Heuristic Analysis

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

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       9  |   1    10  |   0    10  |   0    10  |   0
    2       MM_Open      7  |   3    10  |   0     8  |   2     8  |   2
    3      MM_Center     9  |   1     7  |   3    10  |   0    10  |   0
    4     MM_Improved    9  |   1     8  |   2     6  |   4     8  |   2
    5       AB_Open      7  |   3     6  |   4     5  |   5     6  |   4
    6      AB_Center     6  |   4     5  |   5     5  |   5     4  |   6
    7     AB_Improved    4  |   6     6  |   4     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:      72.9%        74.3%        70.0%        71.4%
```

Heuristic plays a very important part in the game-play agent, it provides the evaluation of the next move. Here are three heuristics provided by sample_players.py.

open_move_score

The basic evaluation function described in lecture that outputs a score equal to the number of moves open for your computer player on the board. It's not a very efficient evaluation function because it only cares about the computer player's move, it knows nothing about opponent's move.

center_score

Outputs a score equal to square of the distance from the center of the board to the position of the player. This is only for testing, based on the evaluation above, it's probably the worst evaluation function. It makes computer player only care about moving to the center.

improved_score

The "Improved" evaluation function discussed in lecture that outputs a score equal to the difference in the number of moves available to the two players. It's the best evaluation function so far. The intuition here is for computer player to choose the best move that reduces the number of moves own by opponent. This makes a lot of sense even as a human player.

custom_score_2

This is a simple evaluation function that calculates the distance between 2 players, and then takes the value of the reciprocal function of the distance. The intuition here is to make computer player moving closely to the opponent.

custom_score

This one extends the improved_score and takes blank_spaces into account. The formula is like this (#my_moves - #opponent_moves) / #blank_spaces.

custom_score_3

This is the combination of custom_score and custom_score_2, and it takes the average. I hope to combined two will perform better than the single one, but it doesn't.

Final Thoughts

There are some interpretations from the tournament result.

- No doubt, heuristics will always beat the Random.
- MinimaxPlayer barely beats AlphaBetaPlayer. I think the main reason is due to the search depth (=3) of minimax algorithm. The minimax can't search deep enough to be able to reach the end of the game, so it can't foresee better moves ahead. In the meanwhile, AlphaBetaPlayer with iterative deepening would search more deeply within the given time, that's why it yields better results. For example, AB_Improved against AB_Center is 6:4, but AB_Improved against MM_Center is dominate 9:1.
- Very small difference between AlphaBetaPlayer with different heuristics. I think as long as iterative deepening helps dive deep and the heuristic takes 2 players into account, the faster the algorithm often yields better result because within limited time, the faster algorithm could explore more moves.
- Random starting position in tournament setting does cause performance fluctuations (Running tournament several times yield different results). Certain positions have great advantages in winning the game.