One of the issues I experienced when coding this program was the fact that I missed out on several minor pieces of logic when coding the program that I simply didn’t account for, and experienced issues when debugging and attempting to fix these errors. For example, I had an issue returning 3 initially in the driveCourse method due to the fact that my driveSegment method wasn’t working as expected, and it took a while for me to debug and diagnose this issue. Another issue was that I had a for-loop inside a while-loop for my driveCourse method, but I wasn’t resetting the for-loop index so it wasn’t parsing through all the values in the string like it was supposed to be. Minor errors like this led to a lot of bugs in my code that I had to tediously iron out over time, even though most of the base logic was coded properly the first-time through.

My program’s design consisted of the three required methods: isCourseWellFormed(), driveSegment(), and driveCourse(). For the isCourseWellFormed() method, we were meant to parse through a string and return true if the syntax was completely correct. In order to code this, I made a string, called acceptable, of all the acceptable values that could be found in the string, and looped through to make sure that the input string, course, only had these values within t. Next, I made a string dir of all the directions, and looped through in order to find specific locations in ‘course’ with direction characters. Then I checked the three steps in front of the direction character to weed out directions followed by triple digits.

it looked something like this:

while ( index < string course){

for(string of all acceptable chars){

find out if char in course contained in acceptable string

if it is add to counter

}

if counter == 0 { return false }

}

for (index < course size){

for (index < direction string size) {

if there are 3 digits in front of a direction char{ return false }

}

}

return true;

Next, for the driveSegment() method, I made a direction string ‘d’ once again, this time using it to make sure my input character dir was a valid character. Then, based on whether my dir was east, west, north or south, I changed my row and column index value accordingly, based on the max number of steps I could travel without a barrier or the input maxSteps if there were no blockages.

it looked something like this:

for( string of directions ){

if input direction = char from string

save it

}

if( direction is east){

move col var to the right until it hits a wall or goes out of bounds

or stop based on max steps

}

if west { same except move col down }

if north { same except move row upwards }

if south { same except move row downwards}

Finally, I coded driveCourse(). This was my most buggy code by far, but I was able to fix it eventually. I first once again used a string of directions to find out which directions were in the input string course. Then I saved the index position, moved up index values in course to find the following digits, and calculated the step using driveSegment() based on whether there were 0, 1 or 2 digits following my direction. After that, I checked to see if my step value was lower than the maxDig (dig), which would mean that there was a blockage in my math, leading me to returning 3. Otherwise, I incremented my sc and sr input values according to the direction moved, and incremented nsteps by the number of steps (step) moved. Finally, I checked for whether I reached my final destination, returning 0 if I did and 1 otherwise.

it looked something like this:

while (index < course string size) {

for( string of directions ){

if input direction = char from string

save it

check digits in front

save

calculate number of steps using driveSegment()

}

}

if( direction is east){

move col var to the right based on number of steps

}

if west { same except move col down }

if north { same except move row upwards }

if south { same except move row downwards}

all of this is done while returning the correct numbers for each particular result (0, 1, 2, 3)

When testing my code, I used the following test-cases, based around the main method maze provided in the spec with the specifications pasted below:

setSize(3,4);

setWall(1,4);

setWall(2,2);

setWall(3,2);

**TEST CASES:**

**driveCourse(3, 1, 3, 3, “n2e2S02”, nsteps) - RETURNS: 0, nsteps: 6**

**driveCourse(3, 1, 3, 4, “n2e2s2w”, nsteps)** - supposed to hit wall

initially returned 1 for the function return, 7 for nsteps

I realised I wasn’t checking for wall

**NOW RETURNS: 3, nsteps: 6**

**driveCourse(3, 1, 2, 4, “N2E02s2w02”, nsteps)** - didn’t work initially, returned 1,6, changed code so returned a 3 if max dig travelled according to second drive segment method was 0.

**NOW RETURNS: 3, nsteps: 6**

**driveCourse(3, 1, 2, 4, "n02e03ss1", nsteps)** - check for if variable step (steps that are possible to move) < variable dig (steps moved in course string). Moves east onto a wall.

**RETURNS: 3, nsteps: 4**

**driveCourse(3, 1, 3, 4, “N02E2sSe1”, nsteps) -** moves from start to finish

**RETURNS: 0, nsteps: 7**

**driveCourse(2, 1, 1, 3, “ne”, nsteps) - RETURNS: 1, nsteps: 2**

**driveCourse(2, 1, 1, 3, "e", nsteps) - RETURNS: 3, nsteps: 0**

**driveCourse(2, 4, 2, 4, "wsE", nsteps) - RETURNS: 1, nsteps: 3**

**driveSegment(3, 1, 'n', 1)** - used to return 2 instead of one, fixed by making sure ‘check’ variable less than or equal to maxsteps before returning

**NOW RETURNS: 1**

**driveSegment(1, 1, 'e', 20) - RETURNS: 2**

**driveSegment(2, 4, 'W', 314) - RETURNS: 1**

**driveSegment(3, 2, 'S', 2) - RETURNS: -1 (starts on a wall)**

**isCourseWellFormed(“ne") - RETURNS: true (1)**

**isCourseWellFormed(“abc”) - RETURNS: false (0)**

**isCourseWellFormed(“n022E3s4") - RETURNS: false (0)**

**isCourseWellFormed(“n02E03s4wW1") - RETURNS: true (0)**