Problem:

This study looks at the development of a heuristic for an AI agent in the game of Knights Isolation on a 9x11 grid. The rules and description for the game can be found at here: <https://github.com/alekzandr/ai-nanodegree/tree/master/Adversarial%20Search>

Baseline Heuristic:

The baseline heuristic uses the number of liberties available to the user’s agents minus the number of liberties available to the opponent. Below is a Pseudocode representation for the baseline heuristic.

Define count\_moves(game\_state, player):

player\_location <- game\_state.get\_player\_location(player)

return length(game\_state.get\_liberties(player\_location))

Define baseline\_heurisitc(game\_state);

player\_0\_moves <- count\_moves(game\_state, 0)

player\_1\_moves <- count\_moves(game\_state, 1)

return player\_0\_moves - player\_1\_moves

Custom Heuristics:

For the study we use two variations of the Manhattan distance for our heuristic, Max Distance and Min Distance. The Max Distance variation will favor actions the place the agent farther away from the opponent. Conversely, the Min Distance variation will favor actions that place the agent closer to the opponent.

Define max\_distance(game\_state):

player\_loc <- game\_state.get\_location(player)

opp\_loc <- game\_state.get\_location(opponent)

return

Define min\_distance(game\_state):

player\_loc <- game\_state.get\_location(player)

opp\_loc <- game\_state.get\_location(opponent)

return -

Baseline Results:

Here we collect results from 2 rounds of 100 games using the baseline heuristic in different match parameters. Each round, the agent trades first play initiative against the opponent.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Baseline Heuristic | | | | |
| Search Depth | Move Timeouts (miliseconds) | Fair Match | Number Processes | Win Percentage |
| 1 | 150 | Y | 1 | 18% |
| 2 | 1000 | Y | 1 | 22% |
| 3 | 100000 | Y | 10 | 34.5% |
| 4 | 1000000 | Y | 10 | 31% |

Max Distance Heuristic Results:

Here we collect results from 2 rounds of 100 games using the custom heuristic and matching match parameters to the baseline:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Max Distance Heuristic | | | | |
| Search Depth | Move Timeouts (miliseconds) | Fair Match | Number Processes | Win Percentage |
| 1 | 150 | Y | 1 | 18% |
| 2 | 500 | Y | 1 | 12.5% |
| 3 | 10000 | Y | 4 | 23% |
| 4 | 100000 | Y | 10 | 22.5% |

Min Distance Heuristic Results:

Here we collect results from 2 rounds of 100 games using the custom heuristic and matching match parameters to the baseline:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Min Distance Heuristic | | | | |
| Search Depth | Move Timeouts (miliseconds) | Fair Match | Number Processes | Win Percentage |
| 1 | 150 | Y | 1 | 15.5% |
| 2 | 500 | Y | 1 | 17% |
| 3 | 10000 | Y | 4 | 39% |
| 4 | 100000 | Y | 10 | 44% |

We see more completeness at the expense of computational performance when combining the baseline heuristic with the developed Max Distance Heuristic as a weighted function called the aggressive greedy heuristic.

Define aggressive\_greedy(game\_state, weight\_1, weight\_2):

Return (weight\_1 \* min\_distance) + (weight\_2 \* baseline\_heuristic)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aggressive Greedy Heuristic | | | | |
| Search Depth | Move Timeouts (miliseconds) | Fair Match | Number Processes | Win Percentage |
| 1 | 150 | Y | 1 | 17% |
| 2 | 1000 | Y | 1 | 23.5% |
| 3 | 100000 | Y | 10 | 33% |
| 4 | 1000000 | Y | 10 | % |

Project Questions:

* What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?

The Max Distance, Min Distance, and Aggressive Greedy heuristics all use the distance between players as a feature for evaluating states. The Aggressive Greedy heuristic ties in a known good heuristic of counting the liberties available to each player.

* Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?

We see greater confidence in winning when using deeper searches. This comes at the cost of greater search time. The heuristics developed for this project relies more on accuracy then search speed.

Future Improvements:

In future iterations, we could implement a different search algorithm such as Monte Carlo Tree Search. Additionally, we could implement machine learning systems to learn the best plays in situations from past experience. Reinforcement Learning techniques have proved successful in these domains.