**Project 3 Report**

**Notable Obstacles**

1. A method to cycle through the path instruction string wasn’t immediately obvious to me, and the third function hindered me while processing digits after changing direction. The links provided on the Main CS31 Website aided me greatly to process the data types and overcome this obstacle.
2. For the second function, determineSafeDistance, I had a tough time coming up with the for loop to create the movement in one direction. Certain zyBook excepts helped me tweak my code and helped in replication to suit all four directions.
3. Designing test data to ensure functions worked properly was not easy on this project. The large number of variables that can be changed with the grid made it difficult to acquire an all-encompassing set of test data.

**Pseudocode**

**function hasCorrectForm(string plan)**

if plan ends with digit

return false // to avoid out of range errors

else check for optional digit using if plan at k is a digit

k++

check for another optional digit

k++

if alphabets within the string are not R, L, r, l

return false;

k++

return true if all the above is satisfied.

**function determineSafeDistance(int r, int c, char dir, int maxSteps)**

getRows and getCols

if dir is not a direction letter

return -1

if maxSteps is negative,

return -1

if (r,c) exists at an invalid grid position or a wall,

return -1

define function and use it to set delta (change) values

increment steps

increment/decrement x/y-coordinates

check if row/column is valid for each step

if results in running off the grid or hitting a wall

break

**int obeyPlan(int sr, int sc, int er, int ec, char dir, string plan, int& nsteps)**

set conditions for valid start and end positions

if (sr, sc) exists at an invalid grid position or wall

return 2

if dir is not a direction letter

return 2

if the given plan is not valid

return 2

assign an integer to each cardinal direction

Set nsteps to 0

for loop to process route string

if char in string plan is an alphabet

permute the index of the direction accordingly

calculate index mod 4

if char in string plan is a digit

if the next char is also a digit

steps = (10 \* digit 1) + (digit 2)

else

steps = digit 1

conduct stepwise movement for loop with different direction indices, and stop when invalidity strikes

nsteps++

if start row and start col are equal to end row and end column

return 0

else if the robot did not hit a wall or get to the end position

return 1

**Test Cases**

assert(hasCorrectForm("2R1r")); // Detects correct plan is being accepted.

assert(!hasCorrectForm("1Lx")); // Detects if hasCorrectForm checks presence of unknown character.

assert(determineSafeDistance(3, 1, 'N', 2) == 2); // Checks if the correct safe distance is running successfully.

int len = - 888;

assert(obeyPlan(3, 1, 3, 4, 'S', "LL2R2r2L1R", len) == 0 && len == 7); // To check if obeyPlan sets len.

assert(obeyPlan(1, 1, 4, 4, 'e', "2r3l1l", len) == 0 && len == 6); // Checking a correct plan with successful start and end point.

assert(determineSafeDistance(3, 1, 'q', 3) == -1); // Checking if the function detects incorrect dir.

assert(!hasCorrectForm("w2+n3")); // Checking if the function returns false for the symbol.

assert(obeyPlan(1, 1, 1, 1, 'e', "2rr2r", len) == 0 && len == 4); // Checking for a successful case.

assert(obeyPlan(2, 4, 1, 1, 'n', "L1L2R1L", len) == 1 && len == 4); // Checking for successful pass but without reaching the end point.

**1x1 grid with no walls**

(1, 1, 'n', 1) - make sure that function returns 0 steps, because the robot will hit the edge of a grid after its only step.

(1, 1, 'W', 50) - make sure that function returns 0 steps, because if the robot hits the edge of the grid, just one step is attempted.

(5, 5, 's' ,2) - make sure that the function returns -1 because the start position is invalid (outside the grid).

**1x2 grid with a wall at (1, 2)**

(1, 1, 'e', 1) - makes sure that function returns 0 steps, because robot will hit the wall after its only step

(1, 1, 'e', 50) - make sure that function returns 0 steps, because robot will hit the wall after just one steps are attempted

(1, 2, 'w', 2) - make sure that the function returns -1 because the start position is invalid (on a wall space)

**10x10 grid with walls at (1, 2) to (9, 2) and (2, 7) to (10, 7)**

(4, 1, 10, 10, "N10e2S5sW10", nsteps) - make sure that the function returns 3 and nsteps is unchanged because the path is not well-formed

(0, 1, 1, 1, "", nsteps) - make sure that 0 is not a valid starting position row. Returns 3

(1, 0, 1, 1, "", nsteps) - make sure that 0 is not a valid starting position column. Returns 3

(-10, 1, 10, 10, "", nsteps) - make sure that function returns 3 because the starting position is invalid (above the grid)

(20, 1, 10, 10, "" ,nsteps) - make sure that function returns 3 because the starting position is invalid (below the grid)

(4, 1, 10, 10, "s50E50N50e50s50", nsteps) - make sure that the function returns 2 because it hits the bottom edge of the grid and then sets nsteps to 6

(4, 1, 10, 10, "N50s5N50e50s50", nsteps) - make sure that the function returns 2 because it hits the top edge of the grid and then sets nsteps to 3