Heuristic Analysis

The 2 strategies that were used were Alpha-Beta pruning and Minimax. The minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, the game is usually a two player game and in this case of isolation the game is a two player game. The player then makes moves that maximized the minimum value of the opposition. Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm. It is an adversarial search algorithm that ignores the portion of the search tree that makes no difference to the final choice.

Custom_Score_1: This is the best heuristic function of three and is a combination of both custom score 2 and custom score 3. Here the heuristic is a weighted linear function were the opponent moves (opp_moves) is 2 times more relevant than the agent moves (own_moves). This is to influence the agent to prioritize its moves that minimize opposition moves in order to win.

Custom_Score_2: In this case the opponents moves (opp_moves) were 2 times more relevant than the agents moves (own_moves). In this case the reasoning is to minimize the moves from the opponent before maximizing its moves.

Custom_Score_3: In this case the weights are given priority to the opponent (opp_moves) to see how the agent does.

The script evaluates the performance of the custom score evaluation function again a baseline agent using alpha-beta search and iterative deepening called 'AB Improved'. The three AB Custom agents use ID and alpha-beta search with the custom score functions defined in game_agent.py

Match #	Opponent	AB Improved		AB Custom		AB Custom_2		AB Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	8	2	8	2	10	0	3	7
2	MM_Open	3	7	8	2	8	2	4	6
3	MM_Center	5	5	9	1	9	1	5	5
4	MM_Improved	5	5	4	6	6	4	5	5
5	AB_Open	8	2	4	6	6	4	5	5
6	AB_Center	8	2	6	4	4	6	7	3
7	AB_Improved	3	7	6	4	4	6	7	3
	Win Rate	57.1%		67.1%		64.3%		51.4%	

Analysis

AB_custom algorithm:

```
if game.is_loser(player):
    return float("-inf")

if game.is_winner(player):
    return float("inf")

own_moves = len(game.get_legal_moves(player))
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

score = game.utility(player) + float(own_moves - 2 * opp_moves)

return score
```

AB_custom is a combination of the strategies used in AB_custom_2 and AB_custom_3, and produces the best results. The algorithm consists of first getting the list of legal moves of the player (defined in own_moves) and the list of legal moves of the opponent player (defined in opp_moves). In this heuristic the game will calculate the number of player moves versus 2 times the value of the opponent moves which allows the player to anticipate to a greater depth the opponent moves. The other reason for a better score with this heuristic is because it uses an edge strategy as mentioned in the videos (16 solving 5 by 5 strategy) instead of a center based strategy.

From the results AB_custom_3 did the worst.

References:

Russell, S. J., & Norvig, P. (n.d.). *Artificial Intelligence*.