My code:

```
  def custom_score(game, player):
      if game.is_loser(player): return float("-inf")
      if game.is_winner(player): return float("inf")
      own_moves = len(game.get_legal_moves(player))
      opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
      return float(-opp_moves) float(len(game.get_legal_moves(player)))
  def custom_score_2(game, player):
      if game.is_loser(player): return float("-inf")
      if game.is_winner(player): return float("inf")
      own_moves = len(game.get_legal_moves(player))
      opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
      return float(2*own_moves - opp_moves)
  def custom_score_3(game, player):
      if game.is_loser(player): return float("-inf")
      if game.is_winner(player): return float("inf")
      own_moves = len(game.get_legal_moves(player))
      opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
      return float(own_moves - 2*opp_moves)
```

`custom_score` is only focus on reducing opponent's move, which is inverse to `open_move_score` ;

`custom_score2` and `custom_score3` are variants of `improved_score` . custom_score2 is more favored to increase own_moves, while custom_score3 is more favored to decrease opp_moves.

A full run costs 15 minutes. My result is:

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

 Match #    Opponent     AB_Improved    AB_Custom    AB_Custom_2  AB_Custom_3
                          Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1         Random      10  |   0     10  |   0     7  |   3     9  |   1
    2         MM_Open      8  |   2      7  |   3     6  |   4     5  |   5
    3        MM_Center     8  |   2      8  |   2     9  |   1     9  |   1
    4      MM_Improved     7  |   3      6  |   4     8  |   2     9  |   1
    5         AB_Open      3  |   7      4  |   6     4  |   6     6  |   4
    6        AB_Center     5  |   5      7  |   3     4  |   6     5  |   5
    7      AB_Improved     5  |   5      5  |   5     6  |   4     3  |   7
--------------------------------------------------------------------------
           Win Rate:      65.7%        67.1%        62.9%        65.7%
time cost:  858.7957961559296
```

How to read table? e.g. the first slot: AB_Improve wins 10 and lost 0 over Random

Because the first 2 move is random, the result may have some fluctuation.

For player type, AlphaBetaPlayer > MinimaxPlayer > RandomPlayer. This is expected, because RandomPlayer has no optimization, and AlphaBetaPlayer has iterative deepening that can think "deeper" than depth-limited MinimaxPlayer.

For evaluation functions, the tested 4 functions are quite similar, considering that the first 2 move is random and may cause fluctuation.

I would recommend `custom_score` because the following 3 reasons:

1. rate: it has the **best overall winning rate**
2. complexity: this is actually very straightforward. Because the winning criterior of the eliminate game is no legal move of opponent, it doesn't matter how many moves I have.
3. depth: because opp_moves is considered, its actual depth is **one layer deeper** than `open_score` if given the same depth limitation.