Analysis of Heuristic Functions

Summary

This analysis compares the heuristic functions used in an adversarial game of Isolation that uses minimax, alpha beta search with iterative deepening for game play. Three custom evaluation functions are introduced in addition to the built-in ones provided and are compared at the end in a tournament to determine the performance of each. The minimax, alpha beta functions along with the custom heuristics have been tested and all unit tests passed.

Based on the analysis, custom heuristic 1 is recommended as it consistently demonstrates better performance over the improved score heuristic for the following reasons:

1. On average, it provides a 3.3% improvement on the win ratio compared to the improved score heuristic
2. It provides a higher weightage to the board center which increases the probabilities of a victory
3. Beyond the center of the board, it falls back to the performance of the open move heuristic, so in effect it provides a net improvement considering the previous two points

Built-in heuristic functions

The sample player provided includes the following 4 heuristic functions:

Null score
This is simply a no-op heuristic unless in the case of a leaf node where the board is evaluated for win or loss. In all other cases this function returns zero which does not help the search function determine the strength of the position without having to search all the way to the end. Despite being included in the sample list of heuristics, this was not used in the tournament to determine the effectiveness of the custom heuristics.

Open move score
This heuristic simply looks at the number of open moves for the given player and scores the board based on it. This is based on the assumption that a play with higher legal moves has a better chance of winning than one that has lesser legal moves at a given point in time.

Improved score
This heuristic builds upon the open move score heuristic, where the scoring takes place by taking the opponents moves as well into account. For example, a player who has 3 possible legal moves would score higher when the opponent has only 2 possible legal moves, which would imply that the former player would have a better chance of winning than the latter player. In testing, this has been a very effective heuristic.

Center score
This heuristic determines a score that is equivalent to the square of the distance from the center of the board to the position of the player. This implies that the further away the player is from the center, the higher the score would be. In testing, this has not performed very well, and is almost counter intuitive to the notion that positions closer to the center are stronger than those further away, especially closer to the edges.

Custom heuristic functions

Three custom heuristic functions were implemented in game_agent.py and tested with the tournament script for its evaluation performance against built in heuristics. The three functions are as follows:

Heuristic 1: open moves with higher weights to the center
This heuristic is an extension of the improved score heuristic with the difference that moves from the center of the board are given higher weightage and therefore higher scores than those off center. This is to encourage the choice of the center of the board as that is the most powerful position.

Implementation

```
def custom_score(game, player):
    if game.is_winner(player):
        return POSINF
    elif game.is_loser(player):
        return NEGINF

    points = 0.

    board_center = (int(game.width / 2), int(game.height / 2))

    location = game.get_player_location(player)
    opponent = game.get_player_location(game.get_opponent(player))

    valid_moves_player = get_next_legal_moves(game, player, location)
    valid_moves_opponent = get_next_legal_moves(game, player, 
opponent)

    if location == board_center:
        points += (2 * len(valid_moves_player)) - 
len(valid_moves_opponent)
    elif opponent == board_center:
        points += len(valid_moves_player) - (2 * 
len(valid_moves_opponent))
    else:
        points += len(valid_moves_player) - len(valid_moves_opponent)
    return points
```

Heuristic 2: ratio of valid moves to empty space compared to the opponent
This compares the ratio of valid moves to empty space of the current player to that of the opponent. The player who has a higher score is deemed to have a better chance than the player who doesn't even though not all empty space is navigable for the player given the movement pattern.

Implementation

```
def custom_score_2(game, player):
    if game.is_winner(player):
        return POSINF
    elif game.is_loser(player):
        return NEGINF

    points = 0.

    location = game.get_player_location(player)
    opponent = game.get_player_location(game.get_opponent(player))

    valid_moves_player = get_next_legal_moves(game, player, location)
    valid_moves_opponent = get_next_legal_moves(game, player,
opponent)

    blank_spaces = len(game.get_blank_spaces())

    points += len(valid_moves_player) / blank_spaces -
len(valid_moves_opponent) / blank_spaces

    return points
```

Heuristic 3: ratio of moves falling into a corner to the total number of legal moves compared to the opponent
This heuristic assesses how many moves of both the player and the opponent fall into the corner of the board which is not a desirable position. The ratio of the moves falling into the corner against the total number of legal moves for the player is then compared to the opponent to see which one is in a weaker position (the one with more moves falling into the corner).

Implementation

```
def custom_score_3(game, player):
    if game.is_winner(player):
        return POSINF
    elif game.is_loser(player):
        return NEGINF

    points = 0.

    top = [(0, i) for i in range(game.width)]
    right = [(i, game.width - 1) for i in range(game.height)]
    bottom = [(game.height - 1, i) for i in range(game.width)]
    left = [(i, 0) for i in range(game.height)]
    corners = set(top + right + bottom + left)

    location = game.get_player_location(player)
    opponent = game.get_player_location(game.get_opponent(player))

    valid_moves_player = get_next_legal_moves(game, player, location)
    valid_moves_opponent = get_next_legal_moves(game, player,
opponent)

    player_corner_moves = 0.
    opponent_corner_moves = 0.

    if location in corners:
        player_corner_moves += 1

    if opponent in corners:
        opponent_corner_moves += 1

    for move in valid_moves_player:
        if move in corners:
            player_corner_moves += 1

    for move in valid_moves_opponent:
        if move in corners:
            opponent_corner_moves += 1

    player_corner_ratio = 0.
    opponent_corner_ratio = 0.

    if len(valid_moves_player) != 0:
        player_corner_ratio = player_corner_moves /
float(len(valid_moves_player))

    if len(valid_moves_opponent) != 0:
        opponent_corner_ratio = opponent_corner_moves /
```

```
float(len(valid_moves_opponent))

    return opponent_corner_ratio - player_corner_ratio
```

Performance of custom heuristic functions

Finally, the heuristics are pitted against each other in a tournament where the custom heuristics are compared with the built-in heuristics. Here's the output from the tournament.py script that shows the performance of each compared to each other. For the comparison, three iterations of the tournament.py script is shown below. In the table below, AB_Custom, AB_Custom_2, AB_Custom_3 correspond to custom heuristic 1, 2 and 3 respectively.

Iteration #1

```
 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     9  |   1     7  |   3
    2       MM_Open      6  |   4     8  |   2     5  |   5     6  |   4
    3      MM_Center     9  |   1     5  |   5     8  |   2     5  |   5
    4     MM_Improved    6  |   4     7  |   3     7  |   3     4  |   6
    5       AB_Open      4  |   6     5  |   5     6  |   4     4  |   6
    6      AB_Center     5  |   5     6  |   4     3  |   7     3  |   7
    7     AB_Improved    6  |   4     7  |   3     5  |   5     4  |   6
--------------------------------------------------------------------------
           Win Rate:      61.4%        65.7%        61.4%        47.1%
```

Iteration #2

```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       8  |   2     7  |   3     8  |   2     7  |   3
    2       MM_Open      8  |   2     7  |   3     6  |   4     5  |   5
    3      MM_Center     8  |   2     7  |   3     9  |   1     6  |   4
    4     MM_Improved    5  |   5     8  |   2     8  |   2     6  |   4
    5       AB_Open      6  |   4     6  |   4     5  |   5     1  |   9
    6      AB_Center     4  |   6     5  |   5     6  |   4     5  |   5
    7     AB_Improved    3  |   7     5  |   5     4  |   6     6  |   4
--------------------------------------------------------------------------
           Win Rate:      60.0%        64.3%        65.7%        51.4%
```

Iteration #3

```
 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       7  |   3     9  |   1     9  |   1     8  |   2
    2       MM_Open       7  |   3      9  |   1      6  |   4      7  |   3
    3      MM_Center      7  |   3      7  |   3      8  |   2      8  |   2
    4     MM_Improved     7  |   3      7  |   3      4  |   6      5  |   5
    5       AB_Open       6  |   4      5  |   5      7  |   3      5  |   5
    6      AB_Center      5  |   5      4  |   6      5  |   5      3  |   7
    7     AB_Improved     7  |   3      6  |   4      5  |   5      5  |   5
--------------------------------------------------------------------------
           Win Rate:      65.7%        67.1%        62.9%        58.6%
```

Summary of the average performance of the heuristics

Agent	**Heuristic**	**Average win rate**
AB_Improved (by Udacity)	Improved score	**62.4%**
AB_Custom	**Heuristic 1**	**65.7%**
AB_Custom_2	Heuristic 2	**63.3%**
AB_Custom_3	Heuristic 3	**52.4%**

Based on the results above, the recommendation is Heuristic 1 for the following reasons:

1. On average, it provides a 3.3% improvement on the win ratio compared to the improved score heuristic
2. It provides a higher weightage to the board center which increases the probabilities of a victory
3. Beyond the center of the board, it falls back to the performance of the improved score heuristic, so in effect it provides a net improvement considering the previous two points

References

Russell, Stuart J. et al. “Artificial Intelligence A Modern Approach.”