Evaluation Heuristic Analysis

The evaluation function of number of player moves available minus the number of opponent moves is a good starting evaluation function for isolation.

```
#my_moves - #opponent_moves
```

When a player has a large number of moves available, this is a good thing. The higher the number of player moves available, the more chance the player is to win. The evaluation function above also penalises boards where the number of opponent moves available is large.

This evaluation function also operates well within the allocated time limit for each move. It's important an evaluation function executes quickly to maximise the depth of the game tree searched.

Custom Score 1 - `custom_score`

After several rounds of testing the above evaluation function, it was found it could be improved by attaching a weight to the number of opponent moves (see below).

```
#my_moves - 4 * #opponent_moves
```

With the increased performance after attaching a weight to the number of opponent moves in custom_score, several different weights were tried to find a better result. Weights trialled including, 2x, 3x, squared, 4x, 1.5x and more. Each of the weights had a similar effect on the win rate but 4x seemed to give the most reliable outcomes.

Increasing the number of opponent moves by a factor of 4x further ensured the penalisation of boards where the opponent had more moves available. This simple function achieved an average of over 72% win rate when compared to other functions (see custom_score column below).

Trial	custom_score	custom_score_2	custom_score_3
1	65.7	65.7	68.6
2	78.6	65.7	70
3	70	71.4	71.4
4	70	75.7	68.6
5	75.7	80	67.1
6	80	71.4	71.4
7	74.3	67.1	68.6
8	74.3	71.4	68.6
9	65.7	65.7	68.6
Average	72.70	70.46	69.21

This function was chosen to be the main evaluation heuristic because of three main reasons, firstly, this evaluation function had a higher win rate than others tested (compared to `custom_score_2` and `custom_score_3`). Secondly, the function is easily understood by someone who is new to the problem, enabling them to expand on the function in the future. Finally, the function does not require much compute time to operate, thus allowing the game tree to be searched as far as possible within the allocated time frame.

Custom Score 2 - `custom_score_2`

A weight of 2x gave the second best results of the trialled weights.

```
#my_moves - 2 * #opponent_moves
```

This function managed to achieve a 71% win rate compared to other functions (see custom_score_2 column above).

Custom Score 3- `custom_score_3`

A weight of 3x was used for the final `custom_score` function. It averaged just under a 70% win rate compared to the other functions (see custom_score_3 column above).

```
#my_moves - 3 * #opponent_moves
```

An example of a bad evaluation function involved penalising the number of player moves available whilst favouring boards with a high number of opponent moves available.

```
#opponent_moves - #my_moves
```

When using this evaluation function, the win rate plummeted, as seen in the AB_Custom_2 column below.

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	8	2	10	0	0	10	10	0
2	MM_Open	6	4	7	3	0	10	8	2
3	MM_Center	7	3	9	1	0	10	8	2
4	MM_Improved	7	3	5	5	0	10	7	3
5	AB_Open	5	5	4	6	1	9	4	6
6	AB_Center	7	3	6	4	0	10	6	4
7	AB_Improved	4	6	4	6	0	10	5	5
	Win Rate:	62.9%		64.3%		1.4%		68.6%	

Future evaluation functions could be derived to achieve higher win rates within the allocated timeframe (2-seconds per game move).

The weights used for `custom_score`, `custom_score_2` and `custom_score_3` were discovered by trial and error. A more thorough approach such as using genetic algorithms to discover more ideal weights, Q-learning, or other reinforcement learning techniques to find the best possible evaluation function, however this was beyond my scope at the time of writing.