good choice the heuristic2 but we need to repeate the experiment to get a win percentage. Running the experiment for 15 times and calculating average of win percentages, it's a good choice to select one of firts two Heuristic function. Hence heuristic 1 (weighted improved score) or heuristic2 (active versus opponent moves). Following list shows the experiments with related win percentage for each heuristic. At the end averages are shown:

```
In [ ]: #EXP AB_IMPROVED heuristic1 heuristic2 heuristic3
```

1	60,00%	60,00%	67,10%	58,60%
2	55,70%	65,70%	58,60%	60,00%
3	58,60%	62,90%	65,70%	58,60%
4	57,10%	71,40%	64,30%	54,30%
5	61,40%	64,30%	60,00%	58,60%
6	55,70%	62,90%	71,40%	55,70%
7	60,00%	61,40%	57,10%	62,90%
8	60,00%	67,10%	70,00%	52,90%
9	62,90%	58,60%	58,60%	61,40%
10	54,30%	65,70%	61,40%	62,90%
11	58,60%	64,30%	62,90%	60,00%
12	61,40%	67,10%	70,10%	57,10%
13	47,10%	55,70%	61,40%	64,30%
14	55,70%	64,30%	67,10%	57,10%
15	64,30%	68,60%	64,50%	62,90%
AVG	58,19%	64,00%	64,01%	59,15%

N.B: This is a first version of alphabeta and minimax algorithms and it should be improved as well as the three heuristic functions described in this document

1.5 Recommendation

The heuristic2 function is recommended for following reasons:

1. The result table shows that it performs better in term of win rate (heuristic with best averaged win rate).
2. It's the function with the lowest complexity: a simple division to get a ratio.

3. It's the most intuitive: It helps algorithm to maximize the logic that active player should have more moves in comparison to opponent.