Udacity Isolation Game Agent

In this project students were asked to implement heuristics that would lead to agents that could win at a chess Knight movement isolation game. To find a good heuristic three general categories were explored using combinations of 4 different subscores. Listed below the subscores were

- **Number of moves available for active player (player_moves)**
- **Number of moves available for inactive player (opp_moves)**
- **Cartesian distance from center of board (center)**
- **Cartesian distance from other player (player_distance)**

The values of the first two subscores are directly tied to the final objective of Isolation, to force a turn where the player has no moves left. The other two though were created based on an intuition of the game. In Isolation it may be beneficial to stay in close proximity to the other user, something that will likely block them on over time, or inhibit their chances to move in the future. It also may be likely beneficial to stay towards the center of the board, as being near an edge, and especially corner, would severely limit the number of move options available.

Given these subscores the three metrics tested were

- **Player_moves**
- **Player_moves - opp_moves**
- **A*player_moves - B*opp_moves - C*center - D*player_distance**

In the final heuristic all variables were normalized to one, except the coefficient A,B,C, and D, which were tested using a grid search. For the grid search 192 combinations of coefficients were generated from the set of weights below, and a tournament was played for each.

own_moves = [3, 2, 1]
op_moves = [1, .8, .6, 0]
center_weights = [1, .6, .2, 0]
distance_weights = [1, .6, .2, 0]

Through the tournament the best round ended up with a win percentage of 82% with the weights of (3,1,1,0). Full results of the tournament can be found in console.log.txt. Unfortunately the grid search was not trivial to run, taking upwards of 12 hours at a time to return results.

Unfortunately upon running multiple tournaments again outside of the grid with the same weights the win percentage seemed to be slightly lower, tending to be in the higher 70s.. However it still seemed to perform better than the original other two heuristics which seemed to tend to be in the lower 70s which is why this heuristic was ultimately chosen.

Future actions

In the future the writer may explore more complex heuristics, that switch strategies depending on the length of the game for instance initially weighting towards player_moves before moving towards blocking the player_move. The writer may also look at options to rent high speed compute time to speed up iterations of heuristic testing. Lastly the writer would use the additional compute time to run more trials and generate a distribution of percentages, or determine the minimum number of runs per trial to gain a stable percentage score.

Resources

The code used to test the weights is available in heuristic_testing.py and modified_tournament.py. In modified_tournament.py the modified_main function takes an array of weights, generate a scoring function with those weights, and then runs a tournament for each one, recording the results.