**Heuristic functions**

We evaluated 3 different custom heuristic functions:

1. AB_Custom:

- maximizes the number of open moves for the current player
- aggressively minimizes the number of open moves for the opponent
- maximizes the square of the distance between the two players

This strategy ensures that the player is 'protected' from its opponent's attacks by maximizing the player's number of open moves and the distance between the two players. At the same time, we ensure that the player maintains an active winning strategy by aggressively minimizing the number of open moves for the opponent.

2. AB_Custom_2:

- maximizes the number of open moves for the current player
- aggressively minimizes the number of open moves for the opponent

This is a more aggressive version of the 'improved_score' being used by AB_Improved. It does not maximize the distance between the two players.

3. AB_Custom_3:

- maximizes the number of open moves for the current player
- aggressively minimizes the number of open moves for the opponent
- minimizes the square of the distance between the active player and the center of the board

With this function, we ensure that the player maintains an active winning strategy by aggressively minimizing the number of open moves for the opponent while maximizing its own number of moves. In addition, we minimize the square of the distance between the player and the center of the board, to hopefully avoid the player being cornered on the edges of the board.

**Tournament Results**

To evaluate the goodness of our heuristics, we first ran 50 matches again each opponent, with a time limit of 150 ms before timeout.

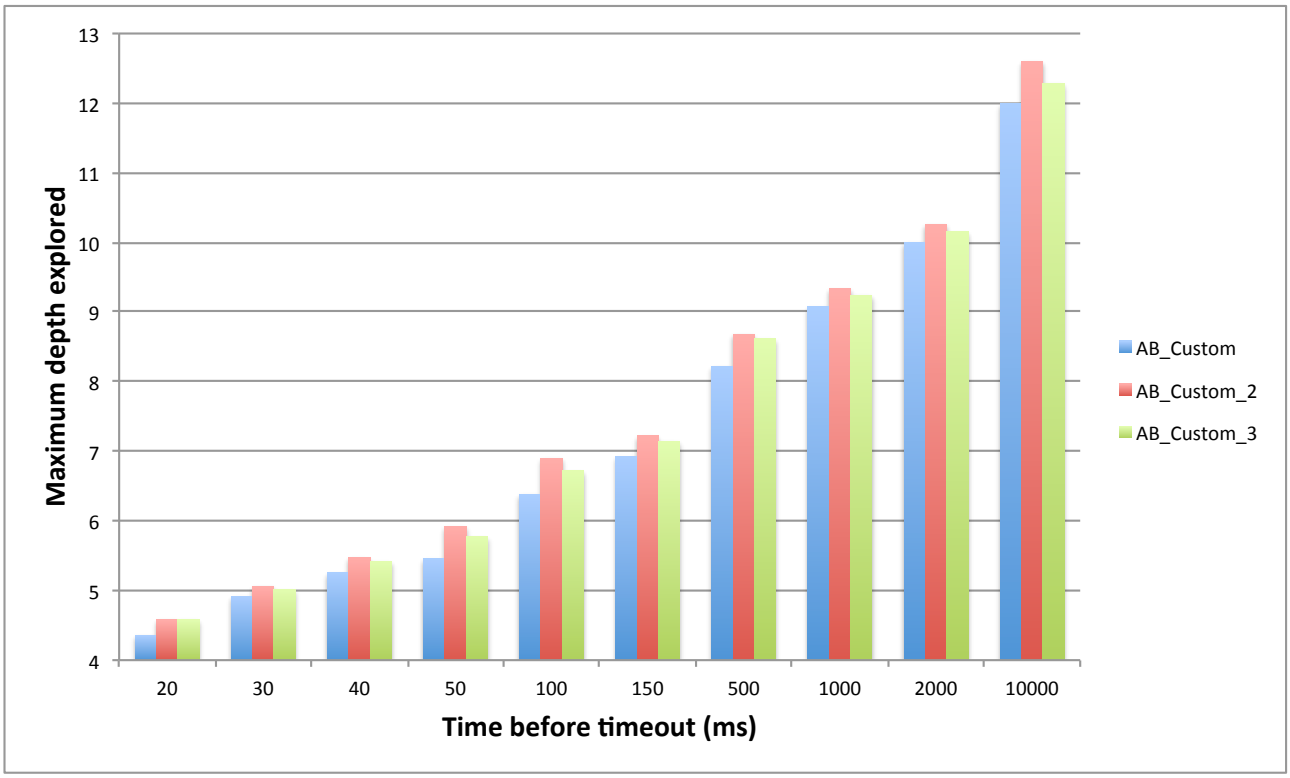
```
                      **************************
                           Playing Matches
                      **************************

 Match #    Opponent    AB_Improved    AB_Custom    AB_Custom_2   AB_Custom_3
                         Won  | Lost    Won  | Lost    Won  | Lost    Won  | Lost
    1        Random       49   |   1     47   |   3     46   |   4     47   |   3
    2        MM_Open      37   |  13     38   |  12     37   |  13     28   |  22
    3       MM_Center     42   |   8     45   |   5     43   |   7     42   |   8
    4     MM_Improved     34   |  16     35   |  15     36   |  14     33   |  17
    5        AB_Open      22   |  28     25   |  25     29   |  21     29   |  21
    6       AB_Center     29   |  21     33   |  17     26   |  24     25   |  25
    7     AB_Improved     24   |  26     25   |  25     26   |  24     27   |  23
--------------------------------------------------------------------------------
            Win Rate:      67.7%        70.9%        69.4%        66.0%
```

From the table above we can see that both AB_Custom and AB_Custom_2 have a higher averaged winning rate than AB_Improved, with win rates of 70.9% and 69.4%, respectively, compared to 67.7%.

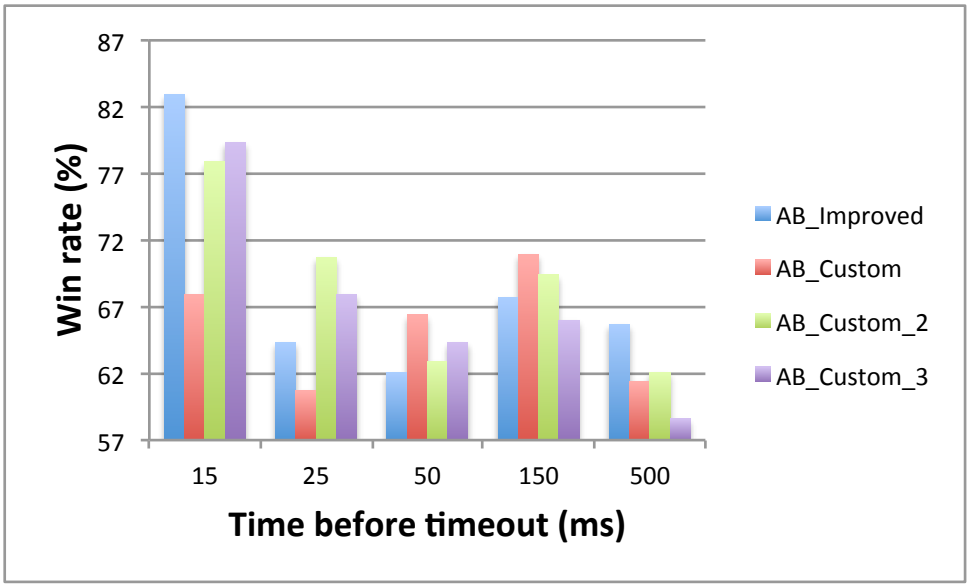
In addition, we see that AB_Custom_2 performs better against all of its opponents, while AB_Custom fails to consistently win against AB_Open and AB_Improved.

To further quantify the results from our heuristics, we first evaluated the depth reached by each of the heuristics during the first move of 50 randomly initialized matches. We compared the results across different times before timeout, from 20 ms to 10 s. Results are displayed below.




As we can see, AB_Custom_2 consistently reaches deeper levels of the search tree, for all of the evaluated timings.

Finally, we computed the win rate for different timings, averaged across 20 matches. As we can see, from our custom functions, AB_Custom_2 performs significantly better than AB_Custom for smaller timings, while AB_Custom slightly outperforms it for the larger timings of 50 ms and 150 ms.



In conclusion, we would chose AB_Custom_2 as a heuristic function for the following reasons:

- when evaluated with 150ms of time before timeout, it beats all of its opponents;
- it takes less time to compute and thus reaches deeper depths on the search tree;
- it is very performant for short computing times (i.e., < 50 ms) and still performs well for larger computing times, although AB_Custom might also be a valid alternative if large computational time is available.