AIND-Isolation - Heuristic Analysis

Key components

There are a few components in composing the three custom heuristic functions:

own_moves:

```python
own_moves = len(game.get_legal_moves(player))
```

own_moves is the number of legal moves that the agent has at its current position.

opp_moves:

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

opp_moves is the number of legal moves that opponent has at its current position.

own_dist:

```python
own_pos = game.get_player_location(player)
own_dist = ((game.width / 2 - own_pos[0]) ** 2 + (game.height / 2 - own_pos[1]) ** 2) ** 0.5
```

own_dist is the distance of the agent from the center of the game board.

game_stage:

```python
blank_spaces = len(game.get_blank_spaces())
game_stage = (game.width * game.height - blank_spaces) / (game.width * game.height * 0.59)
```

game_stage is a number from 0 to 1, where 0 means the game has just started, while 1 means the game has ended. To compute game_stage, we need the number of blank spaces on the board, and also the typical length of a game. `0.59` is a parameter I got after adding logs of the number of average moves in a game to `tournament.py`. The average number of moves per game is 29, hence `29/49 = 0.59`.

Heuristic 1

```python
if game_stage <= 0.2:
    return float(-own_dist)
elif game_stage <= 0.5:
    return float(own_moves - opp_moves - own_dist * 0.1)

else:
    return float(own_moves - opp_moves)
```

For `Heuristic 1`, it starts off the game by moving as close as possible to the center of the board. (i.e. minimum of `own_dist`)
Till the middle of the game stage, it will act by increasing own moves (mobility) while reducing the mobility of the opponent. When there is a tie, the effect of current position (`own_dist * 0.1`) will kick in, and the move which is closer to the center of the board will be picked.
In the later game stage, it will ignore its current position altogether and just maximize the difference between own moves and opponent moves.

Heuristic 2

```python
if game_stage <= 0.2:
    return float(-own_dist)
else:
    return float(own_moves - opp_moves - own_dist * 0.1)
```

`Heuristic 2` is similar to `Heuristic 1` except that current position will always be decisive when there is a tie in heuristic score, i.e. the closer one to the center of the board will get picked.

Heuristic 3

```python
if game_stage <= 0.2:
    return float(-own_dist)
else:
    return float(own_moves - opp_moves)
```

`Heuristic 3` is similar to `Heuristic 1` except that current position will not be considered after the early game stage (20% of the game).

Performance comparison of the three heuristics

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

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost   Avg Moves
    1       Random      46  |   4    48  |   2    45  |   5    49  |   1       25
    2       MM_Open     30  |  20    35  |  15    39  |  11    39  |  11       30
    3      MM_Center    46  |   4    42  |   8    40  |  10    38  |  12       28
    4     MM_Improved   31  |  19    35  |  15    37  |  13    40  |  10       30
    5       AB_Open     26  |  24    27  |  23    23  |  27    26  |  24       30
    6      AB_Center    27  |  23    30  |  20    27  |  23    24  |  26       28
    7     AB_Improved   22  |  28    24  |  26    24  |  26    24  |  26       29
--------------------------------------------------------------------------
           Win Rate:      65.1%        68.9%        67.1%        68.6%
```

Raw data

After having 50 matches against 7 different opponents each, `AB_Custom` with `Heuristic 1` has the highest winning rate of `68.9%`, which is also better than the reference agent `AB_Improved` (65.1%).

All three custom heuristics performs better than the reference agent `AB_Improved`, which shows that it is beneficial to move to the center of the game board in the early game stage, so the player can prevent getting stuck in a corner of the board.

`AB_Custom_2` performs the worst among the three agents, which shows that taking into account the current position all the time and try to get close to the middle of the board can be harmful at the later game stage.

Therefore, I will pick `Heuristic 1` as the best evaluation function because:

1. It has the highest winning rate of `68.9%`, and it performs better than the reference agent `AB_Improved` in 6 out of the 7 opponents.
2. It has taken the concept of game stage into account and adopt different strategies to play the game.
3. It has effectively treated the player and the opponent differently throughout the game to achieve to best result.
   Early game stage (<20% of the game): own board position
   Middle game stage (20%-50% of the game): own mobility, opponent mobility, own board position
   Late game stage (>50% of the game): own mobility, opponent mobility

Future improvement

1. Determine the average length of the game dynamically instead of hardcoding.

   From the data above, it is significant that the games played with the `Random` player are shorter (~25 moves) than with other players (~29 moves). It is because novice players tend to play in a worse way and lose earlier than stronger players, and vice versa.

   In order for the heuristic function to have an accurate `game_stage`, the average length of the game should be updated dynamically after facing the opponents for a few times; otherwise, it may not perform consistently when the strength of the opponent fluctuates.

2. Train the weight of each component with machine learning.

   Right now, the weight of component (`own_moves: 1; opp_moves: -1; own_dist: -0.1`) is decided after a few rounds of trial and error. It could be optimized by training on a neural network to obtain the optimized weights to achieve higher chance of winning.