Necessary Modules

In [10]:
```
import numpy as np
import random
# for inline plots in jupyter
%matplotlib inline
# import matplotlib
import matplotlib.pyplot as plt
# for latex equations
from IPython.display import Math, Latex
# for displaying images
from IPython.core.display import Image
# import seaborn
import seaborn as sns
# settings for seaborn plotting style
sns.set(color_codes=True)
# settings for seaborn plot sizes
sns.set(rc={'figure.figsize':(5,5)})
# makeing Dataframe
import pandas as pd
# for showing progress
from tqdm import tqdm
# euclidean distance
from scipy.spatial import distance
```

In [11]:
```
import sys
!{sys.executable} -m pip install -r requirements.txt
```

```
'C:\Program' is not recognized as an internal or external command,
operable program or batch file.
```

Question 1

Let's see how does our model behave, let's first discuss our events:

1. Someone gets infected.
2. Someone dies.
3. Someone recovers.
4. A recovered person goes through contagtious time.

So, now we will see what happens in each situation:

1. $n_h - = 1, n_r, n_u, n_s + = 1, n_d$
2. $n_h, n_r, n_u, n_s - = 1, n_d + = 1$
3. $n_h, n_r + = 1, n_u + = 1, n_s - = 1, n_d$
4. $n_h + = 1$ or $n_h, n_r, n_u - = 1, n_s, n_d$

and there is also the question that do we need to specify our people in the model, which the answer is no, only knowing the numbers are enough.

since all generations are exponential then we can deduse that with probability of $\frac{n_h\mu}{n_h\mu+p_r n_s\mu+p_d n_s\mu+ln(t_u)}$

In [106]:
```
class SimpleVirus:
    def log(self):
        df = pd.DataFrame(self.log_data, columns=self.keys)
#         df = df.set_index(['time'])
        return df

    def save_episode(self, event):
        self.log_data.append([self.time, event, self.number_healthy, self.numb
er_recovered,
                                    self.number_sick, self.number_dead, self.n
u, self.next_infected,
                                    self.next_recover, self.next_contagious])

    def save_dataframe(self,df):
        df.to_csv('BasicSpreadlog.txt', index=False, sep='\t')

    def __init__(self, n=10**5, nh=10**5-1000, nr=0, nu=0, ns=1000, nd=0, pr=
0.9, ar=0, mu=2000, br=30, tu=14,
                 healthy_flag=False, update_policy=0):

        self.number_healthy=nh   # number of healthy
        self.number_recovered=nr   # number of recavered
        self.nu=nu   # number of u
        self.number_sick=ns   # number of sick
        self.number_dead=nd   # number of dead
        self.number_people=n     # number of people
        self.probability_recover=pr   # probability of recovering
        self.probability_dead=1-pr # probability of death

        self.start_interval=ar # a_r
        self.end_interval=br #b_r

        self.mu=mu  # mu of infection rate
        self.tu=tu  # time of u
        self.update_policy=update_policy

        self.time=0 # time in simulation
        self.next_infected=self.generate_infected() # next infection event
        self.recovery_times=[]

        for i in range(self.number_sick):
            self.recovery_times.append(self.recover_time())

        self.next_recover=min(self.recovery_times) if self.recovery_times!=[]
else float('inf') # next recovery event

        self.u_times=[]
        for i in range(self.nu):
            self.u_times.append(np.random.uniform(low=0, high=self.tu))

        self.next_contagious=min(self.u_times) if self.u_times!=[] else float(
'inf') # next contagious time over event

        self.healthy_flag=healthy_flag # count recovered for getting the virus
again
```

```
        # Initiate log
        self.keys=['time','event', 'healthy', 'recovered', 'sick', 'dead', 'co
nvalescence',
          'next infected', 'next recover', 'next contagious']
        self.log_data=[]
        self.save_episode('initaite')


    def updated_mu(self):
        if self.update_policy == 0:
            return self.mu

        elif self.update_policy == 1:
            return self.number_sick * self.mu

        else:
            return (self.number_sick + self.nu) * self.mu


    def infected_time(self):
        return np.random.exponential(1/self.updated_mu())

    def recover_time(self):
        return np.random.uniform(low=self.start_interval, high=self.end_interv
al)

    def contagtious_time(self):
        return self.tu

    def survived(self):
        condition=random.random()
        if condition<=self.probability_recover: return True
        else: return False

    def generate_infected(self):
        return self.time + self.infected_time()

    def generate_recover(self):
        return self.time + self.recover_time()

    def generate_contagtious(self):
        return self.time + self.contagtious_time()

    def infect(self):
        self.time=self.next_infected
        self.number_healthy-=1
        self.number_sick=self.number_sick+1
        self.recovery_times.append(self.generate_recover())
        self.next_recover=min(self.recovery_times)
        if self.number_healthy==0:
            self.next_infected=float('inf')
        else:
            self.next_infected=self.generate_infected()

    def recover(self):
        self.time=self.next_recover
        self.recovery_times.remove(self.next_recover)
```

```
        self.number_sick-=1
        if self.survived():
            self.number_recovered+=1
            self.nu+=1
            self.u_times.append(self.generate_contagtious())
            self.next_contagious=min(self.u_times)
        else:
            self.number_dead+=1

        if self.number_sick==0:
            self.next_recover=float("inf")
        else:
            self.next_recover=min(self.recovery_times)

    def conatagious(self):
        self.time=self.next_contagious
        self.nu-=1
        self.u_times.remove(self.next_contagious)
        self.number_healthy=self.number_healthy+1 if self.healthy_flag else se
lf.number_healthy

        if self.nu==0:
            self.next_contagious=float('inf')
        else:
            self.next_contagious=min(self.u_times)

    def run(self):
        event=min(self.next_contagious, self.next_infected, self.next_recover)
        if event==self.next_infected:
            self.infect()
            event_name='infection'
        elif event==self.next_recover:
            self.recover()
            event_name='recovery'
        else:
            self.conatagious()
            event_name='convalescence'

        self.save_episode(event_name)

    def simulate(self,TIME):
        while self.time <=TIME or self.number_healthy==0 or self.number_sick==
0:
            self.run()

    def plot_simulation(self, TIME, SAVE=False):
        for i in tqdm(range(TIME)):
            self.simulate(i)
        df=self.log()

        if SAVE: self.save_dataframe(df)

        fig, ax =plt.subplots(1,2, figsize=(10,5))
        for i in ["recovered", "sick", "dead"]:
            sns.lineplot(x="time", y=i, data=df, markers=True, legend='brief',
label=i, ax=ax[0])
```

```
        ax[0].set(xlabel='Time')
        ax[0].set(ylabel='# People')

        sns.lineplot(x="time", y="healthy", data=df, markers=True, legend='bri
ef', label="healthy",ax=ax[1])

        plt.title('Simple Simulation')
        plt.show()
```

In [107]:
```
simpVirus1=SimpleVirus(healthy_flag=False)
```

In [108]:
```
simpVirus1.simulate(20)
simpVirus1.log().set_index(['time'])
```

Out[108]:

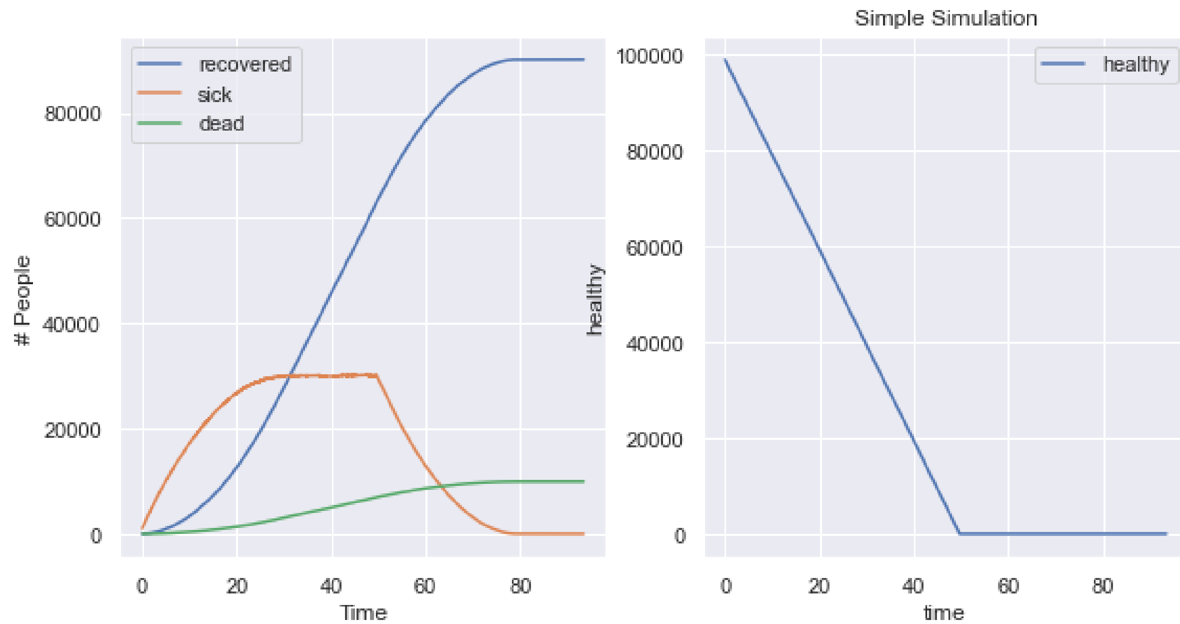
	event	healthy	recovered	sick	dead	convalescence	next infected	next recover	con
time									
0.000000	initaite	99000	0	1000	0	0	0.000169	0.018972	
0.000169	infection	98999	0	1001	0	0	0.000718	0.018972	
0.000718	infection	98998	0	1002	0	0	0.000725	0.018972	
0.000725	infection	98997	0	1003	0	0	0.003084	0.018972	
0.003084	infection	98996	0	1004	0	0	0.003626	0.018972	
...	...	...	...	...	...	...	...	...	
19.998661	infection	59349	12598	26712	1341	11312	19.998780	20.000102	20
19.998780	infection	59348	12598	26713	1341	11312	19.999403	20.000102	20
19.999403	infection	59347	12598	26714	1341	11312	19.999872	20.000102	20
19.999872	infection	59346	12598	26715	1341	11312	20.002063	20.000102	20
20.000102	recovery	59346	12599	26714	1341	11313	20.002063	20.001379	20

54881 rows × 9 columns

A)

In [109]:
```
simpVirus1.plot_simulation(244, SAVE=False)
```

```
100%|██████████████████████████████████████████| 244/244 [02:29<00:00,  1.63it/s]
```



In [31]:
```
simpVirus1.updated_mu()
```

Out[31]:
```
2000
```

In [263]:
```
simpVirus1.number_dead
```

Out[263]:
```
45404
```

In [32]:
```
simpVirus1.number_healthy
```

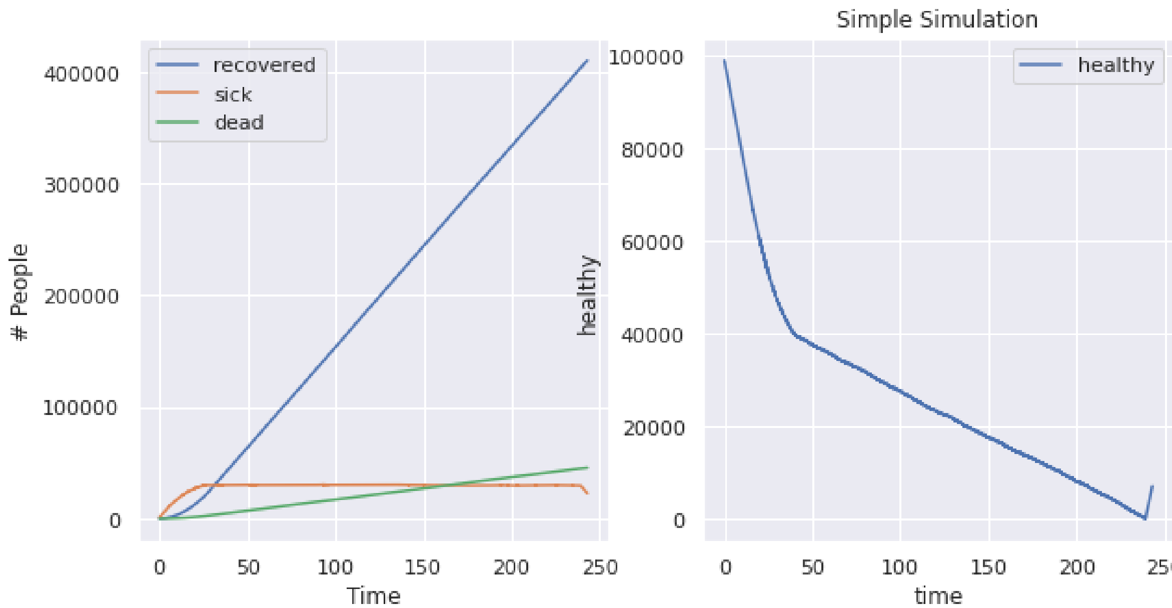
Out[32]:
```
7231
```

B)

In [33]:
```
simpVirus2=SimpleVirus(healthy_flag=True)
```

In [34]:
```
simpVirus2.plot_simulation(244, SAVE=False)
```

```
100%|██████████| 244/244 [11:26<00:00,  2.81s/it]
```



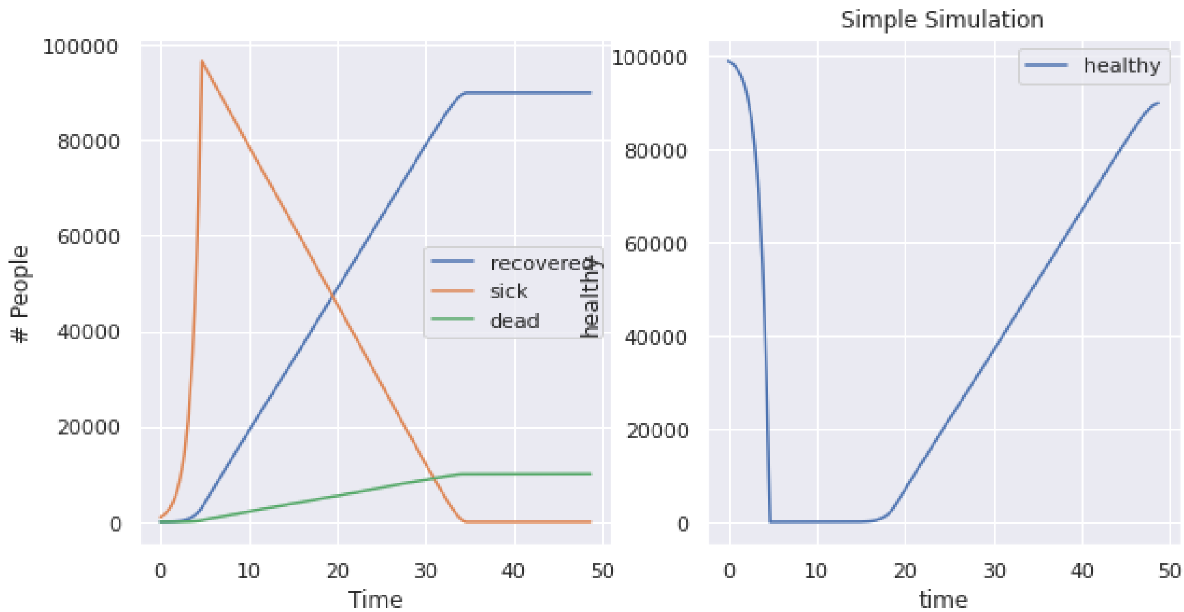
Question 2

A)

In [35]:
```
simpVirus3=SimpleVirus(healthy_flag=True, mu=1 ,update_policy=1)
```

In [36]:
```
simpVirust3.plot_simulation(244, SAVE=False)
```

```
100%|██████████| 244/244 [03:19<00:00,  1.23it/s]
```



In [37]:
```
simpVirus3.number_healthy
```

Out[37]:
```
89957
```

In [38]:
```
simpVirus3.number_dead
```

Out[38]:
```
10042
```

In [39]:
```
simpVirust3.number_recovered
```

Out[39]:
```
89958
```

In [40]:
```
simpVirus3.number_sick
```

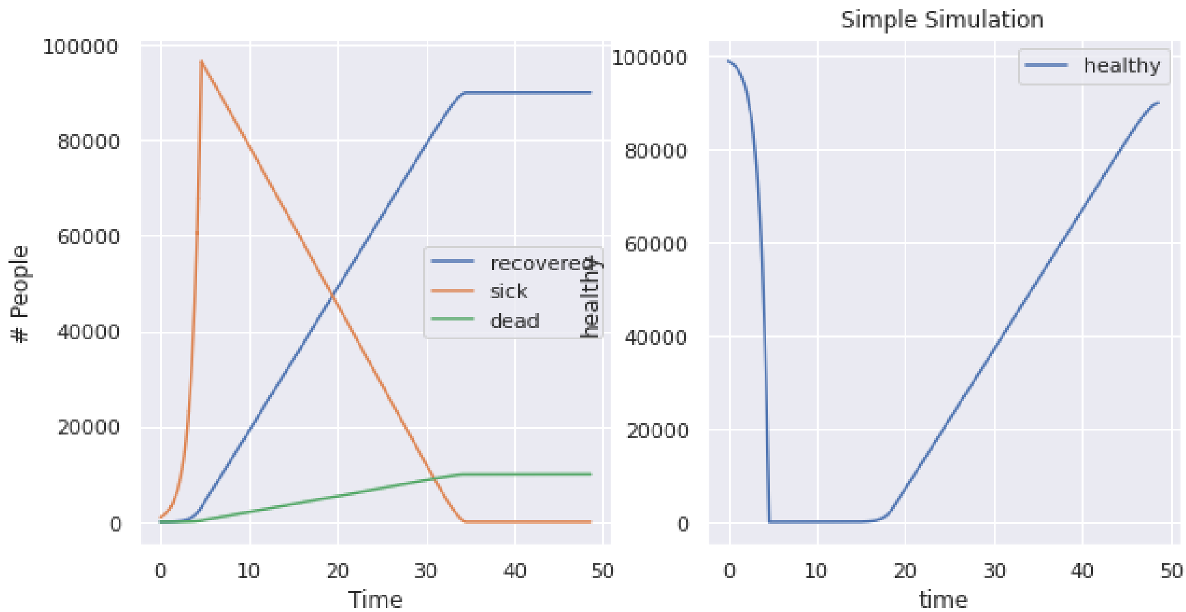
Out[40]:
```
1
```

B)

In [41]:
```
simpVirus4=SimpleVirus(healthy_flag=True, mu=1 ,update_policy=2)
```

In [42]:
```
simpVirus4.plot_simulation(244, SAVE=False)
```

```
100%|██████████| 244/244 [03:19<00:00,  1.22it/s]
```



In [43]:
```
simpVirus4.number_healthy
```

Out[43]: 90002

In [44]:
```
simpVirus4.number_dead
```

Out[44]: 9997

In [45]:
```
simpVirus4.number_recovered
```

Out[45]: 90003

In [46]:
```
simpVirus4.number_sick
```

Out[46]: 1

Question 3

From what I gathered, We will make a $l \times l$ grid (city) and distribute our people in each district. Then we will make a function named *Distance* and we will be the distance of two district. Here we don't really need to think of each person as an individual but only know that the person is from which district. So each district has $n_s, n_h, n_d,$ etc. So each District acts like a small independent experiment, with the assumption of that no two people would get infected at the same time we can then make a ComplexVirus class that control Districts. We also put a condition that each district has at least $\frac{n}{4l^2}$ of population.

Now, Let's make another assumption the number of people getting infected or recover in a day doesn't affect the development of the disease that day.

Introduction

Compartmental models are of great utility in many disciplines and very much so in epidemiology. Let us derive deterministic and stochastic versions of the susceptible-infected-recovered (SIR) model of disease transmission dynamics in a closed population. In so doing, we will use notation that generalizes to more complex systems [(Bretó et al. 2009) (http://dx.doi.org/10.1214/08-AOAS201)](http://dx.doi.org/10.1214/08-AOAS201).

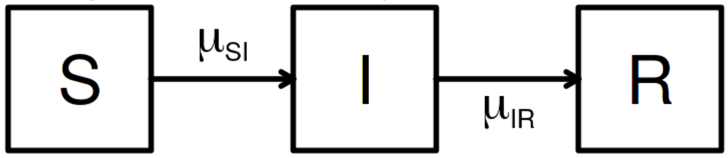


Diagram of the SIR compartmental model.

- Let S, I, and R represent, respectively, the number of susceptible hosts, the number of infected (and, by assumption, infectious) hosts, and the number of recovered or removed hosts.
- We suppose that each arrow has an associated *per capita* rate, so here there is a rate μ_{SI} at which individuals in S transition to I, and μ_{IR} at which individuals in I transition to R.
- To account for demography (birth/death/migration) we allow the possibility of a source and sink compartment, which is not represented on the flow diagram above.
  - We write $\mu_{\bullet S}$ for a rate of births into S.
  - Mortality rates are denoted by $\mu_{S\bullet}$, $\mu_{I\bullet}$, $\mu_{R\bullet}$.
- The rates may be either constant or varying. In particular, for a simple SIR model, the recovery rate μ_{IR} is a constant but the infection rate has the time-varying form

$$\mu_{SI}(t) = \beta\,\frac{I(t)}{N(t)},$$

$$\mu_{SI}(t) = \beta\,\frac{I(t)}{N(t)},$$

  with β being the *contact rate* and N the total size of the host population. In the present case, since the population is closed, we set

$$\mu_{\bullet S} = \mu_{S\bullet} = \mu_{I\bullet} = \mu_{R\bullet} = 0.$$

$$\mu_{\cdot S} = \mu_{S\cdot} = \mu_{I\cdot} = \mu_{R\cdot} = 0.$$

- In general, it turns out to be convenient to keep track of the flows between compartments as well as the number of individuals in each compartment. Let $N_{SI}(t)$ count the number of individuals who have transitioned from S to I by time t. We say that $N_{SI}(t)$ is a *counting process*. A similarly constructed process $N_{IR}(t)$ counts individuals transitioning from I to R. To include demography, we could keep track of birth and death events by the counting processes $N_{\bullet S}(t)$, $N_{S\bullet}(t)$, $N_{I\bullet}(t)$, $N_{R\bullet}(t)$.
  - For discrete population compartment models, the flow counting processes are non-decreasing and integer valued.
  - For continuous population compartment models, the flow counting processes are non-decreasing and real valued.
- The number of hosts in each compartment can be computed via these counting processes. Ignoring demography, we have:

$$
\begin{aligned}
S(t) &= S(0) - N_{SI}(t)\\
I(t) &= I(0) + N_{SI}(t) - N_{IR}(t)\\
R(t) &= R(0) + N_{IR}(t)
\end{aligned}
$$

$$
\begin{aligned}
S(t) &= S(0) - N_{SI}(t)\\
I(t) &= I(0) + N_{SI}(t) - N_{IR}(t)\\
R(t) &= R(0) + N_{IR}(t)
\end{aligned}
$$

These equations represent a kind of conservation law.

- Over any finite time interval $[t, t+\delta)$, we have

$$
\begin{aligned}
\Delta S &= -\Delta N_{SI}\\
\Delta I &= \Delta N_{SI} - \Delta N_{IR}\\
\Delta R &= \Delta N_{IR},
\end{aligned}
$$

$$
\begin{aligned}
\Delta S &= -\Delta N_{SI}\\
\Delta I &= \Delta N_{SI} - \Delta N_{IR}\\
\Delta R &= \Delta N_{IR},
\end{aligned}
$$

where the Δ notation indicates the increment in the corresponding process. Thus, for example $\Delta N_{SI}(t) = N_{SI}(t+\delta) - N_{SI}(t)$.

Compartmental models in theory

The deterministic version of the SIR model

Together with initial conditions specifying $S(0)$, $I(0)$ and $R(0)$, we just need to write down ordinary differential equations (ODE) for the flow counting processes. These are,

$$
\frac{dN_{SI}}{dt} = \mu_{SI}(t)\, S(t), \qquad \frac{dN_{IR}}{dt} = \mu_{IR}\, I(t).
$$

$$
\frac{dN_{SI}}{dt} = \mu_{SI}(t)\, S(t), \qquad \frac{dN_{IR}}{dt} = \mu_{IR}\, I(t).
$$

The simple continuous-time Markov chain version of the SIR model

- Continuous-time Markov chains are the basic tool for building discrete population epidemic models.
- Recall that a *Markov chain* is a discrete-valued stochastic process with the *Markov property*: the future evolution of the process depends only on the current state.
- Surprisingly many models have this Markov property. If all important variables are included in the state of the system, then the Markov property appears automatically.
- The Markov property lets us specify a model by giving the transition probabilities on small intervals together with initial conditions. For the SIR model in a closed population, we have

$$
\begin{aligned}
\mathrm{P}\left[N_{SI}(t+\delta) = N_{SI}(t) + 1\right] &= \mu_{SI}(t)\, S(t)\, \delta + o(\delta)\\
\mathrm{P}\left[N_{SI}(t+\delta) = N_{SI}(t)\right] &= 1 - \mu_{SI}(t)\, S(t)\, \delta + o(\delta)\\
\mathrm{P}\left[N_{IR}(t+\delta) = N_{IR}(t) + 1\right] &= \mu_{IR}(t)\, I(t)\, \delta + o(\delta)\\
\mathrm{P}\left[N_{IR}(t+\delta) = N_{IR}(t)\right] &= 1 - \mu_{IR}(t)\, I(t)\, \delta + o(\delta)
\end{aligned}
$$

$$\begin{aligned}
P\left[N_{SI}(t+\delta) = N_{SI}(t)+1\right] &= \mu_{SI}(t)\,S(t)\,\delta + o(\delta)\\
P\left[N_{SI}(t+\delta) = N_{SI}(t)\right] &= 1-\mu_{SI}(t)\,S(t)\,\delta + o(\delta)\\
P\left[N_{IR}(t+\delta) = N_{IR}(t)+1\right] &= \mu_{IR}(t)\,I(t)\,\delta + o(\delta)\\
P\left[N_{IR}(t+\delta) = N_{IR}(t)\right] &= 1-\mu_{IR}(t)\,I(t)\,\delta + o(\delta)
\end{aligned}$$

- A *simple* counting process is one for which no more than one event can occur at a time (Wikipedia: point process (https://en.wikipedia.org/wiki/Point_process)). Thus, in a technical sense, the SIR Markov chain model we have written is simple. One may want to model the extra randomness resulting from multiple simultaneous events: someone sneezing in a crowded bus, large gatherings at football matches, etc. This extra randomness may even be critical to match the variability in data.
- We will see later, in the measles case study (../measles/measles.html), a situation where this extra randomness plays an important role. The representation of the model in terms of counting processes turns out to be useful for this.

Exercise: From Markov chain to ODE

Find the expected value of $N_{SI}(t+\delta) - N_{SI}(t)$ and $N_{IR}(t+\delta) - N_{IR}(t)$ given the current state, $S(t)$, $I(t)$ and $R(t)$. Take the limit as $\delta \to 0$ and show that this gives the ODE model.

Euler's method for ODE

- Euler (https://en.wikipedia.org/wiki/Leonhard_Euler) took the following approach to numeric solution of an ODE:
  - He wanted to investigate an ODE

$$\frac{dx}{dt} = h(x,t)$$

$$\frac{dx}{dt} = h(x,t)$$

  with an initial condition $x(0)$. He supposed this ODE has some true solution $x(t)$ which could not be worked out analytically. He therefore wished to approximate $x(t)$ numerically.
  - He initialized the numerical solution at the known starting value,

$$\tilde{x}(0) = x(0).$$

$$\tilde{x}(0) = x(0).$$

  Then, for $k = 1, 2, \dots$, he supposed that the gradient dx/dt is approximately constant over the small time interval $k\delta \le t \le (k+1)\delta$. Therefore, he defined

$$\tilde{x}\big((k+1)\delta\big) = \tilde{x}(k\delta) + \delta\, h\big(\tilde{x}(k\delta), k\delta\big).$$

$$\tilde{x}\big((k+1)\delta\big) = \tilde{x}(k\delta) + \delta\, h\big(\tilde{x}(k\delta), k\delta\big).$$

  - This defines $\tilde{x}(t)$ when only for those t that are multiples of δ, but let's suppose $\tilde{x}(t)$ is constant between these discrete times.

- We now have a numerical scheme, stepping forwards in time increments of size δ, that can be readily evaluated by computer.
- Mathematical analysis of Euler’s method (https://en.wikipedia.org/wiki/Euler_method) says that, as long as the function $h(x)$ is not too exotic, then $x(t)$ is well approximated by $\tilde{x}(t)$ when the discretization time-step, δ, is sufficiently small.
- Euler’s method is not the only numerical scheme to solve ODEs. More advanced schemes have better convergence properties, meaning that the numerical approximation is closer to $x(t)$. However, there are 3 reasons we choose to lean heavily on Euler’s method:

  1. Euler’s method is the simplest (the KISS principle).
  2. Euler’s method extends naturally to stochastic models, both continuous-time Markov chains models and stochastic differential equation (SDE) models.
  3. In the context of data analysis, close approximation of the numerical solutions to a continuous-time model is less important than may be supposed, a topic worth further discussion….

Some comments on using continuous-time models and discretized approximations

- In some physical situations, a system follows an ODE model closely. For example, Newton’s laws provide a very good approximation to the motions of celestial bodies.
- In many biological situations, ODE models become good approximations to reality only at relatively large scales. On small temporal scales, models cannot usually capture the full scope of biological variation and biological complexity.
- If we are going to expect substantial error in using $x(t)$ to model a biological system, maybe the numerical solution $\tilde{x}(t)$ represents the system being modeled as well as $x(t)$ does.
- If our model fitting, model investigation, and final conclusions are all based on our numerical solution $\tilde{x}(t)$ (e.g., we are sticking entirely to simulation-based methods) then we are most immediately concerned with how well $\tilde{x}(t)$ describes the system of interest. $\tilde{x}(t)$ becomes more important than the original model, $x(t)$.
- When following this perspective, it is important that one fully describe the numerical model $\tilde{x}(t)$. From this point of view, then, the main advantage of the continuous-time model $x(t)$ is then that it gives a succinct way to describe how $\tilde{x}(t)$ was constructed.
- All numerical methods are, ultimately, discretizations. Epidemiologically, setting δ to be a day, or an hour, can be quite different from setting δ to be two weeks or a month. For continuous-time modeling, we still require that δ is small compared to the timescale of the process being modeled, and the choice of δ does not play an explicit role in the interpretation of the model.
- Putting more emphasis on the scientific role of the numerical solution itself reminds you that the numerical solution has to do more than approximate a target model in some asymptotic sense: the numerical solution should be a sensible model in its own right.

Euler’s method for a discrete SIR model

- Recall the simple continuous-time Markov chain interpretation of the SIR model without demography:

$$\mathrm{P}\left[N_{SI}(t+\delta) = N_{SI}(t)+1\right] = \mu_{SI}(t)\, S(t)\, \delta + o(\delta),$$
$$\mathrm{P}\left[N_{IR}(t+\delta) = N_{IR}(t)+1\right] = \mu_{IR}\, I(t)\, \delta + o(\delta).$$

$$\mathrm{P}\Big[N_{SI}(t+\delta) = N_{SI}(t)+1\Big] = \mu_{SI}(t)\, S(t)\, \delta + o(\delta),$$
$$\mathrm{P}\Big[N_{IR}(t+\delta) = N_{IR}(t)+1\Big] = \mu_{IR}\, I(t)\, \delta + o(\delta).$$

- We look for a numerical solution with state variables $\tilde{S}(k\delta)$, $\tilde{I}(k\delta)$, $\tilde{R}(k\delta)$.
- The counting processes for the flows between compartments are $\tilde{N}_{SI}(t)$ and $\tilde{N}_{IR}(t)$. The counting processes are related to the numbers of individuals in the compartments by the same flow equations we had before:

$$\begin{aligned}\Delta\tilde{S} &= -\Delta\tilde{N}_{SI}\\ \Delta\tilde{I} &= \Delta\tilde{N}_{SI} - \Delta\tilde{N}_{IR}\\ \Delta\tilde{R} &= \Delta\tilde{N}_{IR},\end{aligned}$$

$$\begin{aligned}\Delta\tilde{S} &= -\Delta\tilde{N}_{SI}\\ \Delta\tilde{I} &= \Delta\tilde{N}_{SI} - \Delta\tilde{N}_{IR}\\ \Delta\tilde{R} &= \Delta\tilde{N}_{IR},\end{aligned}$$

- Let's focus $N_{SI}(t)$; the same methods can also be applied to $N_{IR}(t)$.
- Here are three stochastic Euler schemes for N_{SI}:

1. Poisson increments:

$$\Delta\tilde{N}_{SI} \ \sim\ \mathrm{Poisson}\left(\tilde{\mu}_{SI}(t)\,\tilde{S}(t)\,\delta\right),$$

$$\Delta\tilde{N}_{SI} \ \sim\ \mathrm{Poisson}\left(\tilde{\mu}_{SI}(t)\,\tilde{S}(t)\,\delta\right),$$

   where $\mathrm{Poisson}(\mu)$ is the Poisson distribution with mean μ and

$$\tilde{\mu}_{SI}(t) = \beta\,\frac{\tilde{I}(t)}{N}.$$

$$\tilde{\mu}_{SI}(t) = \beta\,\frac{\tilde{I}(t)}{N}.$$

2. Binomial increments with linear probability:

$$\Delta\tilde{N}_{SI} \ \sim\ \mathrm{Binomial}\left(\tilde{S}(t), \tilde{\mu}_{SI}(t)\,\delta\right),$$

$$\Delta\tilde{N}_{SI} \ \sim\ \mathrm{Binomial}\left(\tilde{S}(t), \tilde{\mu}_{SI}(t)\,\delta\right),$$

   where $\mathrm{Binomial}(n,p)$ is the binomial distribution with mean $n\,p$ and variance $n\,p\,(1-p)$.

3. $\Delta\tilde{N}_{SI} \ \sim\ \mathrm{Binomial}\left(\tilde{S}(t), 1-e^{-\tilde{\mu}_{SI}(t)\,\delta}\right)$.

- Note that these schemes agree as $\delta \to 0$.
- What are the advantages and disadvantages of these different schemes? Conceptually, it is simplest to think of (1) or (2). Numerically, it is usually preferable to implement (3).

Compartmental models via stochastic differential equations (SDE)

The Euler method extends naturally to stochastic differential equations. A natural way to add stochastic variation to an ODE $dx/dt = h(x)$ is

$$\frac{dX}{dt} = h(X) + \sigma\,\frac{dB}{dt}$$

$$\frac{dX}{dt} = h(X) + \sigma \frac{dB}{dt}$$

where $B(t)$ is Brownian motion and so dB/dt is Gaussian white noise. The so-called Euler-Maruyama approximation $\tilde{X}$ is generated by

$$\tilde{X}\big(\,(k+1)\delta\,\big) = \tilde{X}(k\delta) + \delta\, h\big(\,\tilde{X}(k\delta)\,\big) + \sigma\sqrt{\delta}\, Z_k$$

$$\tilde{X}\big(\,(k+1)\delta\,\big) = \tilde{X}(k\delta) + \delta\, h\big(\,\tilde{X}(k\delta)\,\big) + \sigma\sqrt{\delta}\, Z_k$$

where $Z_1, Z_2, \dots$ is a sequence of independent standard normal random variables, i.e., $Z_k \sim \mathrm{Normal}(0,1)$. Although SDEs are often considered an advanced topic in probability, the Euler approximation doesn't demand much more than familiarity with the normal distribution.

Exercise: SDE version of the SIR model

Write down the Euler-Maruyama method for an SDE representation of the closed-population SIR model. Consider some difficulties that might arise with non-negativity constraints, and propose some practical way one might deal with that issue.

- A useful method to deal with positivity constraints is to use Gamma noise rather than Brownian noise (Bhadra et al. 2011,@He2010,@laneri10). SDEs driven by Gamma noise can be investigated by Euler solutions simply by replacing the Gaussian noise by an appropriate Gamma distribution.

Euler's method vs. Gillspie's algorithm

- A widely used, exact simulation method for continuous time Markov chains is Gillspie's algorithm (https://en.wikipedia.org/wiki/Gillespie_algorithm) (Gillespie 1977). We do not put much emphasis on Gillespie's algorithm here. Why? When would you prefer an implementation of Gillespie's algorithm to an Euler solution?
- Numerically, Gillespie's algorithm is often approximated using so-called tau-leaping (https://en.wikipedia.org/wiki/Tau-leaping) methods (Gillespie 2001). These are closely related to Euler's approach. Is it reasonable to call a suitable Euler approach a tau-leaping method?

With the above explanation we will change our view to a day to day base. Where everyday people will get infected with a binomial distribution according to where they have been that day. The are two more questions that needs to be address before writting the code. How are we going to handle recovery times and how are we going to handle infection in a place. With the idea that people who recovered and are contagious will not leave the house untill they are healthy again. To each *Place* we can give a triple value of `nh, ns, recovery_times` and get the new `nh, ns, recovery_times` then gather them in a *Place* named city. We will also change recoverd to contagious, and contagious to healthy at the end of the day.

In [13]:
```
class Place:
    def log(self):
        df = pd.DataFrame(self.log_data, columns=self.keys)
#         df = df.set_index(['time'])
        return df

    def save_episode(self):
        self.log_data.append([self.name, self.day, self.number_healthy, self.n
umber_recovered,
                              self.number_sick + (self.qurantine_place.number_
sick if self.qurantine_flag else 0),
                              self.number_dead, self.nu + (self.qurantine_plac
e.nu if self.qurantine_flag else 0), self.mu])

    def save_dataframe(self,df):
        df.to_csv(self.name+'Spreadlog.txt', index=False, sep='\t')


    def initiat_qurantine(self):
        number_qurantine= np.random.binomial(self.number_sick,self.probability
_qurantine)
        if number_qurantine!=0:
            self.number_sick-=number_qurantine
            qurantined=list(np.random.choice(self.recovery_times, number_quran
tine, replace=False))
            self.recovery_times=[time for time in self.recovery_times if time 
not in qurantined]

            self.qurantine_place=Place('qurantined', (float('inf'),float('inf'
)) ,n=number_qurantine, nh=0, nr=0,
                                 ns=number_qurantine, recovery_times=qurantine
d, u_times=[], day=self.day, stay_time=1,
                                 qurantine_flag=False, pq=0.9, nu=0, nd=0, mu=
0, pr=0.9, ar=0, br=30, tu=14,
                                 healthy_flag=True, center=True)
        else:
            qurantined=[]
            self.qurantine_place=Place('qurantined', (float('inf'),float('inf'
)) ,n=number_qurantine, nh=0, nr=0,
                                 ns=number_qurantine, recovery_times=qurantine
d, u_times=[], day=self.day, stay_time=1,
                                 qurantine_flag=False, pq=0.9, nu=0, nd=0, mu=
0, pr=0.9, ar=0, br=30, tu=14,
                                 healthy_flag=True, center=True)

    def qurantine_infection(self, ns):
        number_qurantine = np.random.binomial(ns,self.probability_qurantine)
        infected_number=ns-number_qurantine
        qurantined=[self.day + self.recover_time() for i in range(number_quran
tine)]
        self.qurantine_place.number_sick+=number_qurantine
        self.qurantine_place.recovery_times.extend(qurantined)

        return infected_number

    def qurantine_recover(self):
```

```
        self.qurantine_place.run()
        if self.qurantine_place.number_recovered!=0:
            self.number_recovered+=self.qurantine_place.number_recovered
        if self.qurantine_place.number_healthy!=0:
            self.number_healthy+=self.qurantine_place.number_healthy
        if self.qurantine_place.number_dead!=0:
            self.number_dead+=self.qurantine_place.number_dead

        self.qurantine_place.update(nh=0,nr=0,nd=0)



    def __init__(self, name, location ,n, nh, nr, ns, recovery_times, u_times,
day, stay_time, qurantine_flag, pq=0.9,
                 nu=0, nd=0, mu=2000, pr=0.9, ar=0, br=30, tu=14, healthy_flag
=True, center=False):

        self.n=n
        self.name=name
        self.location= location  # a tuple to show the number of two
        self.number_healthy=nh   # number of healthy
        self.number_recovered=nr   # number of recavered
        self.nu=nu   # number of u
        self.number_sick=ns    # number of sick
        self.number_dead=nd    # number of dead
        self.number_people=n     # number of people
        self.probability_recover=pr   # probability of recovering
        self.probability_dead=1-pr # probability of death
        self.probability_qurantine=pq # probability of qurantine

        self.start_interval=ar # a_r
        self.end_interval=br #b_r

        self.mu=mu  # mu of infection rate
        self.tu=tu  # time of u

        self.day=day # time in simulation
        self.stay_time=stay_time
        self.recovery_times=recovery_times[:]
        if self.recovery_times==[]: self.append_recovery_times(self.number_sic
k)
        self.u_times=u_times[:]
        if self.u_times==[]: self.append_u_times(self.nu)

        self.healthy_flag=healthy_flag # count recovered for getting the virus
again
        self.qurantine_flag=qurantine_flag
        if self.qurantine_flag: self.initiat_qurantine()

        self.center=center

         # Initiate log
        self.keys=['name','day', 'healthy', 'recovered', 'sick', 'dead', 'conv
alescence', 'mu']
        self.log_data=[]
        self.save_episode()
```

```
    def infected_number(self):
        p=(1-np.exp(-self.mu*self.stay_time*(0.005)))
        if(p>1 or p<0): print(self.name, self.day)
        return np.random.binomial(self.number_healthy,p)


    def recover_time(self):
        return np.random.uniform(low=self.start_interval, high=self.end_interv
al)

    def contagtious_time(self):
        return self.tu

    def survived(self, recovered_number):
        nr=np.random.binomial(recovered_number,self.probability_recover)
        return nr, recovered_number-nr

    def append_recovery_times(self, number_of_infected):
        for i in range(number_of_infected):
            self.recovery_times.append(self.day + self.recover_time())


    def append_u_times(self, contagtious_count):
        for i in range(contagtious_count):
            self.u_times.append(self.day + self.contagtious_time())

    def infection(self, Append=True):
        number_of_infected=self.infected_number()
        self.number_healthy-=number_of_infected
        if self.qurantine_flag: number_of_infected=self.qurantine_infection(nu
mber_of_infected)
        self.number_sick+=number_of_infected
        if Append: self.append_recovery_times(number_of_infected)

    def recover(self, Append=True):
        recovered=[time for time in self.recovery_times if time<=self.day]
        self.recovery_times=[time for time in self.recovery_times if time not
in recovered]
        recovered_count=len(recovered)
        self.number_sick-=recovered_count
        contagious_count, death_count = self.survived(recovered_count)
        self.number_recovered+=contagious_count
        self.nu+=contagious_count
        if Append: self.append_u_times(contagious_count)
        self.number_dead+=death_count

    def contagious(self):
        healed=[time for time in self.u_times if time<=self.day]
        self.u_times=[time for time in self.u_times if time not in healed]
        healed_count=len(healed)
        self.nu-=healed_count
        self.number_healthy=self.number_healthy+healed_count if self.healthy_f
lag else self.number_healthy

    def update(self, nh=None, nd=None, nr=None, ns=None, recovery_times=None,
u_times=None, mu=None, stay_time=None):
        if nh!=None: self.number_healthy=nh
```

```
        if nd!=None: self.number_dead=nd
        if nr!=None: self.number_recovered=nr
        if ns!=None: self.number_sick=ns
        if recovery_times!=None: self.recovery_times=recovery_times[:]
        if u_times!=None: self.u_times=u_times[:]
        if mu!=None: self.mu=mu
        if stay_time!= None: self.stay_time=stay_time

    def run(self,Append=True):
        if self.center:
            self.recover(Append=Append)
            self.contagious()

        self.infection(Append=Append)
        if self.qurantine_flag:
            self.qurantine_recover()


        self.day+=1
        self.save_episode()
#         if (self.number_healthy+self.number_dead+self.number_sick+self.nu)>s
elf.n:
#                 print(self.name, self.day-1)

    def distance(place1, place2):
        return  distance.euclidean(place1.location, place2.location)

    def f1(place1, place2, mu, c):
        return mu if Place.distance(place1,place2)<c else 0

    def f2(place1, place2, mu, c):
        d=Place.distance(place1,place2)
        return mu/(1+d) if d<c else 0
```

In [683]:

```
dist=Place(name="Dist", location=(0,0), n=10000, nh=10000-1000, nr=0, ns=1000,
recovery_times=[],
           u_times=[] , day=0, stay_time=1, nu=0, nd=0, mu=0.2, pr=0.9, ar=1,
br=30, tu=14,
           healthy_flag=True, center=True, qurantine_flag=True, pq=0.9)
```

In [677]:

```
for i in tqdm(range(200)):
    dist.run()
```

```
100%|██████████| 200/200 [00:00<00:00, 3324.14it/s]
```

In [678]:
```
dist.log()
```

Out[678]:

	name	day	healthy	recovered	sick	dead	convalescence	mu
0	Dist	0	9000	0	1000	0	0	0.2
1	Dist	1	8992	0	1008	0	0	0.2
2	Dist	2	8976	0	1024	0	0	0.2
3	Dist	3	8963	28	1007	2	28	0.2
4	Dist	4	8959	53	983	5	53	0.2
...	...	...	...	...	...	...	...	...
196	Dist	196	9449	2389	136	303	112	0.2
197	Dist	197	9443	2394	141	303	113	0.2
198	Dist	198	9440	2403	140	304	116	0.2
199	Dist	199	9436	2412	140	305	119	0.2
200	Dist	200	9434	2419	144	306	116	0.2

201 rows × 8 columns

In [680]:
```
dist.nu
```

Out[680]: 12

In [679]:
```
dist.qurantine_place.log()
```

Out[679]:

	name	day	healthy	recovered	sick	dead	convalescence	mu
0	qurantined	0	0	0	898	0	0	0
1	qurantined	1	0	0	905	0	0	0
2	qurantined	2	0	0	920	0	0	0
3	qurantined	3	0	26	903	2	26	0
4	qurantined	4	0	23	881	3	49	0
...	...	...	...	...	...	...	...	...
196	qurantined	196	13	7	127	2	100	0
197	qurantined	197	3	5	132	0	102	0
198	qurantined	198	5	8	131	1	105	0
199	qurantined	199	6	7	131	1	106	0
200	qurantined	200	9	7	132	1	104	0

201 rows × 8 columns

In [412]:
```
dist.qurantine_place.nu
```

Out[412]: 11

In [16]:
```
def partion(n,r):
    if n!=0:
        total=n
        temp = []
        for i in range(r):
            val = np.random.randint(0, total)
            temp.append(val)
            total -= val
        if total!=0: temp[-1]+=total
        random.shuffle(temp)
        assert sum(temp) == n
        return temp
    else:
        return [0]*r
```

In [11]:
```
def partion(n,r):
    normal_pop=list(np.random.normal(n/r,n/r/4,size=r))
    pop=[round(i) for i in normal_pop]
    dif=n-sum(pop)
    add=int(dif/r)
    pop=[i+add for i in pop]
    index=np.random.randint(0, r)
    dif=n-sum(pop)
    pop[index]=pop[index]+dif

    return pop
```

In [101]:
```
partion(100,10)
```

Out[101]:
```
[8, 9, 10, 11, 4, 19, 7, 8, 11, 13]
```

In [14]:
```
class ComplexVirus:
    def log(self, name='districts'):
        logs=[]
        if name=='districts':
            for dist in self.districts:
                for l in dist.log_data:
                    logs.append(l)
            df = pd.DataFrame(logs, columns=self.districts[0].keys)
            df = df.set_index(['name'])

        elif name=='total':
            df = pd.DataFrame(self.log_data, columns=self.keys)

        elif name=='workplace':
            for workplace in self.workplaces:
                for office in workplace:
                    logs.extend(office.log_data)
            df = pd.DataFrame(logs, columns=self.workplaces[0][0].keys)
            df = df.set_index(['name'])

        elif name=='malls':
            for mall in self.malls:
                for shop in mall:
                    logs.extend(shop.log_data)
            df = pd.DataFrame(logs, columns=self.malls[0][0].keys)
            df = df.set_index(['name'])



        return df

    def save_episode(self):
        self.log_data.append([self.day, self.number_healthy,
                              self.number_recovered, self.number_sick, self.nu
mber_dead, self.nu])

    def save_dataframe(self,df):
        df.to_csv('ComplexSpreadlog.txt', index=False, sep='\t')

    def recover_time(self):
        return np.random.uniform(low=self.start_interval, high=self.end_interv
al)

    def contagtious_time(self):
        return self.tu

    def make_malls(self, n):
        ### we don't need the recovery times since we don't handle them here s
o we will make a empty
        malls=[]
        for m in range(n):
            mall=[]
            for i in range(self.lsc):
                for j in range(self.lsc):
                    mall.append(Place(name="Shop"+str(i*self.lsc+j), location=
(i,j), n=0, nh=0, nr=0, ns=0,
```

```
                                      recovery_times=[], u_times=[], day=self.
day, stay_time=1/4, nu=0, nd=0,
                                      mu=self.mu, pr=self.probability_recover,
ar=self.start_interval, br=self.end_interval,
                                      tu=self.tu, healthy_flag=self.healthy_fl
ag, qurantine_flag=False,
                                      pq=self.probability_qurantine, center=Fa
lse))
            malls.append(mall)
        return malls

    def generate_number_workers(self):
        healthy_workers=np.random.binomial(self.number_healthy,self.probabilit
y_working)
        sick_workers=np.random.binomial(self.number_sick - self.number_quranti
ne,self.probability_working)
        return healthy_workers, sick_workers

    def generate_workplace_square(self):
        n=random.randint(1,3)
        return n

    def dist_pop_workplace(self, nh, ns, ws):

        workplaces_healthy_pop=[]
        workplaces_sick_pop=[]

        healthy_pop=partion(nh,self.number_workplaces)
        sick_pop=partion(ns,self.number_workplaces)

        for i in range(self.number_workplaces):
            workplaces_healthy_pop.append(partion(healthy_pop[i], ws[i]**2))
            workplaces_sick_pop.append(partion(sick_pop[i], ws[i]**2))

        return workplaces_healthy_pop, workplaces_sick_pop



    def make_workplaces(self):
        workplaces=[]
        healthy_workers, sick_workers = self.generate_number_workers()
        number_workers = healthy_workers + sick_workers
        ws = [self.generate_workplace_square() for i in range(self.number_work
places)]
        workplaces_healthy_pop, workplaces__sick_pop = self.dist_pop_workplace
(healthy_workers, sick_workers, ws)

        recoverytimes=list(np.random.choice(self.recovery_times, sick_workers,
replace=False))
        index=0
        for m in range(self.number_workplaces):
            offices=[]
            for i in range(ws[m]):
                for j in range(ws[m]):

                    rt=recoverytimes[index:index+workplaces__sick_pop[m][i*ws[
m]+j]]
```

```
                    index+=workplaces__sick_pop[m][i*ws[m]+j]

                    offices.append(Place(name="Office"+str(i*ws[m]+j), locatio
n=(i,j), n=0, nh=workplaces_healthy_pop[m][i*ws[m]+j]
                                         , nr=0, ns=workplaces__sick_pop[m][i*ws[
m]+j],
                                         recovery_times=rt,
                                         u_times=[], day=self.day, stay_time=1/4,
nu=0, nd=0,
                                         mu=self.mu, pr=self.probability_recover,
ar=self.start_interval, br=self.end_interval,
                                         tu=self.tu, healthy_flag=self.healthy_fl
ag, qurantine_flag=False,
                                         pq=self.probability_qurantine, center=Tr
ue))
            workplaces.append(offices)
        return number_workers, ws, workplaces

    def make_house(self):
        # each district needs a
        pass

    def make_districts(self):
        districts=[]
        groups=self.l**2
        min_pop=int(self.number_healthy/4/groups)
        pop_used=min_pop*groups
        partions=partion(self.number_healthy-pop_used,groups)
        population=[part+min_pop for part in partions]
        min_sick=int(self.number_sick/4/groups)
        sick_used=min_sick*groups
        sick_partion=partion(self.number_sick-sick_used,groups) if self.number
_sick!=0 else [0]*groups
        sick=[part+min_sick for part in sick_partion]
#         print(sum(sick)+sum(population))
        for i in range(self.l):
            for j in range(self.l):

                recoverytimes=self.recovery_times[sum(sick[:self.l*i+j]):sum(s
ick[:self.l*i+j+1])]
                utimes=self.u_times[sum(sick[:self.l*i+j]):sum(sick[:self.l*i+
j+1])]

                districts.append(Place(name="district"+str(i*self.l+j), locati
on=(i,j), n=population[self.l*i+j]+sick[self.l*i+j],
                                   nh=population[self.l*i+j], nr=0, ns=sick[sel
f.l*i+j],
                                   recovery_times=recoverytimes, u_times=utimes
,day=self.day, stay_time=1, nu=0, nd=0,
                                   mu=self.mu, pr=self.probability_recover, ar=
self.start_interval, br=self.end_interval,
                                   tu=self.tu, healthy_flag=self.healthy_flag,
qurantine_flag=self.qurantine_flag,
                                   pq=self.probability_qurantine, center=True))


                if self.qurantine_flag:
```

```
                        self.number_qurantine+=districts[i*self.l+j].qurantine_pla
ce.number_sick

        return districts

    def __init__(self, n=10**6, l=5, nh=10**6, nr=0,ns=0, nu=0, nd=0, day=0, w
ork_time=1/4, workplaces=0, malls=0,
                 houses=0, qurantine_flag=False, mu=20, pq=0.9, pr=0.9, ar=0,
br=30, tu=14, healthy_flag=True, f=Place.f1,
                 mu_condition=True, lsc=8, mug=0.36, pw=0.75):

        self.n=n
        self.l=l
        self.lsc=lsc
        self.mug=mug


        self.number_healthy=nh   # number of healthy
        self.number_recovered=nr   # number of recavered
        self.nu=nu   # number of u
        self.number_sick=ns   # number of sick
        self.number_dead=nd   # number of dead
        self.number_people=n     # number of people
        self.number_workplaces=workplaces # number of workplaces
        self.number_houses=houses    # number of houses
        self.number_malls=malls    # number of malls
        self.number_infected_yesterday=0


        self.probability_recover=pr   # probability of recovering
        self.probability_dead=1-pr # probability of death
        self.probability_qurantine=pq # probability of a sick person to qurant
ine itself
        self.probability_working=pw

        self.start_interval=ar # a_r
        self.end_interval=br #b_r

        self.mu=mu  # mu of infection rate
        self.tu=tu  # time of u

        self.day=day # time in simulation
        self.work_time=work_time #work hours

        self.healthy_flag=healthy_flag # count recovered for getting the virus
again

        self.number_qurantine=0
        self.qurantine_flag=qurantine_flag # qurantine or not



        self.recovery_times=[]
        for i in range(self.number_sick):
            self.recovery_times.append(self.recover_time())

        self.u_times=[]
```

```
        for i in range(self.nu):
            self.u_times.append(np.random.uniform(low=0, high=self.tu))

        self.districts=self.make_districts()
        if self.number_malls!=0: self.malls=self.make_malls(self.number_malls)
        if self.number_workplaces!=0: self.number_workers, self.ws, self.workp
laces=self.make_workplaces()

        self.f=f
        self.mu_condition=mu_condition

        self.keys=['day', 'healthy', 'recovered', 'sick', 'dead', 'convalescen
ce']
        self.log_data=[]
        self.save_episode()


    def calc_group_mu(self,places, f=Place.f1, FLAG=True, c=2):
        places_mu=[]
        for place in places:
            mu=0
            for other_place in places:
                if FLAG:
                    mu+=(other_place.number_sick+other_place.nu)*f(place1=plac
e, place2=other_place, mu=self.mu, c=c)
                else:
                    mu+=(other_place.number_sick)*f(place1=place, place2=other
_place, mu=self.mu, c=c)
            places_mu.append(mu)

        return places_mu

    def districts_day(self, stay_time=1 ,f=Place.f1, FLAG=True):
        districts_mu=self.calc_group_mu(self.districts,f=f, FLAG=FLAG)
        for i in range(self.l**2):
            self.districts[i].update(stay_time=stay_time,mu=districts_mu[i])
            self.districts[i].run()

    def dist_pop_malls(self):
        n=np.random.binomial(self.number_healthy,self.mug)
        s=np.random.binomial(self.number_sick - self.number_qurantine,(self.nu
mber_sick - self.number_qurantine)/(self.number_healthy+self.number_sick- self
.number_qurantine))
        healthy_pop=partion(n,self.number_malls)
        sick_pop=partion(s,self.number_malls)
        malls_healthy_pop=[]
        malls_sick_pop=[]
        for i in range(self.number_malls):
            malls_healthy_pop.append(partion(healthy_pop[i],self.lsc**2))
            malls_sick_pop.append(partion(sick_pop[i],self.lsc**2))

        return malls_healthy_pop, malls_sick_pop

    def send_sick_district(self, infected):
        recovery_times=[]
        while infected>0:
            i=np.random.randint(0,self.l**2)
```

```
            if self.districts[i].number_healthy>0:
                sick=np.random.randint(1, min(self.districts[i].number_healthy
, infected)+1)
                if sick<=self.districts[i].number_healthy:
                    self.districts[i].number_sick+=sick
                    self.districts[i].number_healthy-=sick
                    infected-=sick
#                   print(sick)
                    start=len(self.districts[i].recovery_times)
                    self.districts[i].append_recovery_times(sick)
                    recovery_times.extend(self.districts[i].recovery_times[sta
rt:])
#                   print(sick==len(self.districts[i].recovery_times[sta
rt:]))

        return recovery_times

    def malls_day(self, stay_time=1/6 ,f=Place.f1, FLAG=True):
        healthy, sick= self.dist_pop_malls()
        total_infected=0
        for i in range(self.number_malls):
            mall_mu=self.calc_group_mu(self.malls[i],f=f, FLAG=FLAG)
            for j in range(self.lsc**2):

                self.malls[i][j].update(stay_time=stay_time,mu=mall_mu[j], nh=
healthy[i][j], ns=sick[i][j], recovery_times=[], u_times=[])
                self.malls[i][j].run()
                total_infected+= self.malls[i][j].number_sick- sick[i][j]

#         print(total_infected)
        self.send_sick_district(total_infected)

    def add_sick_workers(self, n):
        r=np.random.binomial(n, self.probability_working)
#         print(r)
        rt=list(np.random.choice(self.recovery_times[-n:], r, replace=False))
        counter=set()
        condition=sum([ws**2 for ws in self.ws])
        while r>0:
            m=random.randint(0, self.number_workplaces-1)
#             print(m)
            i=random.randint(0, self.ws[m]-1)
            j=random.randint(0,self.ws[m]-1)
            target=self.workplaces[m][i*self.ws[m]+j]
            if target.number_healthy!=0:
                target.number_healthy-=1
                target.number_sick+=1
                target.recovery_times.append(rt[r-1])
                r-=1
            else:
                counter.add((m,i,j))
            if len(counter)==condition: break

    def work_day(self, stay_time=1/6 ,f=Place.f1, FLAG=True):
        if self.number_infected_yesterday>0:
            self.add_sick_workers(self.number_infected_yesterday)
        for i in range(self.number_workplaces):
```

```
            workplace_mu = self.calc_group_mu(self.workplaces[i],f=f, FLAG=FLA
G)
            for j in range(self.ws[i]**2):
                sick=self.workplaces[i][j].number_sick
                dead=self.workplaces[i][j].number_dead
                recovered=self.workplaces[i][j].number_recovered

                self.workplaces[i][j].update(stay_time=stay_time, mu=workplace
_mu[j])
                self.workplaces[i][j].run(Append=False)

                infected=(self.workplaces[i][j].number_sick-sick)+(self.workpl
aces[i][j].number_recovered-recovered)+(self.workplaces[i][j].number_dead-dead
)
                self.workplaces[i][j].recovery_times.extend(self.send_sick_dis
trict(infected))

    def update(self):
        sum_healthy=0
        sum_dead=0
        sum_recovery=0
        sum_sick=0
        sum_nu=0
        sum_qurantine=0
        recoverytimes=[]
        utimes=[]
        for place in self.districts:
            sum_healthy+=place.number_healthy
            sum_dead+=place.number_dead
            sum_recovery+=place.number_recovered
            sum_sick+=place.number_sick
            sum_nu+=place.nu
            recoverytimes.extend(place.recovery_times)
            utimes.extend(place.u_times)
            if self.qurantine_flag:
                sum_healthy+=place.qurantine_place.number_healthy
                sum_dead+=place.qurantine_place.number_dead
                sum_recovery+=place.qurantine_place.number_recovered
                sum_sick+=place.qurantine_place.number_sick
                sum_qurantine+=place.qurantine_place.number_sick
                sum_nu+=place.qurantine_place.nu
                recoverytimes.extend(place.qurantine_place.recovery_times)
                utimes.extend(place.qurantine_place.u_times)

        self.number_healthy=sum_healthy
        self.number_dead=sum_dead
        self.number_recovered=sum_recovery
        self.number_sick=sum_sick
        self.nu=sum_nu
        self.number_qurantine=sum_qurantine
        self.recovery_times=recoverytimes[:]
        self.u_times=utimes[:]

    def run_day(self):
        if self.number_workplaces!=0: self.work_day(stay_time=self.work_time ,
f=self.f, FLAG=self.mu_condition)
        temp=self.number_sick
```

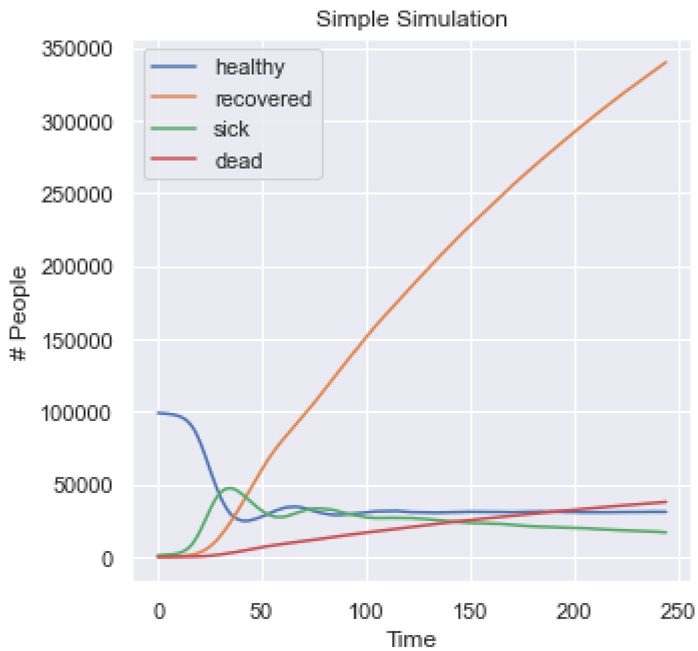
```
        if self.number_malls!=0: self.malls_day(stay_time=1/6 ,f=self.f, FLAG=
self.mu_condition)
        self.districts_day(f=self.f, FLAG=self.mu_condition)
        self.update()
        temp= self.number_sick - self.number_qurantine - temp
        self.day+=1
        self.number_infected_yesterday=temp
        self.save_episode()



    def simulate(self, TIME):
        for i in tqdm(range(TIME)):
            self.run_day()
            if (self.number_healthy+self.number_dead+self.number_sick+self.nu)
!=self.n:
                print('Population:', self.n, 'Healthy:', self.number_healthy,
'sick:', self.number_sick,'dead:', self.number_dead, 'nu:', self.nu,)
                break



    def plot_simulation(self, TIME, SAVE=False):
        self.simulate(TIME)
        df=self.log(name='total')

        if SAVE: self.save_dataframe(df)

#         fig, ax =plt.subplots(1,2, figsize=(10,5))
        for i in ['healthy' ,"recovered", "sick", "dead"]:
           ax = sns.lineplot(x="day", y=i, data=df, markers=True, legend='brie
f', label=i)


        ax.set(xlabel='Time')
        ax.set(ylabel='# People')

#         sns.lineplot(x="time", y="healthy", data=df, markers=True, legend='b
rief', label="healthy",ax=ax[1])

        plt.title('Simple Simulation')
        plt.show()
```

With F1

In [92]:
```
comVirus=ComplexVirus(n=10**5, l=5, nh=10**5-1000, nr=0, ns=1000, nu=0, mu=0.0
02, healthy_flag=True, f=Place.f2, mu_condition=False)
```

In [93]:
```
comVirus.plot_simulation(244)
```

```
100%|████████████████████████████████████████████████████████████
██████| 244/244 [00:53<00:00,  4.57it/s]
```



In [94]:
```
df=comVirus.log(name='total')
df
```

Out[94]:

	day	healthy	recovered	sick	dead	convalescence
0	0	99000	0	1000	0	0
1	1	98904	0	1096	0	0
2	2	98805	24	1166	5	24
3	3	98688	66	1238	8	66
4	4	98570	106	1311	13	106
...	...	...	...	...	...	...
240	240	30866	335867	17268	37317	14549
241	241	30949	336885	17137	37428	14486
242	242	30982	337872	17077	37545	14396
243	243	31000	338953	16900	37658	14442
244	244	30960	339979	16818	37775	14447

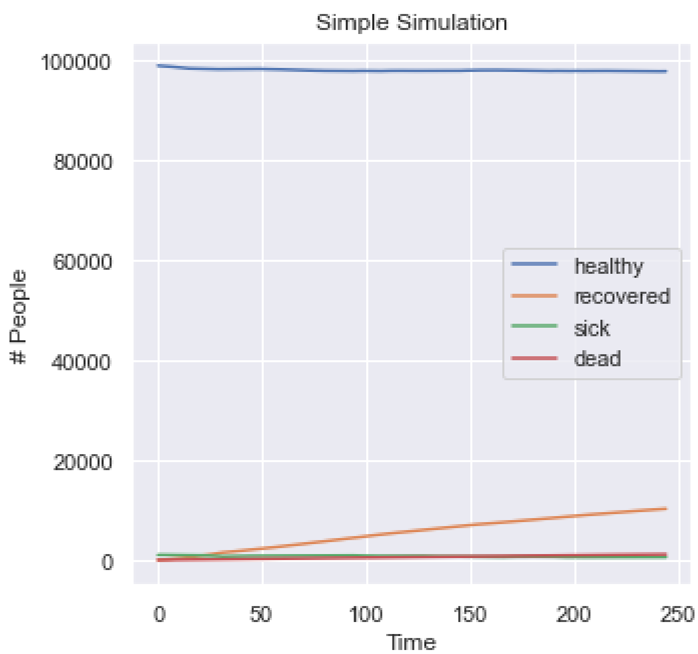
245 rows × 6 columns

With Qurantine

In [95]:
```
comVirus1=ComplexVirus(n=10**5, l=5, nh=10**5-1000, nr=0, ns=1000, nu=0, mu=0.
002, healthy_flag=True, qurantine_flag=True, f=Place.f1, mu_condition=True, pq
=0.9)
```

In [96]:
```
comVirus1.plot_simulation(244)
```

```
100%|██████████████████████████████████████████████████████████████
███████| 244/244 [00:02<00:00, 90.91it/s]
```



In [88]:
```
comVirus1.log(name='total')
```

Out[88]:

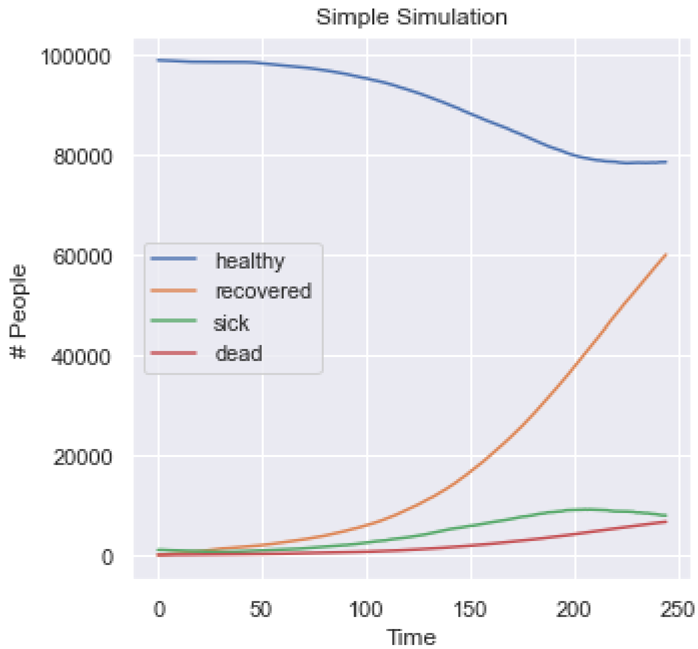
	day	healthy	recovered	sick	dead	convalescence
0	0	99000	0	1000	0	0
1	1	98971	0	1029	0	0
2	2	98944	28	1024	4	28
3	3	98914	64	1014	8	64
4	4	98887	97	1004	12	97
...	...	...	...	...	...	...
240	240	99427	4606	47	469	57
241	241	99431	4608	45	470	54
242	242	99428	4610	47	470	55
243	243	99429	4614	45	470	56
244	244	99432	4616	45	470	53

245 rows × 6 columns

In [117]:
```
comVirus2=ComplexVirus(n=10**5, l=5, nh=10**5-1000, nr=0, ns=1000, nu=0, mu=0.002, healthy_flag=True, qurantine_flag=True, f=Place.f1, malls=7, lsc=4)
```

In [118]:
```
comVirus2.plot_simulation(244)
```

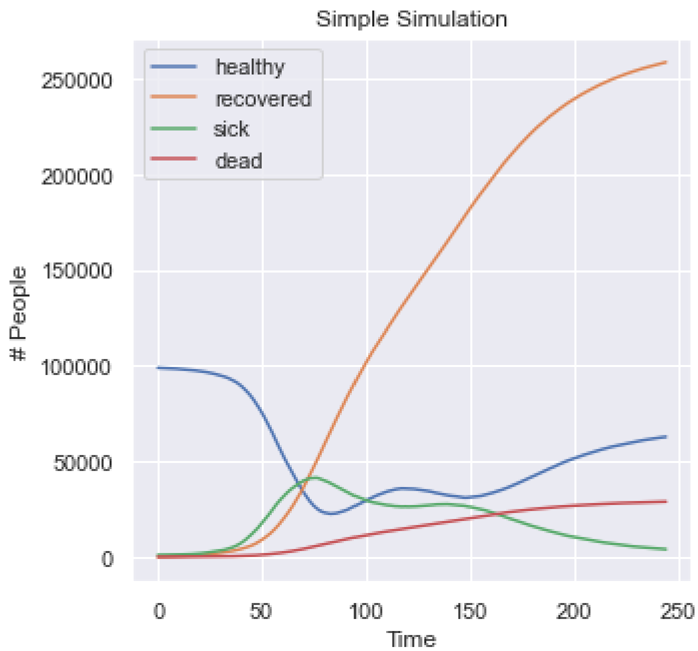
```
100%|██████████| 244/244 [00:13<00:00, 18.57it/s]
```



In [119]:
```
comVirus3=ComplexVirus(n=10**5, l=5, nh=10**5-1000, nr=0, ns=1000, nu=0, mu=0.002, healthy_flag=True, qurantine_flag=True, f=Place.f2, malls=7, lsc=4, work_time=1/4, workplaces=4)
```

In [120]:
```
comVirus3.plot_simulation(244)
```

```
100%|██████████████████████████████████████████████████| 244/244 [00:38<00:00,  6.29it/s]
```



In [26]:
```
comVirus3.log(name='total')
```

Out[26]:

	day	healthy	recovered	sick	dead	convalescence
0	0	99000	0	1000	0	0
1	1	98702	0	1298	0	0
2	2	98325	39	1631	5	39
3	3	97808	74	2111	7	74
4	4	97186	153	2645	16	153
...	...	...	...	...	...	...
240	240	12913	433060	20890	48465	17732
241	241	12964	434347	20752	48609	17675
242	242	12934	435576	20687	48744	17635
243	243	12962	436829	20572	48887	17579
244	244	12940	438016	20532	49018	17510

245 rows × 6 columns

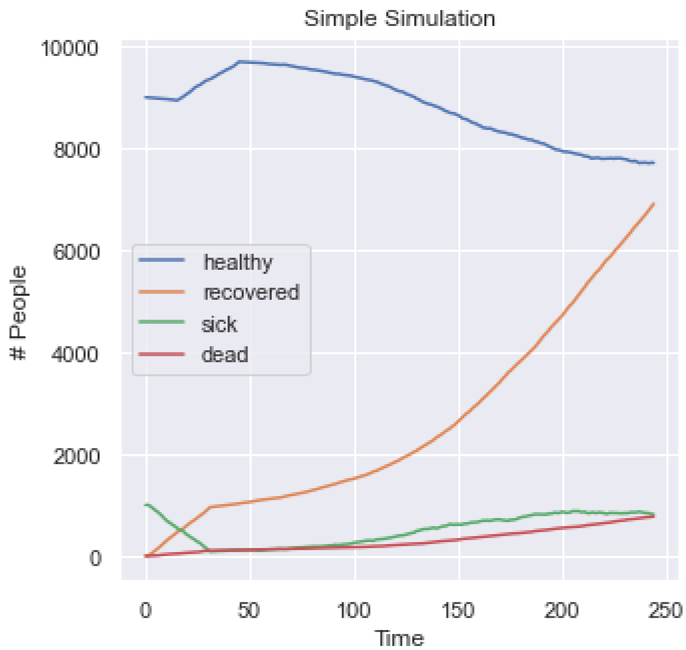
Question 5

for the last question we don't need to do anything but just because our partion function was not great

In [22]:
```python
comVirus4 = ComplexVirus(n=10**4, l=25, nh=10**4-1000, nr=0, ns=1000, nu=0, mu
=0.002, healthy_flag=True, mu_condition=True, qurantine_flag=False, f=Place.f2
, malls=7, lsc=4, work_time=1/4, workplaces=4)
```

In [20]:
```python
comVirus4.plot_simulation(244)
```

```
100%|██████████████████████████████████████████████████████
██████| 244/244 [21:07<00:00,  5.20s/it]
```



In [121]:
```python
comVirus4.log(name='total')
```

Out[121]:

	day	healthy	recovered	sick	dead	convalescence
0	0	99000	0	1000	0	0
1	1	98990	0	1010	0	0
2	2	98979	32	983	6	32
3	3	98970	66	953	11	66
4	4	98956	97	933	14	97
...	...	...	...	...	...	...
240	240	76207	116492	5325	13100	5368
241	241	76366	116802	5252	13146	5236
242	242	76496	117147	5165	13190	5149
243	243	76587	117469	5105	13231	5077
244	244	76665	117763	5096	13260	4979

245 rows × 6 columns

Debug

In [471]:
```
comVirus4 = ComplexVirus(n=10**5, l=10, nh=10**5-1000, nr=0, ns=1000, nu=0, mu
=0.002, healthy_flag=True, mu_condition=False, qurantine_flag=False, f=Place.f
1, malls=7, lsc=4, work_time=1/4, workplaces=1)
```

In [472]:
```
for i in tqdm(range(100)):
    comVirus4.run_day()
```

```
100%|██████████| 100/100 [02:06<00:00,  1.26s/it]
```

In [473]:
```
df=comVirus4.log('malls')
df.loc[df['sick']<0]
```

Out[473]:

	day	healthy	recovered	sick	dead	convalescence	mu
name							

In [475]:
```
df[:20]
```

Out[475]:

	day	healthy	recovered	sick	dead	convalescence	mu
name							
Shop0	0	0	0	0	0	0	0.002
Shop0	1	2	0	0	0	0	0.000
Shop0	2	4	0	0	0	0	0.000
Shop0	3	0	0	0	0	0	0.002
Shop0	4	0	0	0	0	0	0.000
Shop0	5	1	0	0	0	0	0.000
Shop0	6	0	0	0	0	0	0.000
Shop0	7	0	0	2	0	0	0.002
Shop0	8	2	0	0	0	0	0.004
Shop0	9	0	0	0	0	0	0.006
Shop0	10	1	0	0	0	0	0.002
Shop0	11	0	0	0	0	0	0.002
Shop0	12	475	0	4	0	0	0.056
Shop0	13	0	0	9	0	0	0.054
Shop0	14	0	0	0	0	0	0.018
Shop0	15	0	0	0	0	0	0.002
Shop0	16	35	0	0	0	0	0.000
Shop0	17	0	0	0	0	0	0.062
Shop0	18	0	0	3	0	0	0.316
Shop0	19	0	0	0	0	0	0.006

In [393]:
```
len(comVirus4.workplaces[0][0].recovery_times)
```

Out[393]: 1235

In [330]:
```
comVirus4.workplaces[0][0].number_sick
```

Out[330]: 1481

In [163]:
```
len(comVirus4.workplaces[0][0].u_times)
```

Out[163]: 0

In [174]:
```
a=list(range(7))
start=len(a)
a.extend(list(range(9)))

a[start:]
a
```

Out[174]:
```
[0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```