Heuristic analysis

We implemented four heuristic functions to evaluate the score of the player with respect to the current game.

The first one is a sigmoid function defined as follows,

$$\frac{1}{1+\mathrm{e}^{-z}}, z = \#\text{my\_moves} - \#\text{opponent\_moves} \quad (1)$$

The second one is a linear function,

$$\#\text{my\_moves} - 2 * \#\text{opponent\_moves} \quad (2)$$

The last heuristic function is a weighted function in which *move_count* is the total plies up to the current game, *width* and *height* decide the board size.

$$\mathrm{w} * (\#\text{my\_moves} - \#\text{opponent\_moves}), \mathrm{w} = \frac{\text{move\_count}}{width * height} \quad (3)$$

We did some experiment with tournament.py script and the performance comparison of our customized functions and the *improved_score* heuristic from the sample_players.py are shown below.

Table 1

	(1)	(2)	(3)
improved_score	65.71%	57.14%	64.29%
Custom_score	62.14%	67.14%	67.86%

We can notice that heuristic function (1) is almost tied with *improved_score*. While (2) and (3) are better than *improved_score*.

In heuristic function (1), we just scale up and down the difference of the number of moves between one player's and its opponent's, which is analogue to what *improved_score* does. Thus the winning rate of (1) is not more than *improved_score*'s. Otherwise, in (2) and (3), we not only consider the relative number of potential moves between the two players, but also do we pay attention to the depth of the current state in the search tree.

As for heuristic function (2), we apply a factor to the *#opponent_moves*. When there are two nodes and node_1 is deeper than node_2 in the search tree which *#my_moves - #opponent_moves* is equal and the score of node_1 would be larger than node_2's since the *#opponent_moves* of node_1 is less than node_2's (the potential moves would be smaller as the game is going on in general). In this way, we are intended to choose node_1 as the best goal with larger winning chance. It makes sense because the opponent in node_1 has less time (i.e., left moves in the future) to impact the passive situation. The function (3) has the same advantage with (2), so it can beat *improved_score* but just a little.

Neither function (2) nor (3) needs any look ahead search, therefore they are speedy and efficient. The complexity of Function (2) is lower (made of once addition and once multiplication) while compared to once addition and three times multiplication for function (3). Besides that, the winning rate of heuristic (2) exceeded *improved_score*'s by up to 10% and heuristic (2) is the most powerful one among our heuristic functions, so we choose function (2) as our *custom_score*.