Disease Simulation

by

Manuel THOUILLOT, Aurore WENDLING

group SCAN – 73

May 10, 2020

Table of Contents

1. Project Context and Objectives

In the context of the SARS-CoV2 pandemic, our project seems almost cliché. However, we started it without knowing it would be so linked to the current situation. We were interested in simulating how a disease can spread between individuals, and we wanted to create a piece of code that would allow us to change some parameters and see how they would influence the outcome.

As the pandemic emerged, we found several other simulations, and they changed slightly what we aimed for with our own code. In particular, the video from the YouTube channel 3blue1brown interested us a lot (https://www.youtube.com/watch?v=gxAaO2rsdIs), and we used it as an inspiration for our simulation.

In the end, we settled for the following goals :

- simulate the spread of a disease throughout a population, which would be divided in several groups
- model and simulate the behaviour of individuals within a group
- model and simulate the interaction of the groups
- store and show data from the simulation in the form of a graph
- be able to modify parameters before and during the simulation using a graphical interface

We decided to use a very common disease model called the SIR. SIR stands for Susceptible, Infected, Recovered. That means every individual is either susceptible to the disease, infected by it or recovered (or dead, depending on the disease). This model is widely used for modeling the spread of various infectious agents, and can later be improved upon by adding other subcategories.

We decided to implement the following functionalities :

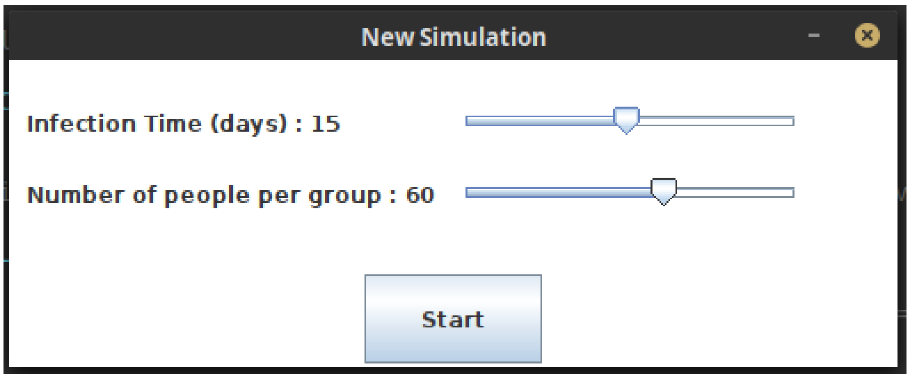
- simulate and represent the spread of the disease (susceptible individuals are shown in green, infected ones in red, and recovered ones in blue).
- individuals may transmit the disease if close enough to each other, depending on the infectiousness of the disease.
- simulate travel between groups : individuals may "teleport" to a random the center of another group, allowing the disease to spread with them.
- simulate social distancing : individuals will tend to stay away from others if asked to.
- simulate the influence of a central point within a group : individuals will tend to visit a central point. This can model shopping, social events and other gatherings.

- simulate testing and quarantining : infected individuals have a probability of being identified as ill. If so, they will be placed in a separate quarantine group without contact with the other groups, until they are not infected anymore.

2. GUI Description

The graphical user interface consists of two windows.

The first is where the user chooses the first parameters of the simulation (number of persons per group and duration of infection), which cannot be modified during the simulation.



Infection Time (days) : 15

Number of people per group : 60

Figure 1: The starting window

The second one is the main window of the simulation. We chose this solution because it seemed clearer to us, since every piece of information the user might need is immediately accessible, and it is not overwhelming since our simulation is rather simple.

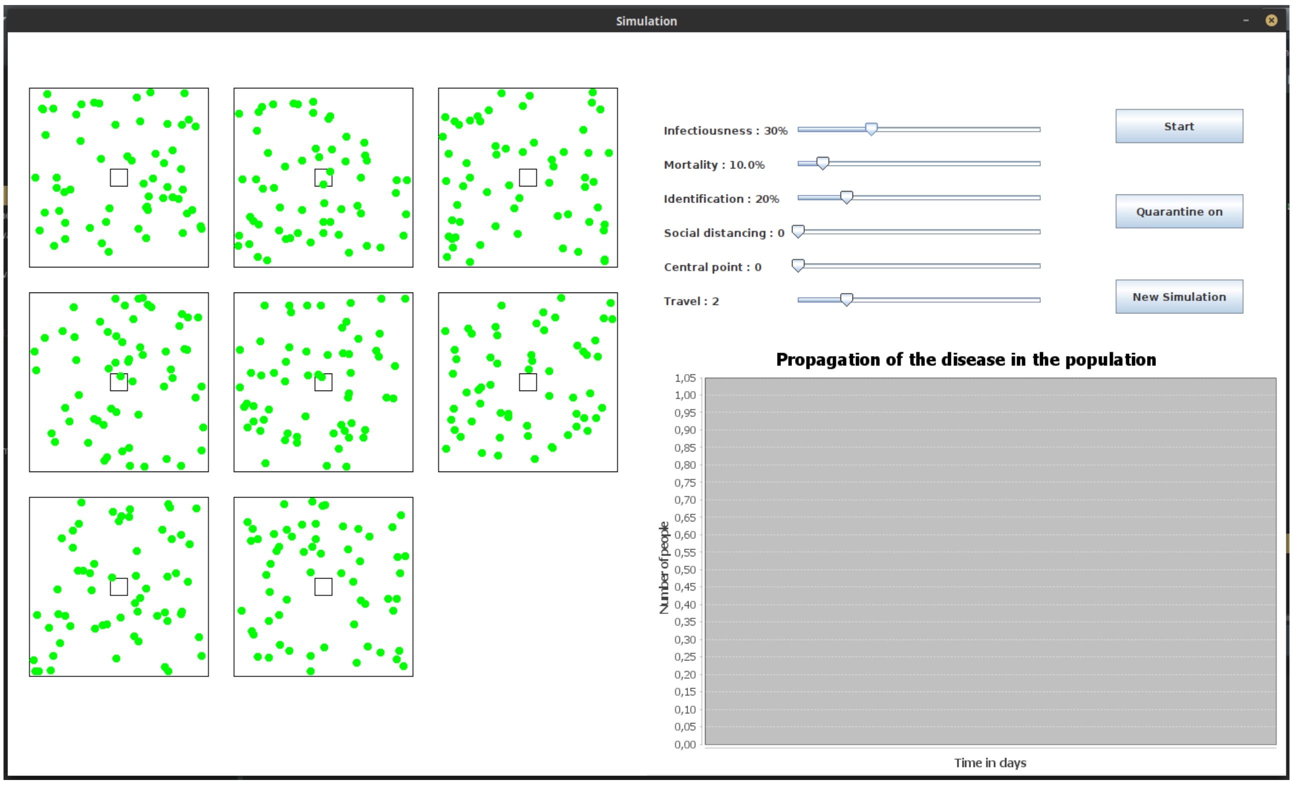
In the top right of this window, the user has the possibility of changing the following parameters of the simulation through sliders :

- infectiousness of the disease
- mortality
- proportion of infected persons which are identified at each iteration
- extent of social distancing
- whether people tend to visit a central point in each group and to which extent
- whether people travel between groups and to which extent.

In addition, with the help of three buttons, the user can start or pause the simulation, start over a new simulation, and toggle the quarantining of identified individuals.

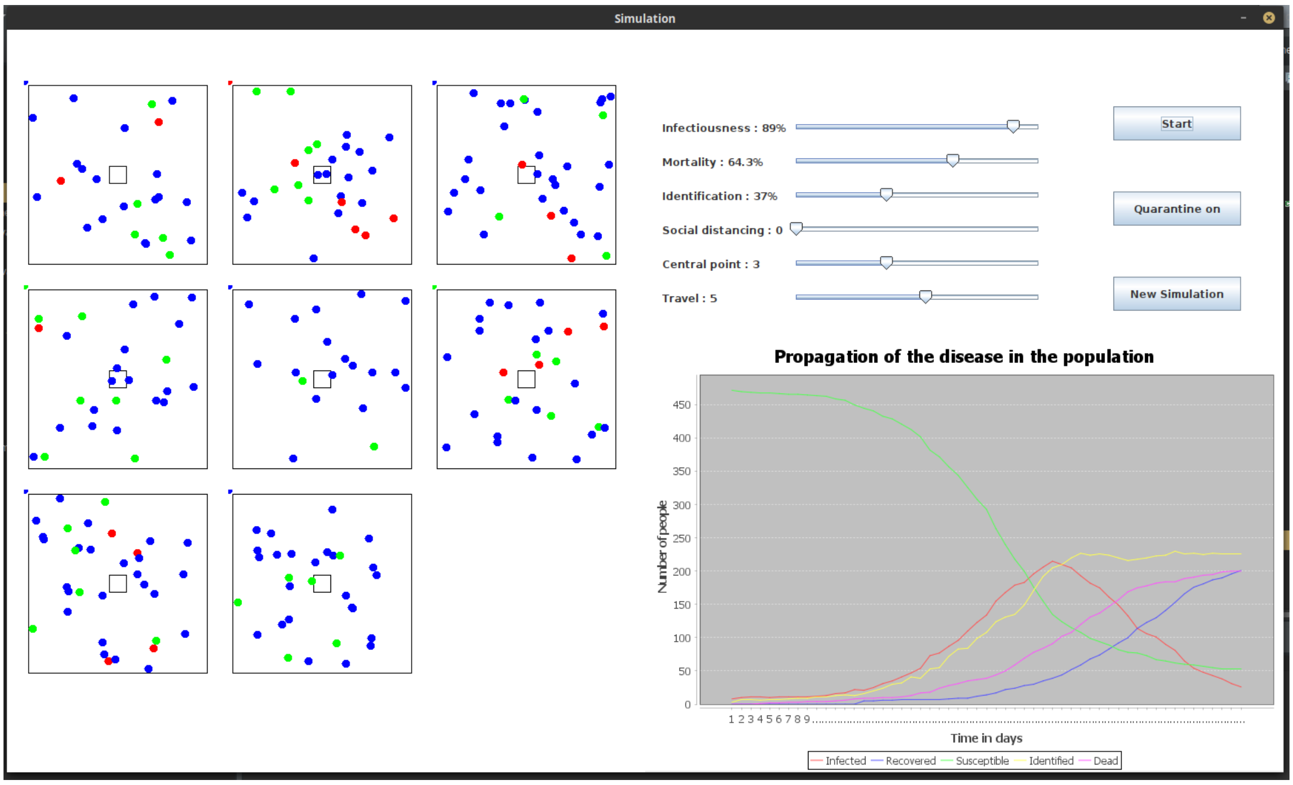
The simulation space consisting of several "boxes", where the individuals interact, is located on the left of the window. Each box possesses a central point, represented as a small square in its middle.

The graph showing the evolution of the situation is located on the right. The curves show the number of susceptible (green), infected (red), identified (yellow), recovered (blue) and deceased (pink) individuals which are present each day. In order to display this graph, we used the JFreeChart Library.



Infectiousness : 30%
Mortality : 10.0%
Identification : 20%
Social distancing : 0
Central point : 0
Travel : 2

Figure 2: Main simulation window before starting the simulation



Infectiousness : 89%
Mortality : 64.3%
Identification : 37%
Social distancing : 0
Central point : 3
Travel : 5

Figure 3: Main simulation window while the simulation is running

In addition, we also wanted to have the possibility of running a simulation with only one group. In this case, only one box would be shown in the space allocated to the simulation, with the addition of the quarantine box if asked for by the user. The simulation had slight issues when implementing quarantine in that context : some individuals were wandering outside of the box. Due to lack of time, we did not try to solve this issue and we deleted this functionality as we thought it did not add much value to our software. We still kept the class, in case we would like to use it later.

3. Algorithm Description

3.1. Basic functioning of the model

We chose to model the spread of the disease empirically, by simulating interactions between individuals moving in a 2D space.

In order to do this, we created an object called “Individual”, to which we attributed :

- A position and a velocity :

  These attributes allow us to represent and displace individuals in the plane. They are modeled as vectors (we created a specific object “Vector” representing a vector possessing x and y components, a norm and an angle).

- A state :

  Each individual is either susceptible, infected, or recovered. If it is infected, it can be identified as such. Moreover, for each individual, we store the time that has passed since the moment it was infected.

In the display, each individual is represented by a moving coloured dot, depending on its state : green if he is susceptible, red if it is infected and blue if it is recovered.

We then created an object “Group” which represents an group of individuals of a given size, which are wandering within the same box. At each iteration, every individual in the group moves and the individuals can infect each other if. An individual has a certain probability of infecting every individual that is close enough to him.

Moreover, at each iteration, a certain proportion of the infected individuals die : they are simply removed from the group.

Regarding the GUI, each Group inherits from JPanel and is visible as a square box in which individuals are wandering.

3.2. Implementing additional functionalities

Starting from this very simple model, we added a number of functionalities in order to observe the effect of the behaviour of the individuals on the spread of the disease.

3.2.1. Social distancing

Given the situation we are currently facing in our daily lives, we obviously wanted to observe the effect of social distancing on the spread of the disease.

In order to model social distancing, we added a repulsive force between individuals. This force is modeled as vector. Each individual exerts a repulsive force on all the other individuals located close enough to him. This repulsive force is inversely proportional to the distance between the two individuals that interact, raised to the power of four. This way, individuals repel much more strongly individuals that are very close to them and much less individuals that are not too close.

For each individual, we sum the contributions of the repulsive force exerted on him by all the other individuals to get the total force it is submitted to. As good mechanics students, we then model the effect of this force on the velocity of the individuals according to Newton’s second law :

Let $\vec{F}_{others}$ the total repulsive force and $\vec{v}$ the velocity of the individual, we write :

$$\vec{F}_{others}=\frac{d\vec{v}}{dt}$$

$$d\vec{v}=\vec{F}_{others}.dt$$

$$\vec{v}'=\vec{v}+d\vec{v}=\vec{v}+\vec{F}_{others}.dt$$

Once the velocity of every individual is updated to take into account the repulsive force exerted by others on him, we move the individuals according to their velocity.

3.2.2. Travel between groups

We then wanted to be able to model the displacement of individuals from one group to another. This was done rather simply : every individual has a given probability of leaving the group it is in to go to a different group. If an individual changes group, its is removed from its departure group and added to its arrival group, where it is placed in the center of the plane. We do not show the displacement of individuals from one group to another, hence the individuals basically “teleport” themselves in the center of another group (or more realistically, they travel by plane to the airport of the arrival group, which is located in its center).

3.2.3. Quarantine

The next functionality we wanted to implement was taking the individuals that are identified as infected to a quarantine zone. In order to do this, we chose to represent the quarantine zone as a Group.

However, we wanted to make sure that an individual leaving the quarantine would go back to the group it initially came from, and not go to any random group. To ensure this, we finally decided to create a distinct quarantine zone for each group and to display them one above the other. This way, we could just transfer identified individuals from their group to its associated quarantined zone and back.

This was rather easy to implement after the previous functionality : since we basically displaced again individuals from one group to another (the quarantine) it was very similar to making the individuals travel between groups.

3.2.4. Central Point

As mentioned earlier, we wanted to see the effect of having the individuals of one group regularly go to a common central point (this could represent going to the market, to school, or any other gathering). Again, because of the current situation, we were particularly interested in seeing the efficiency of social distancing measures while still regularly meeting other people in a school or a store.

The central point of each group is modeled as a small Group, placed in its center. This way, we just need to transfer individuals to this group, and back after a certain time. The individuals have a given probability of going to the central point of their group at each iteration. Again, the individuals were “teleported” to the central point. However, they left the central point normally, by moving away from it in a randomly chosen direction.

4. Class Diagram

Our program is composed of the following classes :

- Disease :

  This class is used to store a number of informations related to the disease and its propagation, which can then be used in all the other classes. The methods it contains are only getters and setters, to modify or retrieve these parameters.

- Vector :

  Each instance of this class represents a vector in the plane, it is defined by its cartesian and polar coordinates. We use it to model the position and velocity of the individuals, as well as the forces exerted on them.

- Individual :

  Each instance of this class represents an individual in the simulation. It possesses a state (susceptible, infected or recovered) as well as a position and a velocity, it can be identified if it is infected.

- Group :

  This class extends JPanel. Each instance of Group represents a collection of individuals that interact together, in the same space. It is represented on a JPanel as square containing moving dots, each of which represents one individual.

- SimulationPanel :

  This class extends JPanel. It is used to display the simulation in the main window. We do not use it directly, since we created to different child classes from this class, in order to display several groups or only one group in the main window.

  This class contains a number of methods to store data to plot the graph, to implement travel between groups, quarantine, travel to central point, some of which are empty and must be overridden by the children classes.

- MultipleGroupsPanel :

  This class extends SimulationPanel. It represents several groups of people and contains specific methods to handle interaction between different groups.

- OneGroupPanel :

  This class extends SimulationPanel. It represents one group (and a quarantine zone) and contains specific methods to handle only a single group. We finally did not use it (as explained previously) because the quarantine did not function properly.

- Simulation :

  This class extends JFrame and implements ActionListener and ChangeListener. It represents the main window of our program, which contains a SimulationPanel, a ChartPanel (from the JFreeChart library) and a panel containing sliders and button to interact with the user.

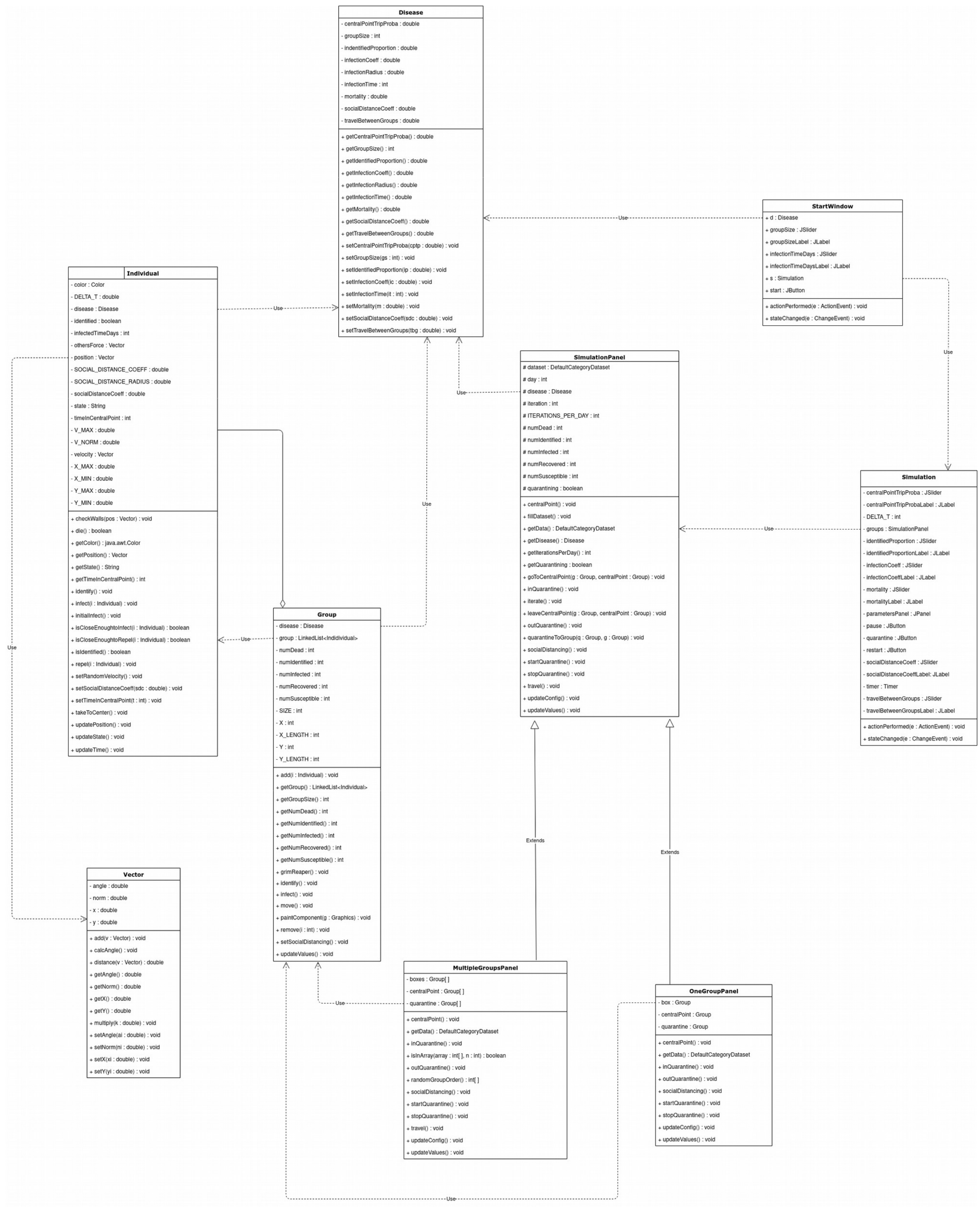
- StartWindow :

  This is the first window to be displayed when we start the program. It allows to set the size of the groups and the duration of the infection, which cannot be modified during the simulation (we thought it was not realistic to be able to modify these parameters while the simulation was running), and to start the simulation.

- Main class :

  The main class of our program simply creates and instance of StartWindow.

The UML diagram, being quite large, it is difficult to read it directly here, so we added it in our .zip file as an image.


Disease
- centralPointTripProba : double
- groupSize : int
- indentifiedProportion : double
- infectionCoeff : double
- infectionRadius : double
- infectionTime : int
- mortality : double
- socialDistanceCoeff : double
- travelBetweenGroups : double
+ getCentralPointTripProba() : double
+ getGroupSize() : int
+ getIdentifiedProportion() : double
+ getInfectionCoeff() : double
+ getInfectionRadius() : double
+ getInfectionTime() : double
+ getMortality() : double
+ getSocialDistanceCoeff() : double
+ getTravelBetweenGroups() : double
+ setCentralPointTripProba(cptp : double) : void
+ setGroupSize(gs : int) : void
+ setIdentifiedProportion(ip : double) : void
+ setInfectionCoeff(ic : double) : void
+ setInfectionTime(it : int) : void
+ setMortality(m : double) : void
+ setSocialDistanceCoeff(sdc : double) : void
+ setTravelBetweenGroups(tbg : double) : void
StartWindow
+ d : Disease
+ groupSize : JSlider
+ groupSizeLabel : JLabel
+ infectionTimeDays : JSlider
+ infectionTimeDaysLabel : JLabel
+ s : Simulation
+ start : JButton
+ actionPerformed(e : ActionEvent) : void
+ stateChanged(e : ChangeEvent) : void
Individual
- color : Color
- DELTA_T : double
- disease : Disease
- identified : boolean
- infectedTimeDays : int
- othersForce : Vector
- position : Vector
- SOCIAL_DISTANCE_COEFF : double
- SOCIAL_DISTANCE_RADIUS : double
- socialDistanceCoeff : double
- state : String
- timeInCentralPoint : int
- V_MAX : double
- V_NORM : double
- velocity : Vector
- X_MAX : double
- X_MIN : double
- Y_MAX : double
- Y_MIN : double
+ checkWalls(pos : Vector) : void
+ die() : boolean
+ getColor() : java.awt.Color
+ getPosition() : Vector
+ getState() : String
+ getTimeInCentralPoint() : int
+ identify() : void
+ infect(i : Individual) : void
+ initialInfect() : void
+ isCloseEnoughtoInfect(i : Individual) : boolean
+ isCloseEnoughtoRepel(i : Individual) : boolean
+ isIdentified() : boolean
+ repel(i : Individual) : void
+ setRandomVelocity() : void
+ setSocialDistanceCoeff(sdc : double) : void
+ setTimeInCentralPoint(t : int) : void
+ takeToCenter() : void
+ updatePosition() : void
+ updateState() : void
+ updateTime() : void
SimulationPanel
# dataset : DefaultCategoryDataset
# day : int
# disease : Disease
# iteration : int
# ITERATIONS_PER_DAY : int
# numDead : int
# numIdentified : int
# numInfected : int
# numRecovered : int
# numSusceptible : int
# quarantining : boolean
+ centralPoint() : void
+ fillDataset() : void
+ getData() : DefaultCategoryDataset
+ getDisease() : Disease
+ getIterationsPerDay() : int
+ getQuarantining : boolean
+ goToCentralPoint(g : Group, centralPoint : Group) : void
+ inQuarantine() : void
+ iterate() : void
+ leaveCentralPoint(g : Group, centralPoint : Group) : void
+ outQuarantine() : void
+ quarantineToGroup(q : Group, g : Group) : void
+ socialDistancing() : void
+ startQuarantine() : void
+ stopQuarantine() : void
+ travel() : void
+ updateConfig() : void
+ updateValues() : void
Simulation
- centralPointTripProba : JSlider
- centralPointTripProbaLabel : JLabel
- DELTA_T : int
- groups : SimulationPanel
- identifiedProportion : JSlider
- identifiedProportionLabel : JLabel
- infectionCoeff : JSlider
- infectionCoeffLabel : JLabel
- mortality : JSlider
- mortalityLabel : JLabel
- parametersPanel : JPanel
- pause : JButton
- quarantine : JButton
- restart : JButton
- socialDistanceCoeff : JSlider
- socialDistanceCoeffLabel: JLabel
- timer : Timer
- travelBetweenGroups : JSlider
- travelBetweenGroupsLabel : JLabel
+ actionPerformed(e : ActionEvent) : void
+ stateChanged(e : ChangeEvent) : void
Group
- disease : Disease
- group : LinkedList<Indidividual>
- numDead : int
- numIdentified : int
- numInfected : int
- numRecovered : int
- numSusceptible : int
- SIZE : int
- X : int
- X_LENGTH : int
- Y : int
- Y_LENGTH : int
+ add(i : Individual) : void
+ getGroup() : LinkedList<Individual>
+ getGroupSize() : int
+ getNumDead() : int
+ getNumIdentified() : int
+ getNumInfected() : int
+ getNumRecovered() : int
+ getNumSusceptible() : int
+ grimReaper() : void
+ identify() : void
+ infect() : void
+ move() : void
+ paintComponent(g : Graphics) : void
+ remove(i : int) : void
+ setSocialDistancing() : void
+ updateValues() : void
Vector
- angle : double
- norm : double
- x : double
- y : double
+ add(v : Vector) : void
+ calcAngle() : void
+ distance(v : Vector) : double
+ getAngle() : double
+ getNorm() : double
+ getX() : double
+ getY() : double
+ multiply(k : double) : void
+ setAngle(ai : double) : void
+ setNorm(ni : double) : void
+ setX(xi : double) : void
+ setY(yi : double) : void
MultipleGroupsPanel
- boxes : Group[ ]
- centralPoint : Group[ ]
- quarantine : Group[ ]
+ centralPoint() : void
+ getData() : DefaultCategoryDataset
+ inQuarantine() : void
+ isInArray(array : int[ ], n : int) : boolean
+ outQuarantine() : void
+ randomGroupOrder() : int[ ]
+ socialDistancing() : void
+ startQuarantine() : void
+ stopQuarantine() : void
+ travel() : void
+ updateConfig() : void
+ updateValues() : void
OneGroupPanel
- box : Group
- centralPoint : Group
- quarantine : Group
+ centralPoint() : void
+ getData() : DefaultCategoryDataset
+ inQuarantine() : void
+ outQuarantine() : void
+ socialDistancing() : void
+ startQuarantine() : void
+ stopQuarantine() : void
+ updateConfig() : void
+ updateValues() : void
Use
Use
Use
Use
Use
Use
Use
Use
Use
Use
Extends
Extends

5. Results and Discussions

In the end, we managed to implement all the functionalities we wanted : travel between groups, quarantine, going to a central point and social distancing, and the simulation works fine.

We were able to observe several interesting phenomena and our simulation confirmed some basic assumptions about disease spread : social distancing is effective at stopping an epidemic in its tracks, but even a few non conforming individuals can significantly increase the spread. Going regularly to a central point decreases also a lot the efficiency of social distancing. Moreover, identifying and quarantining a sizable part of the infected individuals proves to also be tremendously helpful in this regard. Travel can obviously spread a disease between multiple groups. This effect is more pronounced when associated with a tendency of people to meet at a central point.

However, we did not succeed in displaying properly a single group with a quarantine as we initially planned to. However, we did not put much effort in solving this issue since we felt it did add much interest to the program. Sometimes, we observe individuals getting stuck in the top left corner of the group, and we have not managed to figure out why. Here again, little effort was put into solving this issue since it was minor and we lacked time.

We could improve our program in several ways.

We thought of displaying the movement of the individuals from one group to the other instead of “teleporting” them, but given the fact that we had created each Group as a separate panel, it was not so easy to do.

We would also have liked to implement lots other factors because it is very interesting to observe their influence. We thought of differed identification (people would be tested only after a given time, in order to represent the fact that they are not immediately symptomatic), differences in how people respond to the disease (depending on age, for instance), asymptomatic people…

Finally, our graphical user interface is very basic, and not pleasing to the eye. Indeed, we focused our efforts on the model and not on the aesthetic aspect of our software. This could have been addressed if we had had more time.

6. Project Timeline and Members' Contributions

After doing a bit of research on the SIR model, we managed to decide what exactly we wanted to do. It is later, after coming across several other disease spread simulation softwares that we decided to add a compartment aspect to our code, which aimed at representing the fact that in real life, people mostly interact within a given group, and that interactions between groups are less common than intra-group contacts. This resulted in the aforementioned boxes appearing in our simulation.

We started by splitting up the first tasks we thought of. Aurore was tasked with making a first version of the simulation code, while Manuel started coding the first graphical interface. We worked together using GitHub, which was rather handy as long as we were careful to regularly update the files.

For the GUI, Manuel tried using a library called JFreeChart but soon ran into a number of problems. These issue discouraged us to continue the program for some time, so we spent a long time not working on it at all. Then, this led us to try to make graphs without using any library. However, the results were not satisfactory so in the end we went back to using JFreeChart. This allowed us to create clearer graphs without the hassle of having to code absolutely everything. It was also an occasion for us to discover how to use external libraries. It also led Aurore to use the Eclipse IDE instead of Geany, which was a great discovery since it possesses a lot more handy functionalities.

Using this library also helped us realize how important it was to have a clear and complete documentation, so we created a Javadoc for our code.

Since we were only two in the group, each of us had quite a bit of work to do, and since we had totally abandoned the project for some time after facing issues with JFreeChart, we had to work in a rush to finish the program. This was really painful and ended up with us taking more time than normal to finish the project and being very stressed during the last few days. If such a project were to be done again, we would try to organize our work better along time.

Because of this final rush, Aurore demonstrated her ability to tackle several complex tasks at once, without getting overwhelmed. She developed singlehandedly most of the simulation code, and of the final GUI. This is why we believe her work deserves more credit than normal. Manuel finally focused writing the report and the class diagram and on doing some research about the components and libraries we wished to use.

Bibliography

Our main inspiration for this program was a video from the YouTube channel 3blue1brown about the SIR model : https://www.youtube.com/watch?v=gxAaO2rsdIs

We also used a lot the Java documentation : https://docs.oracle.com/en/java/

And the JFreeChart documentation : http://www.jfree.org/jfreechart/api/javadoc/index.html

It is when realizing how important this documentation was to understand and reuse code that we decided to create a Javadoc for our program.