Heuristic Analysis
Isolation

Prepared by
Krishna G. Mohan

SYNOPSIS

The project is to develop an adversarial search agent to play the game of *Isolation*.

Isolation is a deterministic, two-player game of perfect information in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner.

This project uses a version of Isolation where each agent is restricted to L-shaped movements (like a knight in chess) on a rectangular grid (like a chess or checkerboard). The agents can move to any open cell on the board that is 2-rows and 1-column or 2-columns and 1-row away from their current position on the board. Movements are blocked at the edges of the board (the board does not wrap around), however, the player can "jump" blocked or occupied spaces (just like a knight in chess).

Additionally, agents will have a fixed time limit each turn (150ms) to search for the best move and respond. If the time limit expires during a player's turn, that player forfeits the match, and the opponent wins.

OBJECTIVE

The project is to create the adversarial search agent and to create custom heuristics. The effectiveness of the custom heuristics can be tested using the *tournament.py* script. 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*

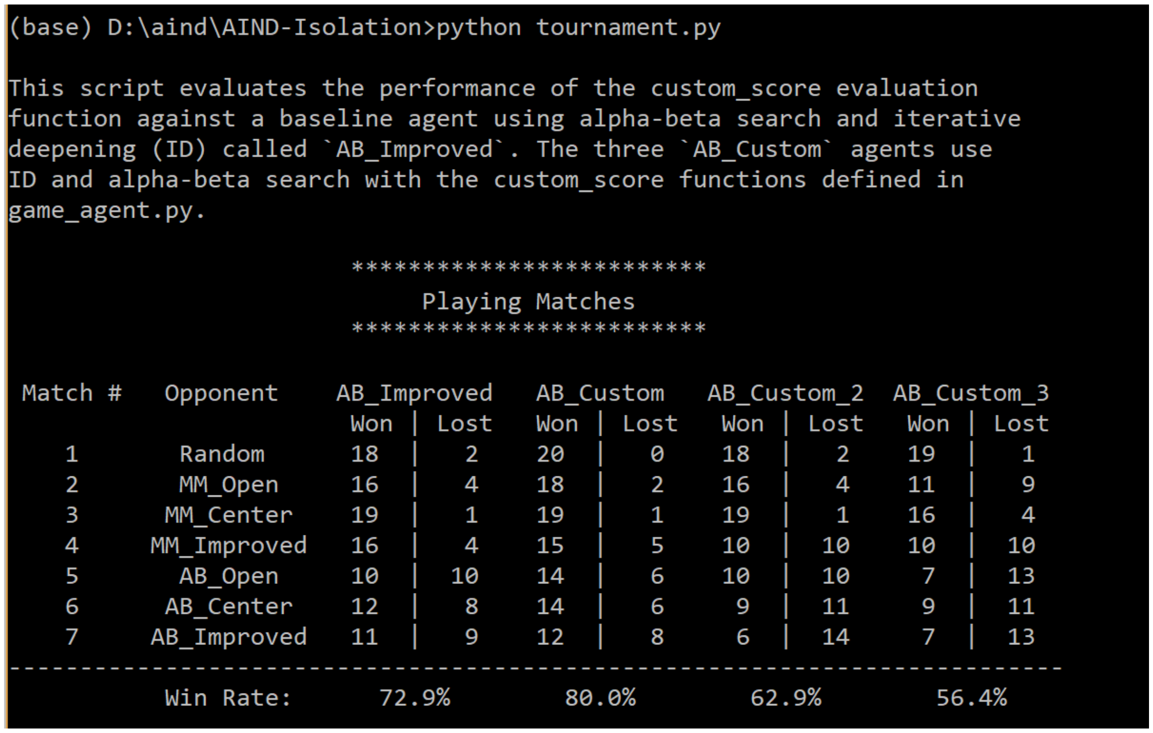
The goal is to develop a custom heuristic such that Student outperforms ID_Improved.

The tournament opponents are listed below:

- Random: An agent that randomly chooses a move each turn.
- MM_Open: MinimaxPlayer agent using the open_move_score heuristic with search depth 3
- MM_Center: MinimaxPlayer agent using the center_score heuristic with search depth 3
- MM_Improved: MinimaxPlayer agent using the improved_score heuristic with search depth 3
- AB_Open: AlphaBetaPlayer using iterative deepening alpha-beta search and the open_move_score heuristic
- AB_Center: AlphaBetaPlayer using iterative deepening alpha-beta search and the center_score heuristic
- AB_Improved: AlphaBetaPlayer using iterative deepening alpha-beta search and the improved_score heuristic

.

RAW RESULTS



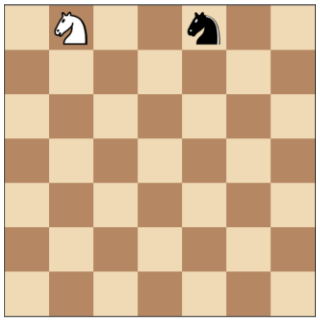
```
(base) D:\aind\AIND-Isolation>python tournament.py

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      18  |   2    20  |   0    18  |   2    19  |   1
    2       MM_Open     16  |   4    18  |   2    16  |   4    11  |   9
    3      MM_Center    19  |   1    19  |   1    19  |   1    16  |   4
    4     MM_Improved   16  |   4    15  |   5    10  |  10    10  |  10
    5       AB_Open     10  |  10    14  |   6    10  |  10     7  |  13
    6      AB_Center    12  |   8    14  |   6     9  |  11     9  |  11
    7     AB_Improved   11  |   9    12  |   8     6  |  14     7  |  13
--------------------------------------------------------------------------
           Win Rate:      72.9%        80.0%        62.9%        56.4%
```

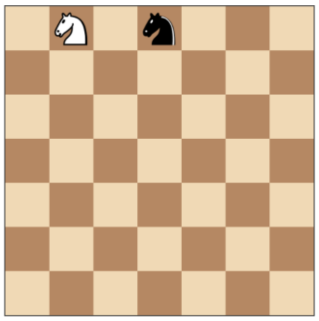
HEURISTIC 1 - CHASE OR RUN (AB_CUSTOM)

Effectiveness: 80%

The heuristic used by AB_Improved is the improved_score. The board position is evaluated by finding the difference between the number of legal moves available to each player.

In the Chase or Run algorithm we take the number of legal moves and adjust it's value based on the relative position of the pieces. If look at the initial position of the pieces, they could be on the squares of the same color or on squares of the opposite color





If the pieces are on squares of different colors and if the player is on a one of the opponent's possible moves, then the player is blocking the opponent. For this the position is rewarded. If the opponent has many possible moves then this blocking is not that great, so the reward should be small. But if the opponent has few possible moves left then the reward should be much greater. The effect is that positions where the player blocks the opponent are given a higher score.

The value is calculated by the formula:

own_moves = own_moves + own_moves * (9. - opp_moves)/8.

So if the opponent has several moves, then the value of the fraction (9-m)/8 will be small – the reward is small. If the opponent has fewer possible moves then the fraction is much larger – the reward is greater. The effect is that the player blocks and *Chases* the opponent

If the pieces are on the same color, then the opponent is *chasing* the player. In this case, the position is to be rewarded only if the opponent cannot block the player on the next move. Again, if the player had several moves, then the opponent block is not that important. But if the player has few moves then then the position should be rewarded well if the opponent cannot block the player on the next move.

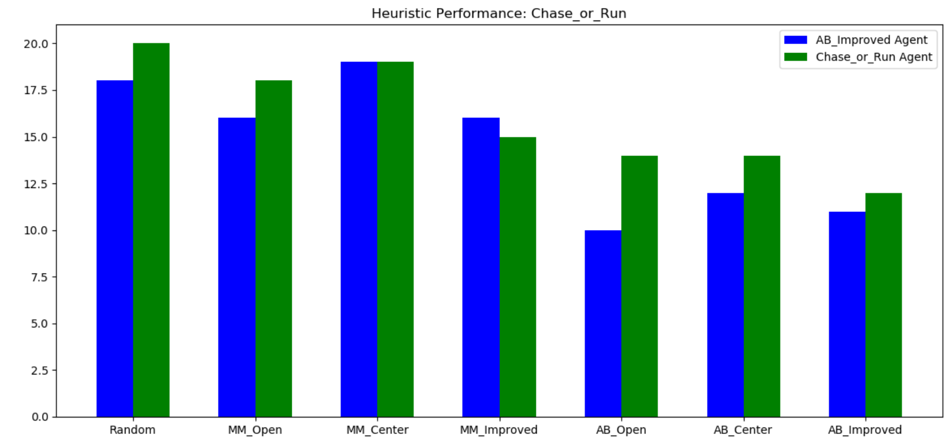
The value is calculated by the formula:

own_moves = own_moves - own_moves * (9.-own_moves)/8.

Note that if the player has several moves then the ratio of (9-m)/8 is small and hence a lower reward. The effect is that if the pieces are on the same colored squares the player *Runs* away from the opponent.

The tournament was played with 20 games and here are the results:

Agents	AB_Improved Agent Wins out of 20	Chase_or_Run Agent Wins out of 20
Random	18	20
MM_Open	16	18
MM_Center	19	19
MM_Improved	16	15
AB_Open	10	14
AB_Center	12	14
AB_Improved	11	12



This heuristic seems to perform better than AB_Improved against all opponents except MM_Improved.

HEURISTIC 2 - LEAST ONWARD SQUARES (AB_CUSTOM_2)

Effectiveness: 62.9%

The idea is to keep the knight on the board for the longest time, which means we try and do a knight's tour algorithm. The knight is moved so that it always proceeds to the square from which the knight will have the fewest onward moves.

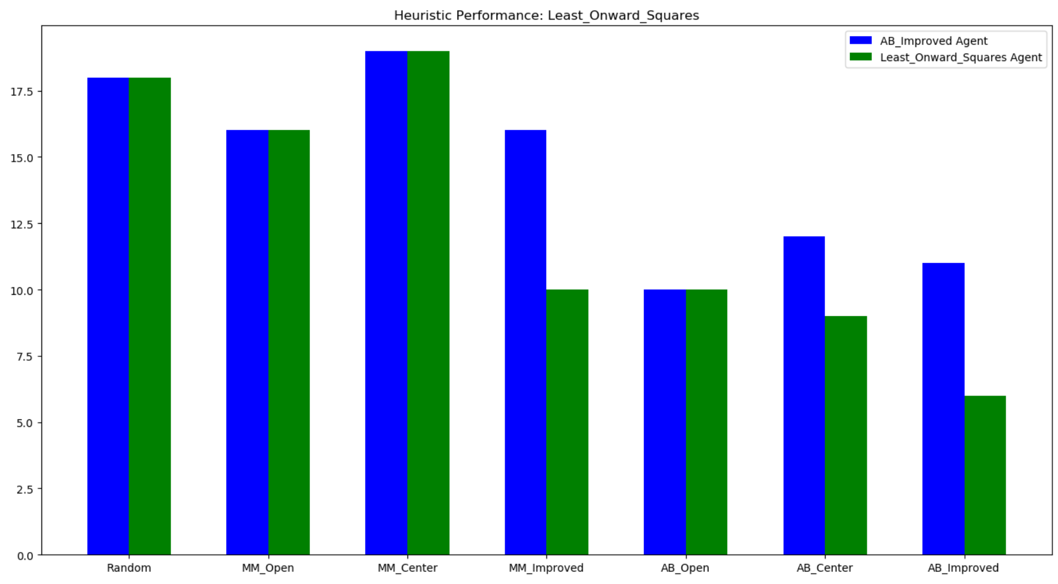
This has one issue though: the knight will try to go to a square with no moves. To counter this, we modify the rule a bit to find the square with the fewest onward moves, but only if it has at least one onward move.

If there is only one onward move, then choose that only if the opponent cannot block the onward move.

This idea comes from Warnsdorf's rule. See https://en.wikipedia.org/wiki/Knight%27s_tour

The tournament was played with 20 games and here are the results:

Agents	AB_Improved Agent Wins out of 20	Least_Onward_Squares Agent Wins out of 20
Random	18	18
MM_Open	16	16
MM_Center	19	19
MM_Improved	16	10
AB_Open	10	10
AB_Center	12	9
AB_Improved	11	6



This heuristic does not perform as well as the AB_Improved heuristic.

HEURISTIC 3 - LONGER TOUR (AB_CUSTOM_3)

Effectiveness: 56.4%

This heuristic evaluates the position by comparing the length of the player's knights tour and the number of open moves to that of the opponent.

The idea is that the player seeks to retain the knight for longer on the board while trying to shorten the opponent's knights tour length. At the same time the player seeks to find the position with the most open moves compared to the opponent.

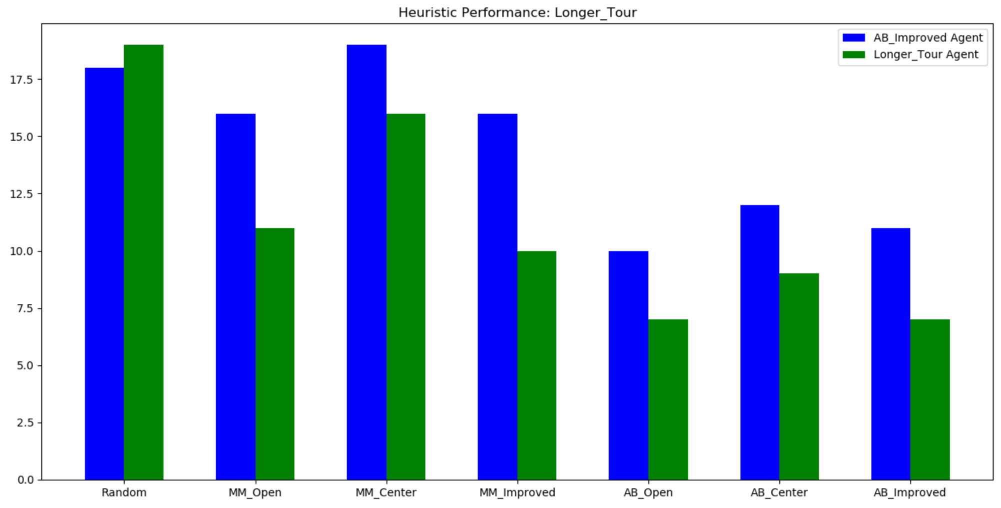
The difference between the lengths + open moves of player to the opponent is the value of the position.

This idea comes from Warnsdorf's rule. The knight is moved so that it always proceeds to the square from which the knight will have the fewest onward moves.

See https://en.wikipedia.org/wiki/Knight%27s_tour

The tournament was played with 20 games and here are the results:

Agents	AB_Improved Agent Wins out of 20	Longer_Tour Agent Wins out of 20
Random	18	19
MM_Open	16	11
MM_Center	19	16
MM_Improved	16	10
AB_Open	10	7
AB_Center	12	9
AB_Improved	11	7



This heuristic performs poorly compared to AB_Improved against almost all opponents.

Conclusion

Here is a summary of the results:

Heuristic	Effectiveness
AB_Improved	72.9%
Chase_or_run	80.0%
Least_Onwards_Squares	62.9%
Longer_tour	56.4%

It seems that taking the *improved score* heuristic (player moves – opponent moves) and then adding valuation based on specific board positions – specifically the colors of the starting squares of the pieces – yields the best results. Trying to retain the piece on the board by identifying knight tours does not produce good results. The *Chase or Run* strategy which takes into account:

- Number of legal moves available to each player
- The color of the starting squares of the pieces

gives the best results.

APPENDIX

Code used to create plots:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

excel = pd.ExcelWriter('results.xlsx')

heuristic_agent = []
ab_improved =[]
index = np.arange(7)
labels = ('Random', 'MM_Open', 'MM_Center', 'MM_Improved', 'AB_Open', 'AB_Center',
'AB_Improved')
bar_width = 0.3

def chart_results(heuristic_agent_name, ab_improved_wins, heuristic_agent_wins):
    data = {'Agents':labels,
            'AB_Improved Agent':ab_improved_wins,
            heuristic_agent_name + ' Agent': heuristic_agent_wins }

    df = pd.DataFrame(data)
    df = df.set_index('Agents', drop=True)
    df.to_excel(excel, heuristic_agent_name)
    excel.save()

    plt.bar(index, ab_improved_wins, width=bar_width, color='b',
            label='AB_Improved Agent')
    plt.bar(index + bar_width, heuristic_agent_wins, width=bar_width, color='g',
            label = heuristic_agent_name + ' Agent')
    plt.title('Heuristic Performance: ' + heuristic_agent_name)
    plt.xlabel = 'Agents'
    plt.ylabel = 'Wins'
    plt.xticks(index + (bar_width/2), labels)
    plt.legend()
    plt.show()


heuristic_agent_name = 'Chase_or_Run'
ab_improved_wins = [18, 16, 19, 16, 10, 12, 11]
heuristic_agent_wins = [20, 18, 19, 15, 14, 14, 12]
chart_results(heuristic_agent_name, ab_improved_wins, heuristic_agent_wins)

heuristic_agent_name = 'Least_Onward_Squares'
ab_improved_wins = [18, 16, 19, 16, 10, 12, 11]
heuristic_agent_wins = [18, 16, 19, 10, 10, 9, 6]
chart_results(heuristic_agent_name, ab_improved_wins, heuristic_agent_wins)

heuristic_agent_name = 'Longer_Tour'
ab_improved_wins = [18, 16, 19, 16, 10, 12, 11]
heuristic_agent_wins = [19, 11, 16, 10, 7, 9, 7]
chart_results(heuristic_agent_name, ab_improved_wins, heuristic_agent_wins)
```