Multiagent simulations in Java

MSc Interactive Entertainment Technology
AI for IET (CS7032)
S. Luz (luzs@cs.tcd.ie)

October 15, 2013

1 Goals

In this lab we will take a look at a Java-based simulation environment called Repast[1], run two simple demos, learn the basics of how to build a simulation, and analyse and modify a simulation of economic agents (Epstein and Axtell, 1996).

2 Setting up the environment

The simulation software you will need may have already been installed on the machines in the IET Lab. If not, please download *Repast*, version 3[2] from the course's software repository[3]. And uncompress it to a suitable directory (folder) in your user area. In the rest of this document we will refer to the location of the Repast directory (folder) on these machines as `REPAST_HOME`.

The libraries you will need in order to compile and run the simulations are located in `REPAST_HOME/` and `REPAST_HOME/lib/`. These include simulation-specific packages (such as `repast.jar`), CERN's scientific computing libraries (`colt.jar`), chart plotting libraries (`plot.jar`, `jchart.jar`), etc.

The source code for the original demos is located in `REPAST_HOME/demo/`. You will find modified versions of some of the demos for use in this lab[4] on the course website. Just download and uncompress them into the same folder where you uncompressed repast3.1.tgz, so that your directory tree will look like this:

```
--|
  |
  |-- repast3.1/
  |
  |-- simlab-0.8/ --|
                    |--- bugs/
                    |--- mousetraps/
                    |--- sscape-2.0-lab/
```

[1] Repast is an acronym for *REcursive Porous Agent Simulation Toolkit*.
[2] Newer versions might work as well, but I haven't tested them.
[3] https://www.scss.tcd.ie/ luzs/t/cs7032/sw/repast3.1.tgz
[4] http://www.scss.tcd.ie/ luzs/t/cs7032/sw/simlab-0.8.tar.gz

The demos can be run via the shell scripts (Unix) or batch files (Windows) provided. These scripts are set so that they will "find" the repast libraries in the above directory structure. If you uncompress repast to a different location, you will have to set the `REPAST_HOME` variable manually in these scripts so that it points to the location of repast on your machine.

Documentation for the simulation (REPAST) and scientific computing (COLT) API's is available in javadoc format in `REPAST_HOME/docs/{api,colt}`, respectively. Further information on how to build and run simulations in Repast can be found in

`REPAST_HOME/docs/how_to/how_to.html`.

3 Running two demos

We will start by running two demos originally developed to illustrate the *Swarm* simulation environment and ported to Repast:

Mouse trap: a simple discrete-event simulation the illustrates the dynamic scheduling capabilities of Repast (and Swarm, the simulation system for which it was first developed). The source code contains detailed commentary explaining how the simulator and model work. Please take a close look at it.

Heatbugs: an example of how simple agents acting only on local information can produce complex global behaviour. Each agent in this model is a 'HeatBug'. The world (a 2D torus) has a spatial property, heat, which diffuses and evaporates over time. The environmental heat property is controlled in `HeatSpace.java`[5]. The world is displayed on a `DisplaySurface`[6] which depicts agents as green dots and warmer spots of the environment as brighter red cells. The class `HeatBugsModel`[7] sets up the environment, initialise the simulation parameters (which you can change via the GUI) distributes the agents on it and schedules actions to be performed at each time 'tick'. Agents release a certain amount of heat per iteration. They have an ideal temperature, and assess at each step their level of 'unhappiness' with their temperature at that step. Locally, since no agent can produce enough heat on his own to make them happy, agents seek to minimise their individual unhappiness by moving to cells that will make them happier in the immediate future. Although all agents act purely on local 'knowledge', globally, the system can be seen as a distributed optimisation algorithm that minimises average unhappiness. The original code has been modified so that it plots the average unhappiness for the environment, showing how it decreases over time.

In order to run a demo, simply, `cd` to its directory (created by uncompressing the lab archive), and use `run.sh` (or `run.bat`). If you want to modify the simulations, and use `compile.sh`[8] (or `compile.bat`) to recompile the code.

The purpose of running these simulations is to get acquainted with the simulation environment and get an idea of what it can do. So, feel free to experiment with different parameters, options available through the GUI, etc. Details on how to use the GUI can be found in the How-To documentation:

`REPAST_HOME/how_to/simstart.html`

[5] uchicago/src/repastdemos/heatBugs/HeatSpace.java
[6] uchicago.src.sim.gui.DisplaySurface
[7] uchicago.src.repastdemos.heatBugs.HeatBugsModel
[8] For emacs+JDEE users, there is a `prj.el` file in `bugs` which you may find useful.

4 Simulating simple economic ecosystems

Now we will explore the *SugarScape* demo in some detail. The demo is a partial implementation of the environment described in chapter 2 of (Epstein and Axtell, 1996). The growth rule G_α (page 23), the movement rule M (page 25), and the replacement rule R (PAGE 32) have been implemented.

A slightly modified version of the original demo is available in the archive you downloaded for the first part of this lab, in `sscape-2.0-lab/`. Shell scripts `compile.sh` and `run.sh` contain the `classpath` settings you will need in order to compile and run the simulation. Remember that **if you are not using the repast3.1 distribution provided** or **if the distribution has not been uncompressed as described above**, you will need to modify the scripts (.bat files) so that variable `REPAST_HOME` points to the right place.

4.1 Brief description of the sscape simulation

At the top level, the directory `sscape-2.0-lab/*` contains, in addition to the scripts, an (ASCII) description of the "sugar landscape": `sugarspace.pgm`. Open this file with your favourite editor and observe that it is filled with digits (0-4) which represent the distribution of "food" on the environment.

The source code for the simulation is in `sugarscape/*.java`[9]. All Repast simulations must contain a *model* class, which takes care of setting up the simulation environment, and at least one *agent* class. The file `SugarModel.java` contains an implementation of a model class, whereas `SugarAgent.java` and `SugarSpace.java` implement simple agents. Note that in this simulation, as is often the case, the environment is also conceptualised as an agent, whose only behaviour in this case is described by the growth rule.

4.2 Exercises

These are some ideas for exercises involving the sscape simulation. Please answer (at least) **three** of the questions below and submit your answers by the end of next week.

1. Modify the topography of the environment by changing `sugarscape.pgm` (copy the original file into, say, `sugarscape.old` first). Make four hills instead of two, or a valley, etc. What effect does the new shape of the world have on the system dynamics?

2. What causes the uneven wealth distribution we see in the original simulation? Is it the agent death and replacement rule R? Is it the uneven distribution of food in `sugarspace.pgm`? Is it the variance in maximum age, or metabolism or vision? Pick one or two parameters of the model with the replacement rule and see how their variance affects the wealth distribution.

3. Modify `SugarModel.java` so that it also displays a time series of agent population. How does the population vary over time, if the default parameters are used? Why?

4. Reset the initial parameters so that the initial agent population decreases, asymptotically approaching the environment's carrying capacity (you might want to change the

[9] uchicago/src/sim/sugarscape/*.java

replacement rule and maximum death age, for instance). What carrying capacity (Epstein and Axtell, 1996, see pages 30-31) values do you obtain for the original maximum values of agent *metabolism* and *vision*?

5. Do an experiment on how the final carrying capacity of the system varies with the initial population. Write down a hypothesis, then design an experiment to test that hypothesis by varying the initial population size. Draw a graph of the result, evaluate the hypothesis, and draw conclusions. (Minar)

6. Do an experiment to discover the effects of maximum metabolism and vision on agent survival.

References

J. Epstein and R. Axtell. *Growing Artificial Societies: Social Science From The Bottom Up.* MIT Press, 1996.

Nelson Minar. Implementation of SugarScape in Swarm. http://www.swarm.org/community-contrib.html.