Swarm Intelligence
Homework 7

Exercise 1: Install ARGoS and Buzz [0 points]

In the remainder of the course, we're going to use the ARGoS multi-robot simulator and the Buzz programming language. To install the necessary software, do the following.

Open a terminal window and install ARGoS with the following commands. Change the folder where you installed your files to your own configuration.

```
# Assuming you are going to download the repository here
$ mkdir -p ~/WPI_SwarmIntelligence/
$ cd ~/WPI_SwarmIntelligence/

# Download the ARGoS sources
$ git clone https://github.com/ilpincy/argos3
$ cd argos3
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DARGOS_DOCUMENTATION=OFF ../src
$ make -j8
$ sudo make install

# This command is necessary only on Linux
$ sudo ldconfig

# Test that it worked: you get a list of recognized plug-ins and no errors
$ argos3 -q all
```

Next, install Buzz:

```
# Assuming you are going to download the repository here
$ cd ~/WPI_SwarmIntelligence/

# Download the ARGoS sources
$ git clone https://github.com/buzz-lang/Buzz
$ cd Buzz
$ mkdir build
$ cd build
$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ../src
$ make -j8
$ sudo make install
```

```
# This next command is necessary only on Linux
$ sudo ldconfig

# Test that it worked: get the following output and no errors
$ bzzparse
Usage:
  bzzparse <infile.bzz> <outfile.basm> [stringlist.bst]
```

Exercise 2: Group Size Detection [100 points]

In this exercise, you are going to implement a simple algorithm for calculating a *very rough estimate* of the size of a group of agents in a completely decentralized manner. This algorithm is called *quorum sensing* and it is used by certain bacteria to know when to launch an attack on the body of the host. The algorithm works as follows:

- The agents are the cells of a $W \times H$ grid. Consider the cases $5 \times 5, 10 \times 10, 20 \times 20$.
- The agents do not know how many they are (that's what they need to estimate!), nor their position in the grid.
- The simulation proceeds in steps in the same fashion as the synchronization algorithm in HW6.
- At any time, the agents can be either *susceptible* or *refractory*.
- At any time, a susceptible agent has a probability P of initiating a "signal wave" by emitting a signal. An agent can initiate a signal only once throughout the duration of an experiment.
- Upon receiving a signal, any susceptible neighbor emits a signal too, thus continuing the "signal wave".
- Any time it emits a signal (initiated or forwarded), the agent enters the *refractory* state. In this state, the agent ignores any received signal and cannot initiate a signal. This state lasts for R steps, after which the agent swicthes back to *susceptible*.
- Any time it emits a signal (initiated or forwarded), the agent increases by 1 its estimate of the group size. The group size is initialized to 0 for all agents.
- In principle, the simulation finishes when all the agents have signalled. However, the agents have no way to know this; therefore, any agent considers itself "done" when it has received no signal for $1/P$ continuous steps. When all agents consider themselves "done", the simulation ends.

The pseudocode of the algorithm can be formalized as follows:

```
input:
  W = grid width [int]
  H = grid height [int]
  P = initiation probability [float]
  R = refractorytimer [int]

init:
  initiated = false [boolean]
  size = 0 [int]
  state = Susceptible

step:
  if (state == Susceptible)
```

```
    if (neighbor signalled)
      emit signal
      state = Refractory
      refractorytimer = R
      size = size + 1
    elif (not initiated) and (random() < P)
      emit signal
      state = Refractory
      refractorytimer = R
      initiated = true
      size = size + 1
    endif
  else
    refractorytimer = refractorytimer - 1
    if (refractorytimer <= 0)
      state = Susceptible
    endif
  endif
```

1. Implement the above algorithm in a language of your choice, as long as it's Python. No need to use ARGoS and Buzz this week. That will be for next week and the rest of the semester.
2. Look for the value of P and R that produces the best estimate across different grid sizes.
3. Does it matter if we use 4-distance or 8-distance?
4. This algorithm is obviously very inaccurate. How would you make this algorithm better? You can propose a modification of this algorithm, or a completely different algorithm.

The latter 3 points should be discussed in a short 2-page report.

Deliverables

The usual deliverable instructions are in order:

Submit an archive called `LastnameFirstname.zip` with the following structure:

```
LastnameFirstname/
  ex1.pdf          (MUST be a PDF!)
  README.txt       (a plain-text file that describes how to run your code,
                    along with the dependencies)
  <your code files>
```