A-Star Project Summary

February 19, 2017

1 Introduction

1.1 Goal

Implement and visualize the A-star pathfinding algorithm and its derivatives, and write a program that was modular (or "abstract"). Implement A-star with a variety of heuristics, both admissible and inadmissible.

1.2 Goals Achieved

1. Implemented A-star with the the Manhattan Heuristic and able to find the most efficient path
2. Implemented a system for selectively choosing which A-star derivative to use as well as grid size, size of obstacles, and the number of obstacles
3. Implemented a crude visualizer of the path that the program selected
4. Implemented all parts of the assignment (to the best of my understanding)
5. Implemented a variety of other heuristics that were either derivatives of the manhattan heuristic

2 Heuristics

1. Manhattan
2. Uniform cost (value returned is always 0)
3. Weighted Manhattan (manhattan distance multiplied by a constant value (2))
4. Exponential
5. Square root

3 Optimizations

Very few optimizations were performed, if any. Beyond the use of a priority queue, for the open set since insertion is O(log n) and ability to perform heapify in O(log n) time, and using a set for the closed set since insertion and retrieval are both O(1), there were no other optimizations that I was aware of. In fact, everything was par-for-the-course, really. All algorithms were written as-is. Space complexity was also neglected, or not really considered since it was doubtful that the program would run out of space. The only "optimizations" was the reduction of code complexity by modularizing a good number of parts of the program, thus reducing code complexity. Use of classes allowed for passing smaller groups of parameters to functions. Other than that, there is not a lot of optimizations done that I know of. The program ran in a reasonable amount of time and that was my benchmark.

4 Difficulties

1. The pseudocode provided was quite confusing, but that was most likely due to me being unfamiliar with the set theory syntax
2. Python, with its large and expansive library unfortunately did not have a min-heap implementation for objects so I had to write my own [1]

[1] There probably was but I could not find it or I failed to grasp that it was meant for that purpose