Assignment 1

Counting Cells

This assignment is based on funny cells. Please look up the following webpage for details: http://www.cse.iitd.ac.in/~aseth/assg/funnycells/funnycells.html In this assignment, you are expected to program *Counting Cells* that can count the number of Susceptible Cells on the map in a distributed manner.

Objective

The objective is to count the number of Susceptible Cells on the map. When multiple counting cells are launched on an arbitrary sized map, each of them must participate in a counting process such that at the end of this process, by accumulating results from all of your cells, the correct number of Susceptible Cells present on the map can be found out.

Specifically, what your cells must do is:

1. Count susceptible cells in mutually exclusive portions of the map. Be careful to not double count!

2. Communicate their counts among each other in a way that enables one of them to accumulate the correct total

3. Log the final result to the console.

Note: It is easy to just have one cell count all the susceptible cells in the network, but that is not the point of the assignment. Your goal should be to count the cells in the least possible time, which implies that all cells must participate in the counting process and add up their results. This is the fun part of the assignment, because you have to create your own protocols for reliable communication in a decentralized manner! To make things more interesting, some cells may also die randomly :O

Implementation

You are given an empty *CountingCellImpl* class which looks like this:

```
public class CountingCellImpl implements CellularProcesses {

    public CountingCellImpl(Integer type, Integer energy, String cellId,
        RateLimBufferedReader in, RateLimPrintWriter out) {}

    public void startCell() {}

}
```

You have to fill in the implementation with having the cells send commands to the server. The source file (***CountingCellImpl.java***) is placed in the funnycells.0.1/src/assg/funnycells/cells folder. On running build.sh, the source code will be compiled and the .class file placed in funnycells.0.1/build/assg/funnycells/cells.

The FunnyCell class, from which CountingCellImpl inherits, actually creates a socket for communicating with the server and passes the input (*in)* and output (*out)* streams to your constructor, which are of type RateLim. To read and write, you can then use the in.readline() and out.println() statements.

For example, to send a MOVECELL message to server (message formats are given on the website), you can use the following set of statements:

```
out.println(assg.funnycells.server.Cell.MOVECELL);
out.println("incx=1");
out.println("incy=1");
out.println("");
```

Other commands you will also use are the DROPCHEM and SENSECHEM commands.

Execution

You first have to launch the server as follows:

java -cp "lib/java-getopt-1.0.13.jar;build" assg.funnycells.server.Coordinator -c funnycells.conf -l [log level]

-c [config file]: The configuration file, default being funnycells.conf present in the current directory
-l [log-level]: 0 (no logging), 1 (info), or 2 (debug)

To test your program, a simple *countingcelllaunch.pl* script is provided which will launch your *counting cells* in a random pattern on a map of arbitrary size and shape between 11x11 and 50x50. As shown in the screenshot in this document, **each row will have exactly one counting cell**. It can be launched as follows:

perl countingcellslaunch.pl [map width] [map height] [number of susceptible cells]

The script ensures that a square contains at the most one susceptible cell.

And remember that whenever you want to test with a different map size, you should change the map size in *funnycells.conf* as well.

Communication

Remember that the cells cannot directly talk to one another. They can only manufacture molecules and dump these molecules into the medium, and other cells in the neighborhood can sense these molecules. Sensing consumes the molecule. You therefore have to develop your own encoding schemes for molecular communication between your cells. Some hints to get you started:

1. Susceptible cells continuously produce presence molecules of the type *gggc-atat-attt-cccg*, where *gggc* is the molecule starting signature, *cccg* is the molecule ending signature, *atat* is the broadcast destination, and *attt* is the presence indicator. So, the very first thing the counting cells have to do is to move around and sense for presence molecules produced by susceptible cells. This can be done by issuing the SENSECHEM command to the server, indicating the prefix you want to sense as *gggc-atat-attt*. The server will respond with the relative locations of any presence molecules, which you can be used to infer susceptible cells in the neighborhood.

2. Keep in mind that susceptible cells produce presence molecules only at fixed intervals, and these molecules also timeout after a while. So the counting cells have to be careful to not move very fast and skip a susceptible cell. The lifetime of a molecule is defined by the *moltimeout* parameter in the configuration file, and the default value is 100ms. The susceptible cells produce presence molecules every *ratetimeout* times *moltimeout*, that is, every 500ms.

3. Depending upon what strategy you use for counting, you may want to develop similar presence molecules so that your counting cells can detect other counting cells in their neighborhood. These can be molecules of the type *gggc-atat-attt-xxxx-cccg*, which are the same as presence molecules by susceptible cells but with an additional quartet to

signify that they are presence molecules for counting cells. The molecules can then be dropped into the medium with the DROPCHEM command.

4. Another thing to keep in mind is that you don't want to sense your own presence molecules! You can do this by delaying sensing to after your presence molecule has expired. And just to confirm, you can also encode your cell-id into the *xxxx* portion of your presence molecule so that you know whether it was your own molecule you sensed or somebody else's.

5. When you want your cells to communicate their individual counts to each other, you again have to encode that information on to a special molecule. You will have to develop your own molecules, maybe something like *gggc-agcg-xxxx-yyyy-cccg*, where the *xxxx* portion can indicate a counting molecule, and the *yyyy* portion can be the count encoded on to the *a t c g* alphabet. Encoding the count can be as simple as finding the base-4 equivalent of the count, and then substituting *a* for *0*, *t* for *1*, *c* for *2*, and *g* for *3*. Cells will then produce and consume molecules through the DROPCHEM and SENSECHEM commands as before.

Logging your result

You are required to use the logger provided with the assignment to output your result. A Logger object can be used to log messages to consoles or files at different levels of importance of the message. This simply involves obtaining a reference of the default **java.util.logging.Logger** instance. A default logger can be created with the following statement:

```
Logger logger = Logger.getDefaultLogger();
```

You can then log messages using the logger.info(String str) method.

At the end of the counting process, one cell must log a string to the console in the following format:

```
SUSCEPTIBLE CELL COUNT = XXX
```

where XXX is the number of susceptible cells counted by your counting cells.

Report

Your report should contain a description of your approach (algorithm) to the distributed counting problem. Use diagrams wherever relevant and keep your description simple and brief. There are no marks for being unreasonably wordy.

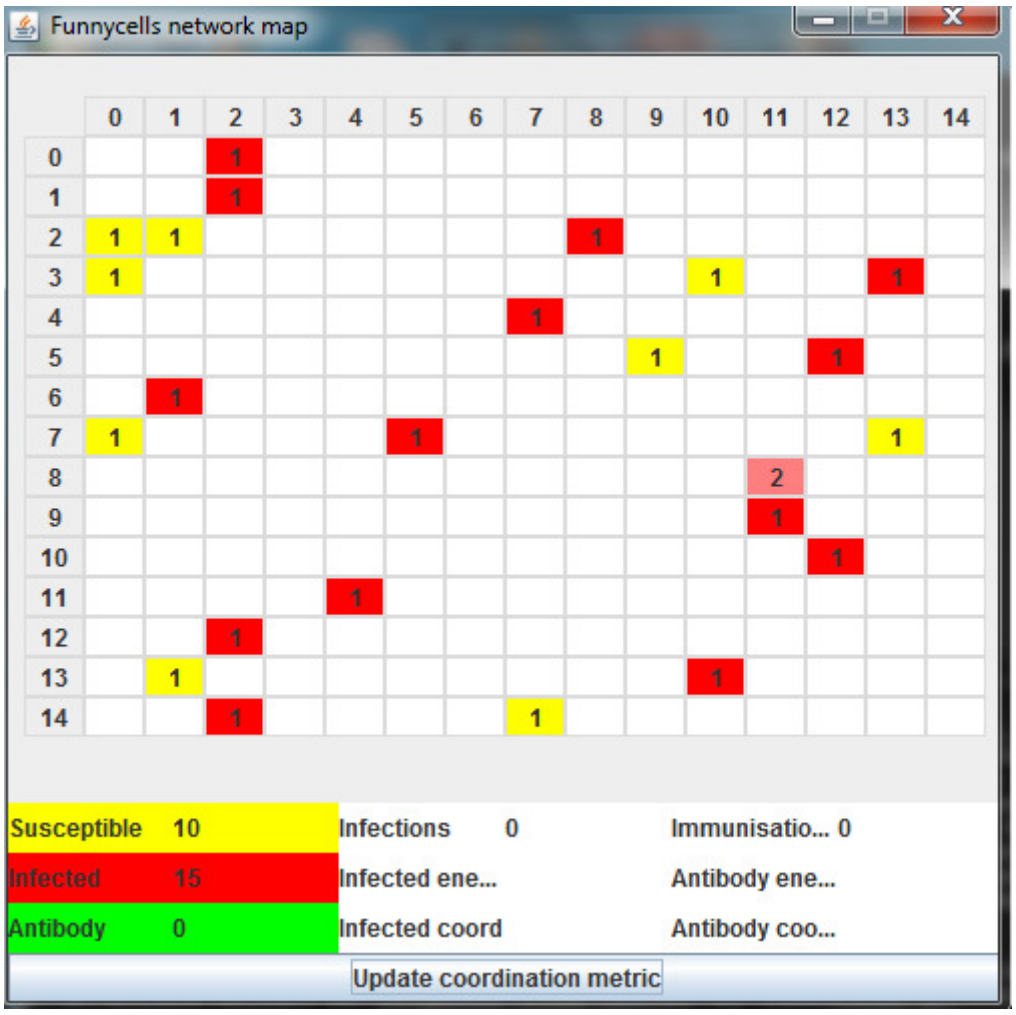
Submission Guidelines

You are expected to adhere to the following submission guidelines:

1. The assignment must be submitted on the moodle online course management system.

2. Your submission *must* contain 2 things: source code ***CountingCellImpl*** and your report in pdf format.

3. Your submission must be a .tar.gz file named as your entry number. The archive must contain one folder with the name as your entry number. This folder must further contain your source file ***CountingCellImpl*** and your report.


Funnycells network map
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 1 1
1 1 1
2 1 1 1
3 1 1 1
4 1
5 1 1
6 1
7 1 1 1
8 2
9 1
10 1
11 1
12 1
13 1 1
14 1 1
Susceptible 10
Infected 15
Antibody 0
Infections 0
Infected ene...
Infected coord
Immunisatio... 0
Antibody ene...
Antibody coo...
Update coordination metric