Jakob Daugherty

John Jolley

Zachary Rump

CS 3050 Final Project Report

Upon first glance, this project seemed simple enough; just move the robots around the map. Initially, we started with an object-orientated approach to the problem, but eventually found a simpler approach. Parsing the map into a two dimensional array proved rather easy. We wrote a simple parsing function to load each character in the map file into a 2D array. To find the starting an ending positions of the robots, we kept track of the x and y positions in the file and simply looked for the applicable symbol when iterating through each character. Once each position has been determined the coordinates are checked to see if the positions are closer than the allowed interference radius R (R is given as a command line argument.) With all input information gathered, it is now time to start moving around the map.

Since, the initial requirements did not require us to find the shortest path for the robot to travel, we used a simple depth first search algorithm to move the robots around the map. This proved to be the most complex in implementation, and debugging. The algorithm starts by looking at adjacent map spaces, checking if the robot has the ability to move to that space. This process continues, recursively, until the robot’s current coordinates match the desired final location. With this example the first robot is to move from S to E, and robot is to move from F to L. The path traveled is stored in a stack structure, and is added to the map before printing. This was the simplest implementation to hold the data, yet proved problematic when running. As you could easily step outside the bounds of the stack; causing a segmentation fault and crashing the program entirely.

In an effort to minimalize situations in which the proximity of the two robots is less than the given value r, we chose to move one robot at a time. Verifying proximity before moving and then moving only one robot at a time. Finally, after adding both paths to the map, the map is then printed out in its entirety. The running time of this particular implementation, is entirely determined by the size of the map. Since the search algorithm has to search all points in the map, in order to find its way around. The running time is, worst-case, O(V + E). Where V is the number of spaces that can be discovered by the robot, and E is the number of adjacent locations that can also be moved to from a particular point. In terms of memory, the program currently can only take in a map no larger than 100 by 100 (this can be changed by in creasing the value of BUFFER).

Working as a team with this project was a learning experience for all. We used GitHub to handle working separately on different aspects of the project. John designed and wrote all of the algorithm code. He also added the stack files and operations. Zach provided the use of his GitHub account as well as set up the repository. He helped to debug and tie the parsing and algorithm programs together, and was a great help when it came to learning Git commands. The parsing and printing functions where written by Jakob. He also wrote the final report for this project.