Heuristic Analysis for Isolation Game

Iris Zeng

To fully understand the game mechanism, I first tried to run the sample player, then played the game with my friend. My heuristic functions are inspired by the sample functions in the sample_player.py, and focus on two factors: the legal moves left and the distance to the center. The number of legal moves left decides if the game ends and how possibly the game is going to terminate. This factor has positive correlation with the final result, which means the more legal moves the position has, the larger possibility to take this position as next step. The distance to the center indicates the number of future moves of the position (edges blocks the possible moves for some positions). This factor is negative correlation with the final result. The larger the distance to the center, the less the possibility to take the position as the next step. Following includes the details about the functions.

1st Heuristic Function: custom_score_3

```
opponent = game.get_opponent(player)
cur_moves = game.get_legal_moves()
opp_moves = game.get_legal_moves(opponent)

cur_center_score = position_score(game, game.get_player_location(player))
opp_center_score = position_score(game, game.get_player_location(opponent))

return float(len(cur_moves) * 1.0 - len(opp_moves) * 1.0
        - (cur_center_score - opp_center_score) / 10.0)
```

The idea is very straight forward. First part is the difference of legal moves between the current player and the opponent player. The second part is the difference between the distance of current location to the center. Since this factor is less important than the first one, it is divided by 10. The function position_score() calculates the Euclid distance between the current position of the player to the center of the board. Since the distance to the center position is negative correlation with the game state score, so game state score is the first part minus the second part.

2nd Heuristic Function: custom_score_2

```
cur_moves = game.get_legal_moves()

cur_center_score = position_score(game, game.get_player_location(player))
```

```
return float(len(cur_moves) * 1.0
        - (cur_center_score / 10.0)
        - sum([position_score(game, move) for move in cur_moves]) / 20.0)
```

This function focus on player side. It calculates the number of legal moves, the distance to the center position and the sum of distance from current position's possible moves to the center position, with numbers to tune the importance of each factor. This function focus on offensive strategies while compare to the first function, and has similar performance as the first one.

3rd Heuristic Function: custom_score_3

```
opponent = game.get_opponent(player)
cur_moves = game.get_legal_moves()
opp_moves = game.get_legal_moves(opponent)

cur_center_score = position_score(game, game.get_player_location(player))
opp_center_score = position_score(game, game.get_player_location(opponent))

cur_possible_moves = sum([position_score(game, move) for move in cur_moves])
opp_possible_moves = sum([position_score(game, move) for move in opp_moves])

return float((len(cur_moves) * 1.0 - len(opp_moves))
        - (cur_center_score - opp_center_score) / 10.0
        - (cur_possible_moves - opp_possible_moves) / 20.0)
```

This function has three parts: the difference between player and its opponent's legal moves, difference of theirs distance to center point, difference of their legal moves' distance to the center. This function considers more factors than the first one, and performance is better than the first one. It combines both offensive and defensive strategies compared to the second function

Conclusion

```
**************************
     Playing Matches
**************************

Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                      Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      9   1        8   2        9   1        9   1
    2       MM_Open     8   2        9   1        7   3        4   6
    3      MM_Center    7   3        8   2        5   5        8   2
    4     MM_Improved   6   4        6   4        5   5        5   5
    5       AB_Open     4   6        4   6        5   5        6   4
    6      AB_Center    5   5        7   3        4   6        5   5
    7     AB_Improved   4   6        5   5        6   4        6   4
--------------------------------------------------------------------
          Win Rate:      61.4%        67.1%        58.6%        61.4%
```

As the tournament result show above, I choose the third solution, because

1. It considers more factors compared to the first one, and it is more comprehensive
2. Compared to the second function, the third one takes both player and opponent into consideration
3. It has highest win rate among the three functions