Isolation Heuristic Analysis

Mingyi Zhang

Definition

I defined three heuristics `custom_score()`, `custom_score_2()` and `custom_score_3()`. All heuristics are given by the difference between the number of legal moves of active player and inactive player. The general formula of the heuristics H

$$H = a \times N_{\text{active}} - b \times N_{\text{inactive}}$$

where a and b are two parameters that $a + b = 1$; N_{active} is the number of legal moves of the active player, while N_{inactive} is of the inactive player. For the three heuristics, I set a and b as

- `custom_score()`: $a = 0.05, b = 0.95$. The number of legal moves of the inactive player dominants the heuristic. The player tends to consider the move that the opponent has fewer moves.
- `custom_score2()`: $a = 0.5, b = 0.5$. The numbers of legal moves of both the active player and the inactive player have the same contribution to the heuristic. Player will maximize the number of its own moves while minimize the number of the opponent's moves.
- `custom_score3()`: $a = 0.95, b = 0.05$. The number of legal moves of the active player dominants the heuristic. The player tends to play the move that maximize the possibility of its own moves.

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(0.05 * act_moves - 0.95 * 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      36  |   4    35  |   5    33  |   7    27  |  13
    2       MM_Open     25  |  15    29  |  11    29  |  11    30  |  10
    3      MM_Center    29  |  11    31  |   9    33  |   7    28  |  12
    4     MM_Improved   27  |  13    23  |  17    27  |  13    23  |  17
    5       AB_Open     19  |  21    23  |  17    21  |  19    20  |  20
    6      AB_Center    24  |  16    23  |  17    23  |  17    27  |  13
    7     AB_Improved   19  |  21    19  |  21    23  |  17    20  |  20
--------------------------------------------------------------------------
          Win Rate:      63.9%        65.4%        67.5%        62.5%
```

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

We can find that

- the win rates of all players are around 65%.
- the player `AB_Custom_2` with heuristic `custom_score2()` outperforms the baseline player `AB_Improved`. `AB_Custom_2` plays the best against all opponents.

Recommendation

I would recommend `custom_score2()` 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      86  |  14    85  |  15    82  |  18    87  |  13
    2       MM_Open     67  |  33    67  |  33    68  |  32    69  |  31
    3      MM_Center    79  |  21    73  |  27    82  |  18    75  |  25
    4     MM_Improved   66  |  34    69  |  31    63  |  37    66  |  34
    5       AB_Open     53  |  47    54  |  46    55  |  45    51  |  49
    6      AB_Center    58  |  42    56  |  44    63  |  37    61  |  39
    7     AB_Improved   50  |  50    57  |  43    55  |  45    55  |  45
--------------------------------------------------------------------------
          Win Rate:      65.6%        65.9%        66.9%        66.3%
```

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