Isolation Game Heuristic Analysis

by Menghe Lu

In this short study we build a game-playing agent.

We have two Players in game-agent.py: MinimaxPlayer who chooses a move using depth-limited minimax search. AlphaBetaPlayer who chooses a move using iterative deepening minimax search with alpha-beta pruning. And it should return a good move before the search time limit expires for both of them. It may return (-1, -1) if there are no available legal moves.

MinimaxPlayer uses depth-limited minimax search algorithm. It evaluates each leaf node using a heuristic evalution function, in the maximizing layer, we get the largest of the child node values, and in the minimizing layer, we get the smallest of the child node values. We calculate from leaf to root node.

AlphaBetaPlayer uses Alpha-beta pruning algorithm. It contains alpha whose initial value is negatie infinity and beta whose initial value is positive infinity. These two values represent the maximum score that the maximizing player limits and the minimum score that the minimizing player limits. We could prune the node if for a certain node the minimum score that the minimizing player limits is less than the maximum score that maximizing player limits. i.e. beta<=alpha.

Concerning the AlphaBetaPlayer alphabeta function depth, the more depths the better result for player, but it requires much more time if adding one level in depth. So we should choose a good balance for that number, I have set depth to 1000 depending my computer.

I have three heuristics in custom score functions which uses game and player as parameters. All of these three, I need to check if game is loser or winner for the player at first. Then calculate the player legal moves and the opponent player legal moves.

*I try to block the opponent's available moves, I give more weight on opponent's move so the custom score could be calculated as float(own_moves - (opp_moves * 2)) or float(own_moves * own_moves – 2 * opp_moves * opp_moves).*

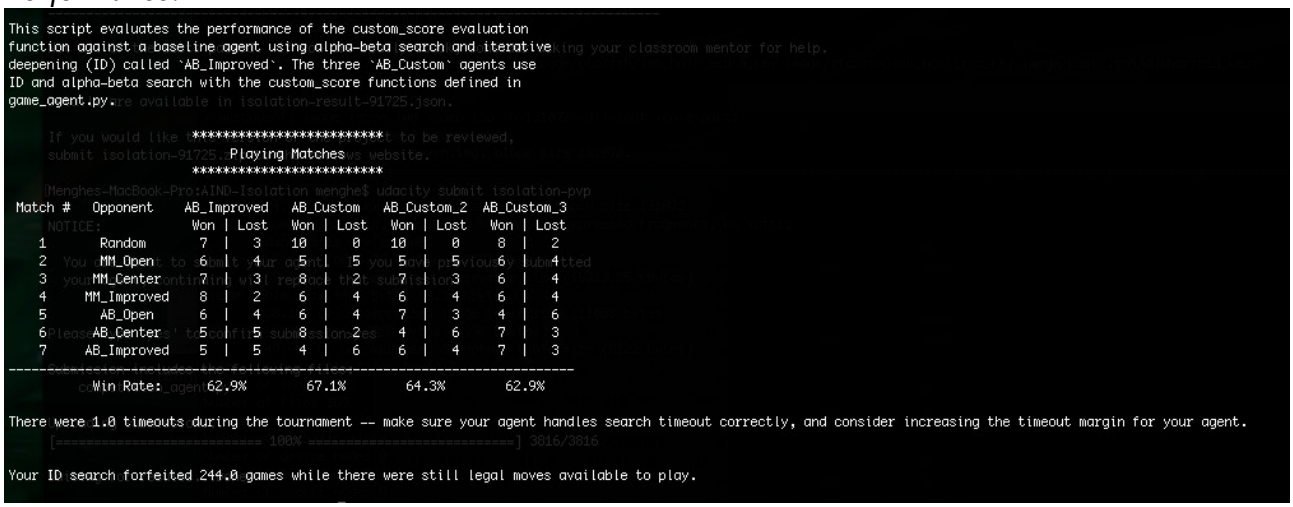
I can also calculate the sum of the available moves for each legal move, and use the sum in my custom score, so it could look like float(own_legal_moves - opp_legal_moves + len(own_moves) – len(opp_moves)).

I can also calculate the distance between player location and the board center. It's usually like the further distance , the less possible legal moves. So my calculate could look like float(own_moves - opp_moves - center_distance(game, game.get_player_location(player))) or float(own_moves - opp_moves

- center_distance(game, game.get_player_location(player))

+ center_distance(game, game.get_player_location(game.get_opponent(player))))

Performance:



```
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       7  |   3    10  |   0    10  |   0     8  |   2
    2      MM_Open       6  |   4     5  |   5     5  |   5     6  |   4
    3     MM_Center      7  |   3     8  |   2     7  |   3     6  |   4
    4   MM_Improved      8  |   2     6  |   4     6  |   4     6  |   4
    5      AB_Open       6  |   4     6  |   4     7  |   3     4  |   6
    6     AB_Center      5  |   5     8  |   2     4  |   6     7  |   3
    7   AB_Improved      5  |   5     4  |   6     6  |   4     7  |   3
--------------------------------------------------------------------------
        Win Rate:      62.9%        67.1%        64.3%        62.9%

There were 1.0 timeouts during the tournament -- make sure your agent handles search timeout correctly, and consider increasing the timeout margin for your agent.

Your ID search forfeited 244.0 games while there were still legal moves available to play.
```

In the four heuristic models, we can see AB_Custom has better performance than AB_Improved has. Because we consider not only the move but also center_distance, consider not only player but also the opponent in AB_Custom heuristic model. Player usually has more potential legal moves if he is near the center of board.
AB_Custom is more complicate to calculate than AB_Custom_3 but it has a better result. It takes not too much more time than other three but better performance.

Concerning AlphaBetaPlayer and MinimaxPlayer, there are have almost the same performance but AlphaBetaPlayer is faster than MinimaxPlayer because it prunes the branches that cannot possibly influence the final decision.
In the seven cpu agents, AB_Center is the best heuristic model because center_score function has the better performance and AlphaBetaPlayer has deeper search in the game tree than MinimaxPlayer has if in the same search time. But it treats only player's move not opponent move.