Building a game playing agent:

Introduction:

This project involved building a game playing agent for a modified version of isolation (ie a player can only move as a knight can in chess, as opposed to the usual version which lets a player move like a queen in chess). In class it was proved that searching to endgame is a computationally intensive task, which in most cases will not complete a run in our lifetime. To address this issue, the heuristic evaluation functions were introduced; these functions provided a measure of the goodness of a board based on certain criteria. With the help of these evaluation heuristics we can avoid searching to end game.

Objective:

The objective of this report is to compare custom evaluation heuristics that were implemented by the student to a predefined score called as Improved score. This improved score is the number of moves a player has subtracted by the number of moves the player's opponent has.

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      9  |   1    10  |   0     9  |   1     8  |   2
   2       MM_Open     7  |   3     6  |   4     9  |   1     8  |   2
   3      MM_Center   10  |   0     9  |   1     9  |   1     7  |   3
   4     MM_Improved   7  |   3     6  |   4     8  |   2     8  |   2
   5       AB_Open     5  |   5     5  |   5     4  |   6     5  |   5
   6      AB_Center    7  |   3     5  |   5     6  |   4     4  |   6
   7     AB_Improved   5  |   5     4  |   6     6  |   4     4  |   6
--------------------------------------------------------------------------
          Win Rate:      71.4%       64.3%       72.9%       62.9%
```

Custom Score 1:

For the first custom score, a metric called **chebyshev distance** was used. This metric provides distance in number of boxes, as opposed to a normal Euclidean distance, which gives the distance of a straight line between two points.

The intuition behind using this metric came from the implementation of center_score function in sample_players.py. But instead of calculating the direct

Euclidean distance, it made more sense to compare distance in terms of boxes as the basic unit of a game board is a box.

The performance of this metric as we can see from the figure 1, is better in the part where only minimax is implemented, but not as good when alpha-beta search is implemented. This shows us that this function is not a very good evaluation metric as it is not as efficient when pruning is done.

Custom Score 2:

This score takes its intuition from improved score function. Instead of subtracting the number of moves the opponent has left we weight that feature causing the computer player to be more aggressive. This gives us the best result. As we can see from figure 1, this heuristic function has a higher win rate than the ab_improved player.

Custom Score 3:

For the third custom score, a metric called **Minkowski distance** was explored. This metric is a normalized version of Euclidean and Manhattan distance. In figure 2 the green line represents Euclidean distance and the other lines represent Manhattan distance.

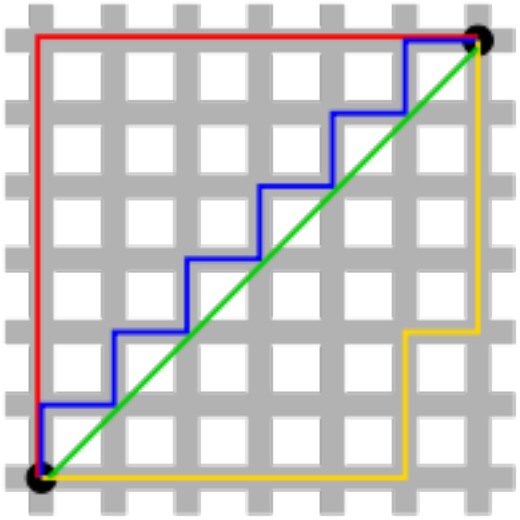

Figure 2. Green: Euclidean, other colors: Manhattan
From Wikipedia

The intuition behind choosing this metric was similar to chebyshev distance. Multiple distance metrics from the page metrics were tested, in an attempt to exceed the ab_improved agent.

Recommendation:

From the data it is clear that custom score 2, is the best pick. The reason for this is because

1. It evaluates the goodness of a board sufficiently.
2. Simple to compute.
3. Consistently produces win rate than ab_improved agent.

One thing that was observed during the runs was that simpler functions tend to give better results. This is in accordance with what was recommended in class as more complex a function get more time it takes which only lets us search fewer levels as compared to a much simpler function.