Assignment 2: Winning the Funny Cup™!

Alright! This is now the grand finale that we've all been waiting for! There are about 19-20 teams, so we will form 4 groups of about 5 teams each. Each group will have a winner with every team pitted against all other teams in the group (${}^{n}C_{2}$ sets of matches per group), and the winners from the 4 groups will compete for the Funny Cup. Here's how it will all happen:

1. **Implementation:** Choose a name for your team, say **FunnyTeam**, and choose a name for your class, say **assg.funnycells.cells.FunnyTeamCellImpl**.

2. **Molecular communication rules:** Your cell will be instantiated automatically through scripts either as a **RED (infected)** cell or a **GREEN (antibody)** cell. The nature/color of the cell isn't important for this assignment, what is important is that you note the type specified during instantiation of your cell, and produce molecules accordingly. That is, if your cell has been instantiated of type *assg.funnycells.server.CellState.INFECTED*, you must behave as an infected cell and produce infection molecules. Similarly, if it is of type *assg.funnycells.server.CellState.ANTIBODY*, you must behave as an antibody. A code snippet to parameterize this is given in the Appendix.

3. **Game rules:**

    a. Each game will last for 120 seconds, and will be played on a 15x15 grid with 7 initial cells per team and 12 susceptible cells. The maximum energy per cell will be 600000 molecular units. Energy renewals are not available. The layout of cells on the grid will be random. Remember that a single infected/antibody cell cannot infect a susceptible cell; you need a quorum of at least two cells to trigger a successful infection. When infected, the susceptible cell will become yet another infected cell and participate in the infection process.

    b. Wall bouncing will be turned off. So, the topology will effectively be that of a toroid rather than a finite grid.

    c. To win a game, the winning team must have more than 10 cells than the losing team after 120 seconds, and the number of remaining susceptible cells must be less than or equal to 5. Otherwise the game will be declared as a draw. This effectively means that your aim is to increase the population of your cells to a little more than double of your initial strength of 7, and not leave enough susceptible cells around to give a chance to your opponent to make a comeback.

    d. Each pair of teams will play a set of 6 games with each other – 2 points will be awarded to the winner, or 1 each in case of a draw. The winner of a set will be declared as the winner between the two teams.

e. After all the ${}^{n}C_2$ sets of matches are run per group, the winner from the group will advance for the finals. If there is a tie in a group, the tie will be resolved by playing another set of 6 matches between the tied teams, but this time on a clustered layout of the susceptible cells.

f. The winners from the four groups will play another ${}^{4}C_2$ matches of 6 games each to find the ultimate Funny Cup winner!

4. **Testing:** The *funnygroup.pl* script will be used to run games for an entire group. It automatically starts the server and all the cells, so you don't have to start the server separately.

   perl funnygroup.pl [*group list file*] [*games per set*] [*timeout*] [*scenario*] [walled]

   Here, [*group list file*] is the list of teams in the group, with each line in the file containing a CSV of *team-name, class-name.* [*games per set*] is 6, [*timeout*] is 120000, [*scenario*] is either *random* or *cluster,* and [*walled*] is 0 for this assignment. For testing purposes, you can pit your own implementation against itself by making a dummy group list file of 2 lines as follows:

   FunnyTeam,assg.funnycells.cells.FunnyTeamCellImpl
   FunnyTeamDuplicate,assg.funnycells.cells.FunnyTeamCellImpl

   Let's call it *mygroup.lst*. Then you can run this as:
   perl funnygroup.pl mygroup.lst 6 120000 random

   You can also test your implementation against the reference implementations, with and without pheromones: *assg.funnycells.cells.AdiPheCellImpl* and *assg.funnycells.cells.AdiNopheCellImpl*.

5. **Submission:** This will be in several stages.

   a. April 12th demo: This version will be tested to confirm that your implementation works correctly for both types of instantiations as INFECTED and ANTIBODY cells. If it does not, you will have 2 days to turn in a corrected version otherwise you will not be allowed to participate in the tournament.

   b. April 15th mock drill: A mock drill of the entire tournament will be held. You will have a chance to observe the strategies of your competitors and improve your methods.

   c. April 18th: Due date for the final version of your implementation.

   d. April 19th: The Funny Cup tournament.

These deadlines are hard and will not be changed! You should also submit a report explaining your strategy. Marking will be based on your report, the correctness of your implementation, the strategy, and bonus marks will be awarded based on your performance in the tournament.

Appendix

```
package assg.funnycells.cells;

import assg.funnycells.server.CellState;
import assg.funnycells.util.Configuration;

public class FunnyTeamCellImpl implements CellularProcesses {
        String myMolPrefix, opMolPrefix, infectionMol;
        Configuration config;

        public FunnyTeamCellImpl(Integer type, Integer energy, String cellId,
                                 RateLimBufferedReader in, RateLimPrintWriter out) {
                …
                …
                config = FunnyCell.getConfiguration();
                if(CellState.invNewType(type) == CellState.INFECTED) {
                        myMolPrefix = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                      config.getVal(Configuration.RECEPTORS,
                                                    Configuration.INFECTED);
                        opMolPrefix = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                      config.getVal(Configuration.RECEPTORS,
                                                    Configuration.ANTIBODY);
                        infectionMol = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                      config.getVal(Configuration.RECEPTORS,
                                                    Configuration.ANY) +
                                     config.getVal(Configuration.AFFECTORS,
                                                    Configuration.INFECTION) +
                                      config.getVal(Configuration.RECEPTORS,
                                                    Configuration.STOP);
                } else {
                        myMolPrefix = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                      config.getVal(Configuration.RECEPTORS,
                                                    Configuration.ANTIBODY);
                        opMolPrefix = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                      config.getVal(Configuration.RECEPTORS,
```

```
                                                    Configuration.INFECTED);
                        infectionMol = config.getVal(Configuration.RECEPTORS,
                                                    Configuration.START) +
                                       config.getVal(Configuration.RECEPTORS,
                                                    Configuration.ANY) +
                                      config.getVal(Configuration.AFFECTORS,
                                                    Configuration.IMMUNISATION) +
                                       config.getVal(Configuration.RECEPTORS,
                                                    Configuration.STOP);
                }
                …
                …
        }

        public void startCell() {
                // myMolPrefix is prefixed to molecules you produce for internal communication
                // opMolPrefix is what you look for while sensing for the opponent's cells
                // infectionMol is the infection (or immunization) molecule
                …
        }
}
```