In this project, I used 3 different heuristic functions for my agents.

Let's have some pre-defined terms in order to give a better understanding of my thoughts. You can easily know what the variable represents since the naming is straightforward.

```
own_moves = game.get_legal_moves(player) // list of legal moves of the current player
opp_moves = game.get_legal_moves(game.get_opponent(player)) // list of legal moves of
 the opponent player

base_score = len(own_moves) - len(opp_moves)

w, h = game.width / 2, game.height / 2
y, x = game.get_player_location(player) // current player location coordinate
a, b = game.get_player_location(game.get_opponent(player)) // opponent player locatio
n coordinate
```

Here, I have a `base_score` which is used in all the three heuristic functions since I think this is always a good measure for the game though sometime it's good to add a coefficient for it, here I just use 1.

There are common terms in my heuristic functions. `float((h - y) ** 2 + (w - x) ** 2)` and `float((h - a) ** 2 + (w - b) ** 2)` are inspired by the `center_score` function in `sample_players.py`, stands for **the distance from the player's current position to the center** and **the distance from the opponent's current position to the center**. These are considerable measures, intuitively like the `open_move` method.

We use some combinations of these two distance as penalty coefficients.

In the first heuristic function, I involve `1 / max(move_count, 1) * sum of these two distance` as the penalty. There is still one question here, since intuitively, as a beginner to the Isolation game, I don't think the sum of these two distance is a significative measure. Here I think I can leave this as an open question and dig it deeper later on.

```
return base_score - 1. / max(game.move_count, 1) * math.sqrt(
        float((h - y) ** 2 + (w - x) ** 2) + float((h - a) ** 2 + (w - b) ** 2))
```

In the second heuristic function, `1 / max(move_count, 1) * the distance from the player to the center` is used as the penalty.

```
return base_score - 1. / max(game.move_count, 1) * math.sqrt(
        float((h - y) ** 2 + (w - x) ** 2))
```

The last heuristic function is more straight-forward. We use the base_score minus the difference of **the distance from the player's current position to the center** and **the distance from the opponent's current position to the center**. This is the most simple one.

```
return base_score - float((h - y) ** 2 + (w - x) ** 2) + float((h - a) ** 2 + (w - b)
 ** 2)
```

I have tried several different params in my `tournament.py`.

Given the agent 300ms to search:

```
NUM_MATCHES = 20  # number of matches against each opponent
TIME_LIMIT = 300  # number of milliseconds before timeout
```

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	37	3	37	3	35	5	36	4
2	MM_Open	26	14	23	17	21	19	25	15
3	MM_Center	32	8	36	4	35	5	36	4
4	MM_Improved	29	11	30	10	24	16	26	14
5	AB_Open	17	23	21	19	22	18	21	19
6	AB_Center	21	19	22	18	24	16	25	15
7	AB_Improved	22	18	22	18	19	21	21	19
	Win Rate:	65.7%		68.2%		64.3%		67.9%	

Given the agent 350ms to search:

```
NUM_MATCHES = 20  # number of matches against each opponent
TIME_LIMIT = 350  # number of milliseconds before timeout
```

****
Playing Matches
****

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	35	5	37	3	39	1	36	4
2	MM_Open	27	13	27	13	31	9	31	9
3	MM_Center	36	4	34	6	35	5	38	2
4	MM_Improved	27	13	36	4	28	12	25	15
5	AB_Open	21	19	23	17	19	21	24	16
6	AB_Center	23	17	23	17	20	20	19	21
7	AB_Improved	18	22	19	21	23	17	16	24
	Win Rate:	66.8%		71.1%		69.6%		67.5%	

Compare to the default time limit setting:

```
NUM_MATCHES = 20  # number of matches against each opponent
TIME_LIMIT = 150  # number of milliseconds before timeout
```

****
Playing Matches
****

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	34	6	36	4	34	6	32	8
2	MM_Open	27	13	29	11	26	14	28	12
3	MM_Center	34	6	29	11	29	11	29	11
4	MM_Improved	26	14	31	9	23	17	29	11
5	AB_Open	23	17	23	17	21	19	24	16
6	AB_Center	25	15	20	20	20	20	14	26
7	AB_Improved	19	21	16	24	19	21	21	19
	Win Rate:	67.1%		65.7%		61.4%		63.2%	

Based on the result above, I'd like to choose my first heuristic function to recommend.

First, it gives us the best result. It's hard to have a better result compared with the AB_Improved agent, since with a relatively simpler heuristic function, such an agent can search deeper and, in most cases can give a better evaluation of the current game. Given a litte more time limit, my first heuristic outperforms the AB_Improved agent.

Also, my three heuristic all involved with using **the distance from the player's current position to the center** and **the distance from the opponent's current position to the center**. The time complexity of these three heuristic are same from the big O perspective. More ituitively, both `max(game.move_count, 1)` and `math.sqrt()` are not time-consuming operation overall.

Last, the first heuristic is easy to understand and implement. It's a combination of some well-known heuristic functions.