Heuristic Analysis for Isolation Game Playing Agent

By Joseph Mejorada

Introduction

In this project, a game playing agent was created to simulate the gameplay for the multiplayer turn-based board game Isolation using the minimax algorithm to search through the game tree and the alpha-beta pruning algorithm to eliminate branches in the game tree to increase efficiency. These algorithms use a scoring model to determine which turns to take next defined by a heuristic. There are three custom heuristics that were used to define the scoring model in this project.

Heuristic Models Logic

Heuristic #1: custom_score

The first custom heuristic imagines the isolation game board with four axes: one vertical axis, one horizontal axis, one axis that extends from the top left corner to the bottom right corner, and one axis that extends from the top right corner to the bottom left corner (see **Figure 1**).

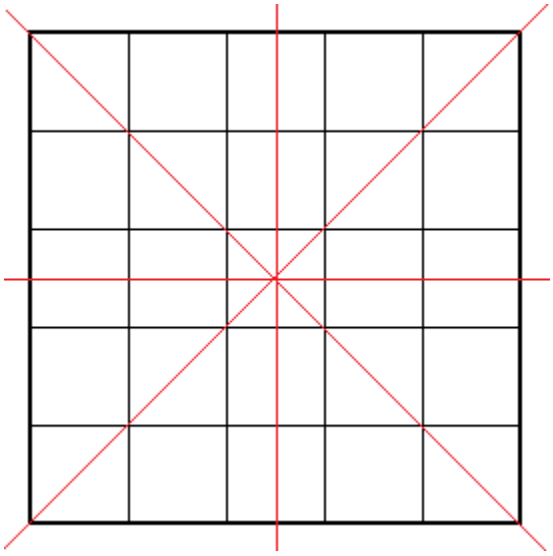

Figure 1: 5x5 game board with red lines depicting different axes.

If the center of the isolation game board is available, moving to the center is first prioritized as it would give a higher chance of winning the game. Next, the chances of winning are increased by prioritizing a move that best mirrors the opponent's most recent position along the axes after checking to see if the center location is available. This rule assumes that as long as the moves of the opponent are mirrored, the player would be able to trap the opponent while having the last turn. Finally, moves that would

result in maximizing the player's own available moves while minimizing the opponent's available moves are prioritized with less weight than the other two rules.

Heuristic #2: custom_score_2

The second heuristic, like the last rule in Heuristic #1, prioritizes moves that maximize the player's own available moves while minimizing the opponent's available moves. However, the difference with this heuristic is that it puts more weight in minimizing the opponent's available moves. This would cause the player to chase and trap the opponent.

Heuristic #3: custom_score_3

The third heuristic attempts to corner the opponent into one of the four quadrants on the game board (see **Figure 2**). The player executes this strategy by first attempting to take the center position. Next, the player prioritizes moves along the vertical and horizontal axes and shares either the x or y coordinates of the opponent's last position. This attempts to create a partition in a shape of a box with the opponent inside the box.

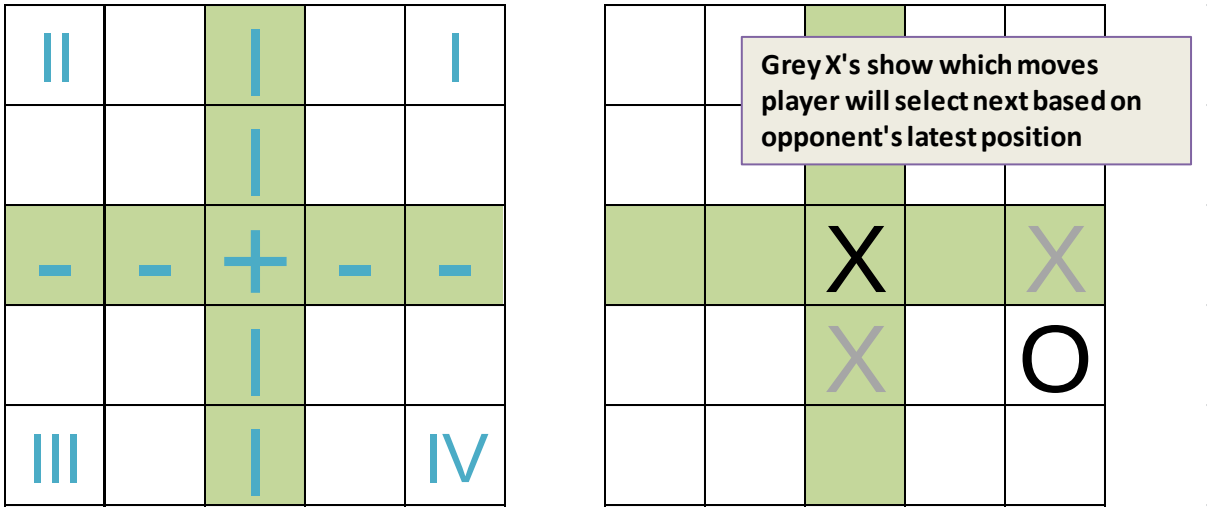

Figure 2: Left image shows the game board split into four quadrants. Green spaces show which moves the player will prioritize. Right image shows the player taking the center position and the opponent currently in one of the four quadrants.

Test Results

The script tournament.py was used to test the three custom heuristics by simulating matches between the game playing agent (which utilizes the three heuristics) and seven different opponents, with each opponent having its own heuristic. Since running the script resulted in varying results each time, the script was executed five times to gather more data and reduce variability (see **Figure 3**).

	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
Opponent	Won	Lost	Won	Lost	Won	Lost	Won	Lost
Win Rate	65.14%		62.86%		66.29%		52.57%	
Random	42	8	38	12	43	7	38	12
MM_Open	33	17	34	16	32	18	29	21
MM_Center	38	12	39	11	41	9	30	20
MM_Improved	31	19	31	19	30	20	22	28
AB_Open	28	22	24	26	31	19	20	30
AB_Center	31	19	25	25	34	16	20	30
AB_Improved	25	25	29	21	21	29	25	25

Figure 3: Results outputted by script 'tournament.py' after running the script five times.

Recommendation

Based on the results depicted in **Figure 3**, 'AB_Custom_2' (Heuristic #2) proves to be a slightly better heuristic compared to the other heuristics. This disproves the assumption that 'AB_Custom' (Heuristic #1) would be more effective despite it using a more sophisticated code. 'AB_Custom_3' (Heuristic #3) proves to be a significantly ineffective heuristic compared to the other heuristics and may be removed from the code. Based on this data, it would be recommended that Heuristic #2 would be used as the primary heuristic as it results in slightly more wins with little difficulty in implementation as it only results in seven lines of code.

.