Projet S2
L3 Magistère Economiste Statisticien

Kevin Godin-Dubois

February 9, 2022

Contents

Listings

1 Generalities

As with the previous semester's, this project is designed to test your programming skills in the context of a complex system simulation. However, in addition to the previous expectations, a special emphasis will be placed on generating,

exploiting and synthesizing the dynamics generated by your system. You will work in teams of three, to design, develop and write your experiment as outlined in the following sections. The main deliverable for this session are:

- the application itself (source code, configuration file)
- an *individual* summary (pdf)
- a 4 pages article (pdf)
- an oral presentation (pdf)

The use of english is encouraged, but by no means necessary, and, depending on the quality, will result in a bonus point (see table 1). Similarly, you can use LaTeX for your article (see the skeleton on moodle) for additional gratification.

2 Co-evolutionary model

In the same spirit as the previous project, you will work on a complex system. This time, however, we will address a somewhat contemporary problem: epidemiology. We will be using the article from Dorin (2005) as a rough reference. While it makes for an interesting reading, and another introduction to Artificial Life, we are only going to take a few ideas from this work. First, the relationship between hosts and diseases will be an emergent process instead of a single mathematical function. Second, both hosts and diseases will be agents in our system albeit with different behavioral rules. Third, diseases will have a much faster mutation rate but can only survive inside a host resulting in a delicate balance for their continued survival.

2.1 Simulation

As with the previous project, your agents will evolve in a "physical" environment. In the context of this experiment, it does not need to be very complex: a simple square area with hard boundaries. However, it is still recommended to back it with a QGraphicsScene to straightforwardly handle agent collisions detection (see code appendix 1). In pseudo-code your simulation should proceed as follow:

```
1  Initialization
2  for n timestep
3     for every host h
4         step h
5
6         for every neighbor n of h
7             h infects n ?
8             h mates with n ?
9     spontaneous infection ?
```

The initialization procedure generates random *Host* agents and, additionally, infects one of them with a random, compatible, disease. Thereafter, each agent is updated once every step of the global loop.

Lines 7-9 indicate stochastic events, that subjected to a throw from the `random` module. Indeed, while most interactions will be biased by host/disease parameters, the final decision will rest in the hands of the random number generator (RNG). For instance the infection chance depends on compatibility between the disease and its potential host but even an extremely unlikely contamination can occur.

2.2 Hosts

The most visible agents in this simulation are the hosts and contain a single intrinsic parameters (the color) as well as a few state variables (position, health, infection status). The color will be expressed as a red, green, blue triplet and stored as a QColor. For simplicity, its is recommended to use the function `QColor.fromRgbF(r,g,b)` where all arguments are in the $[0, 1]$ range. The state variables will describe the current position (QPointF) of the agent, its health (in $[0, 1]$) and whether or not it is infected. This latter can be set to `None` whenever the host is free of any disease.

You must devise an appropriate (simple) behavior for the hosts so that they move around the environment in a "plausible" manner. They need to have regular contact with other agents to propagate diseases without falling into either total immobility or Brownian motion. Additionally, two hosts in sufficiently close proximity have a probability of mating with one another thereby producing an offspring. This probability is directly proportional to the color match between both individual's colors (expressed either as a Cartesian or Manhattan distance): the lower the distance, the better the match.

Given that hosts have a single intrinsic field the crossover procedure is as illustrated by figure 1. Reproductions should be frequent enough to maintain the population even when faced with epidemics but still low enough not to saturate (i.e. reach the upper limit). It may also be a good idea to add some mutation to the newborn agent (see next section) to prevent an excessive convergence of alleles.

2.3 Disease

The second type of agents will model a simple disease with an airborne method of contamination. Unlike the hosts, diseases cannot exist on their own in the environment: they must be carried by a host to survive and propagate. They have a single state variable (time of infection) and a few intrinsic properties:

color as with hosts, this is an RGB triplet

virulence defines the infection rate and health damage

duration how long it lasts in an host

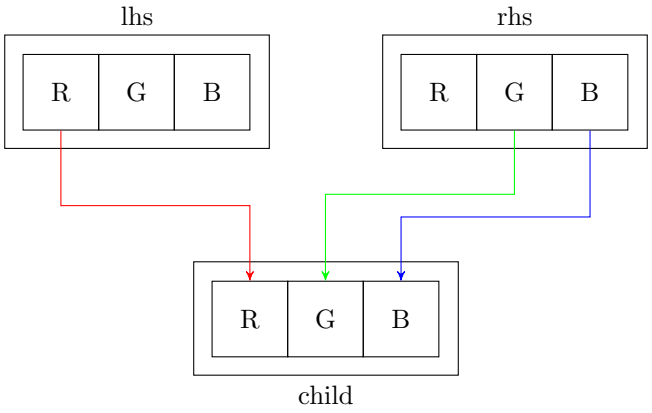

Figure 1: Example of a crossover operation. The child gets its red component from the left-hand side (lhs) parent and the green and blue from the right-hand side (rhs) parent.

mutation optional parameter impacting the mutation speed

The state variable is a timer, started at the beginning of the infection. When said timer expires the disease is considered cured and removed from the host.

Infection occurs when a diseased host (h) comes near a sane host (n). Let d be disease carried by h:

```
1  s = susceptibility(n, d) #in [0, 1]
2  if random.random() < f(s) * virulence(d):
3      infect n with d
```

where $f(s)$ transforms the susceptibility into an infection rate with desirable properties.

Finally, diseases have an extremely high mutation rate compared with their hosts. To simulate this behavior they will be mutated *every* step of the simulation. The corresponding operator is described in figure 2. Thus for each field (RGB color components, virulence, duration, mutation) you must define a lower and upper bound and determine how δ is computed (normal, uniform... distribution). If you implement the mutating mutation rates this delta is impacted by the disease's corresponding parameter (M).

3 Experimental guidelines

The objective of this project is not only to design a complex system of interaction and co-evolution between hosts and diseases but also to test various hypothesis on the dynamics. I this context, you must how best to manage your system to streamline the analytic process. This section highlights some addi-

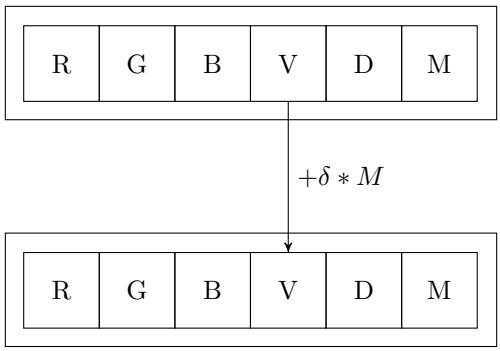

Figure 2: Example of a single mutation. First a field is selected, then a variation (of the appropriate type) is applied.

tional elements that, while not part of the epidemiology model, will simplify its subsequent use.

3.1 Configuration & Logging

As mentioned during the practical sessions, you are encouraged to design your application as much modularity as necessary. Two components are of particular importance in this context: the configuration which will hold your system-wide parameters and the logger that will keep a trace of hosts/diseases interactions. The former will initially be very similar to the `Constants` module of your Boids project with the added twist that you should be able to change its contents via a file input. In this manner you will be able to perform experiments in different conditions while still keeping track of what where the specific values of each parameters. It is recommended to pay particular attention to the RNG seed as it will have a large impact on your run and should thus be either a configuration value or a command line argument. The latter will be responsible of writing to the disk every relevant piece of information generated by your simulation for further analysis and comparison between runs. Please note that, due to the stochastic nature of this experiment, you will need to perform your analysis on a sample of statistically significant size.

3.2 Graphical rendering

Given that the objective is to collect data, you should endeavor to perform as many runs in as short a timeframe possible. As such, having an update loop ticking at a fixed frequency (e.g. 25 frames per second) is counter-productive. Nonetheless there are uses for a graphical interface if only for debugging purposes. Additionally, highlighting specific components of your model or showing the result of a particular intricate interaction are best communicated visually. The implication is that your simulation should be executable both as a plain loop (with no widgets) and with a full graphical interface.

A note of caution: in this model diseases cannot exist outside a host, meaning that drawing a disease can only occur in the `paint` method of a Host. You are free to choose any appropriate representation for both types of agents as long as: a) it is easy to see which hosts are infected and b) the color components of both types of agents are clearly visible.

4 Deliverable

As enumerated in the introduction, this project includes four deliverable. Except for the application itself, which has already been detailed previously, the specific requirements for each will be spelled out in the following sections.

4.1 Article

As a showcase of your work, you will produce a small (4 pages) article-like document that will summarize you decisions in implementing the model and synthesize the most prominent results obtained. Refer to the skeleton provided on moodle for the basic structure of this document which follows the classical academic format[1]:

Abstract a 10 lines summary of the article

Model step-by-step description of your model components with rationalization of your choices

Results important dynamics with qualitative/quantitative analysis

Conclusion summarizes findings and points out potential future improvements

Again, English is appreciated but not required, as is the use of LaTeX. You are, however, required to submit the document, on moodle, in a **pdf** format.

4.2 Presentation

Once more, the course will be concluded by a 15 minutes presentation in which you will both demonstrate your system and highlight the methodology and results of your analysis. As such it will mostly follow the same organization as your article with additional content to address the pedagogical aspects of the project. Namely this will include a reflection on the role distribution by type (code, statistics, redaction...) and coding task (hosts, diseases, gui...) as well as the temporal organization and how well you met the assignment's goals. The presentation shall be supported by slides which you will have to submit on moodle in **pdf** format.

[1] With the exception of the state of art

4.3 Individual summary

Similarly, you will produce an individual summary of this experience, detailing your personal contribution in the project. You are strongly encouraged to take into consideration both the positive and negative aspects of this endeavor, especially compared to the previous semester's introspection. This **pdf** document should be about 2 pages long and should *not* be a code review but, instead, must provide valuable insights for similar future tasks.

Annex: Notation guidelines

Task	Total	Details	Sub-tasks
Code	10		
		2	Simulation
		2	Hosts
		2.5	Diseases
		1.5	Configuration
		1	Logging
		1	GUI
Article	5		
		.5	Abstract
		1.5	Model
		1	Methodology
		1.5	Results
		.5	Conclusion
Presentation	3		
		1	Demonstration
		1	Model
		1	Results
Summary	2		
		.5	Roles
		1	Introspection
		.5	Comparison
Bonuses	2		
		1	English
		1	LaTeX

Table 1: Indicative notation guidelines for the evaluation of the project

Task	Total	Details	Sub-tasks
Simulation	2		
		.5	Initialization
		.5	Step
		1	Random infections
Hosts	2		
		.5	Intrinsic/state variables
		.75	Behavior
		.75	Mating
Disease	2.5		
		.5	Intrinsic/state variables
		.75	Infection
		.75	Mutation
		.5	Mutating mutations
Configuration	1.5		
		.5	Config module
		.5	Read
		.5	Write
Logging	1		
		.5	Dynamics file
		.5	Population snapshots
GUI	1		
		.25	Hosts
		.25	Diseases
		.5	Interface

Table 2: Indicative notation guidelines for the evaluation of the code.

Annex: Code snippets

Listing 1: Detecting neighbors of Host h

```
neighbors = [
    n for n in h.collidingItems() if isinstance(n, Host)
]
```

Listing 2: Processing command-line arguments to read/write a configuration file

```
if len(sys.argv) > 1:
    config = sys.argv[1]
    if os.path.exists(config):
        Constants.read_config(config)
    else:
        with open(config, "w") as c:
            Constants.print_config(c)
```