ISOLATION GAME HEURISTIC ANALYSIS

Chiranjeeb Kataki

GAME RULES:

This project aims to develop an adversarial search agent to play the game “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.

This particular project uses a version of Isolation where each agent is restricted to L-shaped movements only (like a chess knight). The agent also has a fixed time limit per turn.

FINDING A HEURISTIC:

I decided to keep the evaluation heuristics simple; the reasoning was that this would allow more searches of the game tree since there is a time bound for returning a move. The heuristic could be made more complex, but that would mean taking more time, which takes away from deeper searches.

I also played the game against several people, and the feedback I got from most of them in terms of their playing strategy was that they used some form of available moves counting. I factored this into my design process.

For each heuristic, I implement a simple check to return +/- infinity if there are no moves available for the player/opponent; this indicates an immediate win/loss.

HEURISTIC 1:

With this heuristic, we simply try to aggressively get more available moves than our opponent. The main logic is:

```
score = own_moves - 2*opponent_moves
```

We then multiply this score with the game move number to accentuate the goodness or badness the further along the game we are. This product is the final returned score. The point behind this being that a baseline score of (let's say) 3, is better 10 moves in, compared to at the beginning of the game.

HEURISTIC 2:

With this heuristic, we work with the ratio of moves available to us versus our opponent. In a way, this is similar to our first heuristic in that we still depend on a relationship between number of available moves. The logic is:

```
if own_moves > opponent_moves
      score = own_moves/opponent_moves
else
      score = -opponent_moves/own_moves
```

We then multiply this score with the game move number to accentuate the goodness or badness the further along the game we are. The product is the final returned score.

HEURISTIC 3:

This is a slightly more complex heuristic (relatively speaking). We still base our score on the number of moves available, but we use slightly different methods depending on the game stage.

In the early game stage, since it's hard to justifiably say if a position is really good or not, we simply try to stay ahead of our opponent in terms of available moves.

We get progressively aggressive depending on the game state; once the number of blocked spaces is more than blank spaces, we switch to a slightly more aggressive method. After the number of blocked spaces is more than twice the number of blank spaces, we switch to our most aggressive strategy.

The reasoning for this is that as we reach towards the end of the game, the difference in open moves counts for more (both for positive and negative outcomes).

The logic is:

```
if blocked < blanks
      score = own_moves - opponent_moves
else-if blocked > 2*blanks
      score = own_moves - 2.5*opponent_moves
else
      score = own_moves - 2*opponent_moves
```

We then multiply this score with the game move number to accentuate the goodness or badness the further along the game we are. The product is the final returned score.

TOURNAMENT RUNS:

The three custom heuristics were pitted against these six CPU agents:

- Random: agent plays random moves
- MM_Open: minimax agent using open_move heuristic with search depth 3
- MM_Center: minimax agent using center_score heuristic with search depth 3
- MM_Improved: minimax agent using improved_score heuristic with search depth 3
- AB_Open: alpha-beta agent using iterative deepening and the open_move heuristic
- AB_Center: alpha-beta agent using iterative deepening and the center_score heuristic
- AB_Improved: alpha-beta agent using iterative deepening and the improved_score heuristic

The heuristics used by them are:

- Open_move heuristic: score is simply number of open moves
- Center_score heuristic: score is square of the distance from center of the board
- Improved_score heuristic: score is difference in number of open moves of player versus opponent

I modified the supplied tournament script so that during each run, the four agents (AB_Improved, and three user agents) play 20 games against the CPU agents; in total each agent plays 220 games against each of the CPU agents.

The table below describes the overall win percentages of the four agents during each tournament run. An agent plays 20 games against each of the 7 CPU agents, for a total of 140 games per run.

Tournament run (140 total games per run)	**AB_Improved**	**Heuristic 1**	**Heuristic 2**	**Heuristic 3**
1	70.00%	74.30%	64.30%	71.40%
2	70.00%	67.10%	65.70%	72.90%
3	80.00%	70.00%	64.30%	67.10%
4	61.40%	68.60%	65.30%	71.40%
5	71.40%	70.00%	75.70%	71.40%
6	68.60%	64.30%	71.40%	72.90%
7	68.60%	65.70%	68.60%	65.70%
8	72.90%	70.00%	67.10%	77.10%
9	65.70%	72.90%	70.00%	71.40%
10	62.90%	65.70%	67.10%	68.60%
11	65.70%	71.40%	68.60%	74.30%
MEAN	68.84%	69.09%	68.01%	71.29%
MEDIAN	68.60%	70.00%	67.10%	71.40%
STANDARD DEVIATION	5.11%	3.16%	3.42%	3.23%

CONCLUSION:

Looking at the data, we see that heuristic 3 is better (albeit by a small margin) over the other two heuristics. It also performs better than the AB_Improved agent, and has a smaller standard deviation (meaning it is more consistent). Based on this, I selected heuristic 3 for my custom agent.