Heuristics Analysis

Summary of Heuristic Functions

- `custom_score`
    - the differences between the number of possible destinations in the next `k` steps for each player.
    - The idea is explained below
- `custom_score_2`
    - the number of possible destinations in the next `k` steps for the current player.
    - The idea is explained below
- `custom_score_3`
    - the number of legal moves of the current player that are also legal moves of the opponent.
    - The idea is that we let the current player always move to opponent's possible locations, such that the opponent has less legal moves. Because the current player goes first, if the opponent only has one legal move and the current player can actually take that destination before the opponent, then the opponent will lose.

Explanation of First Two Score Functions

According to the alpha go paper (Silver et al, 2016), a perfect information game has an optimal valuation function *v(s)* for each state *s* that accurately evaluate the state and guides to a winning action.

The heuristics used in the `sample_players` uses the number of legal moves in the next step, here I go a little further: I am calculating how many possible destinations the current player can move to in the next `k` steps. The larger the `k`, the better we know the number of possible destinations between the current state and the final state. However, the larger the `k`, the longer time we need for the evaluation.

Note that, given the current location on the board, if the player can only move in "L-shape", there may be locations on the board that the current player can never move to. So, the motivation is to calculate what are possible locations on the board that the current player can actually move to. To calculate the number of possible destinations, I start with the current state and use Breath-First-Search (BFS) to expand other possible destinations on board. The BFS stops when the cumulative path length is larger than `k`.

The `custom_score` and `custom_score_2` uses this idea. The `custom_score_2` calculates the number of possible legal moves of the current player. The `custom_score` made an improvement over `custom_score_2` : it calculates the differences of the possible legal moves in the next `k` steps between two players. (Similar to the `AB_Improved` v.s. `AB_Open`)

The parameter `k` can be chosen to balance the available time and evaluation accuracy. From my testing, when `k==2` or `k==3` , the `custom_score` outperforms `AB_Improved` , but when `k == 5` , the `custom_score` cannot outperforms `AB_Improved` . Note that `k==0` is equal to `game.get_legal_moves()`

Results (`k==3`)

I have run the `tournament.py` for three times. Overall, the `custom_score` outperforms the `AB_Improved` , although my `custom_score` does not outperform the `AI_Improved` in every situation. The `custom_score_2` can also outperform the `AB_Improved` in several rounds.

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     8  |   2     9  |   1     8  |   2
2        MM_Open       6  |   4     7  |   3     6  |   4     6  |   4
3       MM_Center      7  |   3     7  |   3     5  |   5     7  |   3
4      MM_Improved     7  |   3     7  |   3     5  |   5     5  |   5
5        AB_Open       4  |   6     6  |   4     6  |   4     4  |   6
6       AB_Center      6  |   4     7  |   3     6  |   4     6  |   4
7      AB_Improved     5  |   5     5  |   5     5  |   5     5  |   5
--------------------------------------------------------------------------
Win Rate:             62.9%        67.1%        60.0%        58.6%
```

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      6  |   4     8  |   2     9  |   1     7  |   3
    2        MM_Open     6  |   4     7  |   3     7  |   3     7  |   3
    3       MM_Center    9  |   1    10  |   0     8  |   2     7  |   3
    4      MM_Improved   6  |   4     6  |   4     5  |   5     5  |   5
```

```
     5         AB_Open       5   |   5       6   |   4       5   |   5       6   |   4
     6        AB_Center      5   |   5       6   |   4       9   |   1       3   |   7
     7       AB_Improved     6   |   4       6   |   4       3   |   7       2   |   8
--------------------------------------------------------------------------
          Win Rate:       61.4%         70.0%         65.7%         52.9%
```

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      6   |   4    10  |   0     6  |   4     8  |   2
     2        MM_Open     5   |   5     6  |   4     7  |   3     3  |   7
     3       MM_Center    7   |   3     7  |   3     6  |   4     5  |   5
     4      MM_Improved   7   |   3     7  |   3     8  |   2     6  |   4
     5        AB_Open     7   |   3     5  |   5     6  |   4     4  |   6
     6       AB_Center    5   |   5     5  |   5     5  |   5     5  |   5
     7      AB_Improved   5   |   5     7  |   3     6  |   4     4  |   6
--------------------------------------------------------------------------
          Win Rate:       60.0%        67.1%        62.9%        50.0%
```

Conclusion

My `custom_score` and `custom_score_2` evaluation function can outperforms the `AB_Improved` in many cases. The reason my evaluation function can outperform the `AB_Improved` is that my evaluation function provides a better estimation of the state s than the `AB_Improved` one.

My evaluation functions look forward the next `k` steps, and calculate the number of possible destination in all these `k` steps. They provide a better evaluation of the current state than the number of next legal moves. The difference between the number of possible destinations for each player is a better evaluation than the number of possible destinations of a single player. This may be because of the competitive nature of the isolation game.

To achieve a higher probability of winning, I suggest using my `custom_score` function. There are several reasons for this recommendation:

1. It is the best score. It is consistently better than the `AB_Improved`
2. It is fast and easy to scale. The parameter `k` can be tuned according to the computer than runs the program. If the computer is super fast, when we can simply increase the number `k` and can get better valuation score. If the computer is slow, we can decrease `k`. We don't need to change other parts of the code, and this score is quite scalable. In contrast, other

scores don't have the potential to provide a better evaluation of the current board state, even if the program is running on a fast computer.

3. This is easy to implement. The main idea is similar to limited depth BFS, and it is a natural extension of the `AB_Improved` score