Game Playing Agent Heuristics Analysis

This document evaluates three heuristic functions used by the Game Agent implementing the ID and alpha beta search methods.

The below table provides the winning rate of the game agent 3 heuristic functions and the baseline AB_Improved function

	Win Rate (Run 1)	Win Rate (Run 2)	Win Rate (Run 3)	
Custom_Score	72.9	74.3	81.4	
Custom_Score2	67.1	71.4	65.7	
Custom_Score3	65.7	77.1	64.3	
AB_Improved	65.7	68.6	68.6	

Based on the test results and functionality, out of the 3 heuristics the recommended heuristic is Custom_Score which is based on the number of legal moves available for the player and the player position as

- The function is simple and fast to compute
- It takes in to account the positions around the board center - most likely position for winning
- The winning rate of this heuristic is high than other two functions

Custom_Score

This heuristic function returns "inf" if the player is the winner and "-inf" if the player loses the game. It considers a positive value - indicating the player has better chance of winning, a negative value indicating the player has chance of losing and 0 if it is a draw. It returns the difference in the number of moves available to the player and the number of moves available to the opponent.

It also considers if the player is at the center or around the center of the board and adds 1.5 or .5 to the score above.

Test Run Results

Match #	Opponent	Won | Lost	Won | Lost
1	Random	10 | 0	7 | 3
2	MM_Open	7 | 3	9 | 1
3	MM_Center	8 | 2	9 | 1
4	MM_Improved	9 | 1	8 | 2
5	AB_Open	6 | 4	6 | 4
6	AB_Center	7 | 3	7 | 3
7	AB_Improved	4 | 6	6 | 4

Win Rate: 72.9%

Implementation:

```
def custom_score(game, player):
    if game.is_loser(player):
        return float("-inf")

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

    credit = 0.0

    center = ((int(game.width/2), int(game.height/2)))
    r,c = center
    directions = [(-1,-2), (-1,2),(1, -2), (1, 2), (2, -1), (2, 1), (-2, -1), (-2, 1)]
    off_center = [(r + dr, c + dc) for dr, dc in directions if in_bounds(game, r + dr, c + dc)]
    player_location = game.get_player_location(player)
    if player_location == center:
        credit = 1.5
    elif player_location in off_center:
        credit = 0.5

    own_moves = len(game.get_legal_moves(player))
    opp_moves = len(game.get_legal_moves(game.get_opponent(player)))
    return float(own_moves - opp_moves) + credit
```

Custom_Score2

This heuristic function returns the highest score if the player occupies the center space or the player available moves are in the diagonally opposite space of the opponent. Otherwise, returns the difference in the no of legal moves available to the player and the opponent

Test Run Results

Match #	Opponent	Won | Lost
1	Random	8 | 2
2	MM_Open	8 | 2
3	MM_Center	7 | 3
4	MM_Improved	7 | 3
5	AB_Open	5 | 5
6	AB_Center	6 | 4
7	AB_Improved	6 | 4

Win Rate: 67.1 %

Implementation:

```
def custom_score_2(game, player):
  if game.is_loser(player):
     return float("-inf")

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

   center = ((int(game.width/2), int(game.height/2)))
   player_location = game.get_player_location(player)
   opponent_location = game.get_player_location(game.get_opponent(player))
   if (player_location == center):
      return float("inf")
   else:
      corners={(0,0), (0, game.width-1), (game.height-1,0), (game.width-1, game.height-1)}
      if opponent_location in corners and player_location in corners:
         return float("inf")
      else:
         return float(len(game.get_legal_moves(player)) -
len(game.get_legal_moves(game.get_opponent(player))))
```

Custom_Score3

This heuristic function returns the Manhattan distance between the player and the opponent.

The Manhattan distance gives the no of steps needed in horizontal and vertical path – the allowed L shaped moves. The winning rate using this has been less than the other two heuristics, as it is not considering the winning positions and if the player can block the opponent.

Test Run Results

Match #	Opponent	Won | Lost
1	Random	10 | 0
2	MM_Open	7 | 3
3	MM_Center	9 | 1
4	MM_Improved	5 | 5
5	AB_Open	6 | 4
6	AB_Center	4 | 6
7	AB_Improved	5 | 5

Win Rate: 65.7%

```
def custom_score_3(game, player):
if game.is_loser(player):
      return float("-inf")

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

```
own_location = game.get_player_location(player)
opp_location = game.get_player_location(game.get_opponent(player))
distance = abs(own_location[0]-opp_location[0]) + abs(own_location[1]-opp_location[1])
return float(distance)
```