Name: Sidharth Sudhir

UID: 805 759 182

CS31 Project 3 Report

Obstacle # 1: Indexing beyond the length of a string

When writing the first draft of code, I collected the length of the string so that it could be used in the for loops and if statements beyond. One thing I noticed about ‘plan portions’ was that they always had to end with an ‘R’, ‘L’, ‘r’, or ‘l’. If they ended with an integer (the number of steps to move), it was an incorrect plan portion/plan altogether. So, I introduced an if-statement at the start to immediately return False to the hasCorrectForm function if it came across a plan that ended in an integer. When implementing this if-statement, I tried to access the character at the end of the string called plan by using plan.at(k), where k was the length of the string. This obviously led to me trying to access a character that was beyond the length of the string. Luckily, because I have prior experience, I was able to figure out that this was an error quickly and switched it to plan.at(k-1).

Obstacle # 2: Problems with determineSafeDistance()

When writing the code for this, I had a lot of confusion about the directions in how to detect the walls/ends of the grid. This didn’t arise from my ability to write logic, but instead because of my ignorance towards the indexing of the table b/w 1 and 0. To fix this, I physically drew out a table with row and column numbers on a sheet of paper and manually mapped how my for and if statements should be written to successfully break when the car runs into a wall or the end of the rxc grid.

Additionally, while running the different tests, certain cases caught errors in my code for this function. These were fixed using minor changes in logic and syntax.

Pseudocode

*hasCorrectForm function pseudocode*

bool hasCorrectForm()

collect length of integer plan;

check if plan string is empty:

-If so, wrong form

-Return wrong form if final

Iterate through plan string,

-return wrong form if 3 consecutive integers,

-Check if alphabets are not L and R.

-Return false if either are true

-Return true otherwise

*determineSafeDistance function pseudocode*

Int determineSafeDistance()

test that all inputs into functions meet guidelines;

detect walls and the limits of the grid and finds the maximum steps till that point

For each direction,

Add or subtract unit from each position to move across x, y.

Calculate max steps

*obeyPlan function pseudocode*

Int obeyPlan()

- Check that the parameters of the function meet all the requirements

iterate through every character of the string ‘plan’

-if the character is a letter, turn the direction of the car using changeDirection

If not,

Find the max number of steps you can move forward determineSafeDistance

If safedistance = nsteps in obeyplan function,

Add nsteps and continue.

Else

Add maxsteps to nsteps and move forward.

Test Cases

setSize(3,4);

setWall(1,4);

setWall(2,2);

setWall(3,2);

assert(hasCorrectForm("5rL00L0R09R7L")); // normal obeyable test case to see if the code can handle 0s

assert(hasCorrectForm("2R1r")); // simple obeyable test case

assert(!hasCorrectForm("1Lx")); // false test case

assert(determineSafeDistance(3, 1, 'N', 2) == 2); // checking output with suitable input

assert(determineSafeDistance(3, 1, 'q', 3) == -1); // checking output for unsuitable input

int len;

len = -999; // so we can detect whether obeyPlan sets len

assert(obeyPlan(3,1, 3,4, 'S', "LL2R2r2L1R", len) == 0 && len == 7); // simple obeyable test case

assert(obeyPlan(3,1, 3,4, 'N', "1Lx", len) == 2 && len == -999); // test case with unsuitable plan

assert(obeyPlan(2,4, 1,1, 'w', "3R1L", len) == 3 && len == 1); // obeyable case with different direction

assert(obeyPlan(2,1, 4,4, 'N', "rr2l2l", len) == 1 && len == 4); // obeyable test case with repetition

These test cases test whether the functions take in the right input, and whether the output matches the specified output.