A summary of heuristics functions for 7X7 isolation game:

I used 5 different heuristic functions. I also tried to optimize these functions using brute force optimization (scipy.optimize.brute). Some of them ended up as failures, but two of them were more successful than ID_improved.

Heuristic#1:

```
open_moves = game.get_legal_moves(player)
self_x_std = np.std([x[0] for x in open_moves])
self_y_std = np.std([y[1] for y in open_moves])
score = alpha * self_x_std + beta * self_y_std
return score
```

The idea of this heuristic was to examine whether the legal moves available to use were well dispersed on the board or whether they were concentrated on one side of the board. My rationale for designing this heuristic was to prevent the agent from getting stuck on one side of the board. I did not get much luck with this, even after optimizing the values of alpha and beta.

Heuristic#2:

```
opponent_moves = game.get_legal_moves(game.get_opponent(player))
opponent_x_std = np.std([x[0] for x in opponent_moves])
opponent_y_std = np.std([y[1] for y in opponent_moves])
score = 1 / (alpha*opponent_x_std + beta*opponent_y_std + 0.0001)
```

The idea of this heuristic was very similar to heuristic #1. I thought if push the opponent into moving into one side only, I could get success, but this was not very successful. Several values of alpha and beta were tried through brute force.

Heuristic #3:

```
alpha = player.score_alpha
beta = player.score_beta
self_open_moves = len(game.get_legal_moves(player))
opponent_open_moves = len(game.get_legal_moves(game.get_opponent(player)))
score = (self_open_moves ** alpha) / ((opponent_open_moves + 0.01) ** beta)
```

This was a successful heuristic. What I did here was to look at the relationship between self open moves and opponent open moves in a nonlinear way. The reason why this was more successful than ID_improved was that the moves that gives us the same score in ID_improved gives different scores using this heuristic. For example:

Self_open_moves = 5

Opponent_open_moves = 1

ID_improved = 4 and heuristic#3 = 25

Self_open_moves = 4

Opponent_open_moves = 0

ID_improved = 4 and heuristic#3 = 1600

I also optimized for different values of alpha and beta and alpha=2 and beta=3 was found to be optimal.

Heuristic#4:

```
current_position = game.get_player_location(player)
opponent_position = game.get_player_location(game.get_opponent(player))
possible_moves = [(2, 1), (2, -1), (-2, 1), (-2, -1), (1, 2), (1, -2), (-1, 2), (-1, -2)]
opponent_possible_moves = [(opponent_position[0] + x[0], opponent_position[1] + x[1]) for x in possible_moves]
if current_position in opponent_possible_moves:
    score = 15
else:
    alpha = 2
    beta = 3
    open_moves = game.get_legal_moves(player)
    self_open_moves = len(open_moves)
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    opponent_open_moves = len(opponent_moves)
    score = (self_open_moves ** alpha) / ((opponent_open_moves + 0.01) ** beta)
```

Heuristic#4 builds on to heuristic#3. The idea here is to check whether our agent is blocking one of the opponent's possible moves. If this is the case we return 15, otherwise we return heuristic#3. In my initial trials, I thought this heuristic was promising but was not really that much better than heuristic#3.

Heuristic#5:

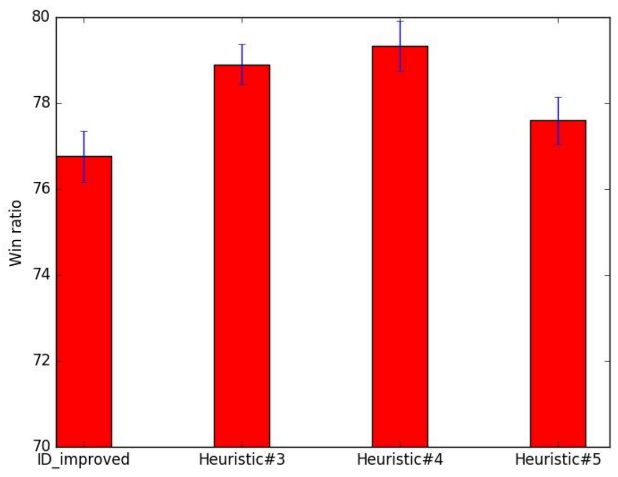
```
current_position = game.get_player_location(player)
opponent_position = game.get_player_location(game.get_opponent(player))
possible_moves = [(2, 1), (2, -1), (-2, 1), (-2, -1), (1, 2), (1, -2), (-1, 2), (-1, -2)]
opponent_possible_moves = [(opponent_position[0] + x[0], opponent_position[1] + x[1]) for x in possible_moves]
self_possible_moves = [(current_position[0] + x[0], current_position[1] + x[1]) for x in possible_moves]

if current_position in opponent_possible_moves:
    score = 15
elif opponent_position in self_possible_moves:
    score = -10
else:
    alpha = 2
    beta = 3
    open_moves = game.get_legal_moves(player)
    self_open_moves = len(open_moves)
    opponent_moves = game.get_legal_moves(game.get_opponent(player))
    opponent_open_moves = len(opponent_moves)
    score = (self_open_moves ** alpha) / ((opponent_open_moves + 0.01) ** beta)
```

Heuristic#5 builds on to heuristic#4. I additionally check here whether my opponent is actually blocking me and return a low negative score to prevent this situation. However, this was not helpful at all and performance was less thant heuristic#4:

Overall performance summary:

Here are the average of win ratios obtained by running tournament.py for 30 times. ID_improved 76.8%, heuristic#3 78.9%, heuristic#4 79.3%, heuristic#5 77.5%. Error bars are standard errors of the mean. I think heuristic#3 is the winner, because it has better performance than ID_improved. Secondly, it allows non-linear relationships between self open moves and opponent open moves to be represented. Finally even though heuristic#4 has slightly better performance than heuristic#3, heuristic#3 is computationally simpler and only slightly more complex than ID_improved. In conclusion, I am submitting heuristic#3 as the final heuristic.


Win ratio
ID_improved
Heuristic#3
Heuristic#4
Heuristic#5