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 and inactive player.
- `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 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(act_moves - inact_moves)
```

Result

I set the parameter `NUM_MATCHES`, number of matches against each opponent, to be 20 in the `tournament.py`, so in total, each heuristic has 40 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      34  |   6    37  |   3    27  |  13    33  |   7
    2       MM_Open     28  |  12    25  |  15    23  |  17    23  |  17
    3      MM_Center    33  |   7    33  |   7    25  |  15    27  |  13
    4     MM_Improved   25  |  15    23  |  17    21  |  19    28  |  12
    5       AB_Open     16  |  24    21  |  19    17  |  23    21  |  19
    6      AB_Center    24  |  16    26  |  14    21  |  19    27  |  13
    7     AB_Improved   21  |  19    17  |  23    15  |  25    18  |  22
--------------------------------------------------------------------------
          Win Rate:      64.6%        65.0%        53.2%        63.2%
```

Figure 1: `tournament.py` results with 40 matches each and 150ms time limit.

We can find that the player `AB_Custom` with heuristic `custom_score()` outperforms the baseline player `AB_Improved`. `AB_Custom` plays the best against all opponents.

Recommendation

I would recommend `custom_score()` because it has the best win rate. It consider the active and inactive player equally. And it is simple enough to explain and it is fast to compute.

```
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      77  |  23    81  |  19    74  |  26    81  |  19
    2       MM_Open     61  |  39    62  |  38    56  |  44    63  |  37
    3      MM_Center    73  |  27    72  |  28    74  |  26    75  |  25
    4     MM_Improved   62  |  38    62  |  38    51  |  49    64  |  36
    5       AB_Open     54  |  46    50  |  50    49  |  51    48  |  52
    6      AB_Center    61  |  39    60  |  40    46  |  54    56  |  44
    7     AB_Improved   51  |  49    50  |  50    46  |  54    48  |  52
--------------------------------------------------------------------------
          Win Rate:      62.7%        62.4%        56.6%        62.1%
```

Figure 2: `tournament.py` results with 100 matches each and 150ms time limit.