Heuristic Analysis
Isolation Game Playing Agent

I tested several different evaluation heuristics functions when creating my game playing agent. I noticed that most of the evaluation functions produced results from around 67 - 74 percent with a large variation of results when with the default value of 5 matches or 20 games. To normalize the scores and to protect against a small sample size, I used 250 matches or 1000 games. This produced much more stable results between runs. I also noticed that most evaluation functions performed relatively closely, between the high 60 and low 70% win rate. I was worried that my heuristic function was not working well. To test this theory, I created a heuristic designed to lose. This "bad" heuristic function took the opponent moves - 2 * my open moves which is the opposite of a relatively good function. This bad function still produced a win rate of 56.40%, while the opposite of this function (my open moves - 2 *the opponents open moves) produced a win rate of 71.20%.

In conclusions, the heuristic functions behaved relatively similar with results in the high 60's to low 70's percent win rate. This makes sense because most functions are some variation of the my open moves vs the opponents open moves. I did find it interesting that even when trying to lose, I was able to produce a better than even win rate of 56.40%. The best performing heuristic with a 72.40 win rate was simply taking my own moves / my opponents open moves.

ID Improved	72.06
custom_score_my_open_moves	69.41
custom_score_simple	72.30
custom_score_diff_in_mine_and_double_opponent	71.20
custom_score_diff_in_opp_and_double_mine	56.40
custom_score_diff_in_mine_and_double_opponent_chase_incase_of_tie	71.39
custom_score_diff_in_mine_and_double_opponent_run_away_incase_of_tie	70.69
custom_score_diff_in_mine_and_double_opponent_closest_to_center_tie	71.74
custom_score_divide_own_by_opponent	72.40
custom_score_diff_in_free_percent_of_board	71.44

ID Improved - this was the default test case: my open moves - opponent's open moves

custom_score_my_open_moves - simply return the my open moves without regard to the opponent's open moves

custom_score_simple - this is the same as ID Improved my open moves - opponent's open moves
custom_score_diff_in_mine_and_double_opponent - my open moves - 2 * opponent's open moves

custom_score_diff_in_opp_and_double_mine - this was designed to be a poor heuristic. It is the opponent's open moves - 2 * my open moves

custom_score_diff_in_mine_and_double_opponent_chase_incase_of_tie - this was designed to be similar to custom_score_diff_in_mine_and_double_opponent by taking my open moves - 2 * the opponent's open moves. In addition it would try to make a move close to the opponent in case of a tie. So it would first take the custom_score_diff_in_mine_and_double_opponent and double it then subtract the distance between my player and the opponent. This will have the effect of scoring moves close to the opponent better than moves further away, while still weighting the difference in our moves more importantly than our distance apart.

custom_score_diff_in_mine_and_double_opponent_run_away_incase_of_tie - this is designed to be similar to custom_score_diff_in_mine_and_double_opponent_chase_incase_of_tie but to rank moves further away from the opponent higher than moves closer to the opponent. In case of a tie, run away from the opponent.

custom_score_diff_in_mine_and_double_opponent_closest_to_center_tie - similar to the above two functions, but it would rank moves closer to the center of the board higher than moves further away.

custom_score_simple - my open moves - the opponent's open moves.

custom_score_divide_own_by_opponent - this was the best heuristic with a win rate of 72.40. It simply took my open moves / my opponent's open moves.

Reasons for picking custom_score_divide_own_by_opponent as the best heuristic function:

1. It has the best score overall and consistently performs better throughout all of my trial runs.

2. It takes into account both my open moves and the opponent's open moves. This is better than a strategy that only evaluates my open moves.

3. It is extremely fast. The function simply divides two numbers after querying both my moves and the opponent's open moves. While a lot of the functions I created used these numbers, several also computed either the distance between my player and the opponent or the distance between my player and the center of the board. Not only do these require the function to query both positions, but to also require a more expensive operation by computing the distance between two points. This function is simple and fast and only requires the same open moves properties that most of the other functions use. It should perform as fast or faster than all but one function (custom_score_my_open_moves which was my worst performing heuristic aside from the one designed to lose).

4. The goal of the game is to make the last move, limiting the opponents moves to 1 is a very desirable outcome. Since the opponent's open moves must be an integer greater than zero since zero moves would already produce a win with a score of infinity, the ratio rewards my moves that produce only one move for the opponent unless there is very strange circumstance. For example a 5 to 1 would score the same as a 10 to 2, 15 to 3 but those scenarios are not very likey, at least as far as I can tell with the dozens of games I have played by hand. There simply isn't enough of a variance to see a move that could produce a 5 to 1 or a 15 to 3. While this might be possible with a very large board, it isn't something that I have come across with a standard 7 * 7 board that we have tested against. When an opponent only has one open move, the denominator will be 1 and I will have a much higher score. If we compare this strategy with my open moves - 2 X the opponents open moves we can see a difference in the following. If the open moves are 3-1 or 5-2, my open moves - 2x the opponent's open moves would both score a 1. With my moves / the opponent's moves, they would score a 2 and a 3 and a 2.5 respectively. Incase of a tie, the program would default to the first option that was tested. Without making a much more complicated and expensive heuristic, I would rank 3-1 as a more desirable move, which would score higher with this heuristic. While this isn't a guaranteed better move based on the specific layout of the board, it would be a much better move without making the heuristic much more complex (taking into account a database of moves for example, like the IBM machine Deep Blue that won a chess match vs Gary Kasparov).