a) Notable Obstacles:

* A first obstacle was having many different variables working on different functions. This was solved by leaving all variables to a local level, avoiding variables to affect each other.
* Verifying if a letter was upper or lower case, this was solved by generalizing using the toupper() function.
* Initially I didn’t setup the library, so an obstacle was making the code to detect a wall. This was solved after adding the grid library and using the isWall().
* When checking if there was a wall outside the bounds ended the program, created a function that checked if the position is the grid
* When checking for the digits following a character, if the character was at the end of the word an error occurred. This was solved by doing an if case to avoid exceeding the string size.

b) Code description:

**isRouteWellFormed:**

The code verifies that the route has only defined characters and can be read correctly.

Pseudocode:

Set initial return variable to true

check if route is empty, if its not continue

go through every character

if character is a digit

if the first character is digit, return variable is set to false

if there’s any 3 digit number, return variable is set to false

if character is letter

if character isn’t upper or lower n, s, w or e, return variable is set to false

if character isn’t number or letter, return variable is set to false

return variable, if no errors were encountered, it will return true

**navigateSegment:**

The code checks the maximum steps possible to advance in a given direction

Pseudocode:

if the starting point isn’t on the grid return -1

if the starting point is on top of a wall return -1

if the maxSteps inputed is negative return -1

if the direction char is North

repeat from 0 to the max possible steps

advance one row up (r—)

check if the new position isn’t on the grid or is a wall, return the number of cycles

if no obstacles are encountered return the max possible steps

if the direction char is South

repeat from 0 to the max possible steps

advance one row up (r++)

check if the new position isn’t on the grid or is a wall, return the number of cycles

if no obstacles are encountered return the max possible steps

if the direction char is East

repeat from 0 to the max possible steps

advance one row up (c++)

check if the new position isn’t on the grid or is a wall, return the number of cycles

if no obstacles are encountered return the max possible steps

if the direction char is West

repeat from 0 to the max possible steps

advance one row up (c—)

check if the new position isn’t on the grid or is a wall, return the number of cycles

if no obstacles are encountered return the max possible steps

if the direction char isn’t upper or lower n, s, e or w return -1

**navigateRoute:**

The code reads the route and navigates through it, the function returns a number depending of how successful the route was and it also changes a variables value to the number of steps given in total.

create variables to store: actual Row position, actual column position, accumulate

number of steps and instant maximum number of steps (for navigateSegment)

call isRouteWellFormed

if false return 2

if starting point or ending point is out of bounds return 2

if starting point or ending point is over a wall return 2

call moveAnalyzer

analyzes the string and stores in two separate arrays the direction and the magnitude intended in the code for each letter.

repeat for each letter in the string code

call navigateSegment for the maximum steps in a given direction

add the number previously obtained to a variable storing total steps

if the the numbered returned by navigate segment isn’t maxSteps

set nsteps to the accumulate number of steps

return 3

if the direction letter is North

subtract the result from navigateSegment to the actual Row position

if the direction letter is South

add the result from navigateSegment to the actual Row position

if the direction letter is East

subtract the result from navigateSegment to the actual Column position

if the direction letter is West

add the result from navigateSegment to the actual Row position

set nsteps to accumulated number of steps

if the robot reached the endpoint

return 0

else, if the robot didn’t reach the endpoint

return 1

c) Test scenarios:

Basic Testing

* grid bigger than (20, 20) (setSize(100,21)); Invalid size
* getting a correct sized grid (setSize(10,10)); Grid creates successfully
* setting a wall in an invalid position (setWall(21, 21)); The wall can’t be created
* setting a wall in a valid position (setWall(1,1)); The wall creates successfully
* invalid start point for draw function (draw(-2, 2, 2, 2)); Error, grid not created
* invalid endpoint for draw function (draw(2, 2, -1, 200)); Error, grid not created
* draw works (draw(2, 2, 2, 2)); the grid is outputed

isRouteWellFormed Testing

* Route starts with a digit (isRouteWellFormed(“4w3nesw4”)); Function returns false
* Route has a 3 digit number (isRouteWellFormed(“W123ens”)); Function returns false
* Route has a character different than s, w, e, n or a digit (isRouteWellFormed(“HolaQueAse9”)); Function returns false
* Route receives upper and lower case n, S, W, e (isRouteWellFormed(“EsWnwN”)); Function returns true
* Route receives a mix of n, s, w, e and numbers of 0, 1 and 2 digits (isRouteWellFormed(“NS2e03w1s11”)); Function returns true
* Route is empty (isRouteWellFormed(“”)); Function returns true

**\*\*\* All the testing was done under the following grid:**

**setSize(10, 10)**

**setWall at:**

**1- (2,4)**

**2- (2,6)**

**3- (4,4)**

**4- (4,6)**

**5- (6,4)**

**6- (6,6)**

**7- (7,7)**

**8- (8,8)**

**9- (9,9)**

navigateSegment Testing

* maxSteps is negative (navigateSegment(2, 2, ’N’, -2)); Function returns -1
* starting position is on a wall (navigateSegment(2, 4, ’s’, 3)); Function returns -1
* starting outside the grid (navigateSegment(15, -2, ‘W’, 12)); Function returns -1
* non valid character (navigateSegment(2, 2, ‘k’, 2)); Function returns -1
* encounter a wall (navigateSegment(2, 2, ‘E’, 4)); Function returns 1
* encounter the edge of the grid (navigateSegment(2, 2, ’n’, 5)); Function returns 1
* move without finding an obstacle (navigateSegment(2, 2, ’s’, 4)); Functions returns 4
* move south with no encounters (navigateSegment(2, 2, ’S’, 1)); Function returns 1
* move north with no encounters (navigateSegment(2, 2, ’n’, 1)); Function returns 1
* move east with no encounters (navigateSegment(2, 2, ‘E’, 1)); Function returns 1
* move west with no encounters (navigateSegment(2, 2, ‘w’, 1)); Function returns 1
* receive a lower case letter (navigateSegment(2, 2, ‘w’, 1)); Function returns 1
* receive an upper case letter (navigateSegment(2, 2, ’N’, 1)); Function returns 1
* move to wall directly next (\*wall at 2, 4\*) (navigateSegment(2, 3, ‘E’, 2)); Function returns 0
* start next to an edge and move towards it (navigateSegment(1, 1, ‘w’, 4)); Function returns 0

navigateRoute Testing

* route not well formed (navigateRoute(2, 2, 10, 10, “123NotOkMate123”, nSteps)); Function returns 2;
* start point outside of the grid (navigateRoute(2, -2, 10, 10, “S1E2n3”, nSteps)); Function returns 2;
* end point outside the grid (navigateRoute(2, 2, -2, 40, “S1”, nSteps)); Function returns 2
* start point over a wall (\*wall at 2,4\*) (navigateRoute(2, 4, 10, 10, “n”, nSteps)); Function returns 2
* end point over a wall (\*wall at 2,4\*) (navigateRoute(2, 2, 2, 4, “s2”, nSteps)); Function returns 2
* robot encounters a wall (navigateRoute(2, 2, 10, 10, “E9s3N4W1”, nSteps)); Function returns 3; nSteps = 1
* robot hits the edge (navigateRoute(2, 2, 10, 10, “S10w23”, nSteps)); Function returns 3; nSteps = 8
* robot ends at the endpoint (navigateRoute(2, 2, 10, 10, “s8E8”, nSteps)); Function returns 0; nSteps = 16
* robot doesn’t hit wall but doesn’t reach end (navigateRoute(2, 2, 10, 10, “se3s2w2n2”, nSteps)); Function returns 1; nSteps = 10
* inputing an empty string for code (navigateRoute(2, 2, 2, 2, “”, nSteps)==0 && nSteps==0)); Function returns 0 (Sp and Ep are equal); nSteps == 0