Game Agent Report

Author: Kiran Ramineni

This report is about heuristic analysis for the isolation game agent. We are playing with one knight piece against an opponent.

Heuristic 1

This Heuristic is based on hybrid approach by mixing open moves and board position. This metric is also the most complex of the three. It is the most compute intensive during tree traversal.

code:

```
    if game.is_loser(player):

        return float("-inf")

    if game.is_winner(player):

        return float("inf")

    moves_own = len(game.get_legal_moves(player))

    moves_opp = len(game.get_legal_moves(game.get_opponent(player)))

    board = game.height * game.width

    moves_board = game.move_count / board

    move_diff = float(moves_own - moves_opp *

                      2) if moves_board > 0.33 else (moves_own - moves_opp)

    pos_own = game.get_player_location(player)

    pos_opp = game.get_player_location(game.get_opponent(player))

    m_distance = abs(pos_own[0] - pos_opp[0]) + abs(pos_own[1] - pos_opp[1])

    return float(move_diff / m_distance)
```

Heuristic 2

This Heuristic is based on fraction of open moves for the player vs opponent. If the Opponent has two open moves vs one move for the player, the score is 0.5.

code:

```
    if game.is_loser(player):

        return float("-inf")

    if game.is_winner(player):
        return float("inf")

    own_moves = len(game.get_legal_moves(player))

    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))

    return float((own_moves + 1) / (opp_moves + 1))
```

Heuristic 3

This heuristic is based on the position of the board. It gives weight to being in the center.

code:

```
    if game.is_loser(player):

        return float("-inf")

    if game.is_winner(player):

        return float("inf")

    w, h = game.width / 2., game.height / 2.

    y, x = game.get_player_location(player)

    y_, x_ = game.get_player_location(game.get_opponent(player))

    return float((h - y)**2 + (w - x)**2) - float((h - y_)**2 + (w - x_)**2)
```

Tournament results

```
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       8  |   2    10  |   0     7  |   3     9  |   1
    2       MM_Open      6  |   4     8  |   2     7  |   3     6  |   4
    3      MM_Center     8  |   2     9  |   1     8  |   2    10  |   0
    4     MM_Improved    7  |   3     7  |   3     6  |   4     3  |   7
    5       AB_Open      6  |   4     5  |   5     5  |   5     4  |   6
    6      AB_Center     5  |   5     5  |   5     7  |   3     6  |   4
    7     AB_Improved    5  |   5     5  |   5     5  |   5     3  |   7
--------------------------------------------------------------------------
           Win Rate:      64.3%        70.0%        64.3%        58.6%

Your agents forfeited 246.0 games while there were still legal moves available to play.
```

Various scoring metrics

1. *Complexity* : Heuristic 1 > Heuristic 3 > Heuristic 2
2. *Deep traversal* : Heuristic 1 > Heuristic 3 = Heuristic 2
3. *Prediction Accuracy*: Heuristic 1 > Heuristic 2 > Heuristic 3

Final Recommendation

From the game report above looks like *Heuristic 1* beat the AB_Improved metric while *Heuristic 2* scored same percentage of wins as the AB_Improved metric. Mixing open moves while maintaining board positions seems like a good strategy. Heuristic 1 is the recommended strategy as it consistently beat the AB_Improved baseline metric.