CSE 443 Object Oriented Analysis and Design
Final Project Report

Ahmed Semih Özmekik
171044039

January 18, 2021

Abstract

Detailed explanation of design choices along with the experimental results in the homework.

1 Design Decisions

1.1 MVC Pattern

We did the whole project hierarchy by following the general design pattern of the modal-view-controller, since our program is a program with a graphical interface.

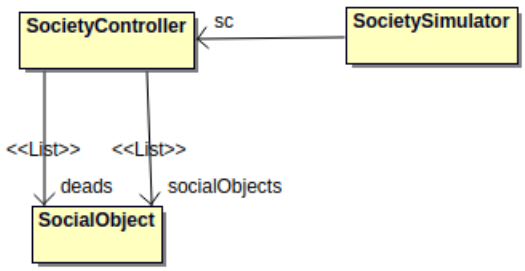

Figure 1

We did the whole project hierarchy by following the general design pattern of modal-view-controller, since our program is a program with a graphical interface.

Our 3 main classes are like this. **SocialObject** represents the model, **SocietySimulator** View represents and **SocietyController** represents Controller.

1.2 State Pattern

1.2.1 First Apply

The most important, significant factor of the **Individual** object was that it had more than one state. That is, an object may be infected, healthy, in a collision state, etc. Anticipating

that there will be many states from the beginning, we decided to use the state design pattern here.

The advantage of the state design pattern is that we have the ability to write different behaviors in different states separately, allowing classes to be loosely coupled and eliminating the confusion in a single code block.

So when a new state arrives, we no longer need to make changes to the core of the code, instead, each state defines its own action. This action is in 3 different forms: **update**, **checkCollision** and **paint**. In other words, while an object has different stateles, it can perform these 3 different behaviors in different ways. As a matter of fact, an infected object and a healthy object do not collide in the same way, or an object in a hospital and a healthy object are not drawn on the screen in the same way.

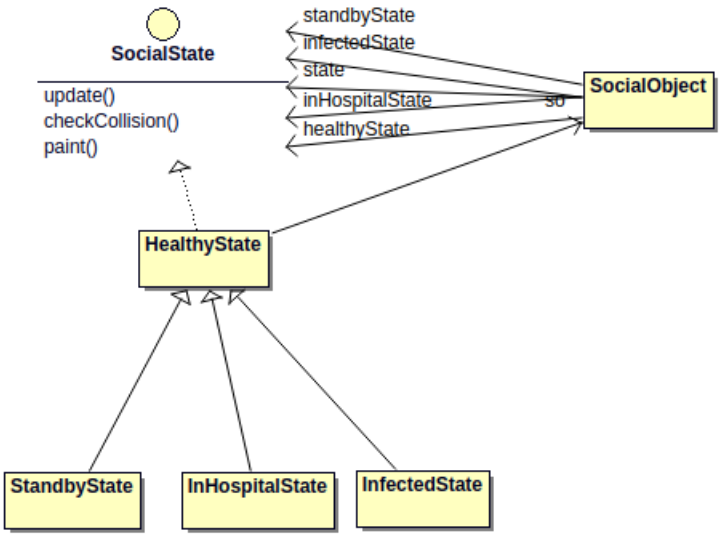

Figure 2

In the class diagram in the figure, you will see the application of the state design pattern to the problem we encounter in our current program.

Of course, the Medaitor class are interested in **checkCollision**, but using them with states has also been beneficial.

HealthyStatus acts as a base status, implementing some convenient methods, and other statuses derive from it to take advantage of them.

1.2.2 Second Apply

We also applied a state design pattern inside the standby state. Because the waiting state also had different states. For example, an infected and a healthy do not expect the same. While an infected is waiting, it can infect a healthy person.

At the time of the collision, either a healthy may have hit an infected or an infected may have hit a healthy. This situation could actually be solved by dividing the existing

StandbyState into 3 different states within our general state design pattern approach. But instead, we wanted to move this problem to **StandbyState** to expand on more flexible and standby states.

So we implemented an additional State design pattern there, too.

1.3 Mediator Pattern

It was necessary for us to use the mediator design pattern in two ways, both social objects represent a model class, so the interaction between them should be handled in a discrete class (i.e. Controller), and we need a successful design when the number of interactions between them is predicted to increase in the future.

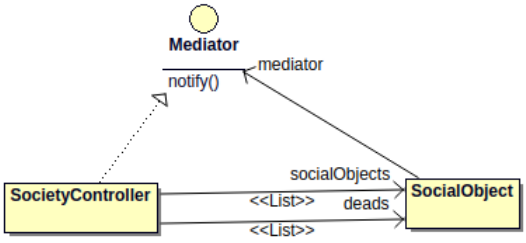

Figure 3

As you can see, the **SocialController** class is the mediator class that handles the interaction between objects. Objects have the function **checkCollision** and they only notify the mediator when there is a collision. Thus, whoever notified, mediator takes care of this collision situation according to their stats.

There is only one interaction for now. However, in the future, as the number of interactions between social objects increases, the mediator design pattern will bring us the advantage of reducing the maintenance cost we want.

1.4 Observer Pattern

We had to plot it in our simulation. What plots have in common is that the x-axes of the two are fed exactly the same values, namely time values. So the x-axis of the plots is the same. What has changed is the X axis. So we needed a generic plot structure that feeds on the same time variable.

And the Plot object had to be aware of when the program state has changed. So, it was supposed to be an observer of this change. This brings to mind exactly the Observer Design Pattern.

The class we define with the name of Plotter is an observer, it draws the plot from the **SocietyController** by taking the dead and infected numbers only when they are completely changed. In this way, we have provided an opportunity for plots that are both efficient and may be needed in the future, fed by different data.

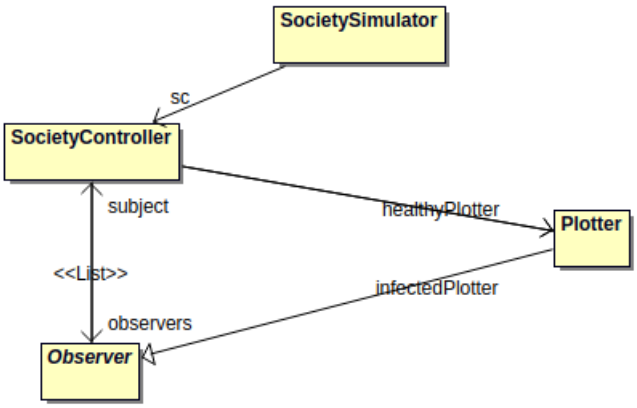

Figure 4

2 Demonstration of the Operations

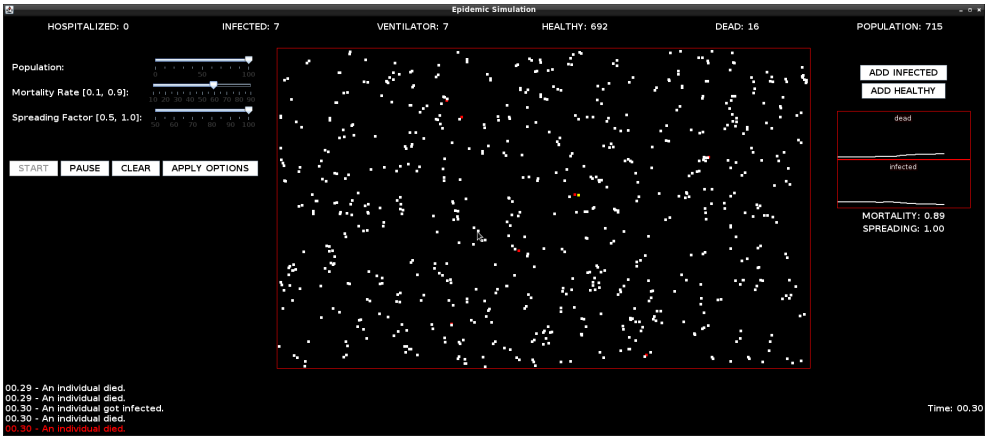

Figure 5

The user can take actions on the interface, such as pausing whenever he/she wants, adding only one healthy or infected individual, or adding individual in bulk, or changing some simulation constants in the run time in the same way.

In addition, on the right side, there is a time dependent graph of the desired dead and infected numbers. Using this section in detail in the next section, some inferences will be made.

At the bottom left, there is a log section that informs the user about some specific events that took place in the simulation. It notifies the user of time-dependent events.

On the left, at the point where we received input from the user, we made sure that we received safe input from the user with a slider. And the options do not apply instantly, they become valid if the apply button below is pressed. Finally, besides the play and pause

button, there is a clear button, which simply clears the whole canvas.

3 Plots and Comments

3.1 Mortality Rate

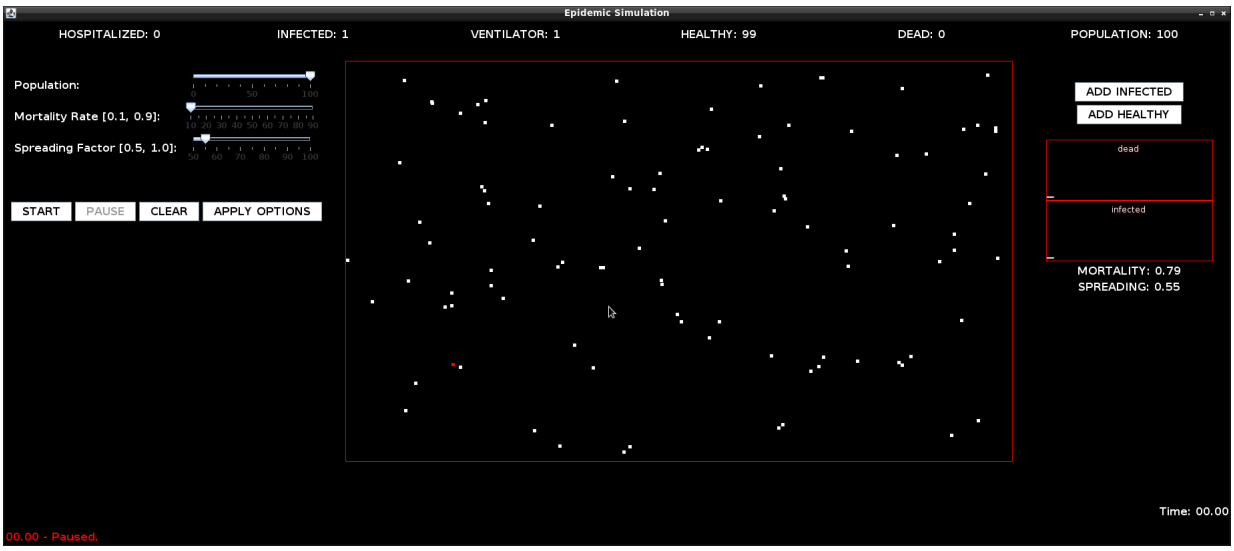

Figure 6

We begin first, calmly, with the lowest mortality rate in a given population. In this part, we added an infected and then we pull the mortality rate to the top, and what we will observe is that all the infected objects die exponentially.

That is, if the mortality rate increases, the spread of the epidemic decreases. If the mortality rate decreases, likewise, the epidemic spreads faster.

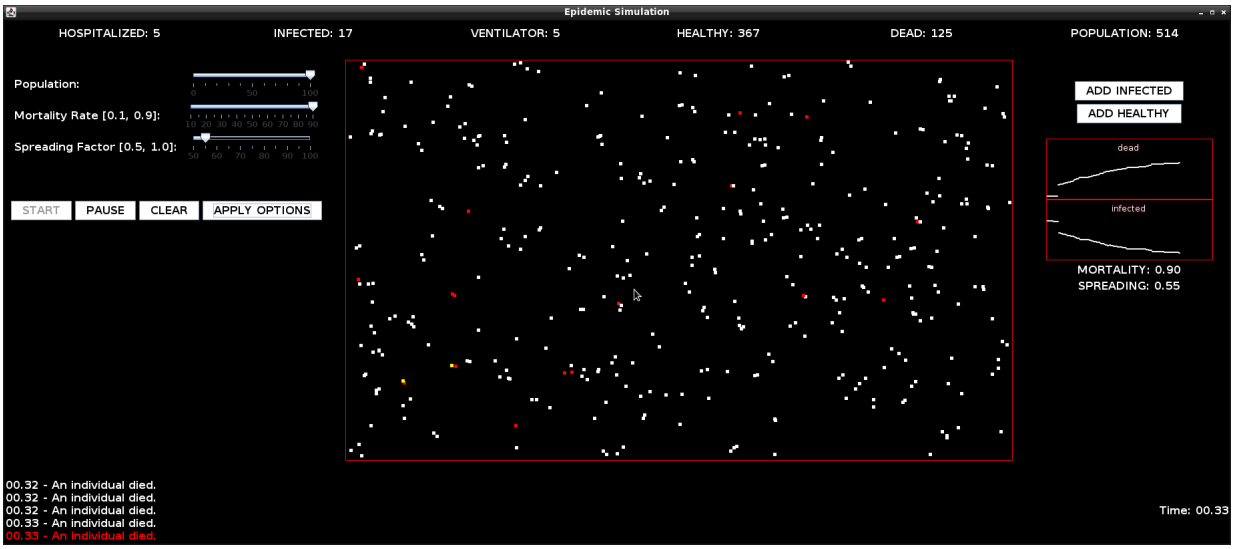

Figure 7

3.2 Spreading Factor

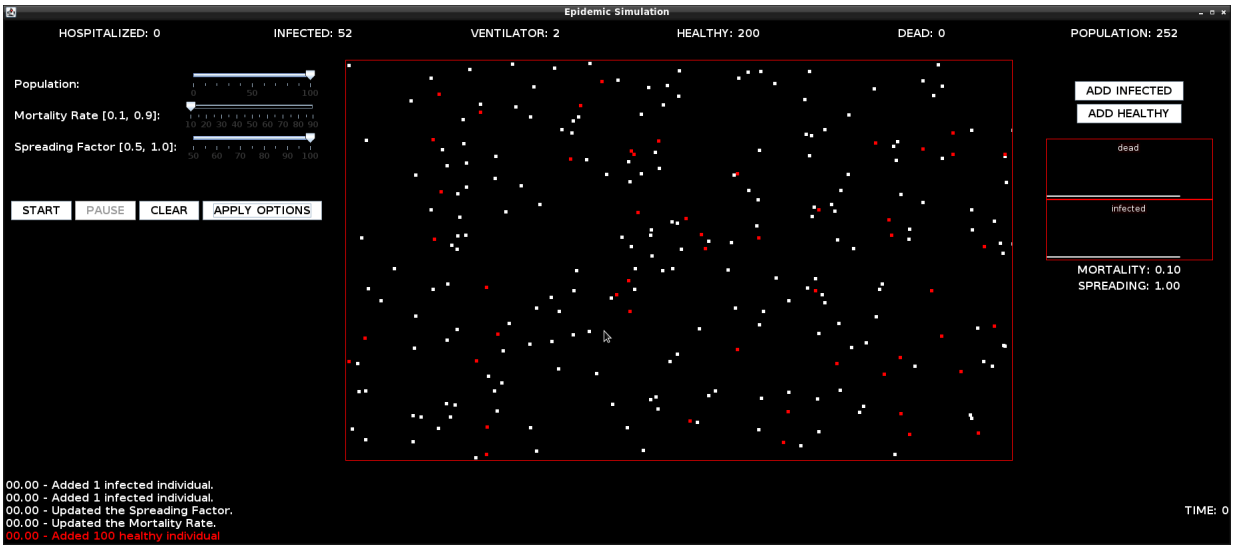

Figure 8

To keep the other variable constant and deal with the spreading factor, we first pull the spreading factor up, then lower the mortality rate so that the infected does not die immediately.

This time, we see that the epidemic is spreading very quickly, exponentially.

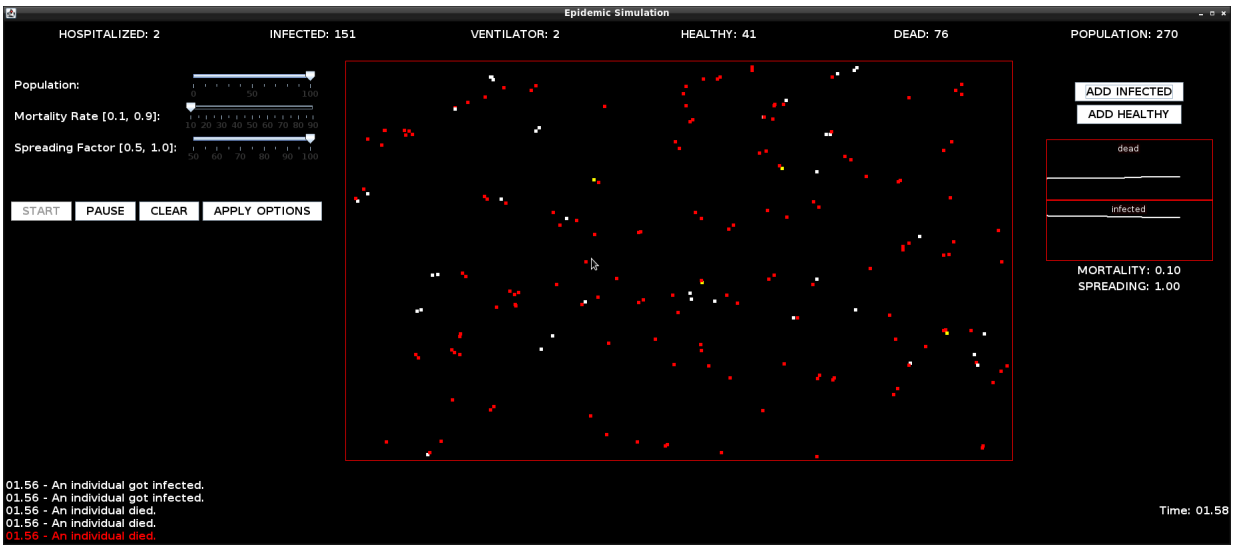

Figure 9

3.3 Sociability

Interestingly, the decrease in sociability increases the spread of the virus, although we are not sure whether this could be counted as a fault of the simulation, the increase in sociability means that any two individuals spend less time, hence the infected ones collide

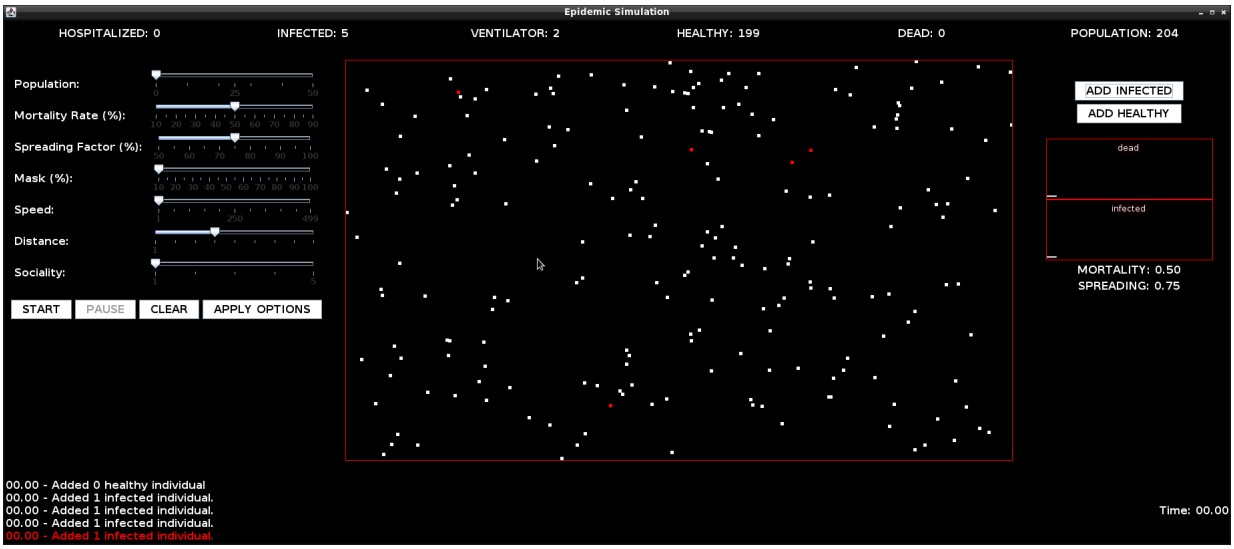

Figure 10

with more people in x time (since the post-collision standby time is reduced) and therefore, more people end up being infected.

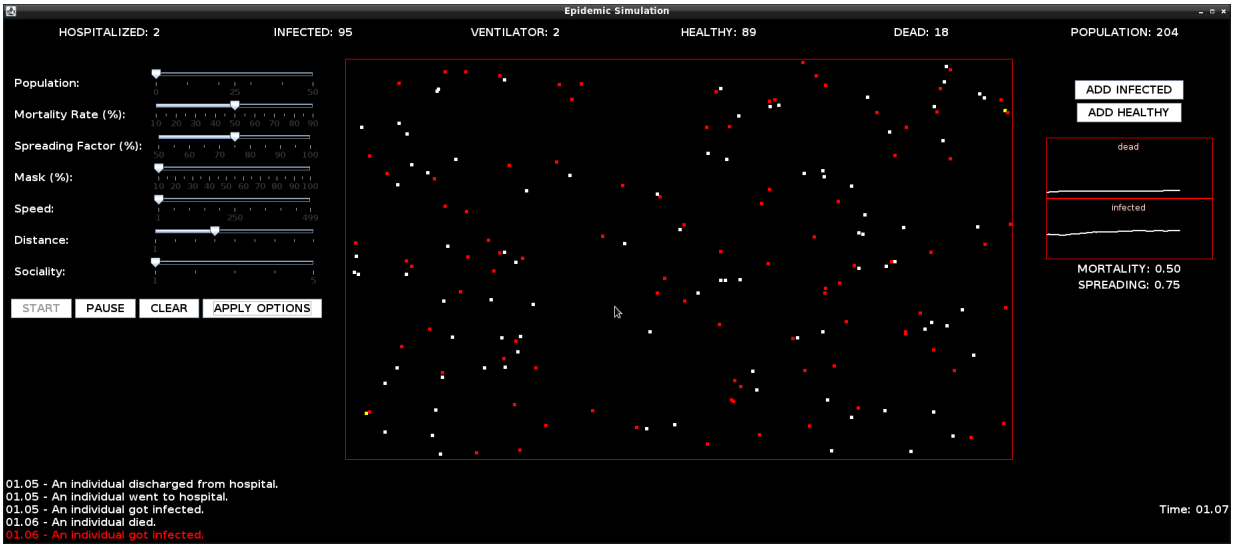

Figure 11

3.4 Distance

The reduction of distance naturally directly increases the spread of the disease.

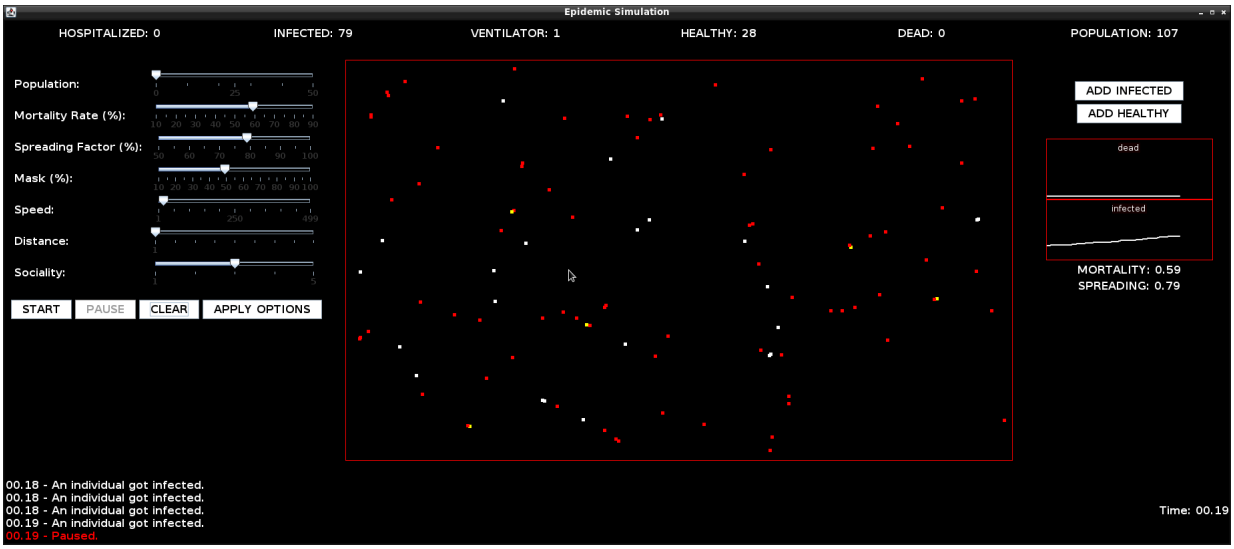

Figure 12

3.5 Mask

When the mask is on , almost no one gets sick.

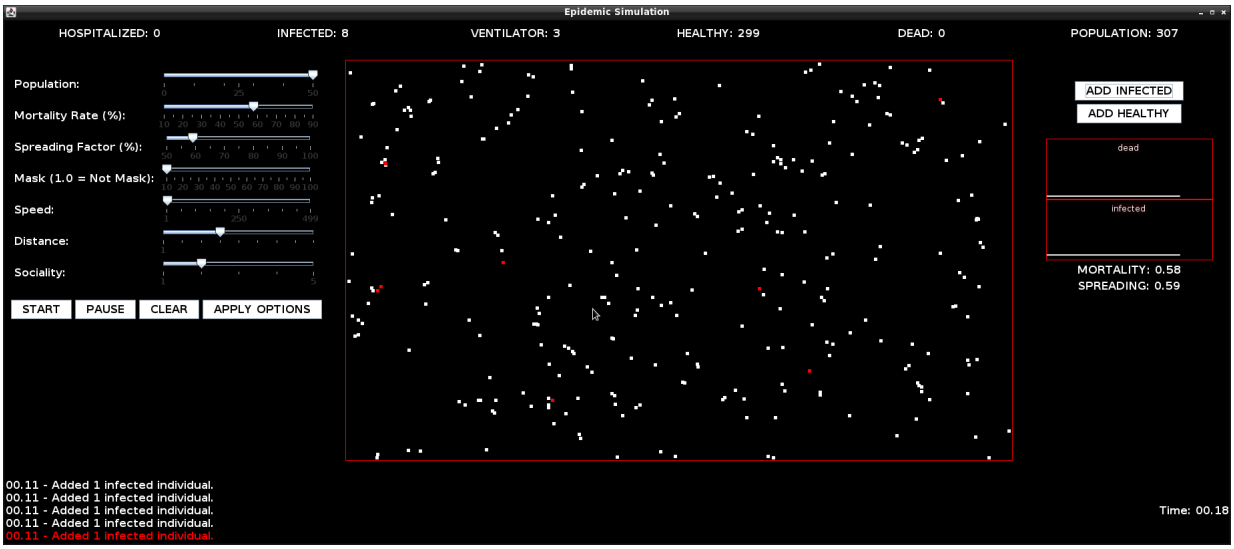

Figure 13