Avery Heizler
Artificial Intelligence Nanodegree
Project 2: Isolation Game

Heuristic Analysis for Isolation Game AI Agent

Introduction:

For this project, students had to experiment with different types of heuristic functions and choose the one of three that performed the best while playing against a baseline agent that used AlphaBeta Search and Iterative Deepening, called "AB_Improved". This paper describes the basis for each heuristic function, its performance against the "AB_Improved" heuristic, analyzes the possible reasons for its performance, and attempts to justify the recommendation of the chosen heuristic function.

Heuristic Function 1:
This heuristic function evaluates the board in a two-fold process.
First it calculates a score of the board state based on the amount of moves available to the player versus twice the amount of available moves to the opponent. If the player has more than twice as many moves available as the opponent, the player will receive a positive evaluation score. The heuristic function then scales the difference in moves via the Manhattan distance between the player and the opponent. If the Manhattan distance between the player and the opponent is large, the player will find it difficult to block the opponent, therefore causing the board state to be penalized, else it will be rewarded.

The reasoning behind this heuristic function is that takes into account the idea that a board state in which the amount of the opponent's moves is few in number and the Manhattan distance between the player and opponent is small rewards the board state. From my playing of the Isolation game I have come to the understanding that when the distance between the player and opponent is minimal, the opponent has less available moves while the player will have an easier time blocking the moves of the opponent. As a result, the player will have a better chance of winning the game.

Opponent	**AlphaBeta_Improved_Score**	**AlphaBeta_Custom_Score**
Random	19 – 1	17 - 3
Minimax_Open_Move_Score	13 – 7	14 - 6
Minimax_Center_Score	16 – 4	17 – 3
Minimax_Improved_Score	10 – 10	12 – 8
AlphaBeta_Open_Move_Score	9 – 11	13 – 7
AlphaBeta_Center_Score	11 – 9	11 – 9
AlphaBeta_Improved_Score	11 – 9	11 - 9
Win Rate:	63.6%	67.9%

Avery Heizler
Artificial Intelligence Nanodegree
Project 2: Isolation Game

This heuristic function performs the best out of all the ones I have tried implementing, though its performance may not be very impressive. This particular heuristic funciton performs better against Minimax_Open_Move_Score, Minimax_Center_Score, Minimax_Improved_Score, and AlphaBeta_Open_Move_Score, but worse against Random. This may be due to the fact that it focuses mainly on the difference between the available moves to each player and on the Manhattan distance between the two players.
Please see game_agent.py for the code of Heuristic Function 1 ("custom_score").

Heuristic Function 2:
This heuristic function places importance on the possibility of limiting the opponent's future moves. It assigns a higher evaluation score to the board state if the possibility of future moves for the opponent in minimal; otherwise it will assign a lower evaluation score.

The reasoning behind this heuristic is simple. Continuously rewarding the board states in which the opponent is forced to have less available moves in the current game state as well as successive future game states will eventually result in a state where there will be no more available moves for the opponent while there will be available moves for the player. This is the ideal situation for a player in the game Isolation, as the player who makes the last move wins the game.

Opponent	**AlphaBeta_Improved_Score**	**AlphaBeta_Custom_Score_2**
Random	19 – 1	18 – 2
Minimax_Open_Move_Score	13 – 7	12 – 8
Minimax_Center_Score	16 – 4	15 – 5
Minimax_Improved_Score	10 – 10	14 – 6
AlphaBeta_Open_Move_Score	9 – 11	10 – 10
AlphaBeta_Center_Score	11 – 9	10 – 10
AlphaBeta_Improved_Score	11 – 9	10 – 10
Win Rate:	63.6%	63.6%

This heuristic function performs just as well as the AlphaBeta_Improved_Score in this instance. It sometimes has performed better, but not consistently. In this instance it performs better against Minimax_Improved_Score and AlphaBeta_Open_Move_Score, but slightly worse against Random, Minimax_Open_Move_Score, Minimax_Center_Score, AlphaBeta_Center_Score, and AlphaBeta_Improved_Score. This may be due to the fact that it is more of a defensive strategy and focuses solely on the available moves for the player and opponent, rather than including other factors in its calculation.
Please see game_agent.py for the code of Heuristic Function 2 ("custom_score_2").

Avery Heizler
Artificial Intelligence Nanodegree
Project 2: Isolation Game

Heuristic Function 3:
This heuristic function evaluates the board in the following manner. If the player and opponent have different amounts of available moves, the function subtracts the amount of available for the opponent from the amount of available moves of the player. If both the player and opponent have the same amount of available moves, then the Manhattan distance for both the player and the opponent relative to the center of the board is utilized. The function calculates the distance of the player to the center of the board and subtracts that from the opponent's distance to the center of the board, then normalizes that difference by dividing by 10. The resulting score will be between -1 and +1. The best-case scenario will be when the opponent's distance from the center is 6 (being in a corner) and the player is at position 0,0 (the center). This would result in a score of +0.6. The worst-case scenario would be when the opponent's position is 0,0 (the center) and the player's distance from the center is 6 (being in a corner). This would result in a score of -0.6. If both the player and opponent were the same Manhattan distance away from the center of the board, the result would be a score of 0.

The reasoning behind this heuristic function is the idea that the more moves available to the player, the better the chances are of winning. However, not all positions are equal. The function further exlpoits the idea that being closer to the center of the board will result in more future available moves, thus having a positional advantage.

Opponent	**AlphaBeta_Improved_Score**	**AlphaBeta_Custom_Score_3**
Random	19 – 1	17 – 3
Minimax_Open_Move_Score	13 – 7	13 – 7
Minimax_Center_Score	16 – 4	16 – 4
Minimax_Improved_Score	10 – 10	12 – 8
AlphaBeta_Open_Move_Score	9 – 11	9 – 11
AlphaBeta_Center_Score	11 – 9	12 – 8
AlphaBeta_Improved_Score	11 – 9	8 – 12
Win Rate:	63.6%	62.1%

This heuristic performs the worst of all three functions in this instance when compared to the performance of AlphaBeta_Improved_Score, though this is not always the case. In this instance it performs better against Minimax_Improved_Score and AlphaBeta_Center_Score, and performs worse against Random, and AlphaBeta_Improved_Score. This may be due to the fact that it relies on a two-fold process in which it first determines whether or not player and opponent have the same available moves, and if that is not the case it relies on the Manhattan distances of both the player and opponent relative to the center of the board. Please see game_agent.py for the code of Heuristic Function 3 ("custom_score_3").

Avery Heizler
Artificial Intelligence Nanodegree
Project 2: Isolation Game

Justification for the Chosen Heuristic:

Based on the results obtained by implementing the three different heuristic functions in the tournament, I recommend using the Heuristic Function 1 based on the following reasons:

1. Heuristic Function 1 performed the best against AB_Improved.
2. Heuristic Function 1 also has consistently performed better in the tournament than any other function used.
3. In order for a heuristic function to perform well it must be able to search to greater depths, therefore it must not be too computationally complex. While Heuristic Function 2 is simpler to compute, it may be too simple by not incorporating more factors into its calculation, such as the Manhattan distance between the player and the opponent. This tradeoff of complexity versus simplicity is cost-effective because the Manhattan distance is not too computationally exhaustive to compute for the results it delivers.