Isolation Heuristic Analysis

Mingyi Zhang

Definition

I defined three heuristics `custom_score()`, `custom_score_2()` and `custom_score_3()`.

- `custom_score()`: return the difference between the numbers of legal moves of the active player `act_moves` and inactive player `inact_moves`, respectively. In order to differ from the predefined `improved_score()`, I introduced pre-factors in front of `act_moves` and `inact_moves`. The pre-factor of `act_moves` is 0.4, a bit smaller than 0.6 of `inact_moves`, which makes the player trying more to minimize the number of opponent's possible moves.
- `custom_score2()`: calculate the number of possible moves of the active player and the average number of possible moves of the opponent in the next step. The score returns the difference between the two numbers.
- `custom_score3()`: return the numbers of legal moves of the active player. The player only trying to maximize its possible move.

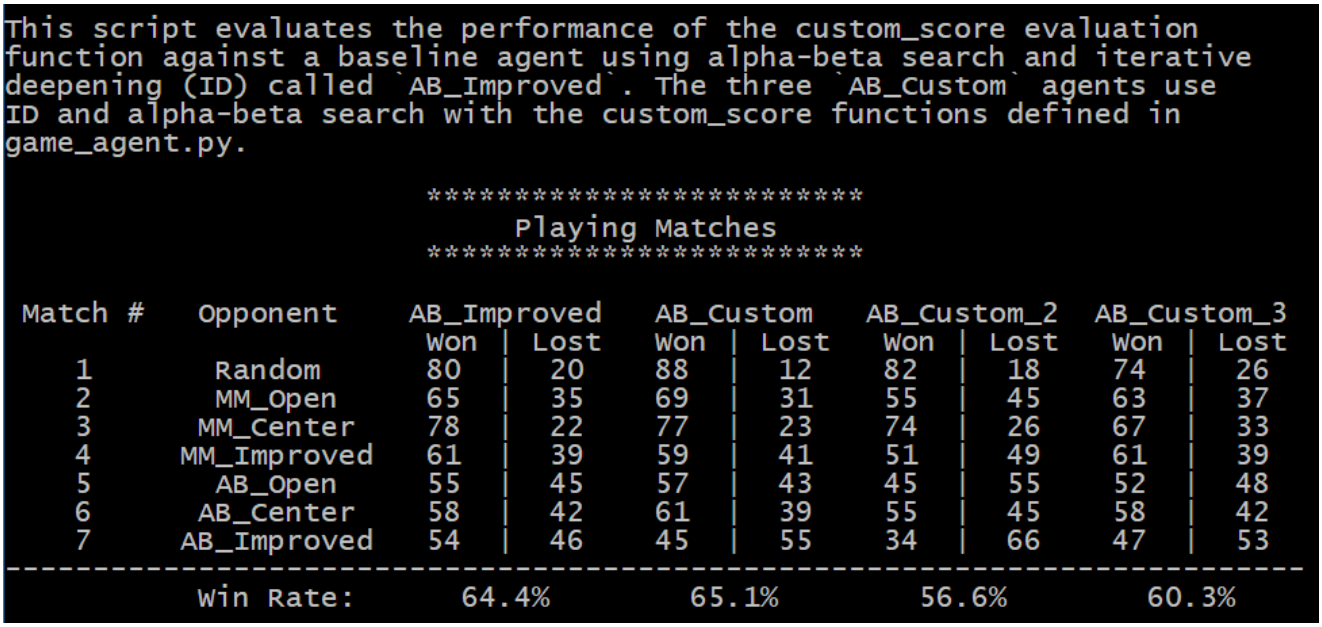
The following is an example of the implementation of `custom_score()`

```
def custom_score(game, player):
    if game.is_loser(player):
        return float('-inf')
    if game.is_winner(player):
        return float('inf')
    # number of possible moves for active player and inactive player
    act_moves = len(game.get_legal_moves(player))
    inact_moves = len(game.get_legal_moves(game.get_opponent(player)))

    return float(0.4 * act_moves - 0.6 * inact_moves)
```

Result

I set the parameter `NUM_MATCHES`, number of matches against each opponent, to be 50 in the `tournament.py`, so in total, each heuristic has 100 matches with each predefined heuristics. The result is



```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                        **************************
                             Playing Matches
                        **************************

 Match #    Opponent    AB_Improved    AB_Custom    AB_Custom_2  AB_Custom_3
                         Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1        Random      80  |  20    88  |  12    82  |  18    74  |  26
    2        MM_Open     65  |  35    69  |  31    55  |  45    63  |  37
    3       MM_Center    78  |  22    77  |  23    74  |  26    67  |  33
    4     MM_Improved    61  |  39    59  |  41    51  |  49    61  |  39
    5        AB_Open     55  |  45    57  |  43    45  |  55    52  |  48
    6       AB_Center    58  |  42    61  |  39    55  |  45    58  |  42
    7     AB_Improved    54  |  46    45  |  55    34  |  66    47  |  53
----------------------------------------------------------------------------
           Win Rate:      64.4%        65.1%        56.6%        60.3%
```

Figure 1: `tournament.py` results with 100 matches each and 150ms time limit.

We can find that

- the player `AB_Custom` with heuristic `custom_score()` has higher Win Rate than `AB_Improved`.
- `AB_Custom` players the best against all opponents except `AB_Improved`. It might due to the fluctuation, because the self-play between two `AB_Improved` have the similar Win Rate as `AB_Improved` against `AB_Custom`.

Recommendation

I recommend `custom_score()` because it has the best Win Rate. It is simple enough to explain and it is fast to compute.