Heuristics analysis

custom_score

This function calculates a total number of moves available to the player several levels deep. It takes a total number of moves available immediately, then adds a number of moves available from each of those moves, then adds a total number of moves available from each of those moves etc.

This function works fast enough because it doesn't "project" each move onto the game board, but instead works directly with blank spaces available on the board. See 'game_agent.py:deep_moves_available' function for implementation details. The final score is calculated as # of own moves - K*# of opponent moves, normalized over # of blank spaces available.

Optimizations:

- K – coefficient K > 1 gradually becomes greater towards the end of the game
- While calculating the total number of available moves, each move counts as 1. But border moves (moves that land on the edge of the board) count as 0.5. This way score becomes higher if a player has more room closer to the center of the board.
- Search deeper towards the end of the game

The intuition behind this heuristic is that it's better than AB_Improved because it not only evaluates current position, but also potential future positions of the player. Key factor here was to make it work fast enough. Speed was achieved by making calculations directly on the set of blank spaces available and by limiting the depth factor to 2-4. This heuristic yields about 80% wins.

custom_score_2

This simple function calculates score as
(# of own available moves) - K*(# of opponent available moves), where K is randomized coefficient. This randomization only slightly improves the performance of the score function, comparing to AB_Improved. This heuristic yields about 72% wins.

custom_score_3

This function works similar to custom_score_3, but it penalizes border moves. Penalizing border moves alone only slightly improves the performance comparing to AB_Improved. This heuristic yields about 70% wins.

Typical tournament

```
                        ************************
                            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    10  |   0     9  |   1
         2        MM_Open      8  |   2     8  |   2     7  |   3     6  |   4
         3       MM_Center    10  |   0    10  |   0     9  |   1     8  |   2
         4     MM_Improved     9  |   1    10  |   0     9  |   1     8  |   2
         5        AB_Open      5  |   5     7  |   3     8  |   2     5  |   5
         6       AB_Center     5  |   5     6  |   4     5  |   5     5  |   5
         7     AB_Improved     3  |   7     7  |   3     5  |   5     6  |   4
      ----------------------------------------------------------------------------
-
                  Win Rate:      71.4%        82.9%        75.7%        67.1%
```

Heuristics tried, but didn't work

Symmetry

Finding symmetry didn't work well. I created several "mutators", which can rotate the board 90 degrees, flip it diagonally. Experiments showed that player moves are rarely symmetrical (especially when some cell on the initial board are randomly marked as not available), therefore looking for symmetry is not worth CPU time. See get_mutation_hashes function for more details.

Finding longest available path

Function 'take_longest_path' calculates the longest path available to the player. I.e. it traces the maximum number of moves this player can take from his current location before getting blocked. The idea was to use this function towards the end of the game, e.g. when only 1/3 of cells is still blank. This strategy, however, didn't perform well in the tournament and was discarded.

Project optimizations

File tournament.py was modified in order to make tournament run faster. It spawns multiple processes to run each round of competition. It adapts to the number of available cores on the computer where it's running. The code is in https://github.com/olehb/AIND-Isolation/tree/tournament_pooled