TouchLab Programming Tasks

As part of your application process, please complete the following 2 programming tasks. Comprehending the tasks as written, good coding style, as well as communication are more important than getting everything ***exactly*** right, so stress what's important (good coding, etc).

In this zip file is a maven project, set up with both eclipse and Intellij Idea projects. You can import these projects and use maven, or start new projects, as long as they're using reasonably coherent source layout. You may also use whatever lib jars you want.

If you don't use eclipse or intellij and would like to use your own IDE that's fine.

Time counts a little bit. Minutes don't matter, but try to complete the projects without interruption.

Task 1 - Grid animation

A grid animation system. The program is provided with a value for width and height of the grid, and then a set of "actor" descriptions. Each actor is initialized, then on each "frame" that we step through, those actors do whatever they're type does to update its status.

Each actor has a "type", position, and direction.

Position is row, column. Zero based.

Direction is 0-7, indicating up, up-right, right, down-right, down, down-left, left, up-left.

Type is one of the following:

- Line - Continue in the specified direction, 1 move per frame
- Still - Ignore direction. Don't move.
- Veer Left - Move 1 spot in the direction specified, but turn counter-clockwise in an ever-widening fashion. *Counter-clockwise would subtract from the direction.* Turn 1 number counter-clockwise the first frame, stay the same direction the next frame, turn one counter-clockwise on the 3rd, stay the same direction for BOTH the 4th and 5th frame, then turn one CC again on the 6th, stay the same direction for the next 3 frames, then turn CC again on the 10th. Add one more frame to skip each time (skip 4, then skip 5, then skip 6, etc).
- Veer Right - Same thing as Veer Left, but turn clockwise instead. *Clockwise adds to the direction.*
- Random - Pick a new spot randomly anywhere on the grid for each frame.

The data file for the actors will be in the following format. One actor per line. A type code, the

row, the column, then the direction. If the data is not needed for the type specified, it MAY not be provided (but don't assume it won't be there). Still needs position, but not direction. Random doesn't need anything. Example:

L,2,5,3
VR,4,2,5
S,1,1
R
L,3,3,2

You can find this data in the file **testgrid.txt** in the zip file.

Each type has a string code: Line(L), Still(S), Veer Left(VL), Veer Right(VR), Random(R).

Program Inputs:

1. Filename for actor descriptions
2. int width of grid
3. int height of grid
4. int number of frames to run

Program Output:
The program should output the frame, type, and position of each actor, to the standard output stream.

For the data set above, with a width and height of 10, and 3 frames:

0,L,2,5
0,VR,4,2
0,S,1,1
0,R,5,9
0,L,3,3
1,L,3,6
1,VR,4,1
1,S,1,1
1,R,3,2
1,L,3,4
2,L,4,7
2,VR,4,0
2,S,1,1
2,R,5,0
2,L,3,5

Note, the VR had a direction of 5 in the first frame, but started in its specified position. In the second frame, we turn to 6 and move 1 in that direction. For the third frame, VR skips a

direction change and stays in direction 6. If we added another frame, the VR would turn again to 7, then stay on 7 for 2 frames, and turn to 0 on the frame after. It would then stay on 0 for 3 frames, and turn to 1 on the frame after that (stay for 4, turn to 2, stay for 5, turn to 3, stay 6, turn to 4, etc). Random just moves around, as expected.

Important Notes

The most complicated parts of this are really the 'Veer' types, so if things are running long, you can make those simpler (maybe just have them turn each frame, or every other frame, rather than changing their values). Getting the output EACTLY right is less important than understanding the basic idea of the specs and seeing how you approach the problem.

Here we are looking for spec comprehension and OOP skill.

Please continue on to the next page for Task 2

Task 2 - Count letter repeats.

You have a simple text file provided as input, as well as a number of lookback spots. You want to count the number of times each letter is duplicated close to itself. A duplicate can be checked up to "lookback" spots. So, for example:

```
aritawtb
```

If you are given that string as the input file, and a lookback value of 3, you will find 1 repeat, of the letter t

```
aritawtb
```

If you are given a lookback of 5, you'd get 1 t and 1 a

```
aritawtb
aritawtb
```

For each letter you look at, you're allowed to look back the supplied number of letters. If there are multiple matches, it only counts once. For example, with a lookback of 3:

```
aaaaa
```

In this string, the first 'a' doesn't count because there's nothing before it. The second 'a' counts. The third 'a' also counts, only once. The fourth 'a' counts as well, only once. Finally the fifth 'a' counts as well, again only once. Your total results would be "a" 4 times.

Program Inputs:

1. Filename of the data file
2. int number of lookback spots (must be greater than 0)

Program Outputs:

Output data to the standard output stream, one line per letter, in the following format:
a-4
b-1
c-3
(continue for the rest of the alphabet)

Important Notes

All counts are case-insensitive. There MAY be characters other than letters in the file. They also count for the lookback, so if you find a period or a space, that counts towards lookback distance (but you only need to COUNT and output letters). See the file **testdups.txt** for an example input. No known limit to input file size is assumed. Could be small like the example, or huge.

Here we are looking for algorithm understanding and practical programming skill.