Build a game-playing agent report

I have conducted a couple of runs of the tournament between my three heuristic functions and the sample players. Here is a summary of the results, the full table results are at the end of the document.

	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3
Run 1 win rate, %	69.3	81.4	71.4	72.1
Run 2 win rate, %	71.7	76.7	73.6	75.9
Run 3 win rate, %	67.9	75.7	69.9	74.6

AB_Custom function

This function is based on the idea of having most places to go from the current state, but not depending on the number of current legal moves, but searching one step ahead and checking the number of legal moves for each legal move in the current position. So always looking for positions where we have the most options to go from. Similar approach of calculating the position score can be done by extending the improved_score function with additional for loops and search with a depth of 2-3 moves ahead, but I've found that the number of scanned positions is lower which is leading to more losses, most probably the performance is not optimal. So the most important part here is tracking of the number of legal moves for each square of the board. I have extended the board to keep an additional weight matrix with the size of the board and initialized at the beginning of the game with max possible moves. On each move that is applied on the board, I lower the number of moves for all squares that has the current move as legal neighbor. Compared to the second custom function, here I calculate the difference between the score for the current player and his opponent. I think this is very important thing for a good heuristic function – always find a position that is a balance between your max possible score and your opponents minimal score.

AB_Custom_2 function

This function is a simplified version of the first one. The idea is to show that for such type of two-player games, it looks better to always keep track not only of your position, but also of your opponent's state. A margin of 5-10% between this function and the first one shows the importance of this.

AB_Custom_3 function

Given the fact that I already got a pretty decent and consistent heuristic function, I wanted to try and add another factor to it. That is the state of the game depending on the number of moves made. So after some tests and checking statistically about the average number of moves until the game is finished, I have split the game in two parts. The first 10 moves (more than 40 blank squares on the board) I wanted my player to try and move to areas that have more free squares and possibly fill it. I based this idea on a 5x5 matrix with weights for each of the free squares around the player. It look like this:

1	**4**	**3**	**4**	**1**
4	3	2	3	4
3	2	Player in center	2	3
4	3	2	3	4
1	4	3	4	1

My idea is that in this 5x5 square, the player doing knight moves, can cover the whole space in multiple ways thus covering a big part of the board and always having some backup moves. I gave different weights for the moves based on the number of jumps needed to reach. Direct moves were awarded 4 points and the furthest with 1.

In the second part of the game, I simply fall back to the initial heuristic function with my best results, because the number of squares is getting low.

Conclusion

Given the fact that the first and the third heuristics result in similar win rate, I'd prefer the third one, because it combines the idea of searching for most possible moves in depth and searching for areas with more free squares. Generally the most important things that I have found through my experiments are:

- Always score the position in a way that is not only good for your player, but bad for the opponent. In most cases this results in searching for the difference between your score and your opponent's score.
- Try to split the game in parts – in the beginning of the game due to the extremely large number of possible positions, it is not so important to reach a significant depth of search, but rather have more positions analyzed. In the first moves of the game there's not many wrong decisions that could be taken
- The good heuristic function must be such that allows tuning. That's why I think that some sort of weights must be used for achieving a good result. Either use variables based only on the number of possible moves or give different values to different decision throughout the game.

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	20	0	19	1	20	0	20	0
2	MM_Open	13	7	20	0	14	6	18	2
3	MM_Center	17	3	18	2	18	2	19	1
4	MM_Improved	14	6	17	3	16	4	16	4
5	AB_Open	9	11	13	7	10	10	11	9
6	AB_Center	13	7	14	6	12	8	10	10
7	AB_Improved	11	9	13	7	10	10	7	13
	Win Rate:	69.3%		81.4%		71.4%		72.1%	

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	91	9	97	3	96	4	94	6
2	MM_Open	76	24	87	13	77	23	78	22
3	MM_Center	87	13	93	7	88	12	92	8
4	MM_Improved	79	21	82	18	77	23	77	23
5	AB_Open	52	48	61	39	61	39	62	38
6	AB_Center	68	32	58	42	65	35	67	33
7	AB_Improved	49	51	58	42	51	49	61	39
	Win Rate:	71.7%		76.6%		73.6%		75.9%	

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

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	35	5	38	2	40	0	37	3
2	MM_Open	28	12	30	10	29	11	31	9
3	MM_Center	36	4	37	3	36	4	36	4
4	MM_Improved	27	13	31	9	31	9	28	12
5	AB_Open	21	19	24	16	21	19	28	12
6	AB_Center	22	18	27	13	23	17	26	14
7	AB_Improved	21	19	25	15	15	25	23	17
	Win Rate:	67.9%		75.7%		69.6%		74.6%	