

Agent-based models of spread of infectious diseases using Python

SIR and SEIR Model

Patrick Vincent N. Lubenia

13 May 2020

Contents

Agent-Based Models	2
Empty class	2
SIR Model	3
Overview	3
The Code	4
Needed Libraries	4
Modeling Task 1a: Attributes of the agents	4
Modeling Task 2c: Behavior of agents	5
Modeling Task 2d: Interaction of agents	5
Updating States	6
Counting	6
Visualization	6
Implementation	8
Day 0	8
Scenario 1	9
Scenario 2	11
SEIR Model	13
Conclusion	17
Appendix A: SIR Model Full Code	18
Appendix B: SEIR Model Full Code	23
References	28

Agent-Based Models

Agent-based modeling is a framework for creating and simulating models of complex systems. It is a mindset wherein a system is described from the point of view of the units constituting it [1]. Agent-based models are computational simulation models that involve numerous discrete agents. They show a system's emergent collective behavior resulting from the interactions of the agents. In contrast to equation-based models, each agent's behaviors in an agent-based model are described in an algorithmic fashion by rules rather than equations. Agents in the model do not typically perform actions together at constant time-steps [2]. Their decisions follow discrete-event cues or a series of interactions.

Depending on one's objectives, agents: are discrete entities, may have internal states, may be spatially localized, may perceive and interact with the environment, may behave based on predefined rules, may be able to learn and adapt, and may interact with other agents. Generally, agent-based models: often lack central supervisors/controllers and may produce nontrivial "collective behavior" as a whole.

The following scientific method-based approach must be kept in mind when designing an agent-based model:

1. Specific problem to be solved by the model
2. Availability of data
3. Method of model validation.

In order to be scientifically meaningful, an agent-based model must be:

1. Built using empirically-derived assumptions, then simulate to produce emergent behavior: for predictions; or
2. Built using hypothetical assumptions, then simulate to reproduce observed behavior: for explanations.

Once a code has been programmed, its basic implementation structure has 3 parts:

1. Initialization
2. Updating
3. Visualization.

Agents are initially placed in the model's environment. The system is then updated according to rules that govern the behavior of the environment and/or agents. Finally, states are visualized in order to appreciate the changes in the system.

The agent-based modeling framework is open-ended and flexible. It may be tempting to be detailed to make a model more realistic. But it must be remembered that increased complexity leads to increased difficulty in analysis. Moreover, the open-endedness of the framework makes it code-intense as lots of details of the simulation must be manually taken care of. Thus, codes must be kept simple and organized.

Empty class

Sometimes, a `class` is created when objects are needed to be flexible enough to be given desired attributes. In

```
1 class performer:  
2     pass
```

the `pass` on line 2 allows one to create an empty class. This class can now be used to create a performer:

```
1 performer_ = performer()
```

It is simple to add attributes to the object called performer, say its location in terms of coordinates, name, and age:

```
1 # Assign attribute x to performer_  
2 performer_.x = 3  
3  
4 # Assign attribute y to performer_  
5 performer_.y = 4  
6  
7 # Assign attribute name to performer_  
8 performer_.name = 'Luca'  
9  
10 # Assign attribute age to performer_  
11 performer_.age = 16
```

This use of `class` is utilized in the model below.

SIR Model

Overview

The Susceptible-Infectious-Recovered (SIR) Model is a compartmental model (members of the population are classified into distinct compartments) in epidemiology. This modeling framework stems from Kermack and McKendrick [3]. During an epidemic, people in a population are classified as

1. Susceptible: Capable of contracting the disease
2. Infectious: Has the disease and is able to transmit it to susceptible people
3. Recovered: Had the disease and no longer has it

In the basic model, a person transitions linearly from being susceptible, to being infectious, and finally to being recovered. A person belonging to particular compartment cannot go back to any previous state. Furthermore, once a person recovers, he is already immune to getting the disease. Complicated models allow movements between compartments, even allowing going back to a previous state.

The SIR Model tracks the number of members in each compartment as time goes by. Starting with a given number of susceptible people, this number declines as they interact with infectious individuals, which, in turn, decrease as people recover. This simple transition between states are modeled using ordinary differential equations.

Let t be time (in days), S the number of susceptible people, I the number of infectious people, R the number of recovered people, N the total number of people in the population, β the transmission rate, and γ the recovery rate. The set of ordinary differential equations describing the change in the compartments over time is given by:

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta I}{N}S \\ \frac{dI}{dt} &= \frac{\beta I}{N}S - \gamma I \\ \frac{dR}{dt} &= \gamma I.\end{aligned}$$

These equations assume random mixing of individuals in the population, regardless of their state. Hence, infectious people are still able to mingle with everyone else. Solving these equations simultaneously gives the following result:

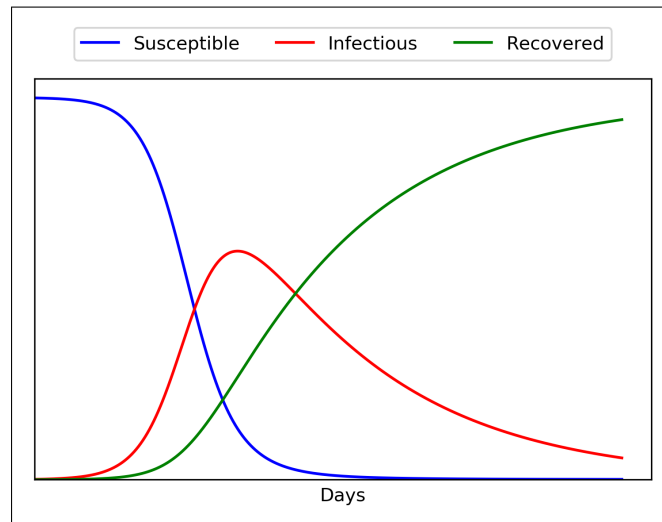


Figure 1: SIR Model. Solution to SIR Model.

Figure 1 shows that at the start of an epidemic, the number of infectious individuals exponentially increase as there are numerous susceptibles people. As infectious people recover, eventually, infectiousness peak, then start to decline. The purpose of the model that follows is to show that agent-based models can simulate the same trend in spread of infectious diseases in a “natural” way without using equations.

This model follows an agent-based modeling framework with the following tasks that must be undertaken [4]:

1. Design the data structure to store the:
 - (a) Attributes of the agents
 - (b) States of the environment
2. Describe the rules for how:
 - (a) The environment behaves on its own
 - (b) Agents interact with the environment
 - (c) Agents behave on their own
 - (d) Agents interact with each other.

Not all these tasks are needed in every agent-based model.

The Code

Needed Libraries

To start the code, the following libraries are imported:

```
1 # For assigning agents to random locations and for choosing random agents
2 import random as rd
3
4 # For visualizing the state of the agents using plot
5 import matplotlib.pyplot as plt
6
7 # For checking if a filename already exists, to avoid overwriting files
8 import os
```

Modeling Task 1a: Attributes of the agents

The first task is to “design the data structure to store the attributes of the agents”. The agents in this model are people with different states (compartments) located in different places. Thus, each agent needs the following attributes:

- Spatial location
- State: Susceptible, Infectious, or Recovered
- Number of days the person has had the disease
- Is the person immune already?

The class called `agent` is initialized:

```
1 class agent:
2     pass
```

A function called `create_agents` is defined to create the population:

```
1 def create_agents():
2
3     # Allow the following variable to be accessed outside the function
4     global agents_list
5
6     # Initialize list of agents
7     agents_list = []
8
9     # Create specified number of agents
10    for i in range(n_agents):
11
12        # Create an agent
13        agent_ = agent()
14
15        # Assign the agent to a random location
16        agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
17
18        # Set the agent to be initially susceptible
19        # 0 = susceptible, 1 = infectious, 2 = recovered
20        agent_.state = 0
21
```

```

22     # Count number of days the agent is infectious
23     agent_.days_infectious = 0
24
25     # Add the agent to the list
26     agents_list.append(agent_)

```

`global` allows variables to be accessed outside the function. It is automatically created when the function is run. Note that this function needs 1 variable to be predefined: `n_agents` (total number of people in the population).

Note: In this model, there is nothing special about the environment where the people are moving around so Modeling Task 1b (“design the data structure to store the states of the environment”), 2a (“describe the rules for how the environment behaves on its own”), and 2b (“describe the rules for how agents interact with the environment”) are skipped.

Modeling Task 2c: Behavior of agents

The next task is to “describe the rules for how agents behave on their own”. This model explores 2 ways the people behave:

1. They go about their usual ways whether they are sick or not
2. Infectious people do not move around (they sort of limit their social contact by not moving around but others are free to engage them).

Scenario 1: In the scenario where people act as if there is no epidemic, the function `move` is very simple to simulate moving around.

```

1 def move():
2
3     # Agent moves to a random location
4     for agent_ in agents_list:
5         agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)

```

Scenario 2: In the scenario where social contact is somewhat limited, the function can be modified easily. Infectious people are excluded from being assigned random coordinates.

```

1 def move():
2
3     # Infectious people do not move
4     for agent_ in agents_list:
5         if agent_.state != 1:
6             agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)

```

Modeling Task 2d: Interaction of agents

The final task is to “describe the rules for how agents interact with each other”. In this model, an infectious person infects non-immune people within a specified radius.

The function `infect` simultaneously infects people within the radius of an infectious in different locations:

```

1 def infect():
2
3     # Allow the following variable to be accessed outside the function
4     global infectious_list
5
6     # Create list of infectious people
7     infectious_list = [agent_ for agent_ in agents_list if agent_.state == 1]
8
9     # Create list of neighbors of infectious
10    for agent_ in infectious_list:
11
12        # Create list of neighbors within the radius specified
13        neighbors = [neighbor for neighbor in agents_list if (neighbor.x - agent_.x)**2 + (neighbor.y - agent_.y)**2 <= infectious_radius**2]
14
15        # Remove person himself from list of neighbors
16        neighbors.remove(agent_)
17
18        # Susceptible neighbors become infectious (recovered individuals are immune)
19        for agent_ in neighbors:
20            if agent_.state == 0:
21                agent_.state = 1

```

Note that this function needs 1 variable to be predefined: `infectious_radius` (radius of neighborhood where people inside can become infectious).

Updating States

Since infectious people eventually recover after a certain number of days, the duration of their infectiousness must be monitored. This updating occurs at the start of each day.

```
1 def monitor():
2
3     # Add count for number of days an infectious person has been infectious
4     for agent_ in agents_list:
5         if agent_.state == 1:
6             agent_.days_infectious += 1
7
8     # Infectious people who reach the end of infectious period recover
9     if agent_.days_infectious > days_to_recover:
10         agent_.state = 2
```

Note that this function needs 1 variable to be predefined: `days_to_recover` (number of days a person stays infectious before recovering).

Counting

To determine the progression of the disease, the number of susceptible, infectious, and recovered people must be counted.

```
1 def count():
2
3     # Allow the following variables to be accessed outside the function
4     global S_count, I_count, R_count
5
6     # Count susceptible
7     S_count = len([agent_ for agent_ in agents_list if agent_.state == 0])
8
9     # Count infectious
10    I_count = len([agent_ for agent_ in agents_list if agent_.state == 1])
11
12    # Count recovered
13    R_count = len([agent_ for agent_ in agents_list if agent_.state == 2])
```

Visualization

2 functions are used for visualization:

1. One to visualize the states of the system
2. Another to graph the trend of the various compartments of the model.

To visualize the states of the system, the function `visualize` is created:

```
1 def visualize():
2
3     # To ensure updated count of compartments
4     count()
5
6     # Initialize subplots
7     fig, ax = plt.subplots()
8
9     # Assign color per compartment
10    agent_color = {0: 'b', 1: 'r', 2: 'g'}
11
12    # Plot each person with his corresponding color
13    for agent_ in agents_list:
14        ax.plot(agent_.x, agent_.y, '.', color = agent_color[agent_.state])
15
16    # Dummy plots for legend
17    ax.plot([], [], 'bo', label = 'Susceptible: ' + str(S_count))
18    ax.plot([], [], 'ro', label = 'Infectious: ' + str(I_count))
19    ax.plot([], [], 'go', label = 'Recovered: ' + str(R_count))
20
```

```

21 # Place legend below the figure
22 ax.legend(ncol = 3, loc = 'upper center', bbox_to_anchor = (0.5, -0.01))
23
24 # Fix the window: choose largest range for random location choices
25 ax.set_xlim([0, 1])
26 ax.set_ylim([0, 1])
27
28 # Title
29 ax.set_title('Day: ' + str(day_count), loc = 'left')
30
31 # Remove extra tick marks on the axes
32 ax.set_xticks([])
33 ax.set_yticks([])
34
35 # Prepare format of file name
36 filename = 'SIR'
37
38 # Starting filename count
39 i = 1
40
41 # Check if filename already exists; add 1 if it does
42 while os.path.exists('{f:d}.png'.format(filename, i)):
43     i += 1
44
45 # Save figure
46 plt.savefig('{f:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)

```

Subplots are used instead of the simpler `plt.plot` so that when the file is run, successive runs of plots do not overlap into one figure. Viewing window should be the as large as possible to include the largest range for random numbers generated for the coordinates. The number of iterations in the title represents day count. The tight specification in `savefig` removes extra white spaces around the figure. Note that this function needs 1 variable to be predefined: `day_count` (number of iterations done).

To show the trend of the various compartments, the function `trend` is created:

```

1 def trend():
2
3     # To ensure updated count of compartments
4     count()
5
6     # Initialize subplots
7     fig, ax = plt.subplots()
8
9     # Plot daily count of each compartment
10    ax.plot(days, S_list, color = 'blue', label = 'Susceptible')
11    ax.plot(days, I_list, color = 'red', label = 'Infectious')
12    ax.plot(days, R_list, color = 'green', label = 'Recovered')
13
14    # Place legend on top of the figure
15    ax.legend(ncol = 3, loc = 'upper center', bbox_to_anchor = (0.5, 1.15))
16
17    # Set the axes to meet at (0, 0)
18    ax.set_xlim(left = 0)
19    ax.set_ylim(bottom = 0)
20
21    # Horizontal axis label
22    ax.set_xlabel('Days')
23
24    # Prepare format of file name
25    filename = 'SIR_Trend'
26
27    # Starting filename count
28    i = 1
29
30    # Check if filename already exists; add 1 if it does
31    while os.path.exists('{f:d}.png'.format(filename, i)):
32        i += 1
33
34    # Save figure
35    plt.savefig('{f:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)

```

The idea is similar to the function `visualize`, except that the daily figures for each compartment are graphed instead, and the legend is placed on top of the figure. Note that this function needs 4 variable to be predefined: `days` (day counts), `S_list` (daily number of susceptible), `I_list` (daily number of infectious), and `R_list` (daily number of recovered).

Implementation

The simulation follows the basic code implementation structure mentioned in the introduction: agents are created (Initialization), then they are allowed to move according to their defined behavior (Updating). Plots are used to visualize (Visualization) what happens to the system before and after the agents move.

Day 0

To follow the SIR model framework, the initial number of susceptible, infectious, and recovered individuals are specified. The total gives the number of agent in the agent-based model. In this model, there are initially 1,000 susceptible, 1 infectious, and 0 recovered. The infectious radius is 0.025, and it takes 7 days for an infectious to recover:

```
1 # Comment all visualize() for faster simulation
2 # But replace with count()
3
4 ## Initial day
5
6 # Start day count
7 day_count = 0
8
9 # Initial values
10 S_initial = 1000
11 I_initial = 1
12 R_initial = 0
13
14 # Initial total population
15 n_agents = S_initial + I_initial + R_initial
16
17 # Infectious radius
18 infectious_radius = 0.025
19
20 # Number of days of infectiousness
21 days_to_recover = 7
22
23 # Initialize list of days
24 days = []
25 days.append(0)
26
27 # Initialize daily count of susceptible
28 S_list = []
29 S_list.append(S_initial)
30
31 # Initialize daily count of infectious
32 I_list = []
33 I_list.append(I_initial)
34
35 # Initialize daily count of recovered
36 R_list = []
37 R_list.append(R_initial)
38
39 # Create the population
40 create_agents()
41
42 # Choose a specified number of random people and make them infectious
43 for agent_ in rd.sample(agents_list, I_initial):
44     agent_.state = 1
45
46 # From the remaining susceptible, choose a specified number of random people and make them recovered
47 for agent_ in rd.sample([agent_ for agent_ in agents_list if agent_.state != 1], R_initial):
48     agent_.state = 2
49
50 visualize()
51 # count()
```

Since created agents are initially susceptible, initial number of infectious people are incorporated. `sample` is used to ensure chosen people are not repeated. From the remaining susceptible people, random individuals are initially assigned as recovered.

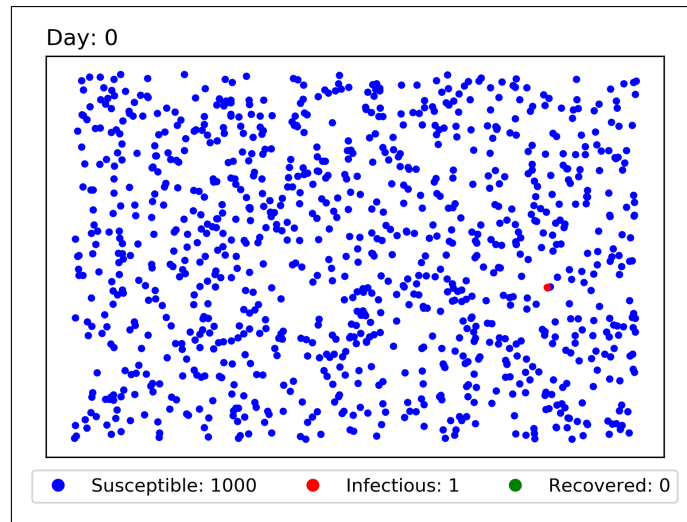


Figure 2: Initial State of the Population. 1,000 susceptible, 1 infectious, and 0 recovered.

As shown in Figure 2, at the end of Day 0, 1 infectious individual appeared. Assume that he only starts infecting other people the following day.

Scenario 1

Consider the first scenario where people move normally as if there is no epidemic. The simulation code goes:

```

1 ## Succeeding days
2
3 # Number of runs
4 runs = 100
5
6 for i in range(runs):
7
8     # Add day count
9     day_count += 1
10
11     # Monitor infectiousness and recovery at the start of the day
12     monitor()
13
14     # Let everyone move
15     move()
16
17     # Visualize new positions
18     visualize()
19     # count()
20
21     # Infections start after the movement
22     infect()
23
24     # Visualize
25     visualize()
26     # count()
27
28     # Update lists
29     days.append(day_count)
30     S_list.append(S_count)
31     I_list.append(I_count)
32     R_list.append(R_count)
33
34     # Stop simulation once number of infectious becomes 0
35     if I_count == 0:
36         break
37
38 # Plot trend
39 trend()

```

The maximum number of runs (corresponding to day counts) is indicated for the simulation. As mentioned in the section Updating States, monitoring occurs at the start of the day

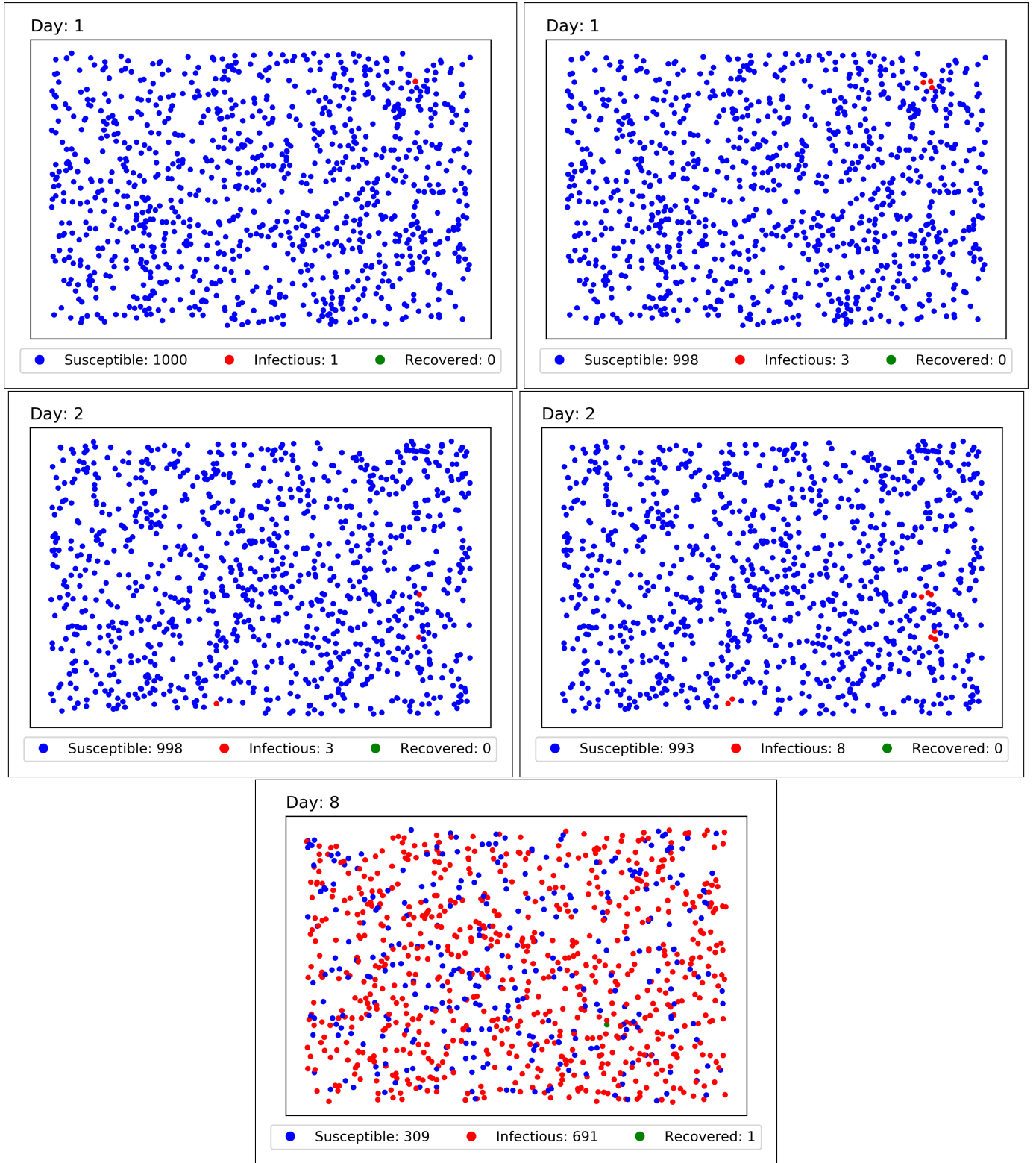


Figure 3: Scenario 1, Some States of the Population. Once infected, an infectious individual starts infecting the following day. The first recovery happens at the start of Day 8, after 7 days of infectiousness.

The top left panel of Figure 3 shows the start of Day 1 after the population moved. The upper right panel shows 2 additional infections at the end of the day. In the middle left panel, the start of Day 2, the 3 infectious individuals scattered around the population. By the end of Day 2, the middle right panel, there were already a total of 8 infectious people. The last panel illustrates the appearance of the recovery of the first infectious person, which happened after 7 days, the recovery time.

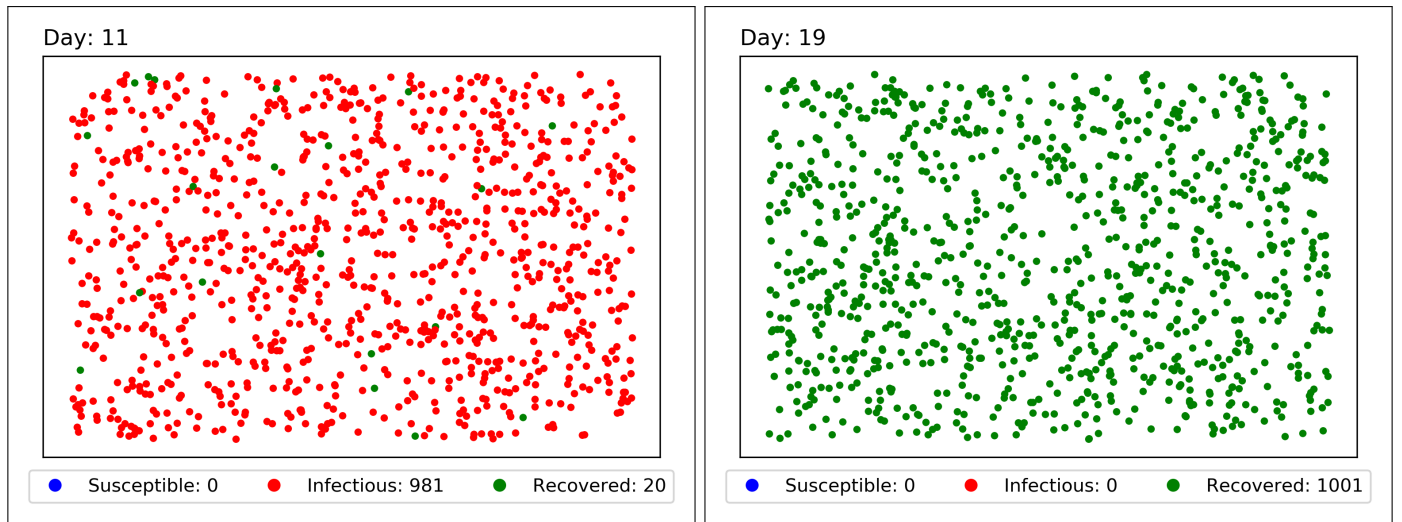


Figure 4: Scenario 1, Some States of the Population. By the end of Day 11, there are no more susceptible individuals. By Day 19, everyone has already recovered.

Figure 4 shows that by the end of Day 11, everyone was already either infectious or recovered. The epidemic has reached its peak, and the infectious are just waiting for recover. By Day 19, everyone had already recovered.

To better appreciate what happened to the population, the following graph is generated:

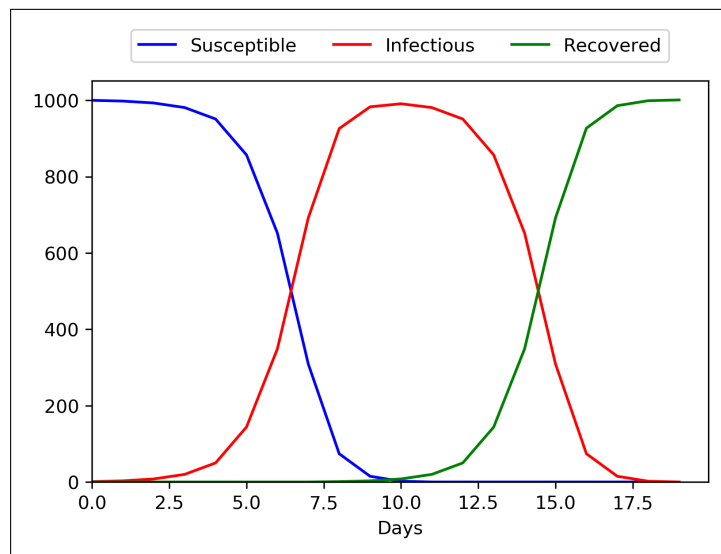


Figure 5: Scenario 1, SIR Model. Daily trend of the number of susceptible, infectious, and recovered individuals.

In Figure 5, as expected in any spread of disease, the number of infectious individuals initially increased exponentially, reached a peak, then declined. The trend of the 3 compartments are similar to the trend shown when the solution to the SIR system of ordinary differential equations in the Overview were graphed.

Scenario 2

Consider now the second scenario where infectious people do not move while they are infectious. Use the alternative `move` function defined earlier.

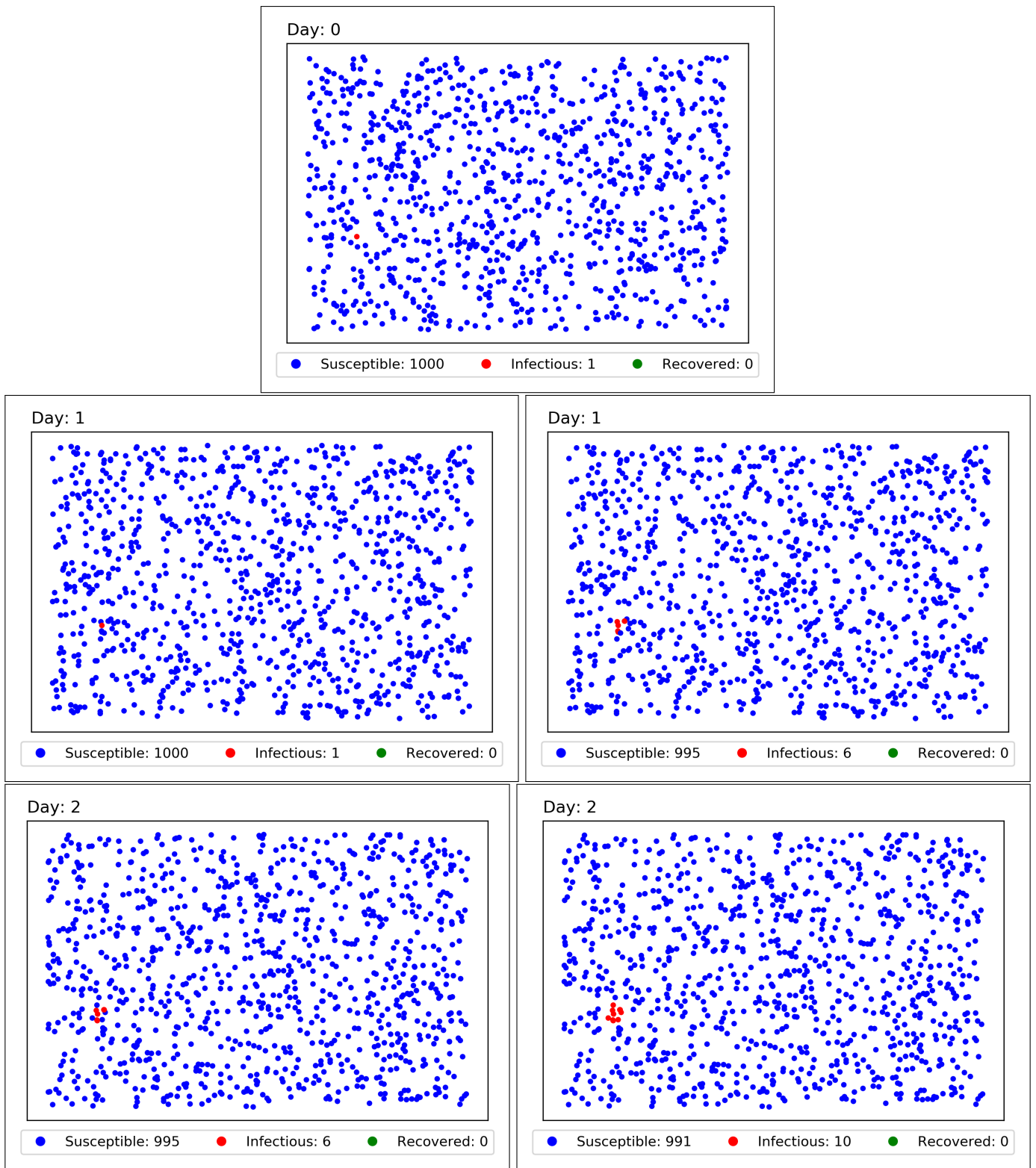


Figure 6: Scenario 2, Some States of the Population. Infectious individuals do not move.

The top panel in Figure 6 shows the initial state of the population. Notice that at the start of Day 1 (middle left panel), the infectious individual has not moved. The rest of the panels show that the infection is largely contained in his area (until other people approach him).

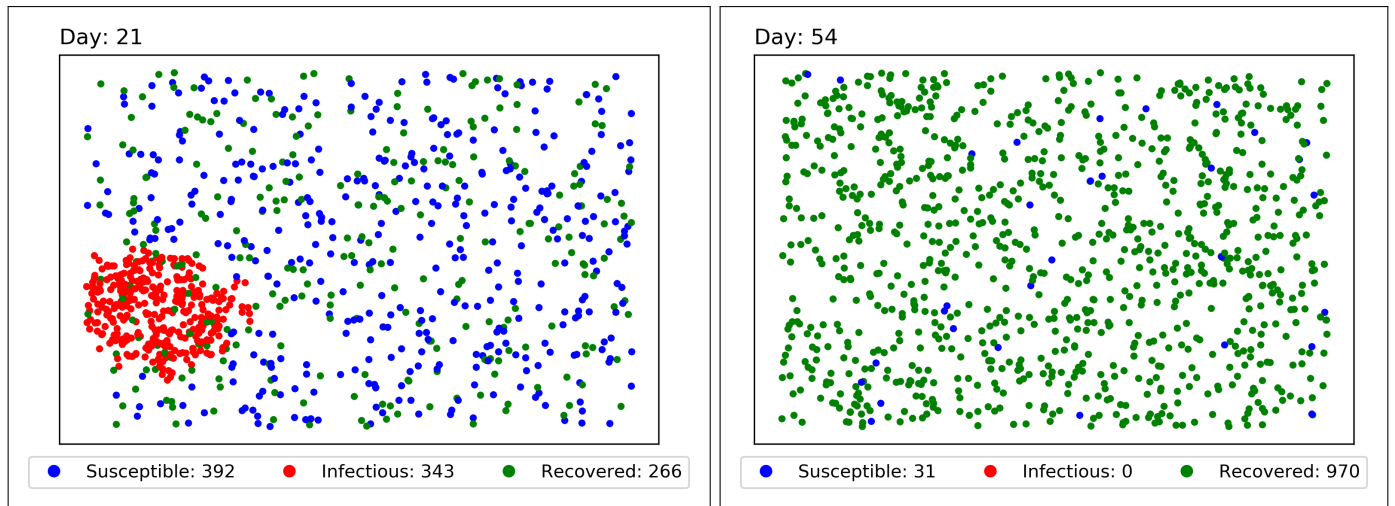


Figure 7: Scenario 2, Some States of the Population. By Day 21, the epidemic is at its peak. And by Day 54, the epidemic has stopped.

In Figure 7, the left diagram shows the peak of the epidemic at Day 21. The epidemic was contained in the neighborhood of the first infected individual. The number of infectious people increased as susceptible individuals came into contact with them: there were no limitations to the movement of non-infectious people. Compared to the first scenario, by the end of the epidemic, there were still some people who were never infected in scenario 2.

The improvement in the containment of the epidemic is evident in the trend graph:

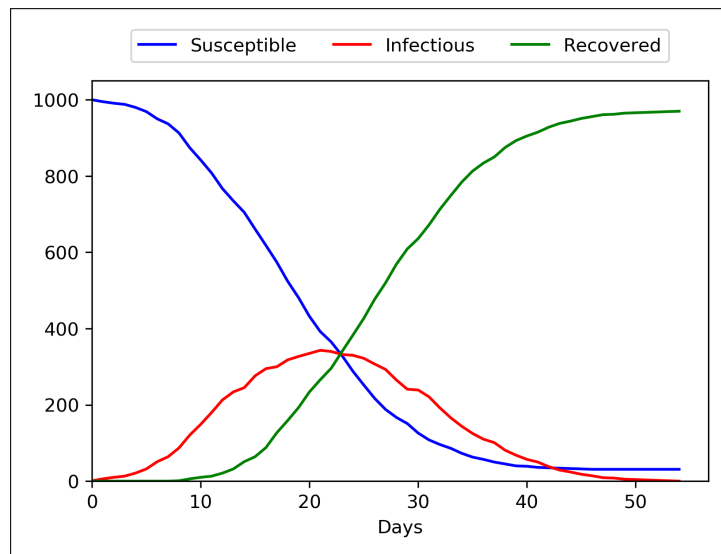


Figure 8: Scenario 2, SIR Model. Daily trend of the number of susceptible, infectious, and recovered individuals.

Figure 8 shows a clear flattening of the curve. Compared to scenario 1, even if the duration of the epidemic was longer in scenario 2, the number of individuals declined. In scenario 1, at the peak of the epidemic, almost everyone was infectious. Having a lot of infectious people at a time can overwhelm the health care capacity of a community. On the other hand, scenario 2 gave a more manageable scenario wherein about only 35% of the population was being taken care of at the peak of the epidemic. Again, the trend of the 3 compartments are similar to the trend shown when the solution to the SIR system of ordinary differential equations in the Overview was generated.

SEIR Model

A modification to the SIR model is the inclusion of an Exposed stage where a person already has the virus causing the disease, but is not yet infectious. Exposed individuals cannot transmit the disease. The time it takes before an infected individual becomes infectious is called the incubation period.

It is easy to incorporate the incubation period in the agent-based model. The following are the modifications for each function.

create_agents: An attribute `days_exposed` is added to count the number of days a person has been exposed. Note that the attribute `state` does not change but an additional state is allowed: 3 for exposed.

```
1 def create_agents():
2     ...
3     agent_.days_exposed = 0
```

move: For the second scenario, both infectious and exposed people are not allowed to move. Originally, only infectious individuals were not permitted.

```
1 def move():
2     for agent_ in agents_list:
3         if agent_.state != 1 and agent_.state != 3:
4             agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
```

infect: After creating a list of all neighbors, susceptible neighbors become exposed. Originally, susceptible neighbors became infectious.

```
1 def infect():
2     ...
3     for agent_ in neighbors:
4         if agent_.state == 0:
5             agent_.state = 3
```

monitor: The code below is added to update the count for the number of days an exposed person has been exposed. If the person reaches the end of the incubation period, he becomes infectious.

```
1 def monitor():
2     ...
3     for agent_ in agents_list:
4         if agent_.state == 3:
5             agent_.days_exposed += 1
6             if agent_.days_exposed > incubation_period:
7                 agent_.state = 1
```

count: Count for the number of exposed people are added.

```
1 def count():
2     ...
3     global E_count
4     ...
5     E_count = len([agent_ for agent_ in agents_list if agent_.state == 3])
```

visualize: Color magenta is assigned for exposed individuals. The dummy plot and legend are updated to include exposed people. And the file name is changed to SEIR.

```
1 def visualize():
2     ...
3     agent_color = {0: 'b', 1: 'r', 2: 'g', 3: 'm'}
4     ...
5     ax.plot([], [], 'mo', label = 'Exposed: ' + str(E_count))
6     ...
7     ax.legend(ncol = 2, loc = 'upper center', bbox_to_anchor = (0.5, -0.01))
8     ...
9     filename = 'SEIR'
```

trend: Like in `visualize`, the plot and legend are updated to include exposed people. Note that the legend is pushed a little bit further up since the layout now contains two rows. The file name is changed to `SEIR_Trend`.

```
1 def trend():
2     ...
3     ax.plot(days, E_list, color = 'magenta', label = 'Exposed')
4     ...
5     ax.legend(ncol = 2, loc = 'upper center', bbox_to_anchor = (0.5, 1.2))
6     ...
7     filename = 'SEIR_Trend'
```

Simulation: The simulation involves no initial exposed individual, and an incubation period of 3 days. Movement is based on scenario 2 where infectious and exposed people do not move.

```
1 ...
2 E_initial = 0
3 ...
4 n_agents = S_initial + E_initial + I_initial + R_initial
5 ...
6 incubation_period = 3
7 ...
8 E_list = []
9 E_list.append(E_initial)
10 ...
11 for agent_ in rd.sample([agent_ for agent_ in agents_list if agent_.state != 1 and agent_.state != 2],
    E_initial):
12     agent_.state = 3
13 ...
14 for i in range(runs):
15     ...
16     E_list.append(E_count)
17     ...
18     if I_count == 0 and E_count == 0:
19         break
```

For the simulation, the initial value for the number of exposed people is added; thus, the variable n_agents must include this value as well. The incubation period parameter must be set. The list of daily number of exposed individuals is initialized. Excluding those assigned to be infectious and recovered, random people are chosen based the number of initial exposed people. The for loop is updated to append the count for the number of exposed people to the initialized list. Finally, the simulation stops before the end if both the number of infectious and exposed individuals have both reached 0.

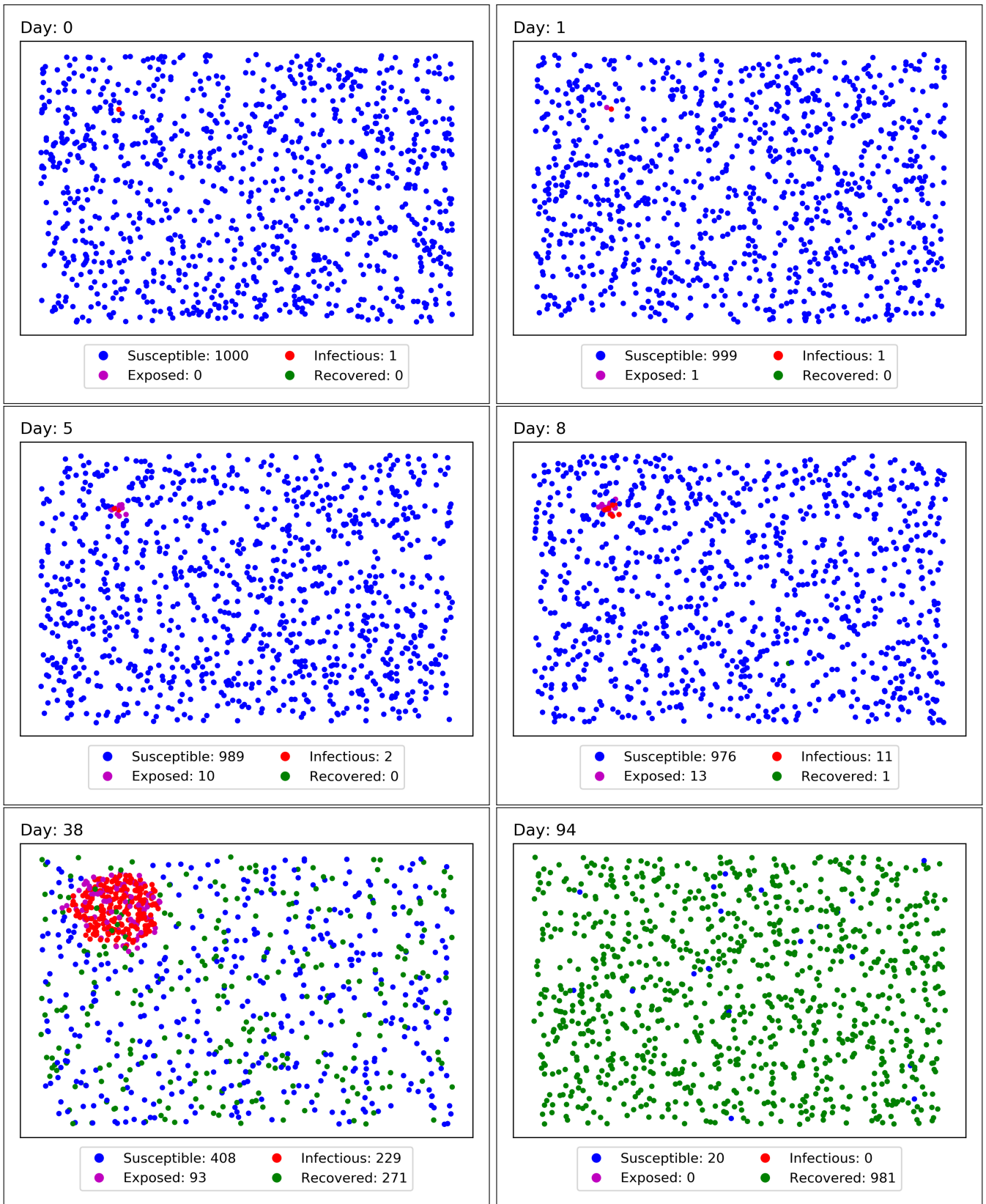


Figure 9: SEIR Model, Some States of the Population. Exposed and infectious individuals do not move.

The lower left panel of Figure 9 shows that the disease at its peak. It was localized in the vicinity of the initial infectious individual. It is noticeable, however, that it took longer for the peak to arrive (Day 38 vs Day 21 in SIR model) because of the incubation period added into the model. It took even longer for the end of the epidemic to come as susceptible people had a lot of opportunities to come into contact with infectious people, even if they were localized (lower right panel).

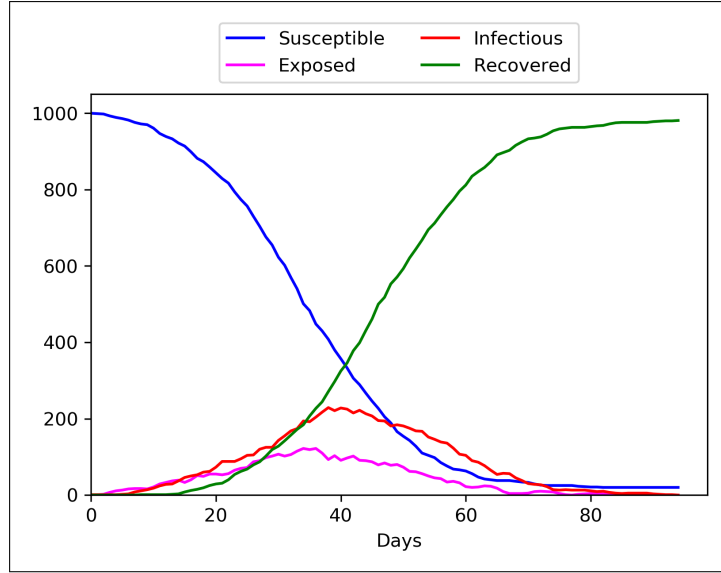


Figure 10: SEIR Model. Daily trend of the number of susceptible, exposed, infectious, and recovered individuals.

Figure 10 shows even more flattening of the curve compared to scenario 2 of the SIR model due to the early arrest of exposed individuals. The trend of the 4 compartments are comparable to the trend shown when the solution to the SIR system of ordinary differential equations in the Overview was generated.

Note: Let E the number of exposed people and σ the latency rate. The set of ordinary differential equations describing the change in the compartments over time for the SEIR Model is given by:

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta I}{N}S \\ \frac{dE}{dt} &= \frac{\beta I}{N}S - \sigma E \\ \frac{dI}{dt} &= \sigma E - \gamma I \\ \frac{dR}{dt} &= \gamma I.\end{aligned}$$

Conclusion

The SIR and SEIR compartmental models are widely used in modeling the spread of infectious diseases. Typically, modelers use the equation-based models because they are easier to implement. However, as shown in this paper, agent-based models, even if more involved in programming, are easier to modify because of the intuitiveness of its components.

One of the most powerful arguments for agent-based models is the ease with which modifications can be incorporated into the model. In this paper, it was quite easy to transition from the SIR to the SEIR framework. In equation based models, this can be done easily by simply adding terms and equations. However, the effects of the added terms are not readily interpretable. They only become apparent when the solution to the system has been generated. And even then, interpretation may be cumbersome especially if one is not skilled in interpreting how ordinary differential equations behave. In an agent-based model, the modifications are logical. The simulation just needs to be run, and the result interpreted based on logic.

Improvements to the model can be easily done. As an example, to simulate quarantine, the random coordinates assigned to agents in the `move` function may be limited to a certain area or a far location. Probabilities may also be incorporated so that the state of being exposed, infectious, or recovered depend on certain probability values. Bounded areas of movement may also be added to simulate social distancing.

Agent-based models are powerful, although they can be very tedious to make. It can be less difficult by following an organized framework so that the different components, e.g., environment, agents, interactions. And once the general model has been setup, modifications can be easily done.

Appendix A: SIR Model Full Code

```
1 ### Needed libraries ###
2
3 # For assigning agents to random locations and for choosing random agents
4 import random as rd
5
6 # For visualizing the state of the agents using plot
7 import matplotlib.pyplot as plt
8
9 # For checking if a filename already exists, to avoid overwriting files
10 import os
11
12
13
14 ### Initialize class to create people ###
15
16 class agent:
17     pass
18
19
20
21 ### Create population ###
22
23 def create_agents():
24
25     # Allow the following variable to be accessed outside the function
26     global agents_list
27
28     # Initialize list of agents
29     agents_list = []
30
31     # Create specified number of agents
32     for i in range(n_agents):
33
34         # Create an agent
35         agent_ = agent()
36
37         # Assign the agent to a random location
38         agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
39
40         # Set the agent to be initially susceptible
41         # 0 = susceptible, 1 = infectious, 2 = recovered
42         agent_.state = 0
43
44         # Count number of days the agent is infectious
45         agent_.days_infectious = 0
46
47         # Add the agent to the list
48         agents_list.append(agent_)
49
50
51
52 ### Everyone moves to a random location ###
53
54 # Scenario 1: Normal day
55
56 # Comment scenario not used
57 # def move():
58
59 # # Agent moves to a random location
60 # for agent_ in agents_list:
61 #     agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
62
63 ## Scenario 2: Somewhat limited social contact for infectious people
64
65 def move():
66
67     # Infectious people do not move
68     for agent_ in agents_list:
69         if agent_.state != 1:
70             agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
71
72
73
74 ### Infection happens within a radius (for non-recovered people) ###
75
```

```

76 def infect():
77
78     # Allow the following variable to be accessed outside the function
79     global infectious_list
80
81     # Create list of infectious people
82     infectious_list = [agent_ for agent_ in agents_list if agent_.state == 1]
83
84     # Create list of neighbors of infectious
85     for agent_ in infectious_list:
86
87         # Create list of neighbors within the radius specified
88         neighbors = [neighbor for neighbor in agents_list if (neighbor.x - agent_.x)**2 + (neighbor.y
- agent_.y)**2 <= infectious_radius**2]
89
90         # Remove person himself from list of neighbors
91         neighbors.remove(agent_)
92
93         # Susceptible neighbors become infectious (recovered individuals are immune)
94         for agent_ in neighbors:
95             if agent_.state == 0:
96                 agent_.state = 1
97
98
99
100 ### Monitor progress of infectious people at the start of the day ###
101
102 def monitor():
103
104     # Add count for number of days an infectious person has been infectious
105     for agent_ in agents_list:
106         if agent_.state == 1:
107             agent_.days_infectious += 1
108
109     # Infectious people who reach the end of infectious period recover
110     if agent_.days_infectious == days_to_recover + 1:
111         agent_.state = 2
112
113
114
115 ### Count members of each compartment ###
116
117 def count():
118
119     # Allow the following variables to be accessed outside the function
120     global S_count, I_count, R_count
121
122     # Count susceptible
123     S_count = len([agent_ for agent_ in agents_list if agent_.state == 0])
124
125     # Count infectious
126     I_count = len([agent_ for agent_ in agents_list if agent_.state == 1])
127
128     # Count recovered
129     R_count = len([agent_ for agent_ in agents_list if agent_.state == 2])
130
131
132
133 ### Visualize population ###
134
135 def visualize():
136
137     # To ensure updated count of compartments
138     count()
139
140     # Initialize subplots
141     fig, ax = plt.subplots()
142
143     # Assign color per compartment
144     agent_color = {0: 'b', 1: 'r', 2: 'g'}
145
146     # Plot each person with his corresponding color
147     for agent_ in agents_list:
148         ax.plot(agent_.x, agent_.y, '.', color = agent_color[agent_.state])
149
150     # Dummy plots for legend
151     ax.plot([], [], 'bo', label = 'Susceptible: ' + str(S_count))

```

```

152 ax.plot([], [], 'ro', label = 'Infectious: ' + str(I_count))
153 ax.plot([], [], 'go', label = 'Recovered: ' + str(R_count))
154
155 # Place legend below the figure
156 ax.legend(ncol = 3, loc = 'upper center', bbox_to_anchor = (0.5, -0.01))
157
158 # Fix the window: choose largest range for random location choices
159 ax.set_xlim([0, 1])
160 ax.set_ylim([0, 1])
161
162 # Title
163 ax.set_title('Day: ' + str(day_count), loc = 'left')
164
165 # Remove extra tick marks on the axes
166 ax.set_xticks([])
167 ax.set_yticks([])
168
169 # Prepare format of file name
170 filename = 'SIR'
171
172 # Starting filename count
173 i = 1
174
175 # Check if filename already exists; add 1 if it does
176 while os.path.exists('{:d}.png'.format(filename, i)):
177     i += 1
178
179 # Save figure
180 plt.savefig('{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)
181
182
183
184 ### Graph trend of compartments ###
185
186 def trend():
187
188     # To ensure updated count of compartments
189     count()
190
191     # Initialize subplots
192     fig, ax = plt.subplots()
193
194     # Plot daily count of each compartment
195     ax.plot(days, S_list, color = 'blue', label = 'Susceptible')
196     ax.plot(days, I_list, color = 'red', label = 'Infectious')
197     ax.plot(days, R_list, color = 'green', label = 'Recovered')
198
199     # Place legend on top of the figure
200     ax.legend(ncol = 3, loc = 'upper center', bbox_to_anchor = (0.5, 1.15))
201
202     # Set the axes to meet at (0, 0)
203     ax.set_xlim(left = 0)
204     ax.set_ylim(bottom = 0)
205
206     # Horizontal axis label
207     ax.set_xlabel('Days')
208
209     # Prepare format of file name
210     filename = 'SIR_Trend'
211
212     # Starting filename count
213     i = 1
214
215     # Check if filename already exists; add 1 if it does
216     while os.path.exists('{:d}.png'.format(filename, i)):
217         i += 1
218
219     # Save figure
220     plt.savefig('{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)
221
222
223
224 ### Simulation ###
225
226 # Comment all visualize() for faster simulation
227 # But replace with count()
228

```

```

229 ## Initial day
230
231 # Start day count
232 day_count = 0
233
234 # Initial values
235 S_initial = 1000
236 I_initial = 1
237 R_initial = 0
238
239 # Initial total population
240 n_agents = S_initial + I_initial + R_initial
241
242 # Infectious radius
243 infectious_radius = 0.025
244
245 # Number of days of infectiousness
246 days_to_recover = 7
247
248 # Initialize list of days
249 days = []
250 days.append(0)
251
252 # Initialize daily count of susceptible
253 S_list = []
254 S_list.append(S_initial)
255
256 # Initialize daily count of infectious
257 I_list = []
258 I_list.append(I_initial)
259
260 # Initialize daily count of recovered
261 R_list = []
262 R_list.append(R_initial)
263
264 # Create the population
265 create_agents()
266
267 # Choose a specified number of random people and make them infectious
268 for agent_ in rd.sample(agents_list, I_initial):
269     agent_.state = 1
270
271 # From the remaining susceptible, choose a specified number of random people and make them recovered
272 for agent_ in rd.sample([agent_ for agent_ in agents_list if agent_.state != 1], R_initial):
273     agent_.state = 2
274
275 visualize()
276 # count()
277
278
279 ## Succeeding days
280
281 # Number of runs
282 runs = 100
283
284 for i in range(runs):
285
286     # Add day count
287     day_count += 1
288
289     # Monitor infectiousness and recovery at the start of the day
290     monitor()
291
292     # Let everyone move
293     move()
294
295     # Visualize new positions
296     visualize()
297     # count()
298
299     # Infections start after the movement
300     infect()
301
302     # Visualize
303     visualize()
304     # count()
305

```

```
306     # Update lists
307     days.append(day_count)
308     S_list.append(S_count)
309     I_list.append(I_count)
310     R_list.append(R_count)
311
312     # Stop simulation once number of infectious becomes 0
313     if I_count == 0:
314         break
315
316 # Plot trend
317 trend()
318
319 ##### end of code #####
```

Appendix B: SEIR Model Full Code

```
1 ### Needed libraries ###
2
3 # For assigning agents to random locations and for choosing random agents
4 import random as rd
5
6 # For visualizing the state of the agents using plot
7 import matplotlib.pyplot as plt
8
9 # For checking if a filename already exists, to avoid overwriting files
10 import os
11
12
13
14 ### Initialize class to create people ###
15
16 class agent:
17     pass
18
19
20
21 ### Create population ###
22
23 def create_agents():
24
25     # Allow the following variable to be accessed outside the function
26     global agents_list
27
28     # Initialize list of agents
29     agents_list = []
30
31     # Create specified number of agents
32     for i in range(n_agents):
33
34         # Create an agent
35         agent_ = agent()
36
37         # Assign the agent to a random location
38         agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
39
40         # Set the agent to be initially susceptible
41         # 0 = susceptible, 1 = exposed, 2 = infectious, 3 = recovered
42         agent_.state = 0
43
44         # Count number of days the agent is exposed
45         agent_.days_exposed = 0
46
47         # Count number of days the agent is infectious
48         agent_.days_infectious = 0
49
50         # Add the agent to the list
51         agents_list.append(agent_)
52
53
54
55 ### Everyone moves to a random location ###
56
57 # Scenario 1: Normal day
58
59 # Comment scenario not used
60 # def move():
61
62 # # Agent moves to a random location
63 # for agent_ in agents_list:
64 #     agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
65
66 ## Scenario 2: Somewhat limited social contact for infectious people
67
68 def move():
69
70     # Exposed and infectious people do not move
71     for agent_ in agents_list:
72         if agent_.state != 1 and agent_.state != 2:
73             agent_.x, agent_.y = rd.uniform(0, 1), rd.uniform(0, 1)
74
75
```

```

76
77 ### Infection happens within a radius (for non-recovered people) ###
78
79 def infect():
80
81     # Allow the following variable to be accessed outside the function
82     global infectious_list
83
84     # Create list of infectious people
85     infectious_list = [agent_ for agent_ in agents_list if agent_.state == 2]
86
87     # Create list of neighbors of infectious
88     for agent_ in infectious_list:
89
90         # Create list of neighbors within the radius specified
91         neighbors = [neighbor for neighbor in agents_list if (neighbor.x - agent_.x)**2 + (neighbor.y
92 - agent_.y)**2 <= infectious_radius**2]
93
94         # Remove person himself from list of neighbors
95         neighbors.remove(agent_)
96
97         # Susceptible neighbors become exposed (recovered individuals are immune)
98         for agent_ in neighbors:
99             if agent_.state == 0:
100                 agent_.state = 1
101
102
103 ### Monitor progress of infectious people at the start of the day ###
104
105 def monitor():
106
107     # Add count for number of days an exposed person has been exposed
108     for agent_ in agents_list:
109         if agent_.state == 1:
110             agent_.days_exposed += 1
111
112     # Exposed people who reach the end of incubation period become infectious
113     if agent_.days_exposed > incubation_period:
114         agent_.state = 2
115
116     # Add count for number of days an infectious person has been infectious
117     for agent_ in agents_list:
118         if agent_.state == 2:
119             agent_.days_infectious += 1
120
121     # Infectious people who reach the end of infectious period recover
122     if agent_.days_infectious > days_to_recover:
123         agent_.state = 3
124
125
126
127 ### Count members of each compartment ###
128
129 def count():
130
131     # Allow the following variables to be accessed outside the function
132     global S_count, E_count, I_count, R_count
133
134     # Count susceptible
135     S_count = len([agent_ for agent_ in agents_list if agent_.state == 0])
136
137     # Count exposed
138     E_count = len([agent_ for agent_ in agents_list if agent_.state == 1])
139
140     # Count infectious
141     I_count = len([agent_ for agent_ in agents_list if agent_.state == 2])
142
143     # Count recovered
144     R_count = len([agent_ for agent_ in agents_list if agent_.state == 3])
145
146
147
148 ### Visualize population ###
149
150 def visualize():
151

```



```

152 # To ensure updated count of compartments
153 count()
154
155 # Initialize subplots
156 fig, ax = plt.subplots()
157
158 # Assign color per compartment
159 agent_color = {0: 'b', 1: 'm', 2: 'r', 3: 'g'}
160
161 # Plot each person with his corresponding color
162 for agent_ in agents_list:
163     ax.plot(agent_.x, agent_.y, '.', color = agent_color[agent_.state])
164
165 # Dummy plots for legend
166 ax.plot([], [], 'bo', label = 'Susceptible: ' + str(S_count))
167 ax.plot([], [], 'mo', label = 'Exposed: ' + str(E_count))
168 ax.plot([], [], 'ro', label = 'Infectious: ' + str(I_count))
169 ax.plot([], [], 'go', label = 'Recovered: ' + str(R_count))
170
171 # Place legend below the figure
172 ax.legend(ncol = 2, loc = 'upper center', bbox_to_anchor = (0.5, -0.01))
173
174 # Fix the window: choose largest range for random location choices
175 ax.set_xlim([0, 1])
176 ax.set_ylim([0, 1])
177
178 # Title
179 ax.set_title('Day: ' + str(day_count), loc = 'left')
180
181 # Remove extra tick marks on the axes
182 ax.set_xticks([])
183 ax.set_yticks([])
184
185 # Prepare format of file name
186 filename = 'SEIR'
187
188 # Starting filename count
189 i = 1
190
191 # Check if filename already exists; add 1 if it does
192 while os.path.exists('{:d}.png'.format(filename, i)):
193     i += 1
194
195 # Save figure
196 plt.savefig('{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)
197
198
199
200 ### Graph trend of compartments ###
201
202 def trend():
203
204     # To ensure updated count of compartments
205     count()
206
207     # Initialize subplots
208     fig, ax = plt.subplots()
209
210     # Plot daily count of each compartment
211     ax.plot(days, S_list, color = 'blue', label = 'Susceptible')
212     ax.plot(days, E_list, color = 'magenta', label = 'Exposed')
213     ax.plot(days, I_list, color = 'red', label = 'Infectious')
214     ax.plot(days, R_list, color = 'green', label = 'Recovered')
215
216     # Place legend on top of the figure
217     ax.legend(ncol = 2, loc = 'upper center', bbox_to_anchor = (0.5, 1.15))
218
219     # Set the axes to meet at (0, 0)
220     ax.set_xlim(left = 0)
221     ax.set_ylim(bottom = 0)
222
223     # Horizontal axis label
224     ax.set_xlabel('Days')
225
226     # Prepare format of file name
227     filename = 'SEIR_Trend'
228

```

```

229 # Starting filename count
230 i = 1
231
232 # Check if filename already exists; add 1 if it does
233 while os.path.exists('{}/{:d}.png'.format(filename, i)):
234     i += 1
235
236 # Save figure
237 plt.savefig('{}/{:d}.png'.format(filename, i), bbox_inches = 'tight', dpi = 300)
238
239
240
241 ### Simulation ###
242
243 # Comment all visualize() for faster simulation
244 # But replace with count()
245
246 ## Initial day
247
248 # Start day count
249 day_count = 0
250
251 # Initial values
252 S_initial = 1000
253 E_initial = 0
254 I_initial = 1
255 R_initial = 0
256
257 # Initial total population
258 n_agents = S_initial + E_initial + I_initial + R_initial
259
260 # Infectious radius
261 infectious_radius = 0.025
262
263 # Incubation period
264 incubation_period = 3
265
266 # Number of days of infectiousness
267 days_to_recover = 7
268
269 # Initialize list of days
270 days = []
271 days.append(0)
272
273 # Initialize daily count of susceptible
274 S_list = []
275 S_list.append(S_initial)
276
277 # Initialize daily count of exposed
278 E_list = []
279 E_list.append(E_initial)
280
281 # Initialize daily count of infectious
282 I_list = []
283 I_list.append(I_initial)
284
285 # Initialize daily count of recovered
286 R_list = []
287 R_list.append(R_initial)
288
289 # Create the population
290 create_agents()
291
292 # Choose a specified number of random people and make them exposed
293 for agent_ in rd.sample(agents_list, E_initial):
294     agent_.state = 1
295
296 # From the remaining susceptible, choose a specified number of random people and make them infectious
297 for agent_ in rd.sample([agent_ for agent_ in agents_list if agent_.state != 1], I_initial):
298     agent_.state = 2
299
300 # From the remaining susceptible, choose a specified number of random people and make them recovered
301 for agent_ in rd.sample([agent_ for agent_ in agents_list if agent_.state != 1 and agent_.state != 2],
302     R_initial):
303     agent_.state = 3
304
305 visualize()

```

```

305 # count()
306
307
308 ## Succeeding days
309
310 # Number of runs
311 runs = 100
312
313 for i in range(runs):
314
315     # Add day count
316     day_count += 1
317
318     # Monitor infectiousness and recovery at the start of the day
319     monitor()
320
321     # Let everyone move
322     move()
323
324     # Visualize new positions
325     visualize()
326     # count()
327
328     # Infections start after the movement
329     infect()
330
331     # Visualize
332     visualize()
333     # count()
334
335     # Update lists
336     days.append(day_count)
337     S_list.append(S_count)
338     E_list.append(E_count)
339     I_list.append(I_count)
340     R_list.append(R_count)
341
342     # Stop simulation once number of infectious becomes 0
343     if I_count == 0:
344         break
345
346 # Plot trend
347 trend()
348
349 ##### end of code #####

```

References

- [1] Bonabeau, Eric. “Agent-Based Modeling: Methods and Techniques for Simulating Human Systems.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, 14 Mar. 2002, pp. 7280-7287.
- [2] Castiglione, Filippo. “Agent Based Modeling.” *Scholarpedia*, Brain Corporation, 29 Sep. 2006, www.scholarpedia.org/article/Agent_based_modeling.
- [3] Kermack, William O, and Anderson G McKendrick. “A Contribution to the Mathematical Theory of Epidemics.” *Proceedings of the Royal Society*, vol. 115, no. 772, 1927, pp. 700-721.
- [4] Sayama, Hiroki. “Agent-Environment Interaction.” *LibreTexts*, 10 November 2019, [https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_\(Sayama\)/19%3A_Agent-Based_Models/19.03%3A_Agent-Environment_Interaction](https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book%3A_Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_(Sayama)/19%3A_Agent-Based_Models/19.03%3A_Agent-Environment_Interaction).