Elena Escalas

CS 31 Project 3 Report

1. **Notable Obstacles**

I initially tried doing navigateRoute without modifying the route string to make it easier on myself, and I got the position values very confused, so eventually I just decided to modify route into newRoute so that I would only have lower case letters to work with instead of numbers. The newRoute I created simply had everything as a letter, so for example if there were ‘e2’ in route, it would be converted into ee and added to newRoute using for-loops

Another one of my big obstacles was I kept getting “isWall invalid value: (0,3)” in my navigateSegment portion. It took me a while to realize that this was because I was decreasing the row by one, and I had made sure r was in the grid by saying that it was less than getRows(), but in actuality it also needed to be greater than zero. When I would input a value for r it wouldn’t exit the loop if r went below 1 because the only restriction on r was that it had to be less than the number of rows, not greater than zero.

I also made the silly mistake many times of setting my stepsMoved to nsteps AFTER I returned a value. I found many cases where I had changed a variable after returning a value, and then the integer would not set because I would return a value and exit the function before being able to set the value to the other integer.

1. **Pseudocode**

bool isRouteWellFormed (string route)

check every character in the string:

if the first character is a digit:

return false;

if the character in this position is not a digit

and is not a letter:

return false;

if character is a letter:

set letter as lowercase letters

if letter is n, s, w, or e:

continue to next character;

else return false;

else if character is a digit:

set digit1 to that digit;

set digit2 to the next character;

if digit1 is negative:

return false;

else if digit2 is a digit:

if digit2 is negative:

return false;

else continue on to next position;

else continue on to the next position;

if every position has been checked and false has not been returned:

return true;

int navigateSegment (int r, int c, char dir, int maxsteps)

set nSteps to -1;

if dir character is invalid: return -1;

if maxSteps integer is negative: return -1;

if the row and column are not within the grid

return -1;

else if the row and column are at a wall

return -1;

else while row and column are within the grid limits

if row and column are a wall

break out of the loop;

else increase nSteps by 1

if the lowercase direction input indicates north:

move the point being tested up one by subtracting 1 from r;

if the lowercase direction input indicates south:

move the point being tested down one by adding 1 to r;

if the lowercase direction input indicates east:

move the point being tested right by adding 1 to c;

if the lowercase direction input indicates west:

move the point being tested left by subtracting 1 from c;

if nSteps is greater than or equal to the maxSteps:

return maxSteps;

else return nSteps;

**To test whether or not positions were valid I created another function:**

bool clearPosition(int row, int column)

if the row or column is greater than the grid size allows:

return false;

if the row or column is less than the grid size allows:

return false:

if the row and column position is a wall:

return false;

else return true;

navigateRoute(int sr, int sc, int er, int ec, string route, int& nsteps)

if route is not well formed or the **start/end positions aren’t valid**:

leave nsteps unchanged and return 2;

set k equal to zero;

create the string newRoute;

while k is not equal to the size of the route string:

if the first character is a digit:

leave nsteps unchanged and return 2;

if the character at position k is a letter and the next character is a zero:

continue to the next character (basically skip this letter)

if the character at position k is a letter:

add the lowercase of that letter to newRoute and move onto next position (increase k by one);

if the character at position k is a digit:

set number equal to the integer version of the character;

if the character before the digit is a letter:

if this number is zero:

continue onto next character;

add letter before the digit the digit number of times to newRoute then move onto the next letter;

if the character before the digit is a number:

convert the two numbers into a two digit number;

add letter before the digit the digit number of times to newRoute then move onto the next letter;

if the newRoute is not well formed: leave nsteps unchanged and return2;

set integer m equal to zero;

set integers mr and mc equal to sr and sc respectively;

set stepsMoved and possible steps equal to zero;

while m is not equal to the size of the newRoute string:

set possible\_steps to navigatesegment for mr, mc, the character at position m, and 1;

if the possible steps are less than one:

set nsteps to stepsMoved and return 3;

else add possible\_steps to steps moved;

if the letter at position m is ‘n’:

subtract possible\_steps from mr (moving row);

else if the letter at position m is ‘s’:

add possible\_steps to mr (moving row);

else if the letter at position m is ‘e’:

add possible\_steps to mc (moving column);

else if the letter at position m is ‘w’:

subtract possible\_steps from mc (moving column);

increase m by one (to check next character in newRoute)

if moving row is equal to end row,

and moving column is equal to end column:

set nsteps to stepsMoved and return 0;

else: set nsteps to stepsMoved and return 1;

1. **Test Data**

bool isRouteWellFormed(string route)

no segment at all: (“”)

no segment is a valid route, and should return as such

a single letter with no digits: (“n”)

one letter is also a valid route

one segment with no digits: (“nswewwesews”)

the program should be able to tell that even without numbers, and only the letters n,s,e, and w the route is still valid

a segment with capitalized letters: (“NsWeSSwEE”)

the program should be able to identify the

a segment with an unidentified letter: (“nSweewnfee”)

f is invalid because it is not one of the 4 appropriate letters indicating a valid direction

a string starting with a digit: (“3nswe”)

a string cannot start with a digit because it indicated steps without first indicating a direction in which the steps lead

a segment with one digit in the proper location: (“nsewswews3sewews4s”)

should be a valid route because there should be between zero and two digits after every letter

a segment ending with a digit: (“nswews8swee9”)

should be a valid route because the digit has an indication of the direction the steps will go

a segment with two digits in a row: (“e2wesw11wew”)

should be a valid route because there should be between zero and two digits after every letter

a segment with three digits in a row: (“we2ewes123ew”)

should not be a valid route because there should be between zero and two digits after every letter

a segment with a non-digit non-letter character: (“ew2\*we5s”)

should not be valid because the character does not indicate a direction nor a number of steps taken in that direction

int navigateSegment(int r, int c, char dir, int maxSteps)

(r,c) not an empty grid position

function should return -1

invalid direction character (not n,s,e, or w in upper or lower case):

function should return -1

maxSteps not proposed number of steps to take in indicated direction:

return -1 because maxsteps is not a valid integer

robot starting at (r,c) could successfully take that number of steps in indicated direction w/o running into a wall or falling off the grid:

function returns initially inputted number of steps

robot starting at (r,c) cannot successfully reach maxSteps because path runs into wall:

test for running into a wall to make sure that value returned equals only the number of steps possible before hitting the wall

robot starting at (r,c) cannot successfully reach maxSteps because path runs off of the grid:

test for running off to the east and west as well as for north and south to make sure that the value returned equals only the number of steps possible for the robot to take before falling off

int navigateRoute(int sr, int sc, int er, int ec, string route, int&nsteps)

(sr, sc) and (er, ec) are valid empty grid positions and route is valid and robot ends at (er,ec):

function should return 0 and correctly set nsteps to the number of steps taken in the route

(sr, sc) and (er, ec) are valid empty grid positions and route is valid, but robot does not end at (er,ec):

function should return 1 and correctly set nsteps to the number of steps taken in the route

(sr,sc) is not a valid empty grid position:

function should return 2 and leave nsteps unchanged

(er,ec) is not a valid empty grid position:

function should return 2 and leave nsteps unchanged

route not valid

function should return 2 and leave nsteps unchanged

(sr,sc) and (er,ec) are valid empty grid positions and route is valid, but route falls off of edge of grid:

function should return 3 and set nsteps to the number of steps

before the route fell off the grid

(sr,sc) and (er,ec) are valid empty grid positions and route is valid, but route hit a wall:

function should return 3 and set nsteps to the number of steps

before the route hit the wall