TDQC

Maze

Jack Spence

14 October 2018

1. Write-Up

1.1 Requirements

Requirements were to read a maze in from a file, and find a path in the maze from the start point, “>”, to the end point, “@”.

1.2 Suggested Features

Suggested feature attempted in my submission is to allow the maze read in to have doors, represented with “+”, allow to have water, which increases the cost of that node as well as being represented by “~”, and allowing the program to generate a map and solve it.

1.3 Syntax

Compile:

1. Make – Make alone will compile the default binary of Maze

2. make profile – Will produce the executable with the profile flags set to produce gmon.out

3. make debug – Produces executable with debug flags set

4. make clean – Cleans the executable as well as any .a, .o. and .outs in the current directory

Usage: ./Maze [filename] [-d] [-w] [-m *size*]

Note: -m with a size will take precedence.

2. Project Design Plans

2.1 Initial Design Plans

My initial design plan was to take the example code we had created in class over the past couple weeks to tackle this map solving. My idea was to modify the airplane example for Dikjstra’s algorithm to accommodate the mazes from the file.

2.2 What didn’t work

I could not come up with a good way to validate the maps. An example map, tests/map05.txt, should not be allowed, but my maze solver will handle it. The next and probably biggest thing was I did not have time to fully implement my maze generating algorithm. The in-place algorithm will generate maps but very sloppily, and some of the maps cannot be solved.

2.3 What went well

When it came time to implement the random generated mazes as well as the mazes from file, the system I had in place handled this very well. Nothing had to be changed other than what maze was actually being parsed by Dikjstra’s.

2.4 Conclusion

Without the examples we had built in class, I would have had an extremely hard time with this project. There was simply a massive amount of moving pieces that were required with my implementation that the dsa2 project on gitlab was essential.

3. Test Procedures

3.1 Test files

I’ve included a few test maps, some were handed out and I created a couple. They are located in tests/ in the root directory.

3.2 Testing Procedure

./maze tests/map01.txt – PASSED Expected: Passed

./maze test/map02.txt – PASSED Expected: Passed

./maze test/map03.txt – PASSED Expected: Passed

./maze test/map04.txt – Invalid map Expected: Invalid Map

./maze test/map05.txt – Solved Expected: Invalid Map

./maze test/door.txt – No Path Expected: No Path

./maze test/door.txt -d – PASSED Expected: Passed

./maze test/water.txt - No Path Expected: No Path

./maze test/water.txt -d – No Path Expected: No Path

./maze test/water.txt -w – PASSED Expected: Passed

./maze test/door-water.txt -w -d – PASSED Expected: Passed

./maze test/door-water.txt -w – No Path Expected: No Path

./maze test/door-water.txt -d – Path avoid water Expected: Path Avoid water

./maze -m 10 - PASSED Expected: Passed # This is chance

./maze -m -10 – No File Expected: No File found

./maze -m 100 – PASSED Expected: Passed #This is Chance

./maze test/map01.txt -m 25 – Generated map Expected: Generated Map