Kelly Yu

Dr. Smallberg

CS 31

October 21, 2021

**1. Notable Obstacles**

Some notable obstacles I faced was checking if we were past the edge of the wall or not. Initially, I kept using the constants MAXROW and MAXCOLS. However, I realized later that using those constants did not make my code flexible enough if I changed the grid size for different test cases. I replaced the constants with the functions getRows and getCols since they changed with the size of the grid. I also had trouble figuring out how many times I should loop the code in the determineSafeDistance function and when the counter variables should be implemented. I didn’t want the counter to count the step as successful before the new space was checked as safe, but I also needed the car to take the step before being able to check the space. Another obstacle I faced was realizing what nsteps represented. For a while, I thought nsteps represented the number of steps the user wanted to take after every turn in the maze. This misunderstanding made me confused about how each of the 3 mandatory functions were related to each other. I later realized that what I though nsteps represented was actually the purpose of the plan. nsteps just tracked the number of successful steps the car completed.

**2. Pseudocode**

declare the three functions

//function 1 -- check if plan is formatted correctly

check every character in the plan if:

- all characters are either letters or numbers

- letters consist of only R’s and L’s when capitalized

- last character in plan is a turn direction

- turn letter always follows 0 to 2 digits

return true if plan meets all conditions; return false otherwise

//function 2 -- check if car can move in specified direction & steps

proceed if:

- position and steps inputs are valid

- create a counter to track number of successful steps

set all direction to uppercase & check the direction we are facing

check if each step is valid in that direction:

take a step

if at a wall or off the edge

stop function; return value of successful steps already taken

if step is valid

add 1 to number of successful steps

if we successfully move the specified number of steps

stop function; return value of specified number of steps

check direction inputs; function fails if input invalid

//function 3 -- check if we can reach endpoint using plan

check if position, direction, and plan inputs are valid (capitalize direction inputs)

create variable to store direction changes

create total successful steps counter

for every character in valid plan, check:

if character is a letter

if letter is upper/lowercase L

depending on current direction, turn car to the left

if letter is upper/lowercase R

depending on current direction, turn car to right

if not a letter, character is a digit

check if we have double digits

if moving that many steps is safe,

move that number of steps

add number of steps to successful steps counter

if not,

add the number of steps we were able to take to the total successful steps

store total successful steps to nsteps

stop checking the plan

if not double digits, we have a single digit

repeat safety check we did for double digits

if we didn’t crash, store total successful steps to nsteps

check if we reached endpoint or not

**3. Test Code**

\*\* using the grid and wall setup from project 3 spec

setSize(3,4);

setWall(1,4);

setWall(2,2);

setWall(3,2);

**hasCorrectForm**

|  |  |
| --- | --- |
| Reason | Test Data |
| Empty plan; no plan portions | assert(hasCorrectForm("")); |
| Plan with 0 digits followed by a letter | assert(hasCorrectForm("L")); |
| Plan with 1 digit followed by a letter | assert(hasCorrectForm("1L")); |
| Plan with 2 digits followed by a letter | assert(hasCorrectForm("12L")); |
| Plan with 3 digits followed by a letter | assert(!hasCorrectForm("123L")); |
| Plan with letters other than R, r, L, l | assert(!hasCorrectForm("12S")); |
| Plan with characters other than R, r, L, l, and digits | assert(!hasCorrectForm("12+")); |
| Plan that does not end with a turn letter | assert(!hasCorrectForm("12")); |
| Plan with at least one plan portion | assert(hasCorrectForm("12L5R")); |

**determineSafeDistance**

|  |  |
| --- | --- |
| Reason | Test Data |
| Cannot start at a wall | assert(determineSafeDistance(2, 2, 'E', 3) == -1); |
| Cannot start off the edge | assert(determineSafeDistance(0, 0, 'E', 3) == -1); |
| Number of steps taken cannot be negative | assert(determineSafeDistance(0, 0, 'E', -3) == -1); |
| Number of steps taken cannot be 0 | assert(determineSafeDistance(0, 0, 'E', 0) == -1); |
| Direction input ≠ upper- or lower-case N, S, W, E characters | assert(determineSafeDistance(0, 0, 'i', 0) == -1); |
| Went go off the edge to (1,0);  furthest is 0 steps to stay in the grid | assert(determineSafeDistance(1, 1, 'N', 2) == 0); |
| Crashed into a wall at (1,4);  furthest is 2 steps to reach (1,3) | assert(determineSafeDistance(1, 1, 'E', 3) == 2); |
| Successfully take all 2 steps South | assert(determineSafeDistance(1, 1, 'S', 2) == 2); |
| Successfully take all 2 steps North | assert(determineSafeDistance(3, 3, 'N', 2) == 2); |
| Successfully take all 1 step East | assert(determineSafeDistance(3, 3, 'E', 1) == 1); |
| Successfully take all 2 steps West | assert(determineSafeDistance(1, 3, 'W', 2) == 2); |

**obeyPlan**

|  |  |
| --- | --- |
| Reason | Test Data |
| Starting off the grid | **int** len;  len=-999;  assert(obeyPlan(0,0, 3,1, 'S', "1LL1L2L2L", len) == 2 && len == -999); |
| Starting at a wall | **int** len;  len=-999;  assert(obeyPlan(2,2, 3,1, 'S', "1LL1L2L2L", len) == 2 && len == -999); |
| Ending off the grid | **int** len;  len=-999;  assert(obeyPlan(1,1, 5,6, 'S', "1LL1L2L2L", len) == 2 && len == -999); |
| Ending at a wall | **int** len;  len=-999;  assert(obeyPlan(1,1, 2,2, 'S', "1LL1L2L2L", len) == 2 && len == -999); |
| Invalid direction input | **int** len;  len=-999;  assert(obeyPlan(1,3, 3,1, 'r', "1LL1L2L2L", len) == 2 && len == -999); |
| Plan is not formatted correctly | **int** len;  len=-999;  assert(obeyPlan(1,3, 3,1, 'S', "2", len) == 2 && len == -999); |
| Successfully get from (1,3) to (3,1) using plan of 6 steps | assert(obeyPlan(1,3, 3,1, 'S', "1LL1L2L2L", len) == 0 && len == 6); |
| Didn’t crash but didn’t reach the endpoint; moved 4 steps | assert(obeyPlan(1,3, 3,1, 'S', "1LL1L2L", len) == 1 && len == 4); |
| Crashed into wall midway through the plan; furthest is 3 steps taken | assert(obeyPlan(1,3, 3,1, 'S', "1LL1L1L1L", len) == 3 && len == 3); |
| Went off the edge midway through the plan; furthest is 2 steps taken | assert(obeyPlan(1,3, 3,1, 'S', "1LL2L1L1L", len) == 3 && len == 2); |