AIND PROJECT 2 Heuristic Analysis

In this project, heuristic scoring function plays a very import role in determine which player is going to be a winner. But if the 'SearchTimeout' occurred, the player always lose the game. So, a 'Good' heuristics must be

1) '**simple**' enough so that not much time is consumed to calculate the score of the board state, and
2) '**accurate**' enough so that it decides which state is advantageous for player, eventually leads to win.

In order to make the best heuristic, I think it is simplicity we need to consider preferentially. Because we use iterative deepening technique and the end of game state is obvious, one player can't move anywhere.

Next, we can think some information we can use in the Board class.

1) move_count
2) Number of the legal moves of each player.
3) Relative, and absolute location in the board, board state
4) Can my movement be limited by the other player?

By considering these, I tried to find a heuristic that could win AB_improved significantly, which use difference of own_moves and opp_moves as heuristic score.

■ Heuristics

1. Custom_1

```
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(1.5 * own_moves - 2*opp_moves)
```

This heuristic have different coefficients to own_moves and opp_moves compared to AB_improved to see what is more important feature.

2. Custom_2

```
return 2*own_moves / (opp_moves+1)
```

Just division of players moves. Not really different from AB_Improved and Custom_1 in terms of it gives high score when own_moves is greater than opp_moves.

3. Custom_3

```
w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)
y2, x2 = game.get_player_location(game.get_opponent(player))
center_dist = abs(w - y) + abs(h - x)
own_moves = set(game.get_legal_moves(player))
opp_moves = set(game.get_legal_moves(game.get_opponent(player)))
if game.move_count > 30:
    hide = bool(own_moves & opp_moves)
    return float(len(own_moves) - len(opp_moves) + center_dist / 3 + hide)
return float(len(own_moves) - len(opp_moves) + center_dist / 3)
```

Now I'm trying to use more information on the board like distance from the center, and move_count, whether player 2 can limit my moves or not. Also, I changed the strategy after 30 moves.

4. Custom_4

```
w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)
center_dist = abs(w - y) + abs(h - x)
own_moves = set(game.get_legal_moves(player))
opp_moves = set(game.get_legal_moves(game.get_opponent(player)))
hide = bool(own_moves & opp_moves)
return float(len(own_moves) - 2*len(opp_moves) + center_dist/3 - hide)
```

-hide, instead of + hide.

5. Custom_5

```
w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)
y2, x2 = game.get_player_location(game.get_opponent(player))
center_dist = abs(w - y) + abs(h - x)
player_dist = abs(y2 - y) + abs(x2 - x)
own_moves = set(game.get_legal_moves(player))
opp_moves = set(game.get_legal_moves(game.get_opponent(player)))
hide = bool(own_moves & opp_moves)

return float(len(own_moves) - len(opp_moves) + center_dist/3 + 1/player_dist + hide)
```

I also looked at the effect of distance between the two players.

6. Custrom_6

```
w, h = game.width / 2., game.height / 2.
y, x = game.get_player_location(player)
y2, x2 = game.get_player_location(game.get_opponent(player))
center_dist = abs(w - y) + abs(h - x)
player_dist = abs(y2 - y) + abs(x2 - x)
own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
count_weight = game.move_count // 10 + 1
return float(own_moves - opp_moves - center_dist/count_weight + 1/(player_dist+count_weight))
```

I adjusted the weights according to move_count.

■ Match results

Intel M-5Y10c CPU @ 0.80GHz 1.00 GHz

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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4  AB_Custom_5  AB_Custom_6
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random     950 |  50    950 |  50    935 |  65    936 |  64    933 |  67    927 |  73    950 |  50
```

1000 games with Random Player. Every heuristic shows approximately 95% of win rate. But why can't I always win? There is so called '**A move of God**', which seems like in poor position in our heuristics but ultimately lead to win.

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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4  AB_Custom_5  AB_Custom_6
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       MM_Open    75  |  25    63  |  37    73  |  27    66  |  34    67  |  33    67  |  33    70  |  30
    2     MM_Center    89  |  11    91  |   9    90  |  10    92  |   8    90  |  10    87  |  13    91  |   9
    3     MM_Improved  78  |  22    70  |  30    71  |  29    71  |  29    78  |  22    72  |  28    70  |  30
    4      AB_Center   62  |  38    59  |  41    67  |  33    59  |  41    61  |  39    68  |  32    64  |  36
--------------------------------------------------------------------------
          Win Rate:      76.0%        70.8%        75.2%        72.0%        74.0%        73.5%        73.8%

There were 78.0 timeouts during the tournament -- make sure your agent handles search timeout correctly, and consider increasing the timeout margin for your agent.

Your agents forfeited 2.0 games while there were still legal moves available to play.
```

This is 100 isolation games with MM player and AB_Center. The winning rate is still good enough, about 70% + alpha.

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

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3  AB_Custom_4  AB_Custom_5  AB_Custom_6
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       AB_Open    60  |  40    53  |  47    58  |  42    57  |  43    55  |  45    43  |  57    52  |  48
    2     AB_Improved  46  |  54    38  |  62    55  |  45    48  |  52    51  |  49    47  |  53    46  |  54
--------------------------------------------------------------------------
          Win Rate:      53.0%        45.5%        56.5%        52.5%        53.0%        45.0%        49.0%

There were 89.0 timeouts during the tournament -- make sure your agent handles search timeout correctly, and consider
 increasing the timeout margin for your agent.
```

However, I failed to find the heuristics that overwhelming AB_Open and AB_Improved. Custom heuristics show 40% to 60% of win rate in 100 matches so these heuristics can be considered as equally powerful.

I think this result can be interpreted for two reasons.

1. Other features except number of players valid moves have no effect on the game.
2. It is helpful but simple heuristics can go down deeper in game-tree in limited time. I think this inference is supported by the difference in win rate against to the Min Max Open heuristic and alphabeta Open heuristic.

■ Sugestion

As introduced in the video lecture, I also tried to implement the 'mirror' strategy that player1 take the center at the beginning and if player2 choose the first location that player1 can move next point symmetrically based on the center. e.g. (p1(3,3) 'center' -> p2(2,5) then -> p1(4,1) ...)). To do this, I modified some code in Board class, but I realize that if I change the codes except game_agent.py, then udacity submit fails. Anyway, if mirror strategy applied, against to player2 selecting first location randomly, since there are 8 possible mirror locations out of total 48 remaining board cells, the strategy would increase the win rate a little (1/2 * 1/6) compared to original heuristic function. So, I suggest applying mirror strategy to one of heuristics. It is very simple and It guarantees victory in possible situation. Here is one part of the code that I had tried.

1. In the heuristic function.

```
# Take the center.
if game.move_count == 1:    # player==player1.
    if y == center_y and x == center_x:
        return float("inf")
# If mirror movement is possible. At the Beginning.
if game.move_count == 3:
    opp_y, opp_x = game.get_player_location(opp)
    if y == 2 * center_x - opp_y and x == 2 * center_y - opp_x:
        return float("inf")

if game.mirror:
    opp_y, opp_x = game.get_player_location(opp)
    if y == 2 * center_x - opp_y and x == 2 * center_y - opp_x:
        return float("inf")
```

2. Board.play method in Isolation.py

```
while True:

    legal_player_moves = self.get_legal_moves()
    game_copy = self.copy()

    move_start = time_millis()
    time_left = lambda : time_limit - (time_millis() - move_start)
    curr_move = self._active_player.get_move(game_copy, time_left)

    # For Mirror Strategy
    if self.move_count == 3:
        center_y, center_x = self.width // 2, self.height // 2
        y, x = self.get_player_location(self._active_player)
        opp_y, opp_x = self.get_player_location(self._inactive_player)
        if y == 2 * center_y - opp_y and x == 2 * center_x - opp_x:
            self.mirror = True
```