Assignment 1 Report

Connor Aleksandrowicz, Alex Dileo, Oleksandr Kalynyk, Marcus Youssif

Optimization of:

- **Uniform Cost Search -** Uniform cost search differs from A* Search by having only the beginning node being inserted into the priority queue, then each node is added only as it is needed. Complexity in the worst case scenario is O(m ^ (1+floor(l/e)))
- **A* Search -** computes cost of path to all open (not blocked) cells and the cost from that cell to the goal. f(n) = g(n) +h(n). At its worse, according to our optimization, it will have a Worst Case run time of O(|E|). We used queue.priorityqueue (as recommended in A* search algorithm) to perform repeated minimums for each node.
- **Weighted A* Search -** When weight is greater or equal to 1, f(n) = g(n) + w*h(n). Optimal when a greater number of expansions are required.
- **#Sequential A* Search -**

Additional Heuristics

- **Manhattan -** this is an admissible heuristic (never overestimates the cost of reaching a goal) because the distance computed for every path will have to be moved at least the number of spots in between itself and its correct position. Computationally inexpensive, as it only requires 2 subtractions and 1 addition.
- **Mini Manhattan -** manhattan distance divided by 4, computationally inexpensive to compute in addition to how Euclidean distance is computed. Minimizes the effect of the h(n) which allows for the result of f(n) to be more focused on g(n).
- **Euclidean Distance -** admissible heuristic, gives the shortest path between current node and goal. Computationally inexpensive, only requires 2 subtractions, the squaring on those results, and 1 addition. Minimizes the effect of the g(n) which allows for the result of f(n) to be more focused on h(n).
- **Euclidean Distance Squared -** self explanatory. Computationally inexpensive, it only requires one more square. Good heuristic as it exaggerates the difference between two paths that may be taken from the goal to the destination.
- **Mini Euclidean -** euclidean distance divided by 4, computationally inexpensive to compute in addition to how Euclidean distance is computed. Minimizes the effect of the h(n) which allows for the result of f(n) to be more focused on g(n).
- **Chebyshev -** allows for all 8 adjacent cells to be reached with a cost of one (manhattan distance does not allow for diagonal moves, only vertical and horizontal, resulting with a diagonal move to cost 2 in Manhattan distance). Computationally, requires 1 multiplication, 5 subtractions, and the calculation of one minimum.
- **Octile -** Similar to Chebyshev distance, with the modification that diagonal distance will have a cost of sqrt(2) rather than 1. Computationally is very similar to Chebyshev as well.

Analysis of 50 Benchmarks

- Over the 50 benchmarks that we ran, Average run time for A*Search is O(|E|), while Weight A* search had an average run time of O(w*|E|), where w is the weight of the node.

- We chose to use different data structures than those shown in examples. We figured that it would be best to use data structures that were efficient at getting and setting data when needed
- Our implementation was efficient because we used the proper data structures to limit the amount of memory required. As suggested, we implemented the priority queue in a way that would efficiently compute the A* Search and Weighted A* Search.
- The most efficient algorithm that ran was the Weighted A* Search. Specifically, when the weight was 2 or greater, but usually a close value to 2, the expansion of the nodes showed to be less than A* or Sequential A*, which means that there was less of a strain on the memory requirement.