HEURISTIC ANALYSIS

Dinesh Bhat
bvdinesh@outlook.com

Table of Contents

Building a Game Playing Agent

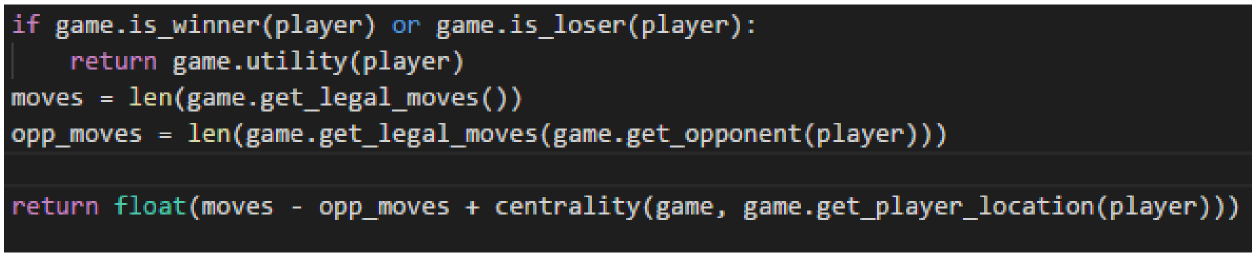
For this application, I have decided to play pen and paper matches of Isolation to understand what I and other potential human players would consider good strategies to win this game.

It became quite apparent to me that there were two main behaviors: offensive and defensive. The offensive behavior consists in trying to obstruct the opponent's game by occupying what would appear to be the best cell for the opponent. This behavior seems a bit short-sighted, but also pays off. The defensive behavior consists in trying to keep as many options as possible open, which results in a general gravitation of the moves towards the center of the board, which is where one has more moves available as opposed to the sides of the board. Therefore, I thought of creating heuristics that combine in different proportions aggressive and defensive strategies. I created a utility function called "centrality" which measures the distance from the center of a certain location in the board.

The following three heuristics were used and tested:

Heuristic 1

The heuristic 1 is implemented on the method "custom_score". This method calculates the heuristic value of a game state from the point of view of the given player.



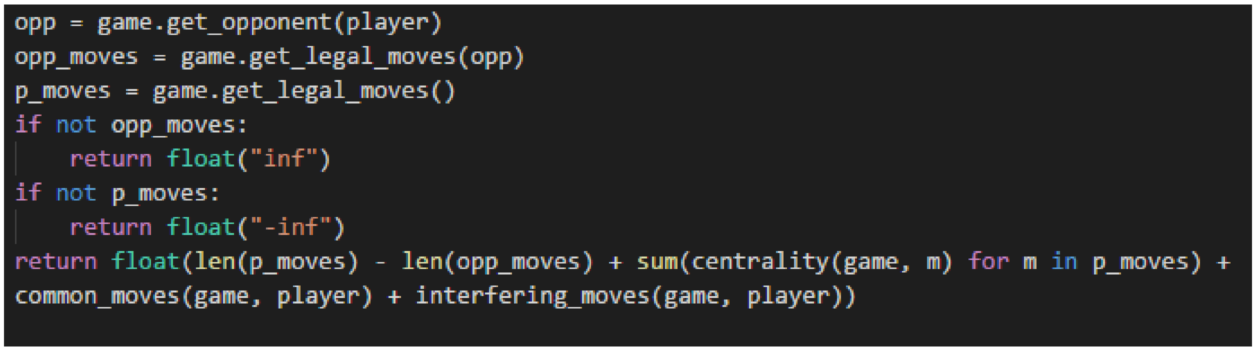
```
if game.is_winner(player) or game.is_loser(player):
    return game.utility(player)
moves = len(game.get_legal_moves())
opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

return float(moves - opp_moves + centrality(game, game.get_player_location(player)))
```

This heuristic adds the number of moves available with a “centrality” value for the potential move. It is a defensive heuristic in that it poses importance on the potential move being as close to the center of the board as possible.

- Heuristic 2

  The heuristic 2 is implemented on the method “custom_score_2”.

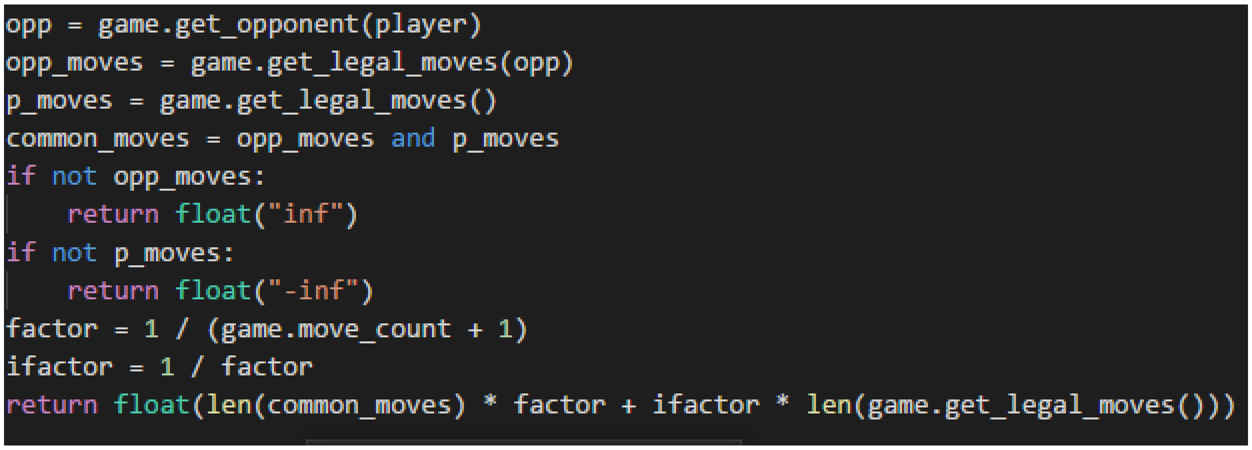


```
opp = game.get_opponent(player)
opp_moves = game.get_legal_moves(opp)
p_moves = game.get_legal_moves()
if not opp_moves:
    return float("inf")
if not p_moves:
    return float("-inf")
return float(len(p_moves) - len(opp_moves) + sum(centrality(game, m) for m in p_moves) +
common_moves(game, player) + interfering_moves(game, player))
```

This heuristic is an all-encompassing scoring function that considers all the possible elements: number of available moves for the player and the opponent, a sum of the centrality of the player moves one ply down, the number of common moves and a value I called “interfering moves” which is the max value of the centrality of potential moves that are common between player and opponent (this indicates whether or not there is an opportunity to steal a really good move off the opponent). There are both defensive and offensive parameters so this seems to be the most balanced heuristic.

- Heuristic 3

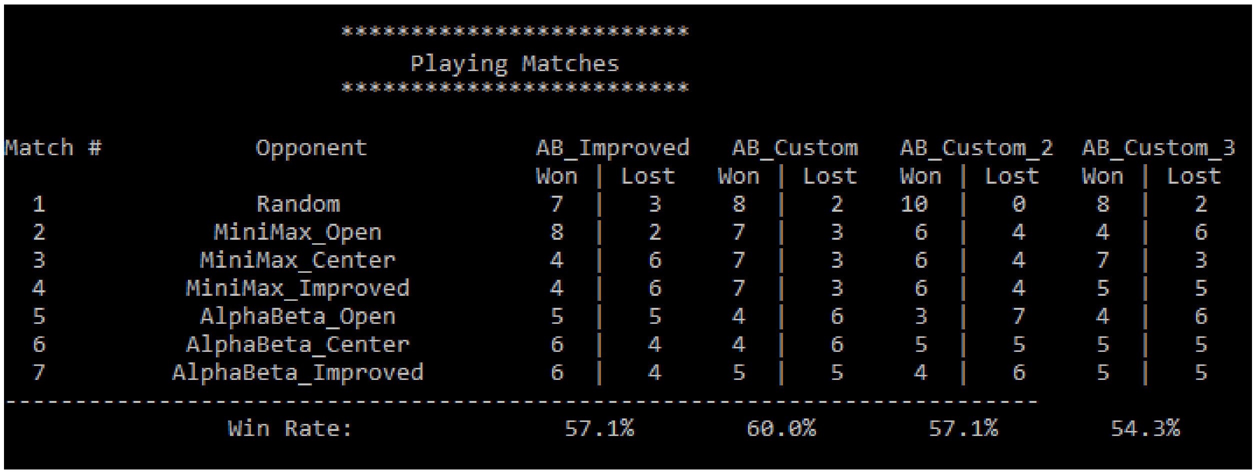
  The heuristic 3 is implemented on the method “custom_score_3”.



```
opp = game.get_opponent(player)
opp_moves = game.get_legal_moves(opp)
p_moves = game.get_legal_moves()
common_moves = opp_moves and p_moves
if not opp_moves:
    return float("inf")
if not p_moves:
    return float("-inf")
factor = 1 / (game.move_count + 1)
ifactor = 1 / factor
return float(len(common_moves) * factor + ifactor * len(game.get_legal_moves()))
```

This heuristic is also a mix of offense and defense, but experimenting with the concept of variable importance of the parameters as the game progresses. Towards the end of the game, a player is expected to have 2 or 3 moves available at best, so the number of moves available should decrease in importance, while the number of common moves should increase in importance.

• Performance



```
                         *************************
                              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      8  |  2     10  |  0      8  |  2
  2             MiniMax_Open            8  |  2      7  |  3      6  |  4      4  |  6
  3            MiniMax_Center           4  |  6      7  |  3      6  |  4      7  |  3
  4           MiniMax_Improved          4  |  6      7  |  3      6  |  4      5  |  5
  5            AlphaBeta_Open           5  |  5      4  |  6      3  |  7      4  |  6
  6           AlphaBeta_Center          6  |  4      4  |  6      5  |  5      5  |  5
  7          AlphaBeta_Improved         6  |  4      5  |  5      4  |  6      5  |  5
--------------------------------------------------------------------------
               Win Rate:                 57.1%        60.0%        57.1%        54.3%
```

Of the three heuristics, number 1 performed the best, demonstrating that the path towards an optimal heuristic is in the weighted sum of all the elements that contribute to evaluating a move rather than an overly aggressive or defensive behavior. This notwithstanding the slowness of the computation of heuristic 1, which may impact the number of iterations in Iterative Deepening, and still perform better than other heuristics.