Narissa Kreutz
Artificial Intelligence NanoDegree

Heuristic Analysis

Summary of Results:

Ultimately I chose the heuristic referred to as AB_Custom_2 (which has since been moved to the custom_score function in game_agent.py as instructed) in the evaluation of tournament.py, primarily because it's performance was superior to the others.

Heuristic	Average Games Won	Standard Deviation
AB_Improved	61.9%	1.8%
AB_Custom (custom_score_2)	65.2%	4.4%
AB_Custom_2 (custom_score)	73.3%	4.7%
AB_Custom_3 (custom_score_3)	65.7%	2.0%

Table 1. Comparison of the average percentage of Games won between the "Improved" Heuristic and the Custom Heuristics

After three rounds of the "tournament" (5 matches per round, see "Tournament Results Summary" section for actual results), the average performance of AB_Custom_2 (custom_score) was 73.3% with a standard deviation of 4.7%. The other two heuristics won the game approximately 65% of the time, all 3 heuristics out-performed the "Improved" heuristic, which had an average of approximately 62% of games won.

This heuristic is a static, easily computed evaluation of the number of moves the player could make (multiplied by 8) minus the number of moves the opponent could make. The number 8 was chosen after multiple rounds of play using tournament.py with different integers used to multiply the variable, where 8 consistently gave better performance than integers such as 2, 4, and 6. No additional improvements in performance were seen when the multiplier was increased past 8.

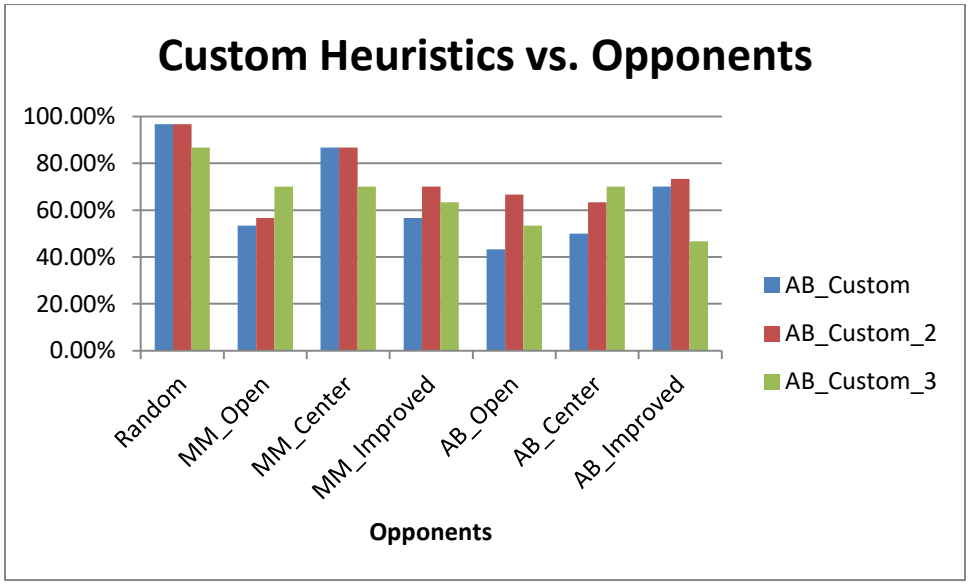

Figure 1. This bar graph shows the percentage of games each custom heuristic won against each opponent.

As shown in the bar graph above, AB_Custom_2 (custom_score, shown in red) is also fairly consistent in opposition to different types of players (the opponent which was strongest against it was MM_Open, otherwise it consistently won against any opponent more than 60% of the time). The other two heuristics were more variable against opponents with different strategies.

Python code snippet for custom_score (AB_Custom_2):

```
76
77      player_moves_left = len(game.get_legal_moves(player))
78      opponent_moves_left = len(game.get_legal_moves(game.get_opponent(player)))
79
80      return float((player_moves_left * 8) - opponent_moves_left)
81
```

The code was relatively computationally efficient, costing 45-50% of my laptop's CPU capacity (a 64-bit AMD E-350 processor), and 5.4 Mb of Memory to run. This was similar across all of the custom heuristics I implemented in game_agent.py, however custom_score_3 required more computational power as it includes a call to find the blank spaces left (in addition to number of player and opponent moves left) as well as a exponential multiplier.

Python code snippet for custom_score_3 (AB_Custom_3):

```
110
111     player_moves_left = len(game.get_legal_moves(player))
112     opponent_moves_left = len(game.get_legal_moves(game.get_opponent(player)))
113     blank_spaces = len(game.get_blank_spaces())
114
115     return float(((player_moves_left/blank_spaces)**2) - (opponent_moves_left/blank_spaces))
116
```

The last heuristic I experimented with divided the number of moves the player could make (again multiplied by 8) by the number of moves the opponent could make (plus 1 to avoid a "divide by zero" error).

Python code snippet for custom_score_2(AB_Custom):

```
42
43      player_moves_left = len(game.get_legal_moves(player))
44      opponent_moves_left = len(game.get_legal_moves(game.get_opponent(player)))
45
46      return float((player_moves_left * 8) / (opponent_moves_left + 1))
47
```

Narissa Kreutz
Artificial Intelligence NanoDegree

Tournament Results Summary:

Round #1:

```
**************************
  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	8	2
2	MM_Open	3	7	7	3	7	3	7	3
3	MM_Center	9	1	9	1	8	2	8	2
4	MM_Improved	5	5	4	6	5	5	6	4
5	AB_Open	5	5	6	4	6	4	3	7
6	AB_Center	5	5	5	5	6	4	7	3
7	AB_Improved	6	4	9	1	7	3	5	5
	Win Rate:	60.0%		71.4%		70.0%		62.9%	

Round #2:

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	8	2	9	1	10	0	9	1
2	MM_Open	5	5	3	7	4	6	7	3
3	MM_Center	9	1	8	2	10	0	6	4
4	MM_Improved	6	4	6	4	10	0	6	4
5	AB_Open	3	7	5	5	7	3	7	3
6	AB_Center	7	3	5	5	8	2	8	2
7	AB_Improved	5	5	7	3	7	3	4	6
	Win Rate:	61.4%		61.4%		80.0%		67.1%	

Round #3:

```
**************************
  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	9	1	9	1
2	MM_Open	7	3	6	4	6	4	7	3
3	MM_Center	9	1	9	1	8	2	7	3
4	MM_Improved	4	6	7	3	6	4	7	3
5	AB_Open	7	3	2	8	7	3	6	4
6	AB_Center	4	6	5	5	5	5	6	4
7	AB_Improved	5	5	5	5	8	2	5	5
	Win Rate:	64.3%		62.9%		70.0%		67.1%	