MOD510 22H| Mandatory Project 4| Coronavirus Goes Randomly Viral

Andreas Nonslid Håvardsen | Daniel Fylling | Sahar Kadkhodamasoum Ali

Abstract

Spreading of viruses through populated areas has been identified as a major concern of spatial epidemology. In order to analyze the spatial and temporal distribution of infectious diseases in a given population with different geographical settings, we need to build dynamic models that predict outbreak behavoiur globally. Transmission of an infectious disease may occur through several pathways;

1. By means of contact with infected individuals
2. By water, airbone inhalation
3. Through vector-borne spread (1).

However, this project focuses on direct contact with an infected individual as the main method of contagious disease transmission. The goal of this project is modeling the dynamics of epidemics based on a Random Walk Model (RWM). In each exercise we model different flows of people from one compartment to another one. In addition, RWS models is going to be compared with the Classical Compartment Model which is used in Ordinary Differential Equation to figure out how epidemic dynamics spread out in the population. We found that assuming the disease transmission rate (β) to be constant, wasn't sufficient to represent the real world.

Throughout this exercise we also observe that the model behaves like a Sigmoid function (5), when it only has infected and susceptible people, whereas we see a skewed normal distribution (6) when adding a big recovery rate.

Introduction

In this project, we use a very simple model: a random walker simulation (which is modeled on an infected person) takes a positive integer number of steps on a 2D square lattice. In each step, the visited site has a probability p of becoming infected. In this case, at the site of the new infection, it branches off another random walker, also with lifetime τ, that can also further spread the contagion to further sites, and so on. Our goal is studying the results from two aspects of temporal and spatial. During this project, it is proved that increasing the number of initial infected persion leads to an decrease in number of time steps for all people being infected.

In this project we use a random walker simulation to simulate the spread of a virus through a population.

The way this works is by using a 2-dimensional grid for positions. Then the population occupy this grid, sharing the disease by having a already infected person and susceptible person step on the same square. When they do, a random chance check decides whether it spread to this person or not.

Excercise 1: Random walk simulation with SI-model

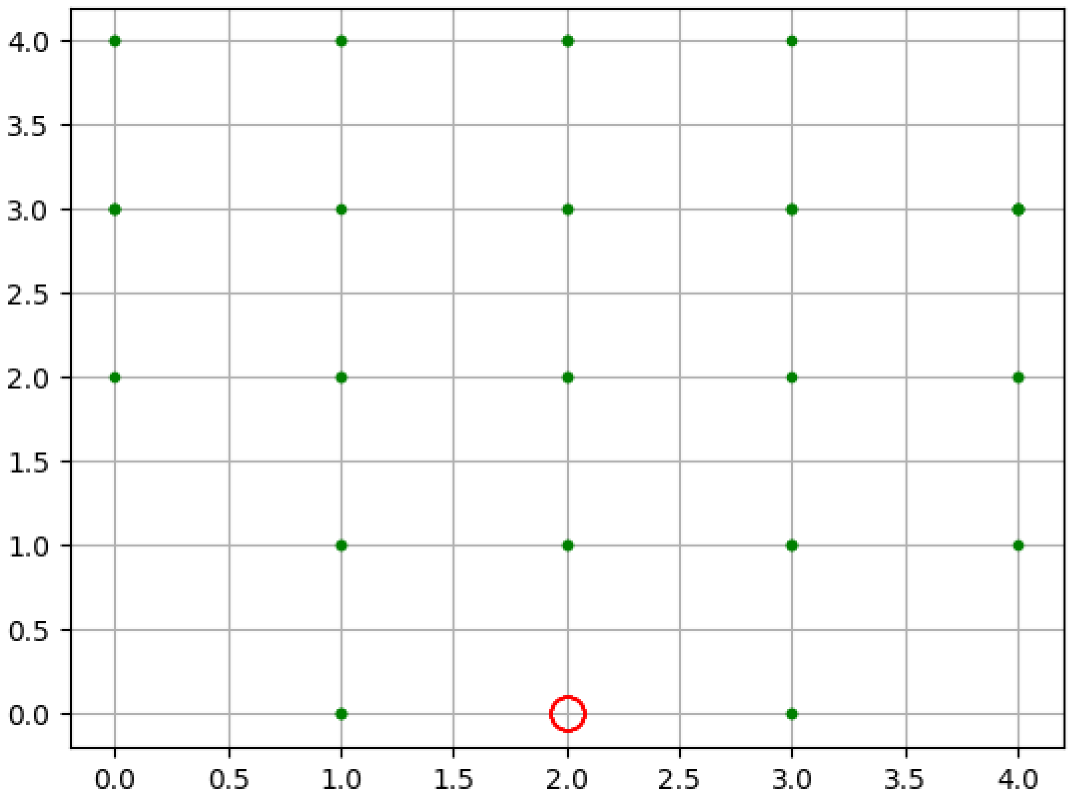
At first, we consider the RWS in the simplest condition which is SI-model (Susceptible-Infected). In this model, a 2-D rectangular lattice with the cordinations of 50*50 is the restricted spatial area. In each time step, people can moved randomly between neighbouring nodes by moving North, South, East and West. Then, the state of people after each time step is defined between terms: Susceptible and infectiuos. Susceptible people has the probability of 0.9 for getting infected (3).

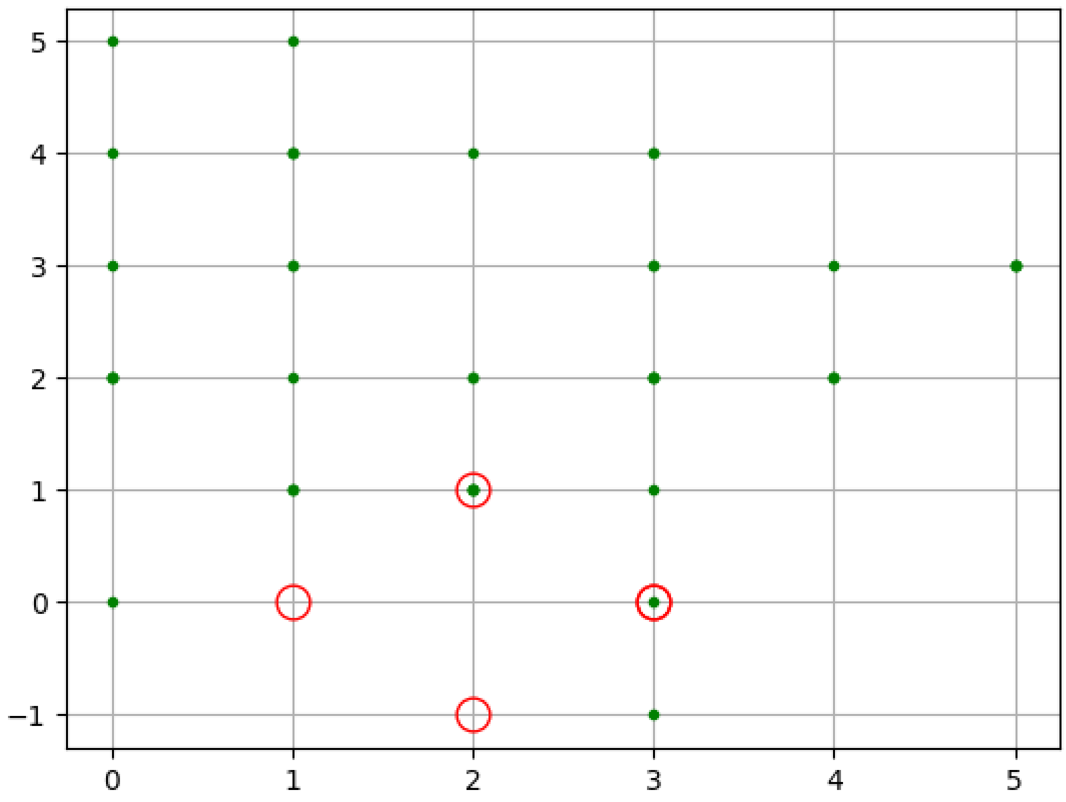
Part 1

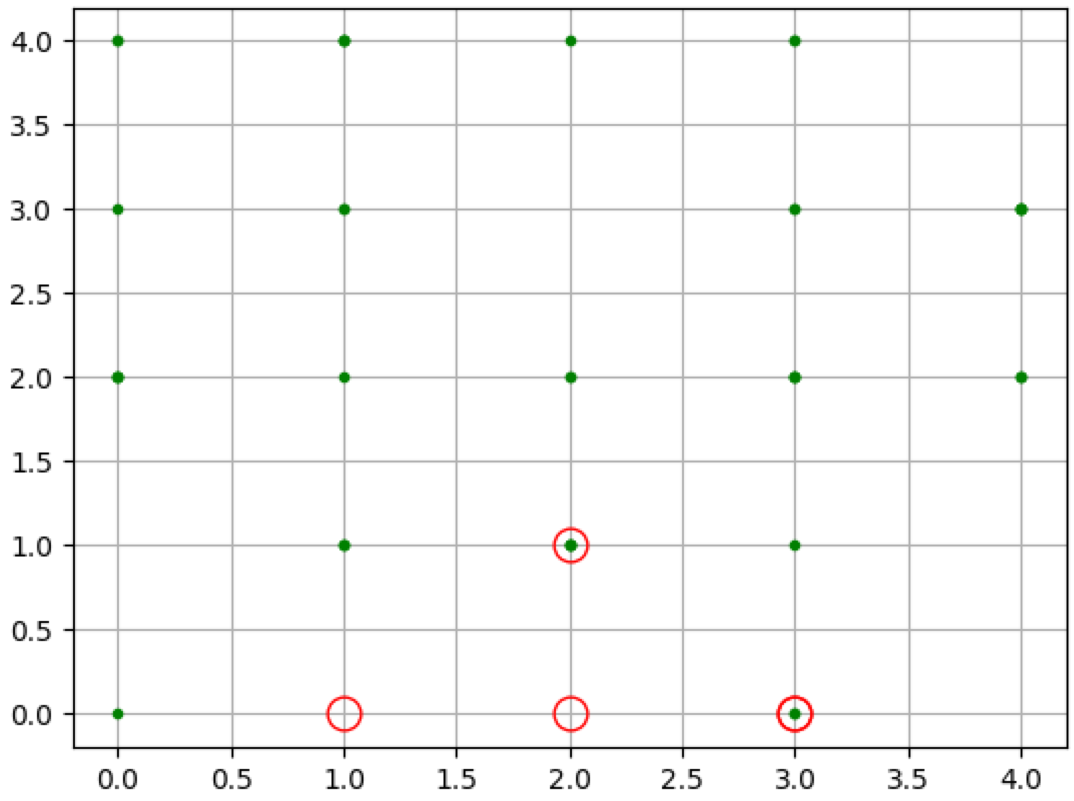
In this part, we coded a class **(RandomlyViral)** for RWS.

In [ ]:
```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
import pandas as pd
import seaborn as sns
import math
import os
from randomlyViral import RandomlyViral
```

In [ ]:
```python
RandomlyViral(simulation_size = 'small' ,no_init_infected=1).single_simulation()
```


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


5
4
3
2
1
0
−1
0
1
2
3
4
5



Plot 1 - t=0
In this plot, there are 50 randomly placed "Walkers", of which one is infected (red circle) all the others are susceptible in a 5x5 grid is shown. We observe that the infectious walkers is seemingly by itself, but this is not neccessarily the case. That is because the walkers first infect (with some probability) those in their own grid cell, before moving. In this plot therefore, the walkers have not yet had the time to move, and might just be many infected people stacked on top of eachother.

Plot 2 - t=1
After running movement function, we can see that the infectious walkers have spread and that some walkers have left the "legal" 5x5 grid area. This is not a real timestep since no data is recorded in this state.*

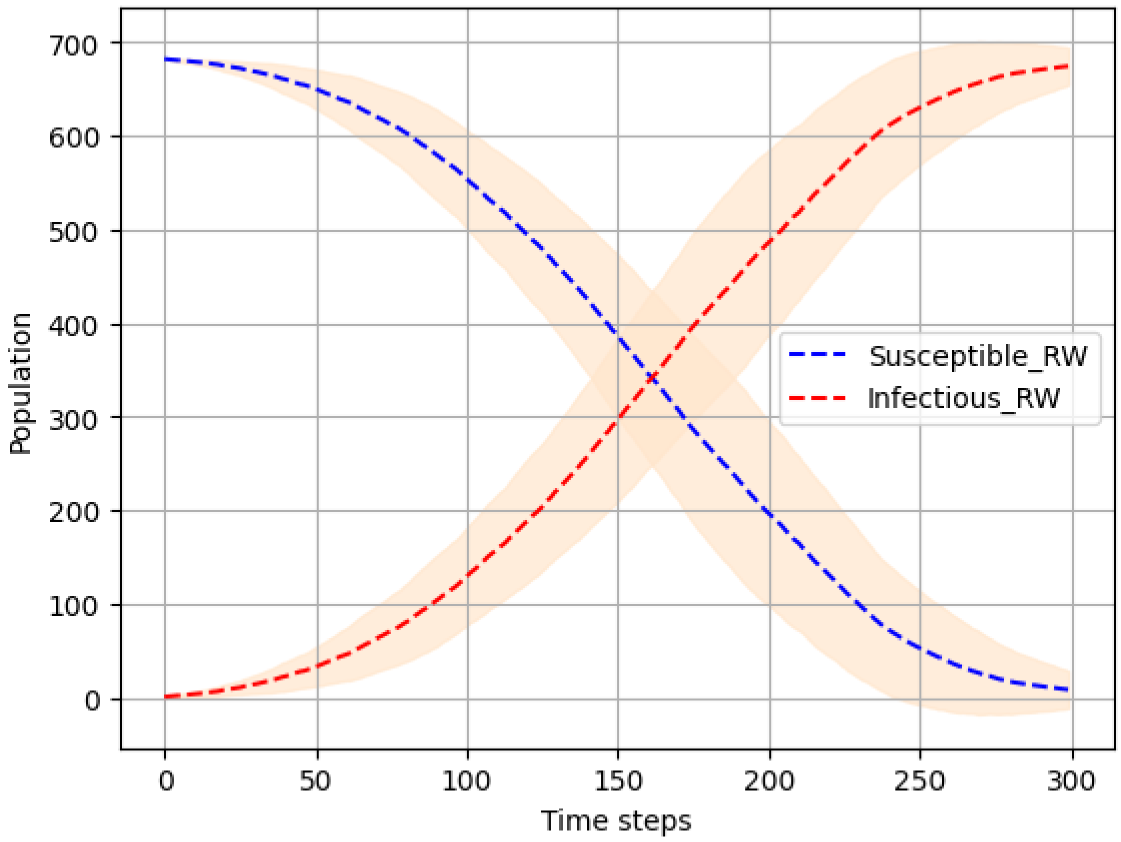
Plot 3 - t=1
After reverting the illegal moves, we can see that all walkers are back in the designated grid.

Escape is not possible.

Part 2

In this part, the number of initially infectious individuals is set to be 1. Then the model is run 100 times and each run consists of 300 time steps. After all the runs is done, the mean and standard deviation of these for both susceptible and infectious people are calculated. Dotted lines represent mean values and colored area represents plus/minus one standard deviation.

In [ ]:
```
RandomlyViral(no_init_infected=1).plot_simulation()
```



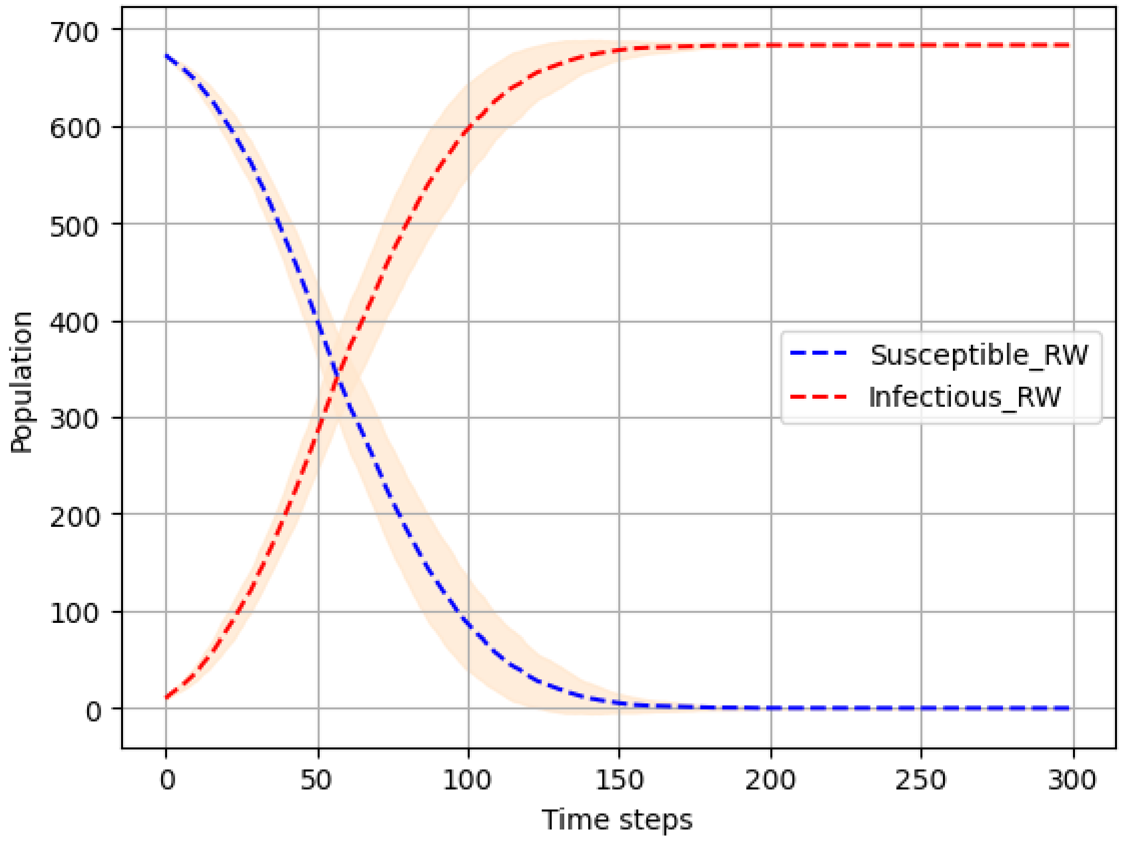
Observations:

- More time steps would be needed in order to see the curves fully flatten out. We expect everyone to end up infected given enough time since there is no way of recovering.
- Standard deviation is small at the beginning, increases to a maximum at around 75% infected population, then decreases back to zero after all variations (within 1 std) have reached 100% infection.
- Around time step 250 subtracting the standard deviation from the mean yields a negative number. This does not mean that some simulations actually showed a negative number. The standard devaition just happens to surpass the mean number of susceptible people at this point. When the standard deviation is subtracted from the mean at this point, we get the apparent "negative" amount of susceptible peolpe. The same effect (but opposite) can be observed for the infected curve, though it is less obvious since there is no hard line indicating the total population.

Part 3

In this part, the number of initially infectious individuals is set to be 10, and run it again. The reason why there is no parameters in the call is because 10 is the default number of initially infected.

In [ ]:
```
RandomlyViral().plot_simulation()
```



Observations:

- Spread is a lot quicker when starting with 10 infected individuals, and standard deviation is less. Both make intuitive sense - a quicker and less random initial spread when starting off with more infected.
- On closer inspection we can see that the spread when startin with 10 infected is a lot faster than for one. Time taken from I=10 to I~=675 (almost full infection) is here about 150 time steps. In the first case, from I=1 to I~=675, the process took about 300 steps. In the first case, from I=1 to I~=10, took approximately 30 days. This means that in the first scenario the time from I=10 to I~=675 is about 270, still almost double of second case. It seems reasonable to think that this difference is related to the fact that in the second case the initial 10 infected walkers are all randomly placed on the grid. In the first case, after reaching I=10, all infected walkers are still relatively close together, and don't interact with as many susceptible people as if they had been spread around the grid.

In our modern society with plenty of fast modes of travel, individuals are not limited to only moving at a constant rate.

Exercise 2: Compare random walk and ODE-based models

In this exercise, we are going to compare the RWS with Classical Compartment Model based on ODE solution.

SI-model based on ODE can be described according to the equations (1) and (2):

$$\frac{dS(t)}{dt} = -\beta(t).\frac{S(t)I(t)}{N} \tag{1}$$

$$\frac{dI(t)}{dt} = \beta(t).\frac{S(t)I(t)}{N} \tag{2}$$

Where:

S(t) is the number of Susceptible people

I(t) is the number of Infectious people

N = S(t)+I(t) is the total population size that in this project equals to 683

β is a constant reflecting the transferability of the disease.

The rate for being infected of a susceptible individual can be calculated by : $\beta(t).\frac{I(t)}{N}$.

In this exercise, we are assuming that β is constant so the solution is equal to equation(3). This assumption is done only for simplification of the project, otherwise in reality β can not be constant since its related to various parameters range from political ideas to people's beliefs:

$$I(t) = \frac{N}{1+\frac{s_0}{I_0}exp-(\beta*t)} \tag{3}$$

Part 1

In this part, the requested parameter is β. To calculate β, we rearrange equation(1). Then equation(4) will become:

$$\beta_n.\Delta t = -\frac{(S_n - S_{n-1})N}{S_n I_n} \tag{4}$$

The value of $\beta.\Delta t$ is calculated for each time step and averaged to return a single constant.

```
In [ ]: RandomlyViral(time_steps=110).print_beta()
```

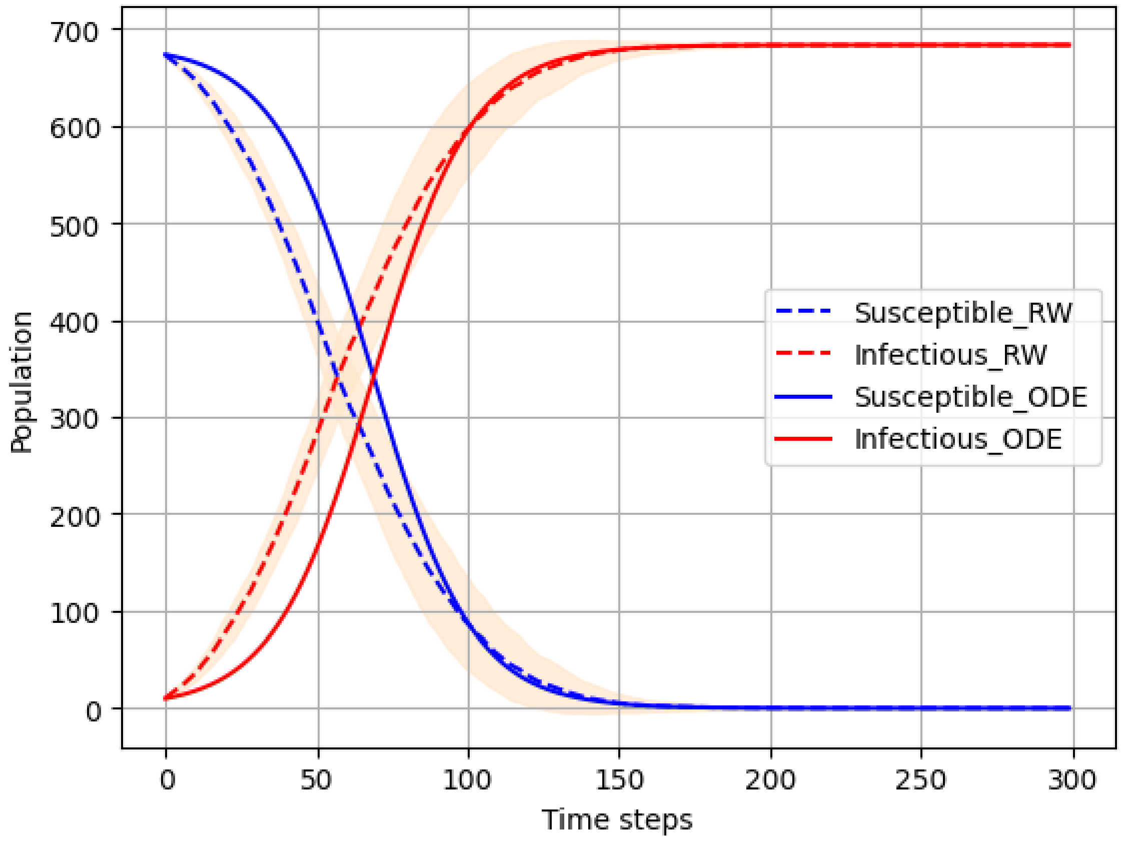
```
0.061998326058314984
```

To avoid dividing by zero in equation(4), the number of time steps had to be reduced (from 300 to 110), as $S_n = 0$ after all people have been infected. Calculating the inverse function does also not work since after everyone have been infected $S_n - S_{n-1}$ will also be zero.

If a condition was put in the calculation function to return zero if encountering the division by zero error, that would affect the mean we are trying to calculate.

_ _Calculation of analytical SI-model and its comparison with RWS model is done in next part*

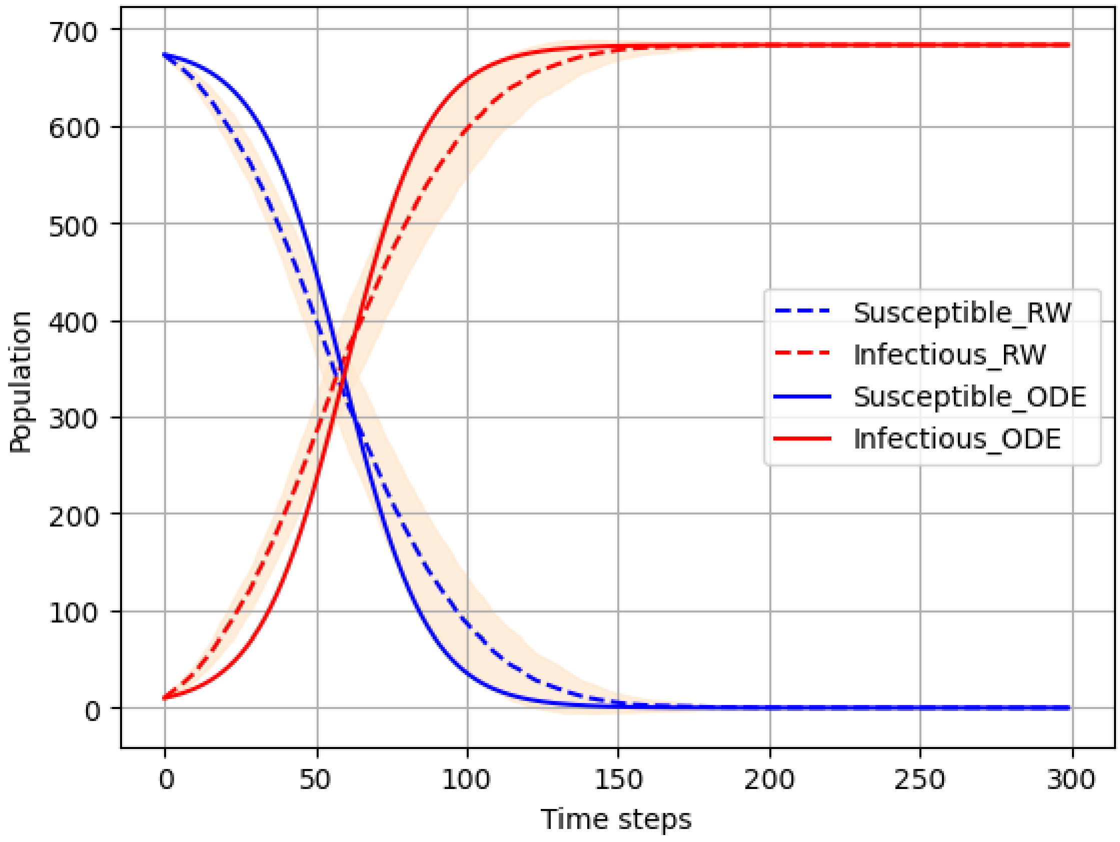
In [ ]:
```
RandomlyViral(model = 'ODE_SI', beta=0.0612).plot_simulation()
```



Given that beta was calculated from the mean values, it makes sense that the fit is best toward the end, where the mean has averaged out.

Below, we will try to maually fit beta to see if we can make it closer overall.

In [ ]:
```
RandomlyViral(model = 'ODE_SI', beta=0.071).plot_simulation()
```



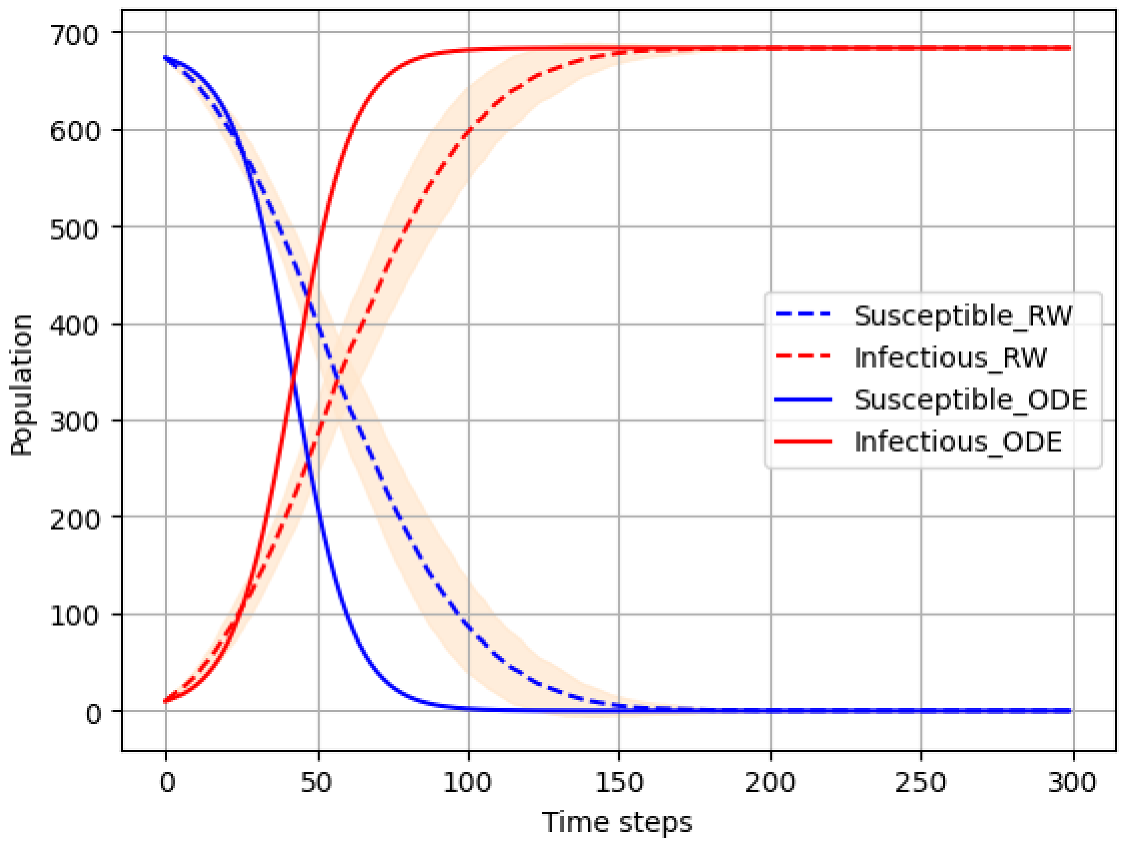
For beta = 0.071, we get a plot where all the lines intersect at the middle and the plot looks really nice and symmetrical. For matching the ODE model to real data, it would be more appropriate to focus on a short section of time, or allow for a variable beta through time, or depending on the infection somehow.

If the random walk output represents the "real data" and we were most interested in matching the early part of the development, beta could be calculated for the time interval we are interested in. Example below shows a calculation of beta based on first 20 time steps and corresponding plot:

```
In [ ]: RandomlyViral(time_steps=20).print_beta()
```

```
0.0980616534670149
```

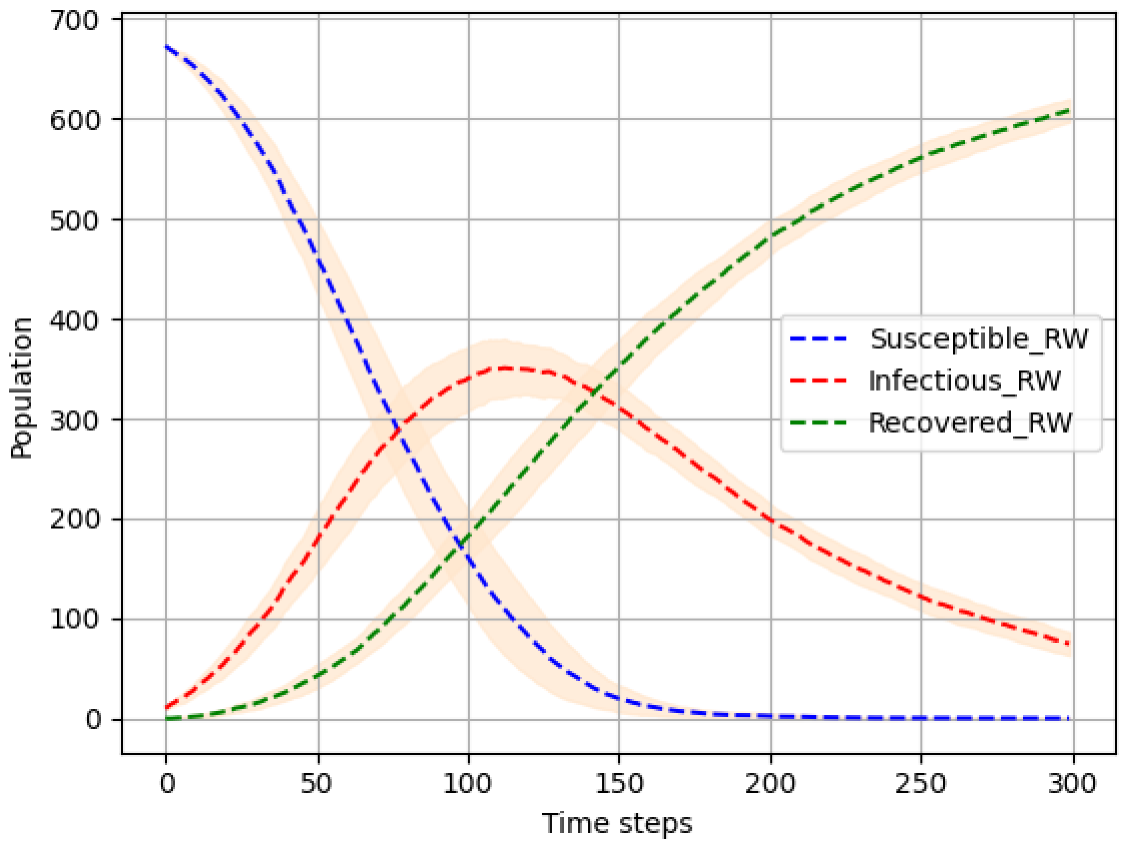
```
In [ ]: RandomlyViral(model = 'ODE_SI', beta=0.1).plot_simulation()
```



Part 2

In this part the SI-RWS is extended to become a SIR-RWS. It means that effect of **Recovery** from the disease is added to the model. The mechanism of recovery is introduced by making a small chance for each infected person to recover, each time step.

In [ ]:
```
RandomlyViral(Recovery = 'on').plot_simulation()
```



The main comment to this plot is recognising how easy it was to add this mechanism. The framework of making Monte Carlo checks for transferring walkers from one "State" to another seems very robust.

Part 3

In this part, the classical compartment model based on ODE is extended to SIR-model. Equation(1) for infectious is still acceptable for this model. On the other hand, susceptible and recovery states has to be changed as we can see in equations(5) and equ(6) respectively.

One term is added which is τ_{sick}, which refers to the average time a person is infected before recovering.

$$\frac{dI(t)}{dt} = \beta(t).\frac{S(t)I(t)}{N} - \frac{1}{\tau_{sick}}.I(t) \tag{5}$$

$$\frac{dR(t)}{dt} = \frac{1}{\tau_{sick}}.I(t) \tag{6}$$

Result of implementation is shown in Part 4 after finding values for β, and τ_{sick}.

Part 4

Like in Exercise 2 - Part 1, in this exercise we calculate β and one more parameter which is τ_{sick}. Then, we used these values in ODE-model.

```
In [ ]: RandomlyViral(Recovery = 'on',time_steps=110).print_beta()
```

```
0.0626927436734449
```

To calculate τ_{sick} we combine equation (6) with a first order approximation of the derivative to yield:

$$\frac{\tau_{sick}}{dt} = -\frac{I_n}{(R_n - R_{n-1})} \tag{7}$$

While calculating τ_{sick} we ran into a different divide-by-zero-error. Due to the low recovery rate, each time step does not guarantee that the number of recoveries is greater than zero. To circumnavigate this issue we calculate the invers of each single value before calculating the mean and inverting back.

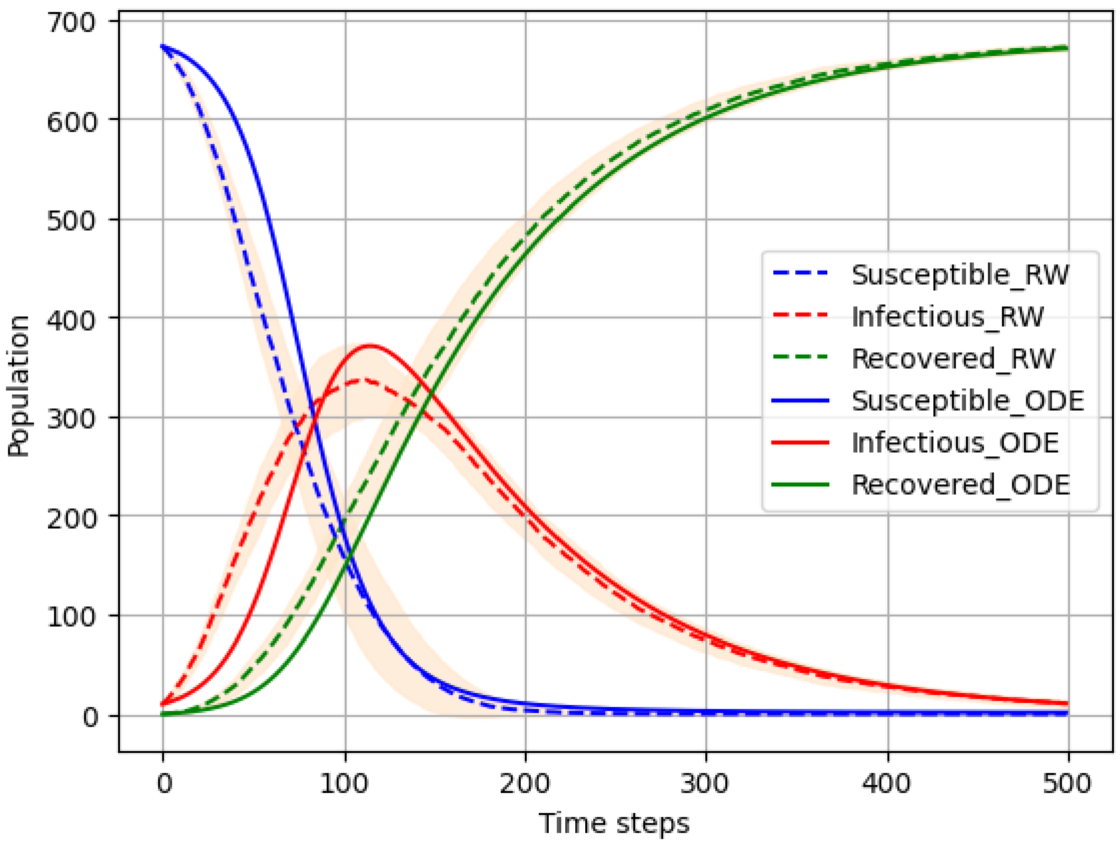
```
In [ ]: RandomlyViral(Recovery = 'on').print_tau()
```

```
97.70749994473935
```

The value for τ_{sick} makes sense seeing that our chance of recovery per time step is 1%. Based on this we can imagine that it will take around 100 days to recover.

Plotting comparisons of ODE and RW results for SIR-model:

```
In [ ]: RandomlyViral(model = 'ODE_SIR', Recovery = 'on', beta = 0.0627, tau = 97.7, tim
```



Observations:

- Considering how differnt the two sets of curves are generated, it is quite intriguing how well they match with only calibrating two constants.

Exercise 3: Implement your own scenario

In this task we could implement anything, but finally decided to add infection waves, removing the border around the grid & finally tried to fit it to some real world data. Quite notably, this fitting to real world data, will only be using infections of susceptible people, and recovery of infected people.

The data used was found at https://github.com/owid/covid-19-data (4)

First, the simplest change is made, which is just to remove the border around the grid which limits any movement that would go outside the grid to be a skipped movement instead.

The motivation to do this, is that when there is a border, there is always a clumping effect at the border, basically from

The changed version, will let it move directly over the the opposite side.

Example:

pos1 = $[x,y]$ = $[0, 0]$

step = $[-1,0]$

previously:

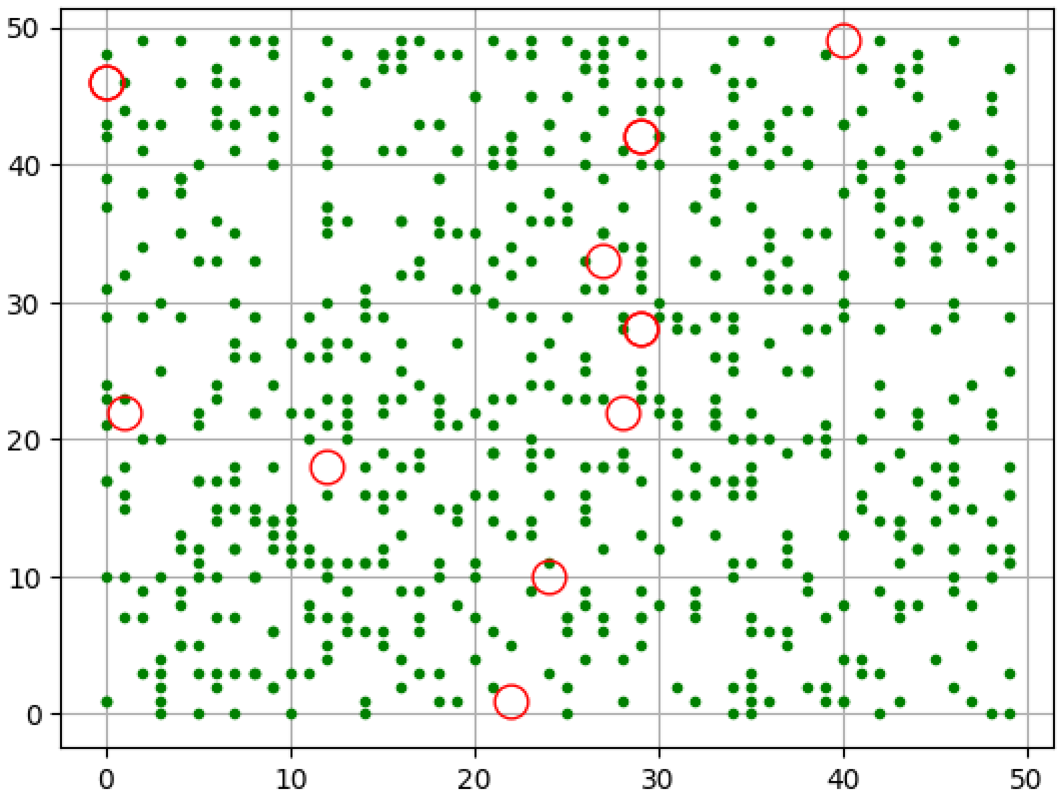
pos2 = $[0-1=0, 0+0=0]$

with borderless mode:

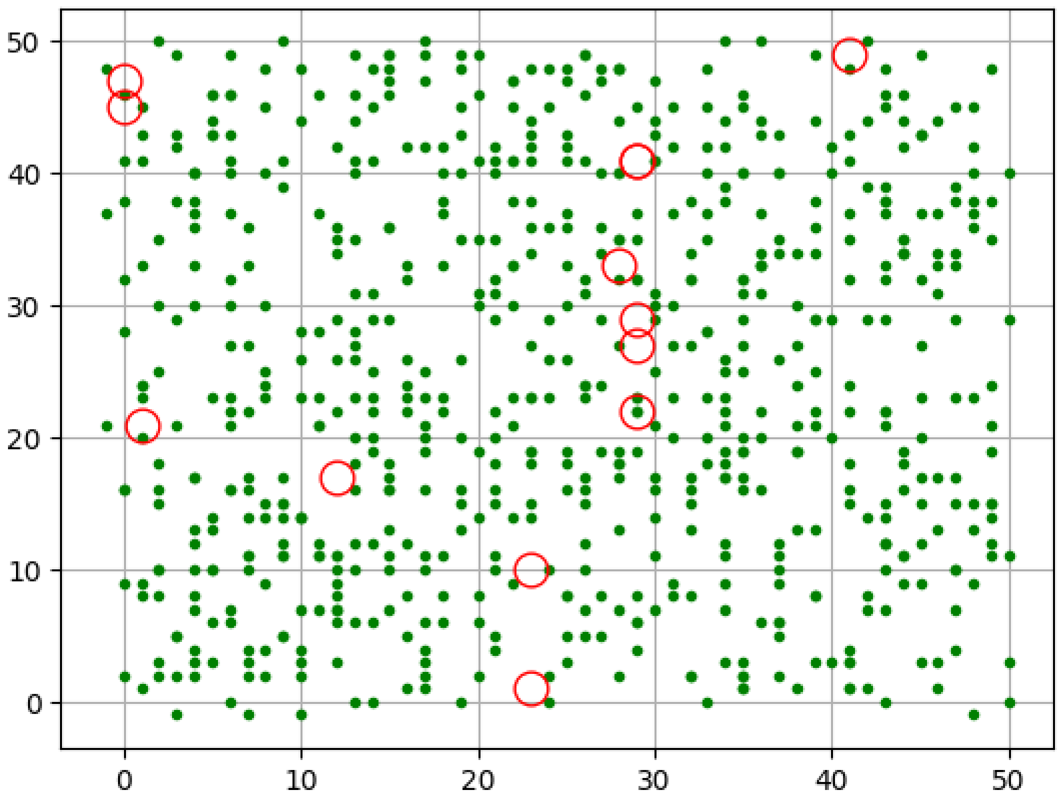
pos2 = $[0-1=x_{\max}, 0+0=0]$

Below is the code used earlier, which show the step and corrections to steps outside of the grid.

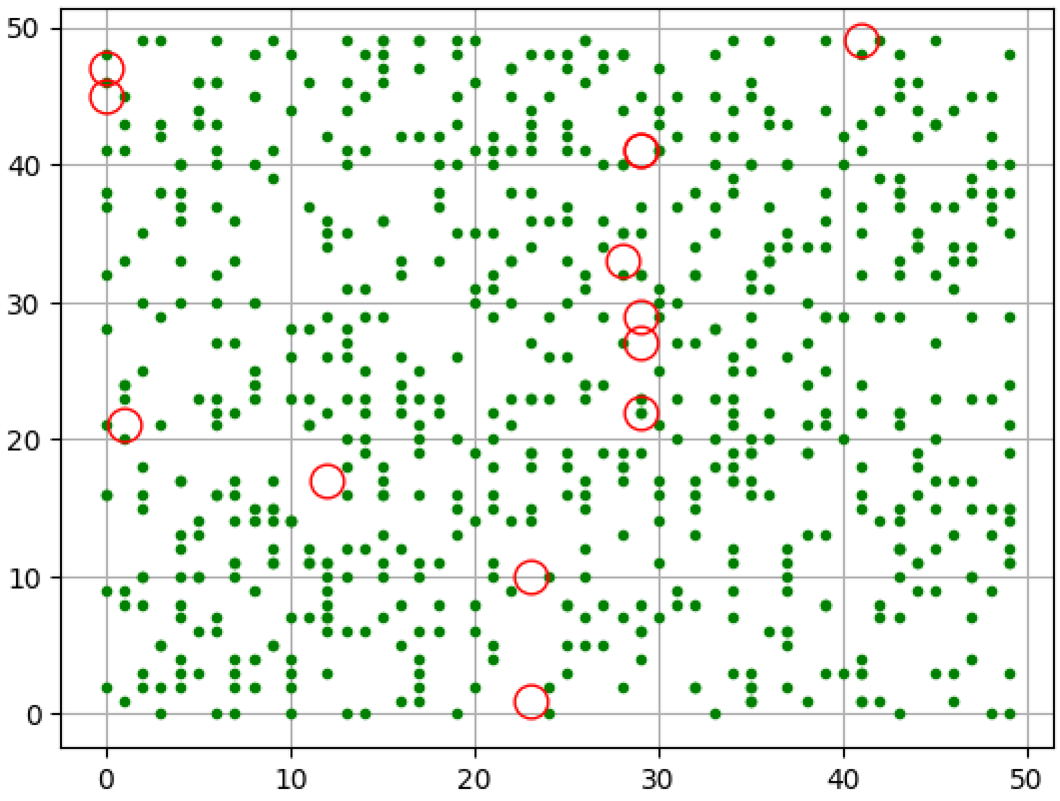
```
In [ ]: seed = 9  # specifically chosen since a step outside the grid happens
        RandomlyViral(Recovery='on',
                      random_seed=seed).single_simulation()
        RandomlyViral(Recovery='on',
                      random_seed=seed, no_border=True).single_simulation()
```

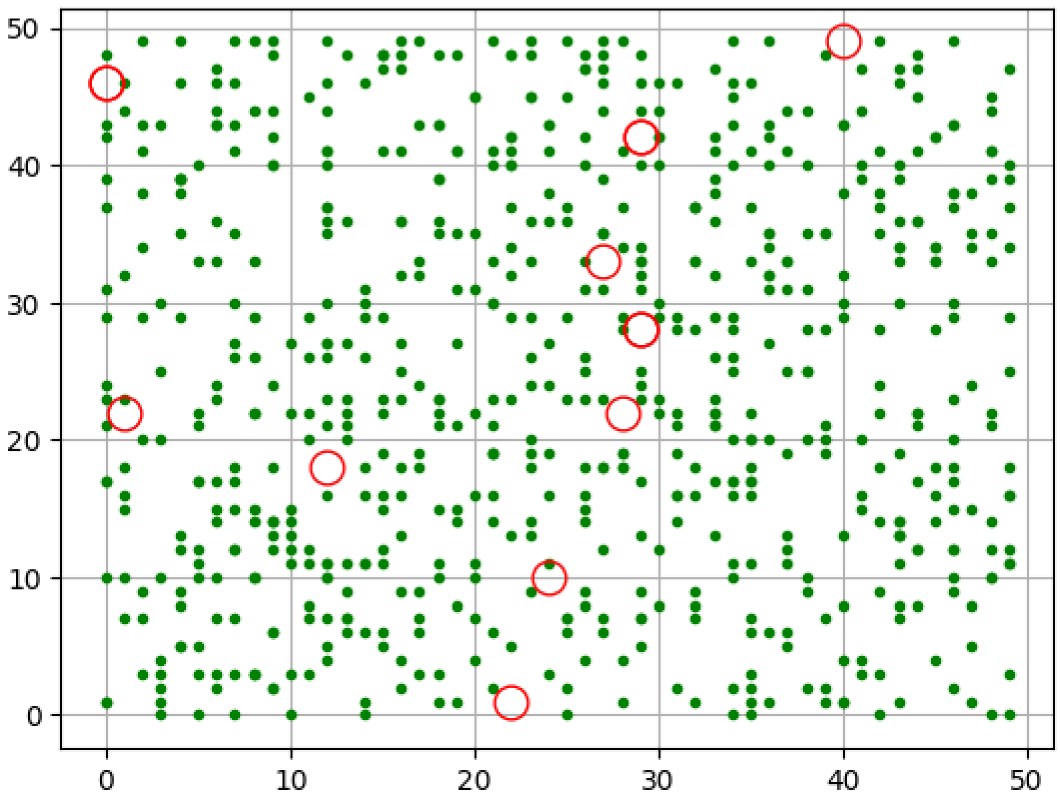


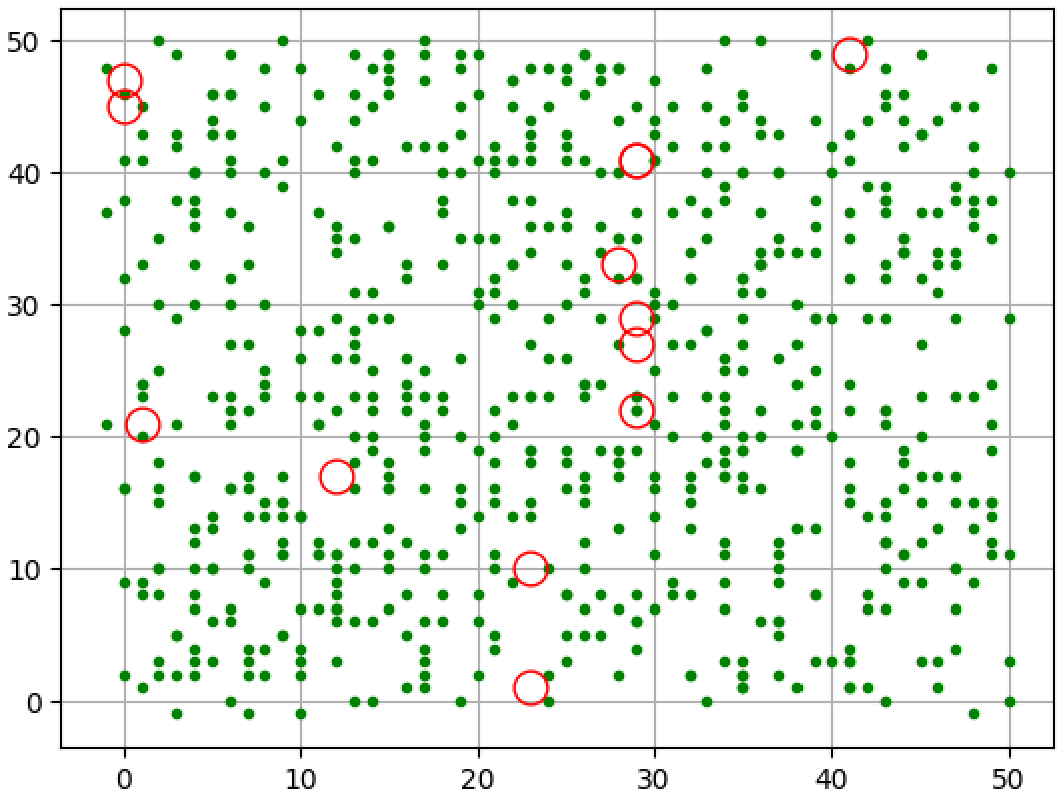
50
40
30
20
10
0
0
10
20
30
40
50

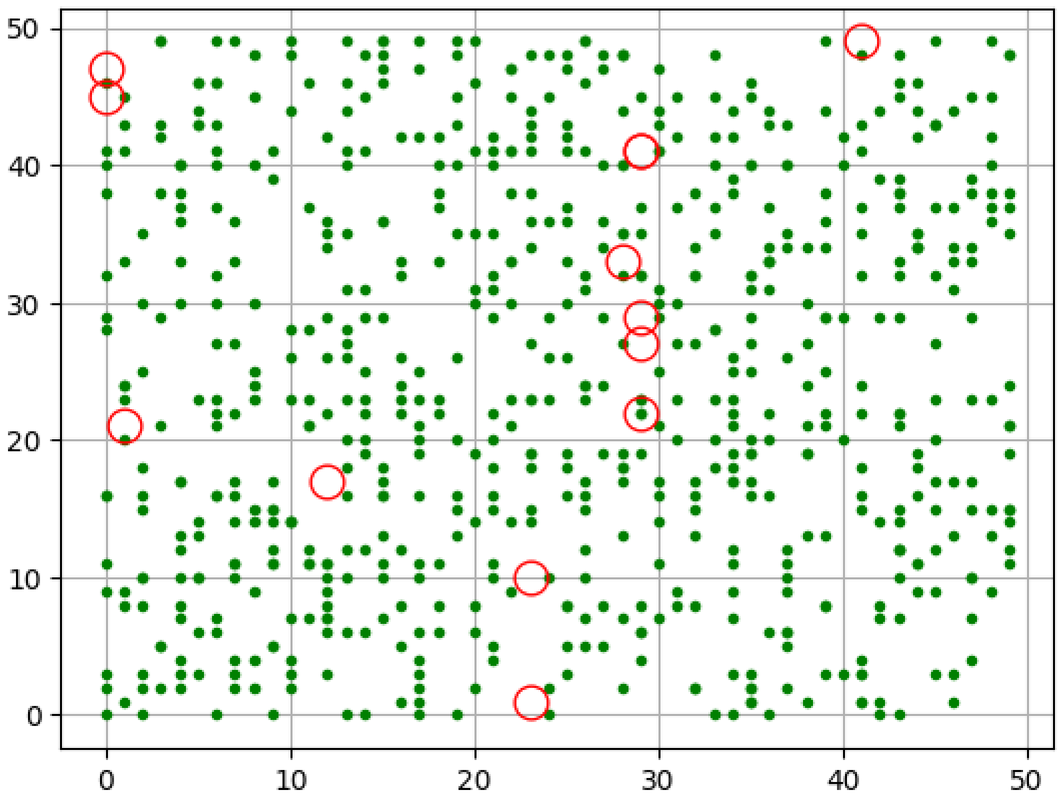


50
40
30
20
10
0
0
10
20
30
40
50


0
10
20
30
40
50
0
10
20
30
40
50


0
10
20
30
40
50
0
10
20
30
40
50



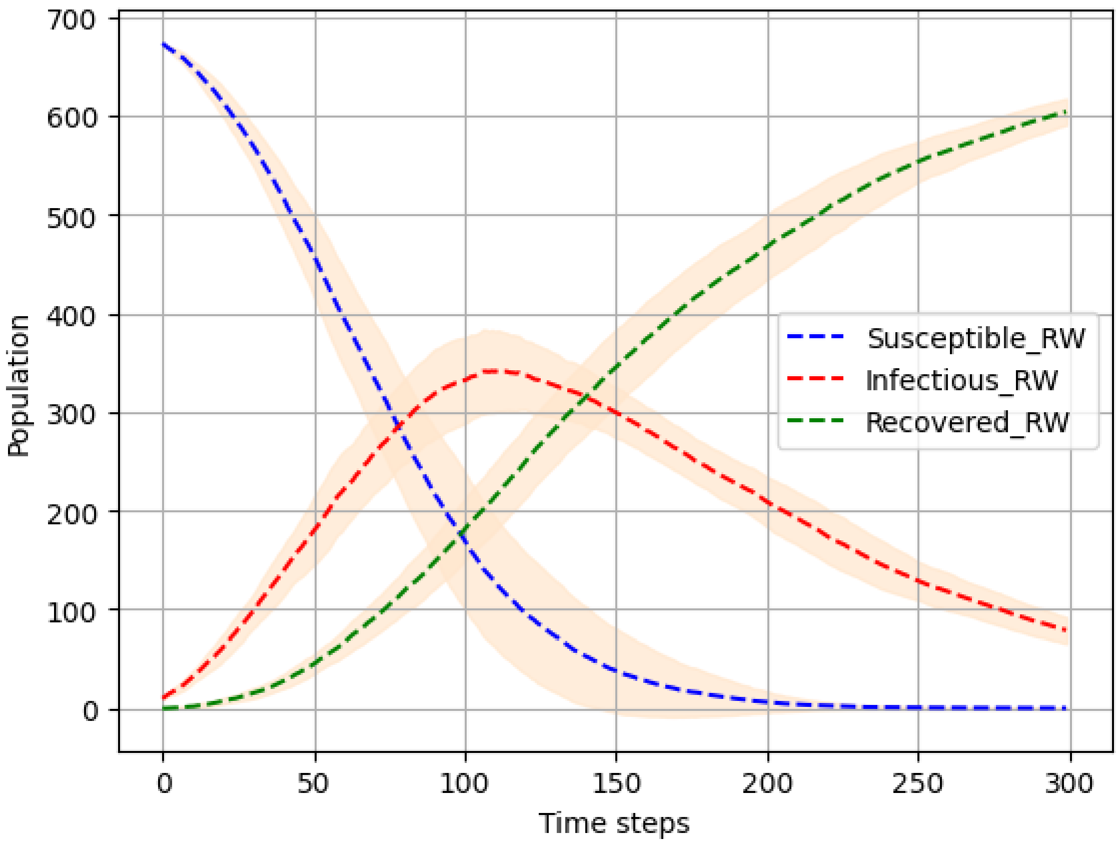


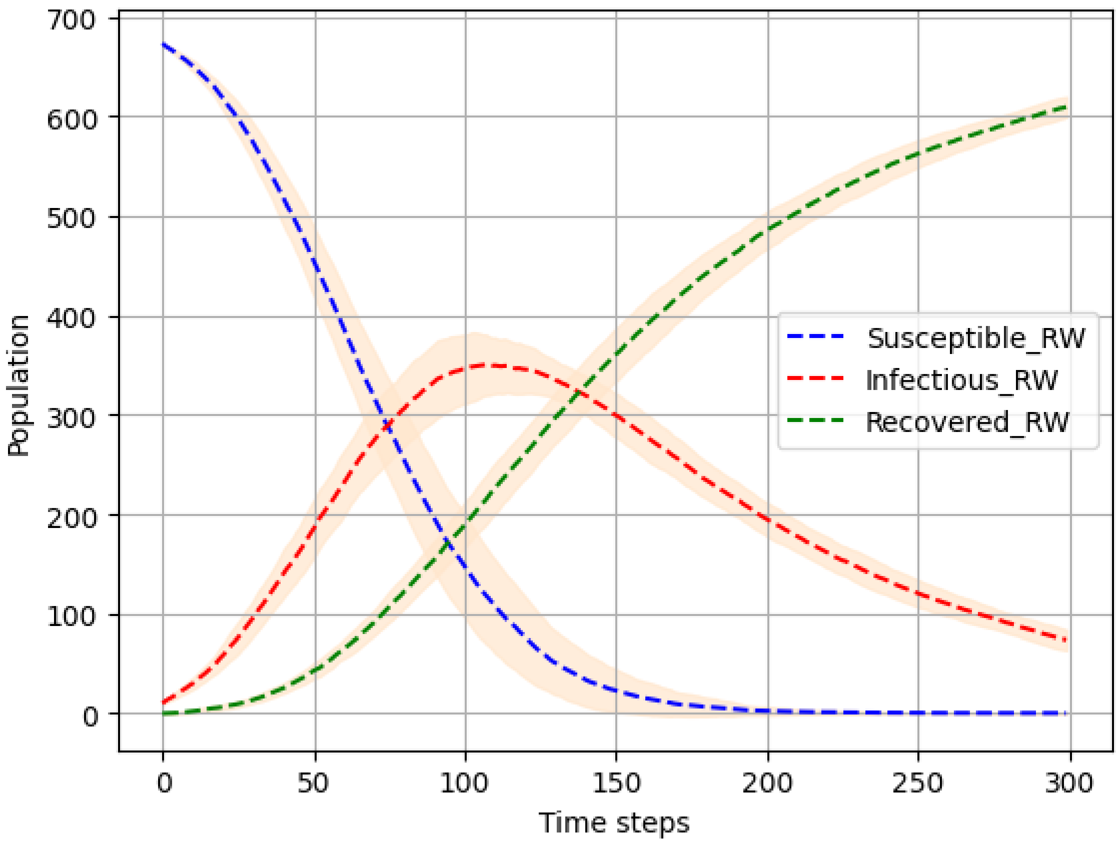
The point of interest here, is the point moving outside the grid at $[x, y] \approx [0, 46]$

Now a actual simulation that uses this change compared to one without, below:

```
In [ ]: iterations = 25  # less computing power
        RandomlyViral(
            iterations=iterations,
            Recovery='on',
            random_seed=seed).plot_simulation()
```

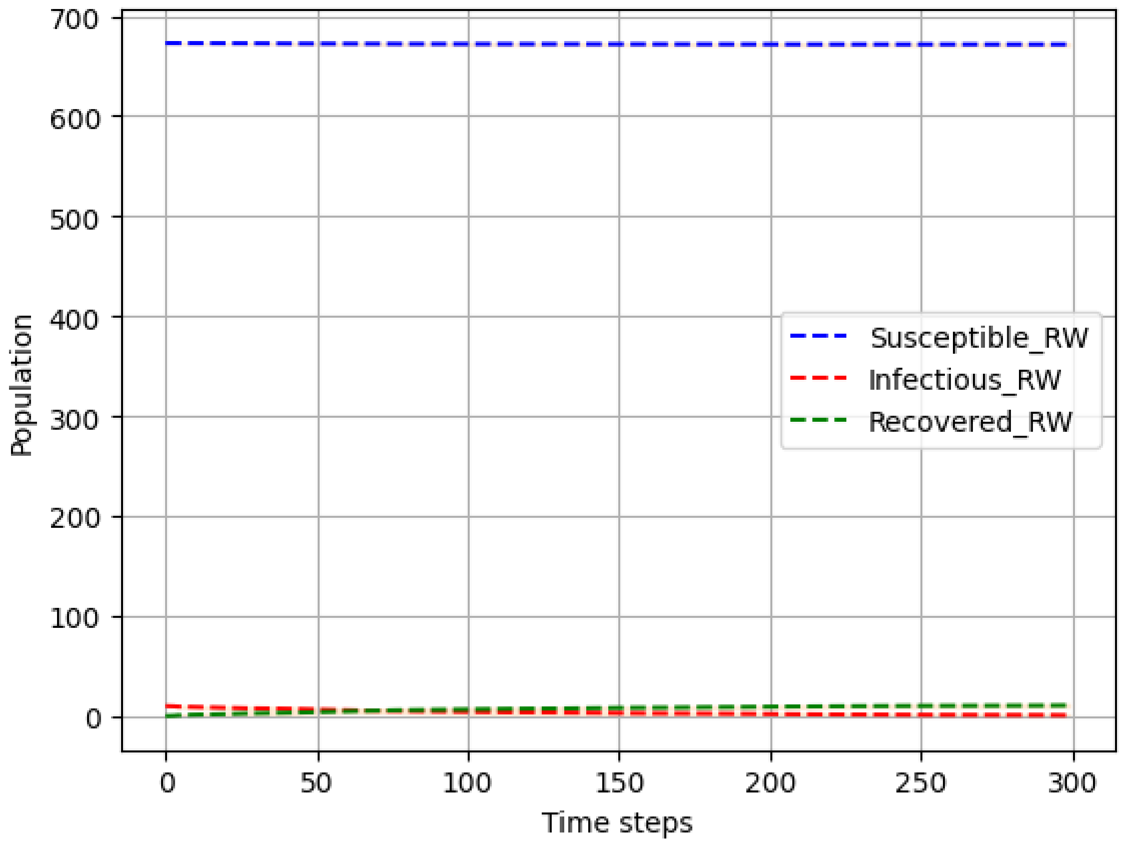
```
RandomlyViral(
    iterations=iterations,
    Recovery='on',
    no_border=True,
    random_seed=seed).plot_simulation()
```

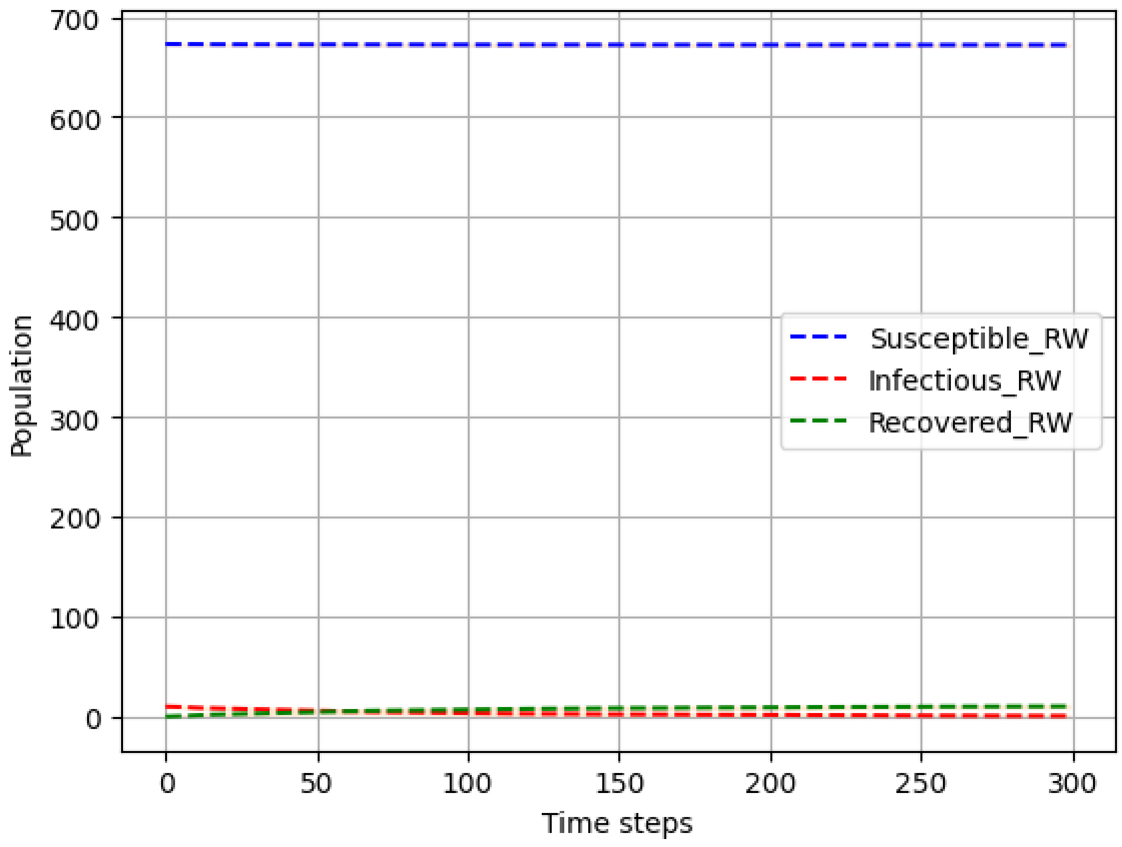




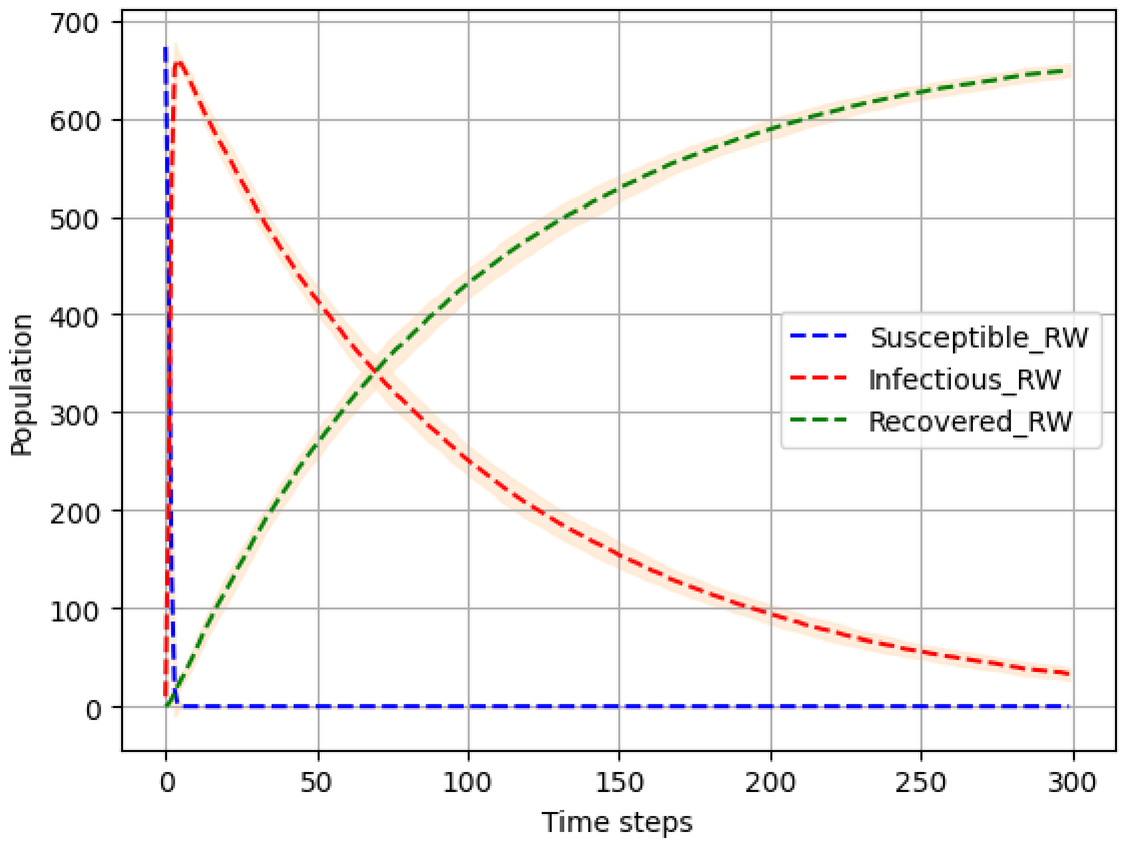
The result is basically a negligable difference. Then to confirm it is always the case, a bigger and smaller grid is used in the same way as above.

In [ ]:
```
# The following is a $10$ times bigger grid:
time_steps = 300
RandomlyViral(
    iterations=iterations,
    nx=500,
    ny=500,
    Recovery='on',
    random_seed=seed).plot_simulation()
RandomlyViral(
    iterations=iterations,
    nx=500,
    ny=500, Recovery='on',
    no_border=True,
    random_seed=seed).plot_simulation()

# The following is a $10$ times smaller grid:
RandomlyViral(
    iterations=iterations,
    nx=5,
    ny=5,
    Recovery='on',
    random_seed=seed).plot_simulation()
RandomlyViral(
    iterations=iterations,
    nx=5,
    ny=5, Recovery='on',
    no_border=True,
    random_seed=seed).plot_simulation()
```

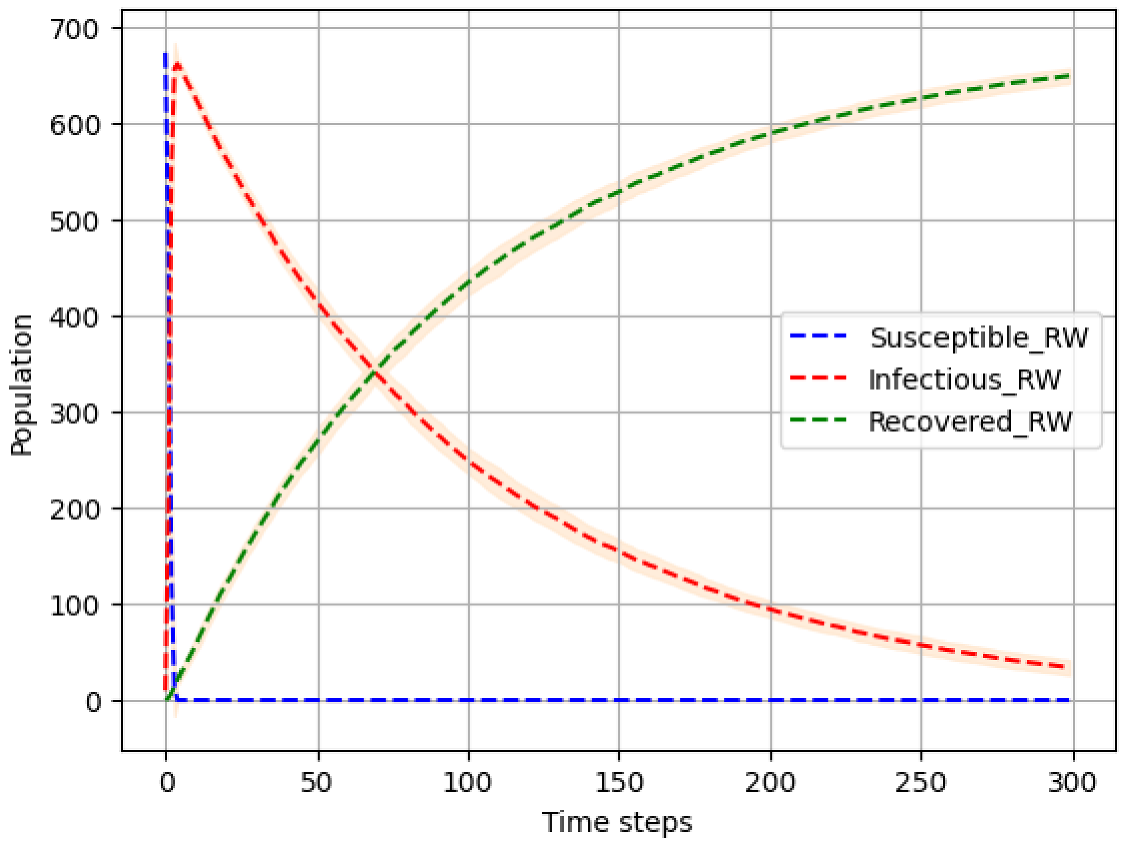




Population
Time steps
Susceptible_RW
Infectious_RW
Recovered_RW
0
100
200
300
400
500
600
700
0
50
100
150
200
250
300



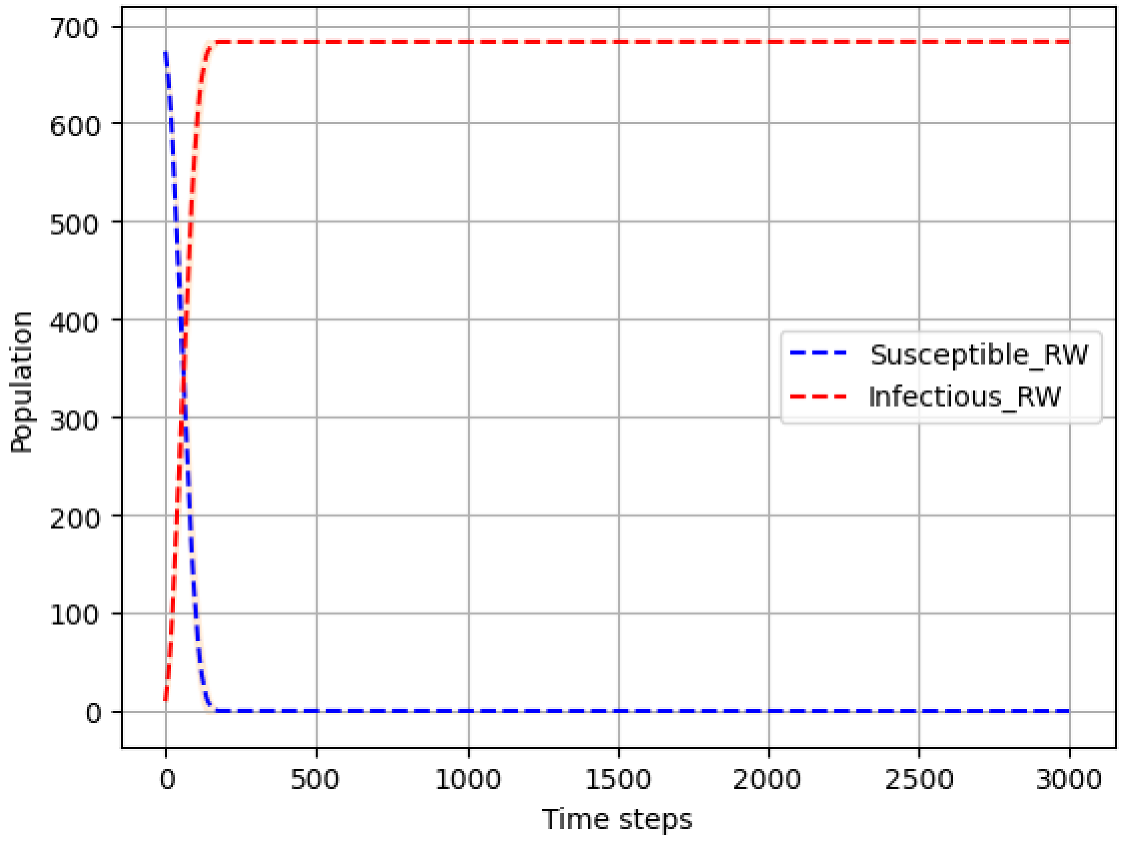
Population
Time steps
Susceptible_RW
Infectious_RW
Recovered_RW
0
100
200
300
400
500
600
700
0
50
100
150
200
250
300

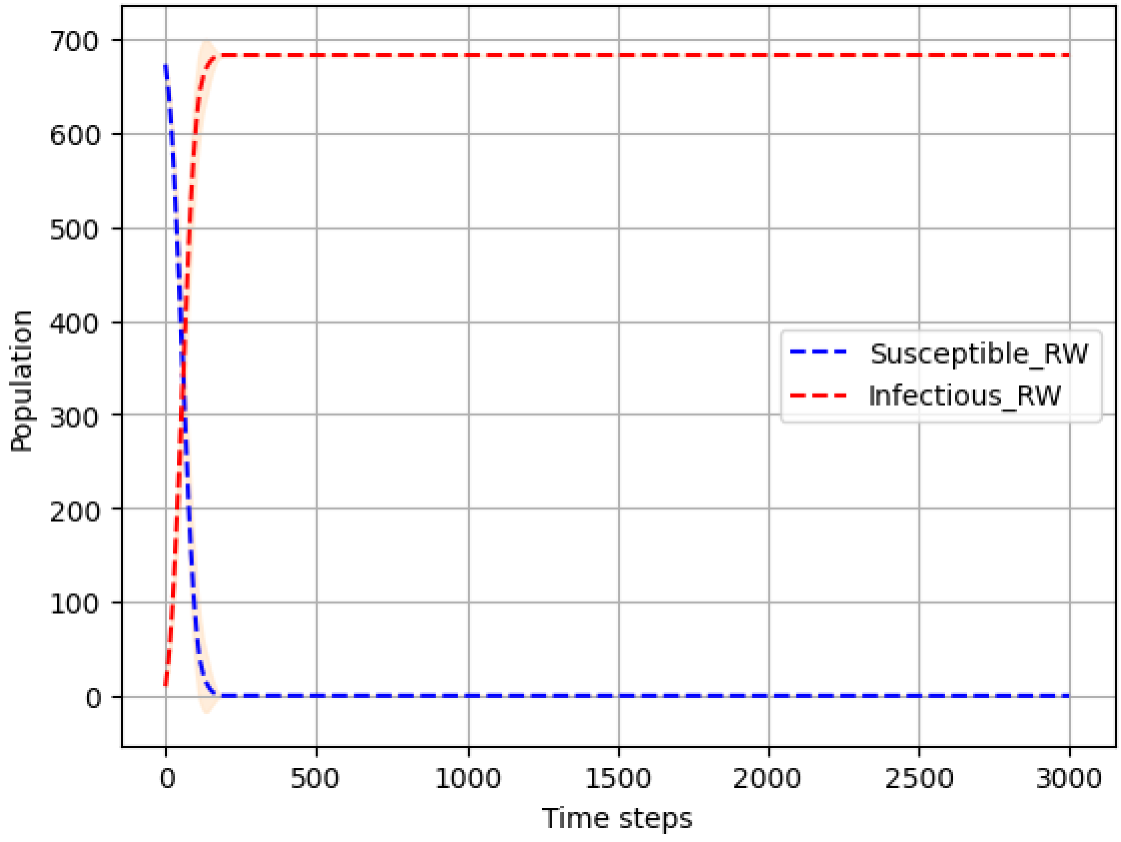


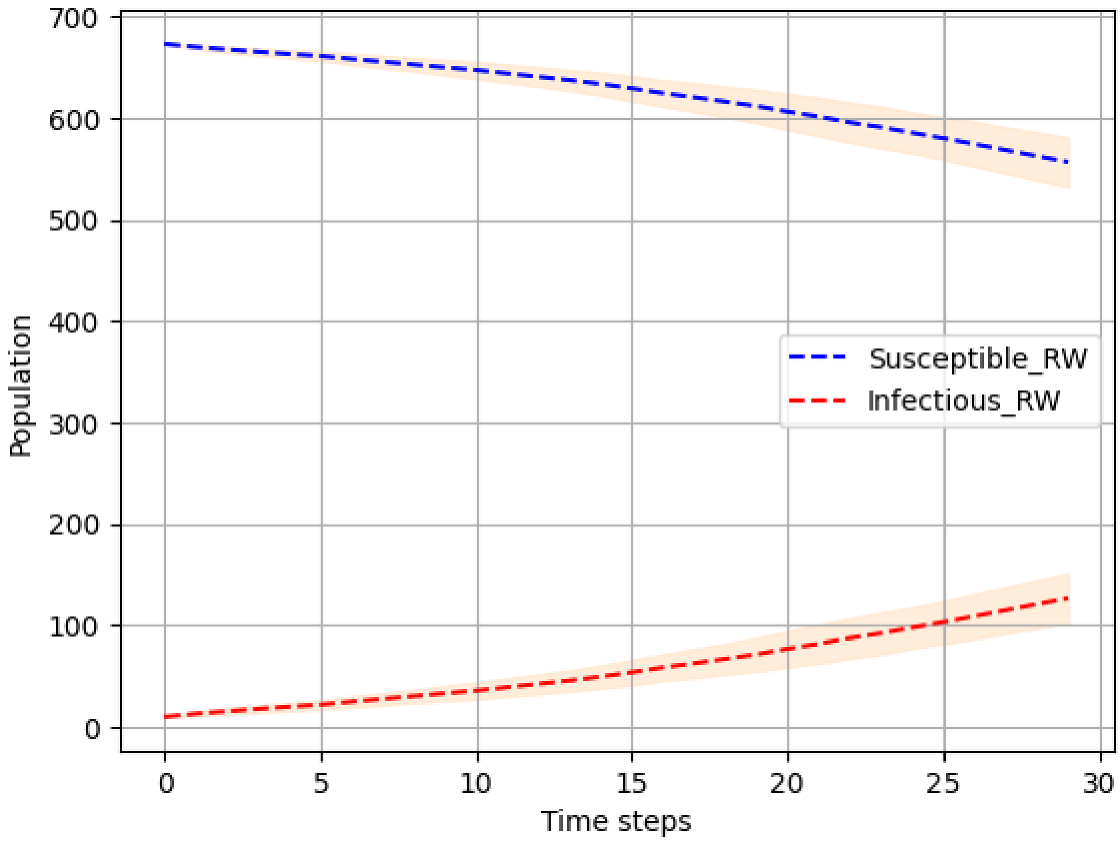
The result is basically a negligable difference, again. Even though this result is incredibly boring, it basically means that the two reactions we expected are practically equally effective in their own right.

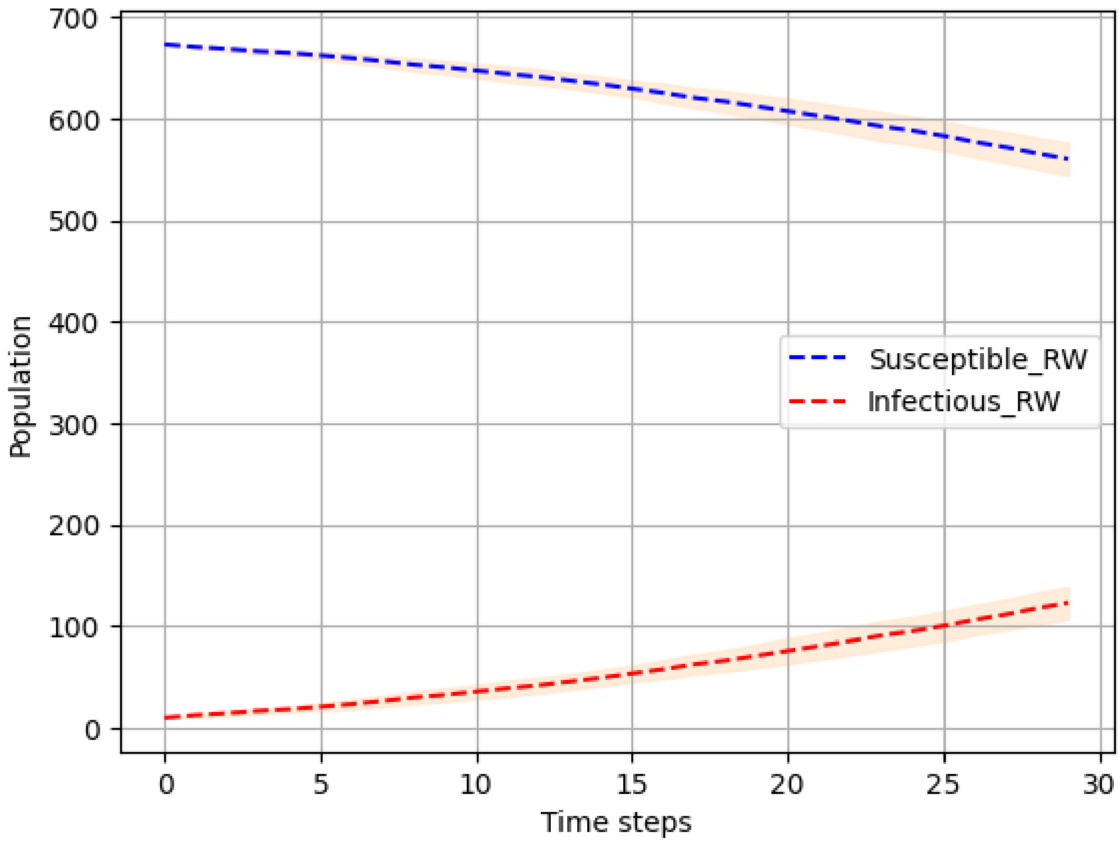
To explore this just a bit more, a change in number of movements is also tested. This is because this gives more time for the expected clumping behaviour to happen in the border case.

```
In [ ]: # The following is a $10$ times number of time steps:
time_steps = 3000
RandomlyViral(
    time_steps=time_steps,
    random_seed=seed).plot_simulation()
RandomlyViral(
    time_steps=time_steps,
    no_border=True,
    random_seed=seed).plot_simulation()

# The following is a $10$ times smaller number of time steps
time_steps = 30
RandomlyViral(
    time_steps=time_steps,
    random_seed=seed).plot_simulation()
RandomlyViral(
    time_steps=time_steps,
    no_border=True,
    random_seed=seed).plot_simulation()
```


Population
Time steps
0
100
200
300
400
500
600
700
0
500
1000
1500
2000
2500
3000
Susceptible_RW
Infectious_RW


Population
Time steps
0
100
200
300
400
500
600
700
0
500
1000
1500
2000
2500
3000
Susceptible_RW
Infectious_RW



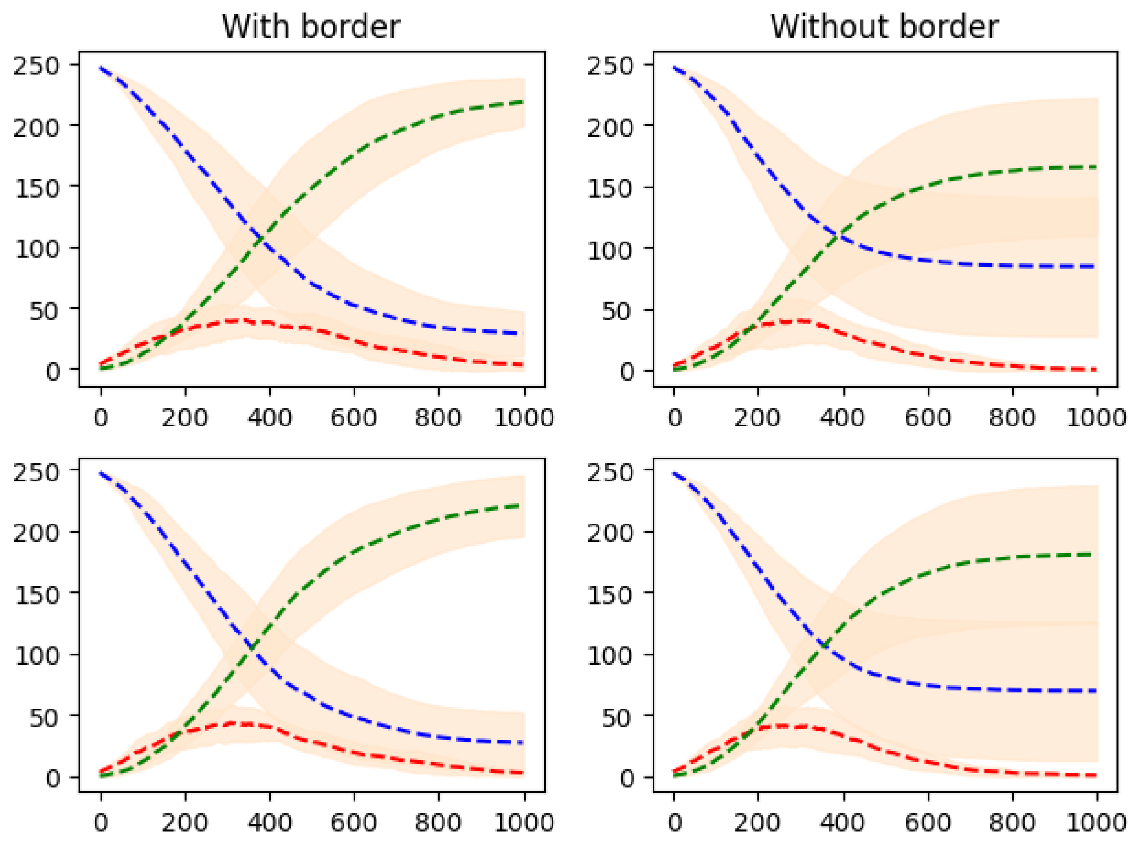


The result is negligable, again.

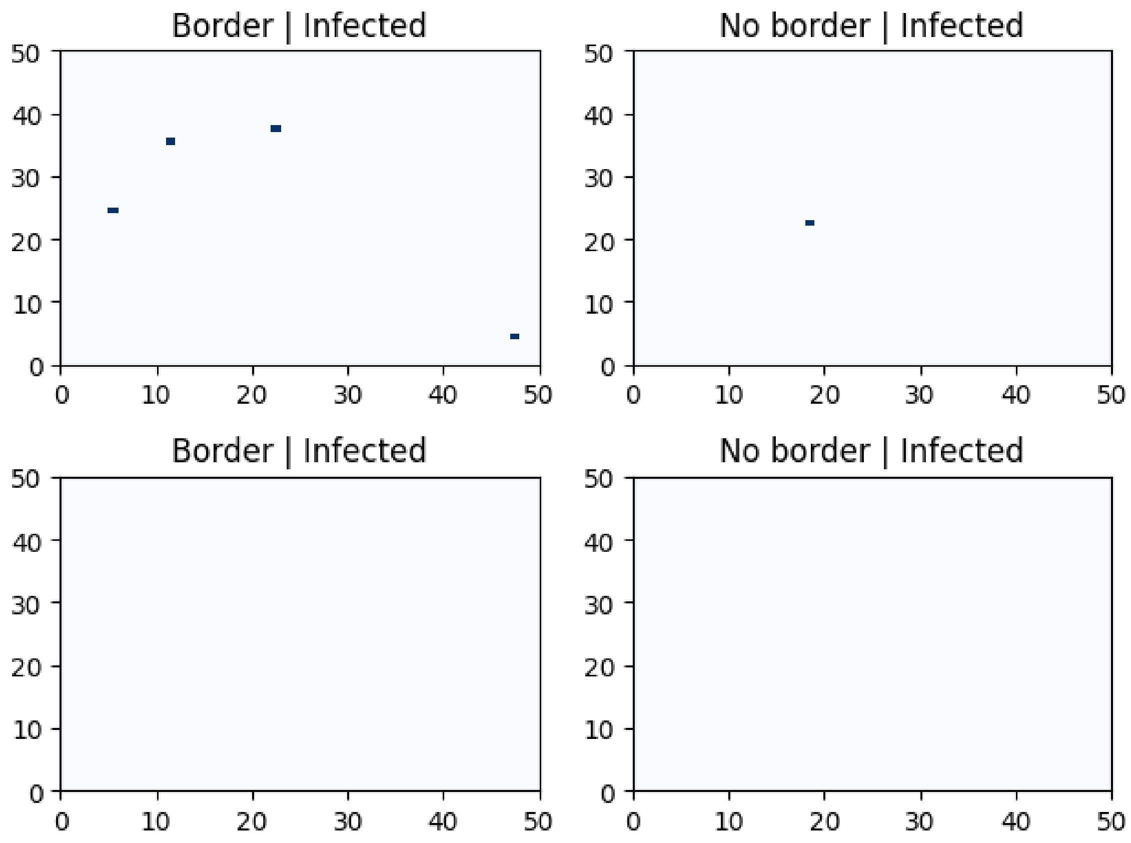
As a last test to show differences between having a border and not, only the infected is plotted in a heatmap. This is show any kind of difference specifically in who gets infected, and where they are located. The run is done with several seeds.

In [ ]:
```
size = 50
pop_size = 250  # size*3
# math.ceil(size*(300*(size/10)))  # size = time_steps/(3*200)
time_steps = 1000
no_init_infected = 3  # math.ceil(pop_size/100)
iterations = 25  # math.ceil(time_steps/pop_size)
seeds = [480, 720]  # np.arange(10*n, 10*(n+1))
recovery_probability = 1e-2
infection_probability = 85e-2
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect


numPlots = 2*len(seeds)
fig, axs = plt.subplots(int(numPlots/2), 2)
axs = axs.T.flatten()
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i])
    noBorder.plot_simulation(ax=axs[int(numPlots/2) + i])

axs[0].set_title('With border')
axs[int(numPlots/2)].set_title('Without border')
fig.tight_layout()
```

```
Current parameters:
size=50
time_steps=1000
pop_size=250
no_init_infected=3
iterations=25
seeds=[480, 720]
recovery_probability=0.01
infection_probability=0.85
```



```
In [ ]: fig, axs = plt.subplots(2, 2)
        for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
            border.plot_grid_heatmap(title="Border | Infected",
                                     numbers=False, pretty=True, state=1, ax=axs[i,0])
            noBorder.plot_grid_heatmap(
                title="No border | Infected", numbers=False, pretty=True, state=1, ax=ax

        fig.tight_layout()
```



From these results the basic and more or less only possible conclusion is:

The variability in a borderless simulation is greater, but the difference in mean value is negligable.

Practically it means that the results give us the following guidance:

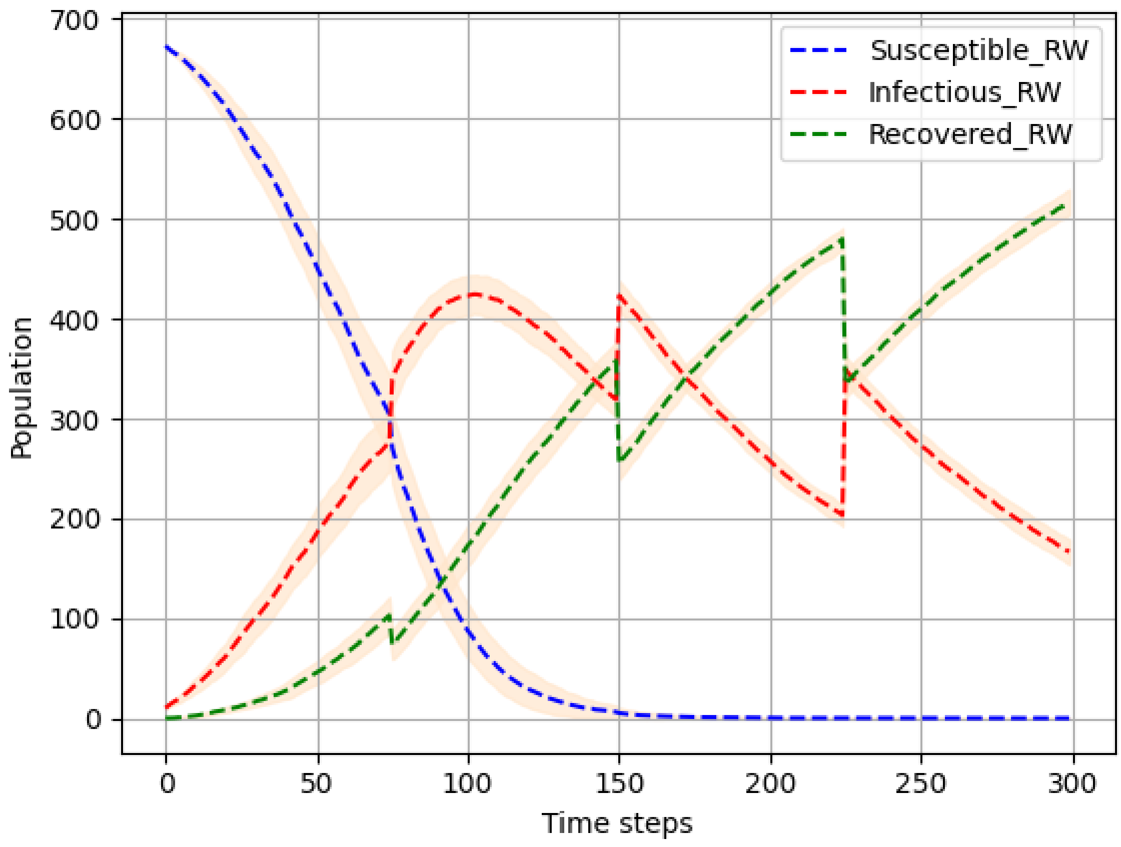
With large enough simulations, it should behave practically the same way, with or without borders, but to be computationally efficient, the simulation with border should be used since it is more stable around the "common" mean.

Following is some simulations done with the default values, and infection waves enabled.

Using 3 waves with probability of infecting;

1. Susceptible people $= 10\%$
2. Recovered people $= 30\%$

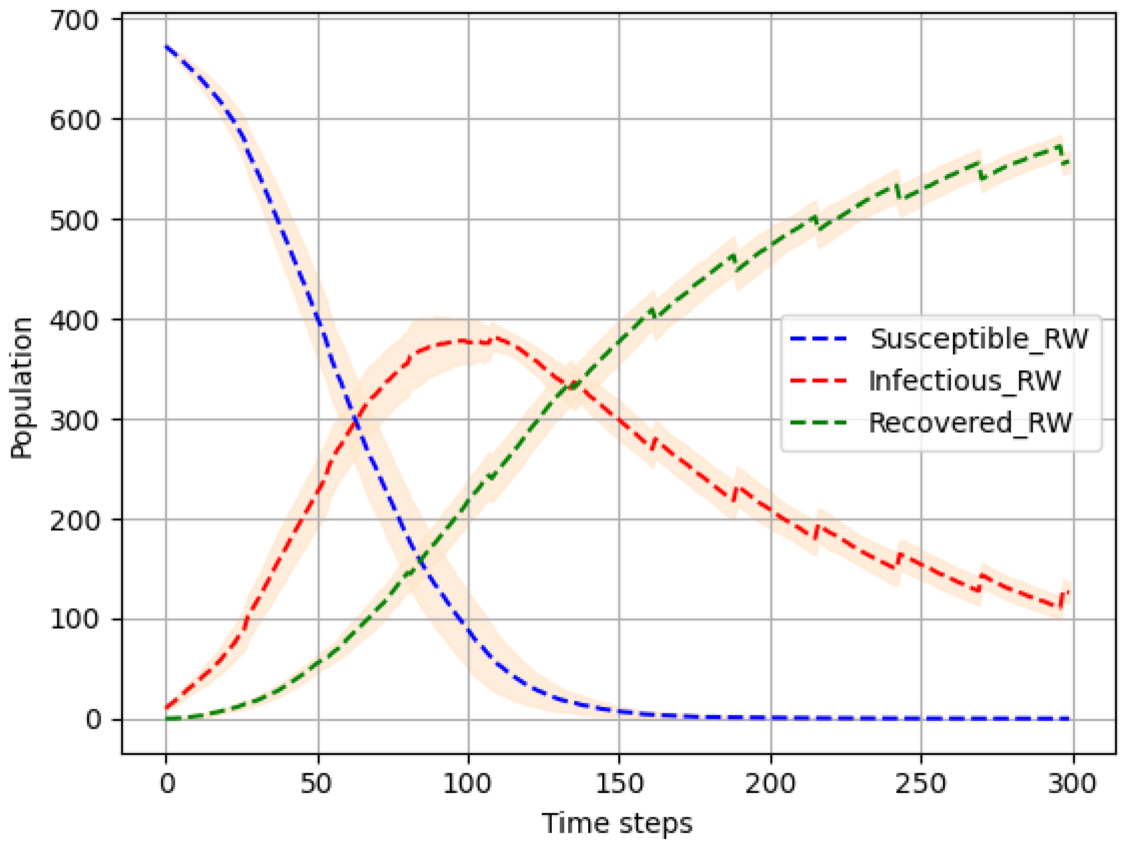
In [ ]:
```
RandomlyViral(Recovery='on', no_border=False,
              wave_susc=0.1,
              wave_reco=0.3,
              num_waves=3,
              infectionWaves=True).plot_simulation()
```



Using 10 waves with probability of infecting;

1. Susceptible people $= 1\%$
2. Recovered people $= 3\%$

```
In [ ]: RandomlyViral(Recovery='on', no_border=False,
                      wave_susc=0.01,
                      wave_reco=0.03,
                      num_waves=10,
                      infectionWaves=True).plot_simulation()
```



It works as expected, and ripples the data as expected.

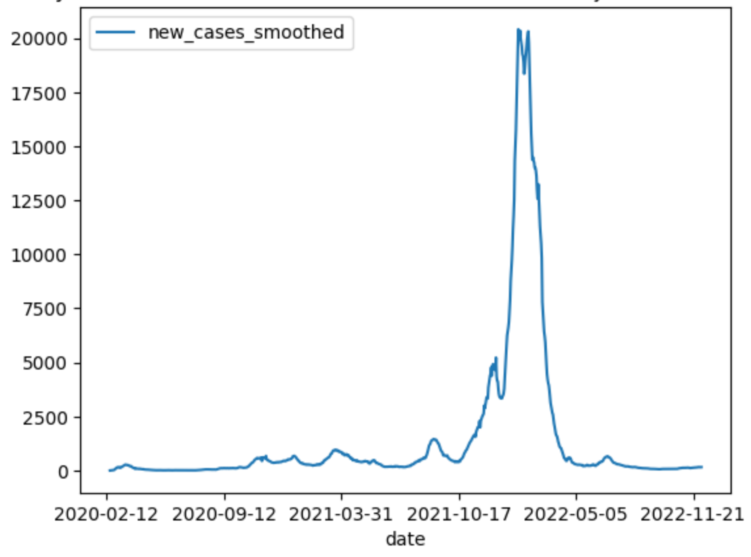
Real world data loaded and visualized below:

In [ ]:
```
cwd = os.getcwd()
cwd = cwd.replace("\\", "/")

path = f"{cwd}/data/NOR-USA-FIN_Data.csv"

df = pd.read_csv(path)
useful_data_df = df.drop(
    df.columns.difference(['iso_code',
                           'date',
                           'new_cases_smoothed',
                           'new_deaths_smoothed',
                           'new_vaccinations_smoothed',
                           'population'
                           ]), 1)
NOR_useful_data_df = useful_data_df[useful_data_df['iso_code'] == "NOR"]
FIN_useful_data_df = useful_data_df[useful_data_df['iso_code'] == "FIN"]
USA_useful_data_df = useful_data_df[useful_data_df['iso_code'] == "USA"]
#   "new_cases_smoothed"
# | "new_deaths_smoothed"
# | "new_vaccinations_smoothed"
y = "new_cases_smoothed"
NOR_useful_data_df.plot(x="date", y=y)
plt.title(f"NOR | {y} \nFrom: {path}")
FIN_useful_data_df.plot(x="date", y=y)
plt.title(f"FIN | {y} \nFrom: {path}")
USA_useful_data_df.plot(x="date", y=y)
plt.title(f"USA | {y} \nFrom: {path}")
```

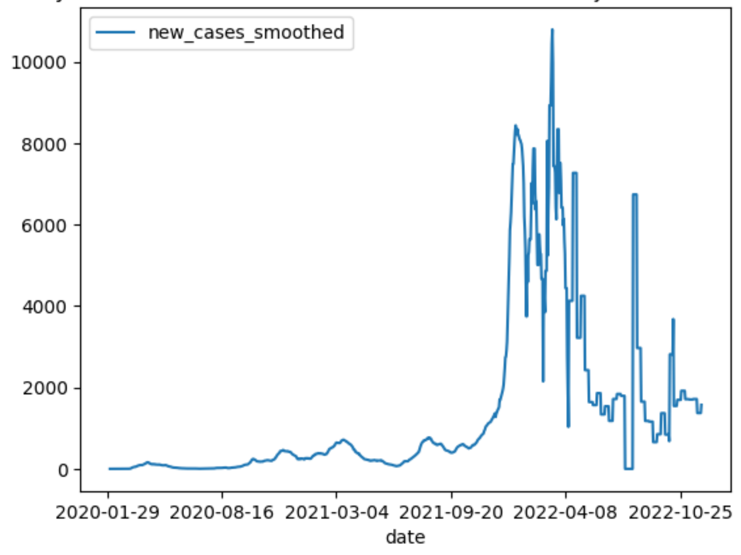
```
C:\Users\danfy\AppData\Local\Temp\ipykernel_14132\1434775432.py:7: FutureWarnin
g: In a future version of pandas all arguments of DataFrame.drop except for the
argument 'labels' will be keyword-only.
  useful_data_df = df.drop(
```

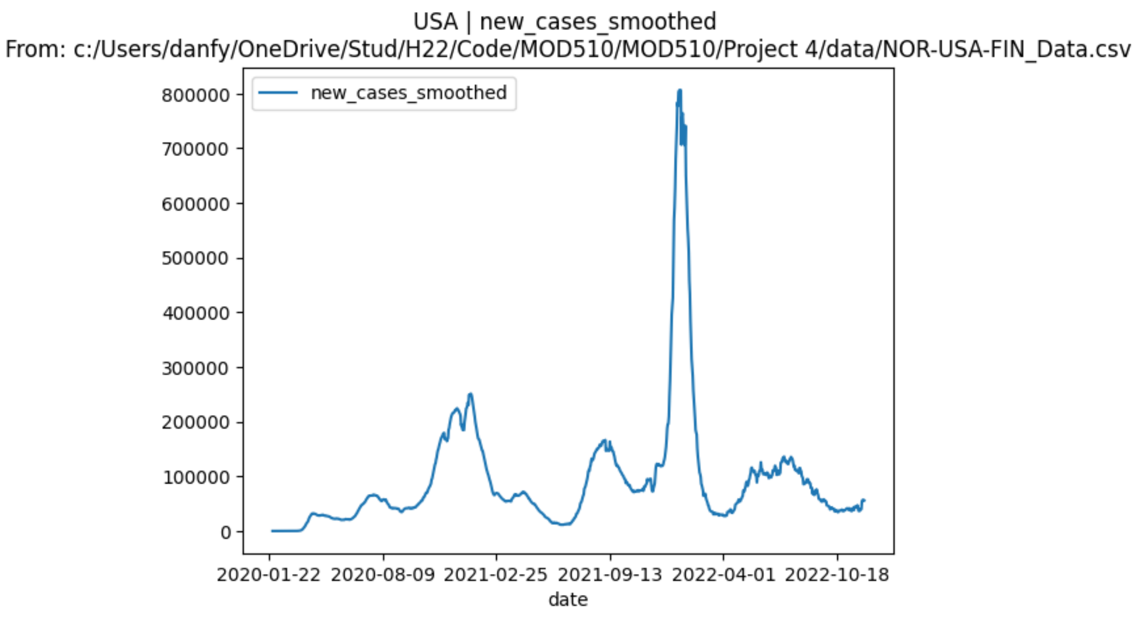
```
Out[ ]: Text(0.5, 1.0, 'USA | new_cases_smoothed \nFrom: c:/Users/danfy/OneDrive/Stud/H
22/Code/MOD510/MOD510/Project 4/data/NOR-USA-FIN_Data.csv')
```

NOR | new_cases_smoothed
From: c:/Users/danfy/OneDrive/Stud/H22/Code/MOD510/MOD510/Project 4/data/NOR-USA-FIN_Data.csv



FIN | new_cases_smoothed
From: c:/Users/danfy/OneDrive/Stud/H22/Code/MOD510/MOD510/Project 4/data/NOR-USA-FIN_Data.csv

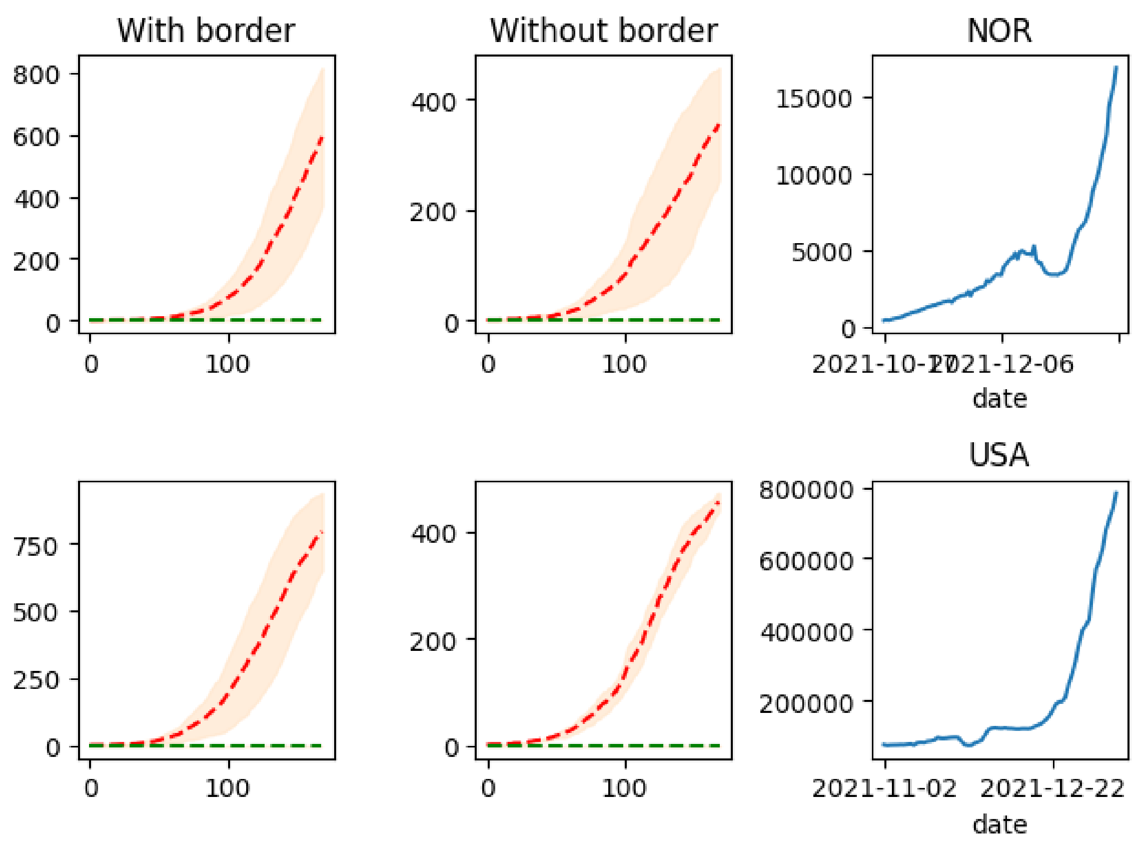




As seen above, the data of new cases, even when smoothed, is quite erratic. We can not hope to be able to match this, but instead will try to match to the peaks of both models, as they occur approximately over the same time/sample interval. The peak is then the total population of our model, so 100% for the model is the found amount in the image above. This is to save computation, because we only have integer values of population and infected, and because the general idea and solving method is the same.

```
In [ ]: # Trying to match results to some data
size=10
time_steps=170
pop_size=1000
no_init_infected=1
iterations=2
seeds=[480, 720]
recovery_probability=0.0
infection_probability=0.005
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect

fig, axs = plt.subplots(len(seeds), 3)
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i, 0], plot_susc=False)
    noBorder.plot_simulation(ax=axs[i, 1], plot_susc=False)

axs[0, 0].set_title('With border')
axs[0, 1].set_title('Without border')

NOR_useful_data_df[600:700].plot(x="date", y=y,
                                 ax=axs[0, 2],
                                 legend=False)
USA_useful_data_df[650:720].plot(x="date", y=y,
                                 ax=axs[1, 2],
                                 legend=False)
axs[0, 2].set_title('NOR')
axs[1, 2].set_title('USA')

fig.tight_layout()
```

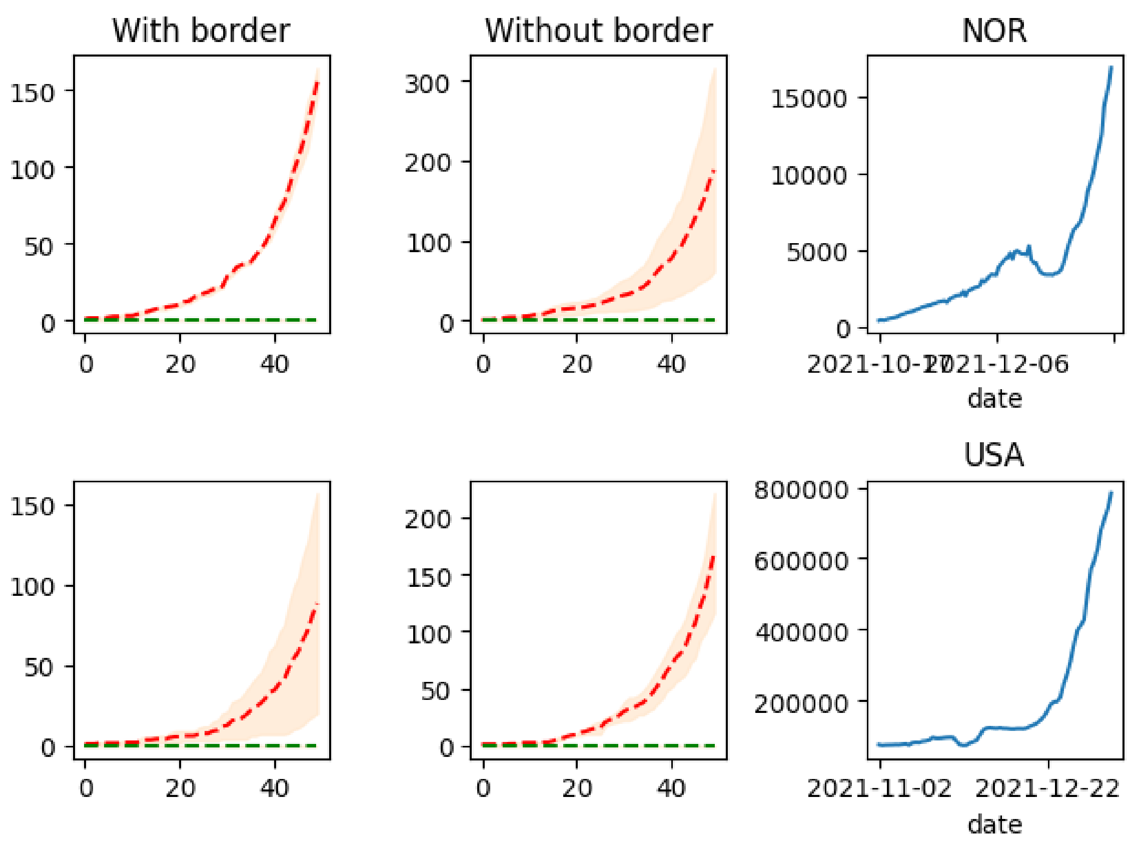
```
Current parameters:
size=10
time_steps=170
pop_size=1000
no_init_infected=1
iterations=2
seeds=[480, 720]
recovery_probability=0.0
infection_probability=0.005
```

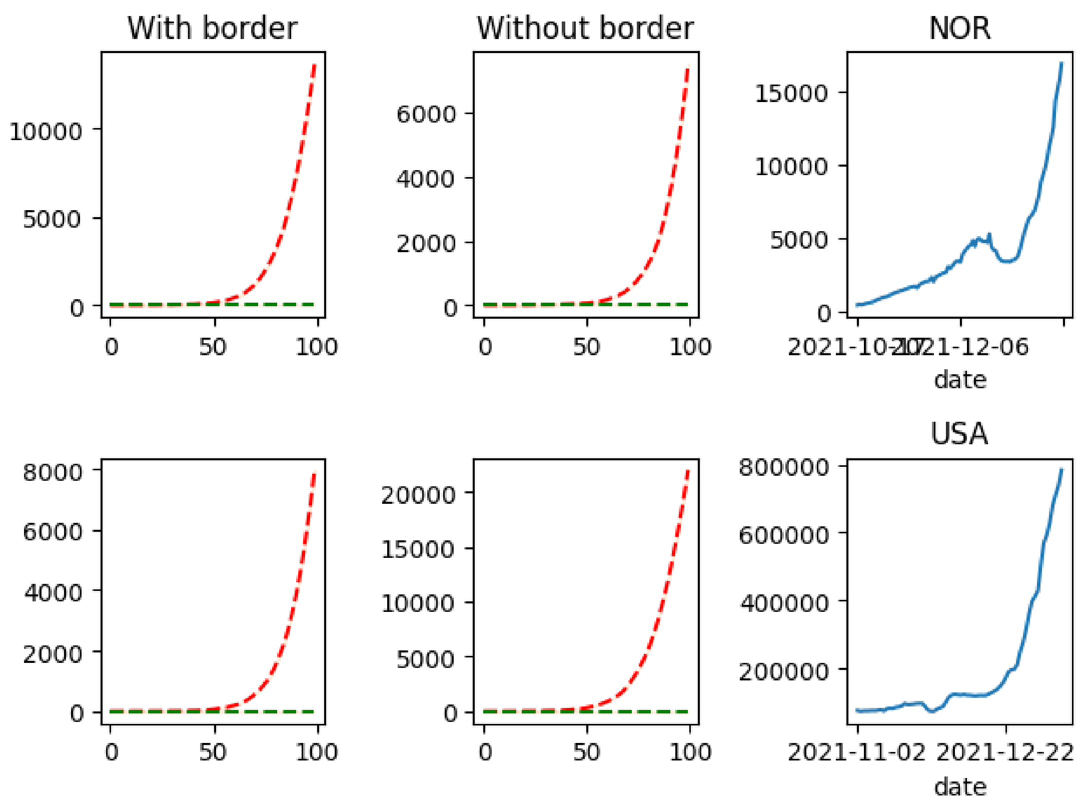


Above is a approximate fit for the Norwegian case.

```
In [ ]: # Trying to match results to some data
size=15
time_steps=50
pop_size=int((100000/(30**2))*(size**2))
no_init_infected=1
iterations=2
seeds=[480, 720]
recovery_probability=0.0
infection_probability=0.001
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect

fig, axs = plt.subplots(len(seeds), 3)
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i, 0], plot_susc=False)
    noBorder.plot_simulation(ax=axs[i, 1], plot_susc=False)

axs[0, 0].set_title('With border')
axs[0, 1].set_title('Without border')

NOR_useful_data_df[600:700].plot(x="date", y=y,
                                 ax=axs[0, 2],
                                 legend=False)
USA_useful_data_df[650:720].plot(x="date", y=y,
                                 ax=axs[1, 2],
                                 legend=False)
axs[0, 2].set_title('NOR')
axs[1, 2].set_title('USA')

fig.tight_layout()
```

```
Current parameters:
size=15
time_steps=50
pop_size=25000
no_init_infected=1
iterations=2
seeds=[480, 720]
recovery_probability=0.0
infection_probability=0.001
```





Above is a approximate fit for the United states case. The inserted image is with more computationally heavy parameters, but better fit.

Summary of exercise 3

The chosen new features added small changes to the simulations, which was quite underwhelming. Nonetheless the features in themselves are not pointless, and specifically contribute to highlighting the difference in variability.

As before specifically for the border and borderless comparison: From these results the basic and more or less only possible conclusion is:

The variability in a borderless simulation is greater, but the difference in mean value is negligable.

Practically it means that the results give us the following guidance:

With large enough simulations, it should behave practically the same way, with or without borders, but to be computationally efficient, the simulation with border should be used since it is more stable around the "common" mean.

The comparison between waves and before is a bit less clear:

There is not too much to say about it, since it is a very basic improvement, but we suspect it would scale in complexity along with any features added later like;

1. Recovered people having a lower, but not 0% chance of becoming infected.
2. Infectious people having area of effect spreading instead of only in-place.
3. Infectious people having variable degrees of infection, meaning;
   A. Different infection probability to those around them.
   B. Different movement pattern, for example every other time step
4. Recovery areas (Hospitals);
   A. Where the infected walk towards.
   B. Odds of spreading when inside is lower.
   C. Recovery rate only exist here.

And of course there are infinitely more, but imagining they are in the solution and then introducing these waves would then be a massively impactful effect, in contrast to how it is now. And so the effect should not be underappreciated. This is noted because we made the conscious decision of choosing 2 new features, and to match to real world data.

When it came to the mapping to the real world data there was several issues which mainly related to computation time that limited our ability to do a proper match. As the model is right now, and also given several more features like death or not letting infected people move, the ability to match it to real data would still be basically as useful as just creating a exponential function of the form:

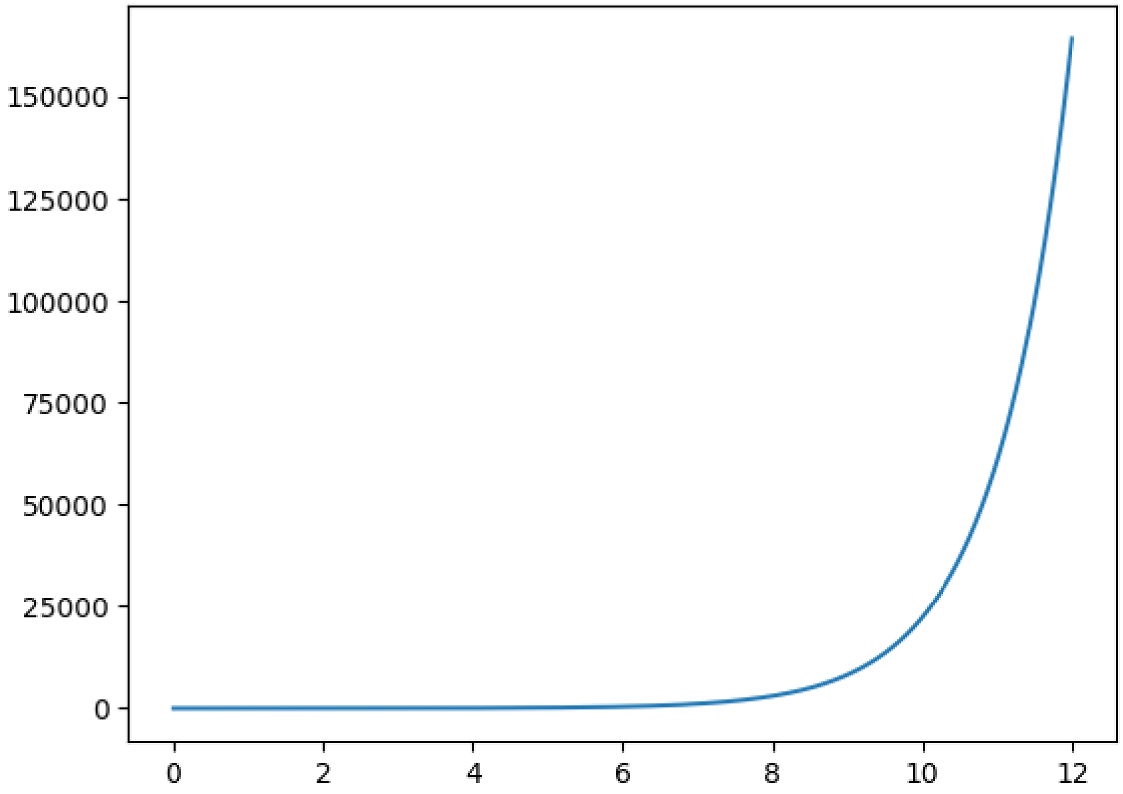
$$f(t) = -(1 - a \cdot e^t)$$

Tweaking the parameters even with rescaled numbers of people and time rate, the probabilities will be using fundamentally wrong assumptions and end up being basically just forced onto the real world data.

This is shown below:

In [ ]:
```
a = 1.01 # 1%
x = lambda t : -(1 - a*math.e**t )

t = np.arange(0,12, 0.001)

plt.plot(t, x(t))
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x1f3bc39ab50>]
```



Or else it would simply be a fake comparison, where we would have a wave of infected people, but by the end everyone is just recovered. This would be completely wrong even though it might look right, as everyone isn't immune directly after in the real world (even if the effects and chance of getting infected is lowered).

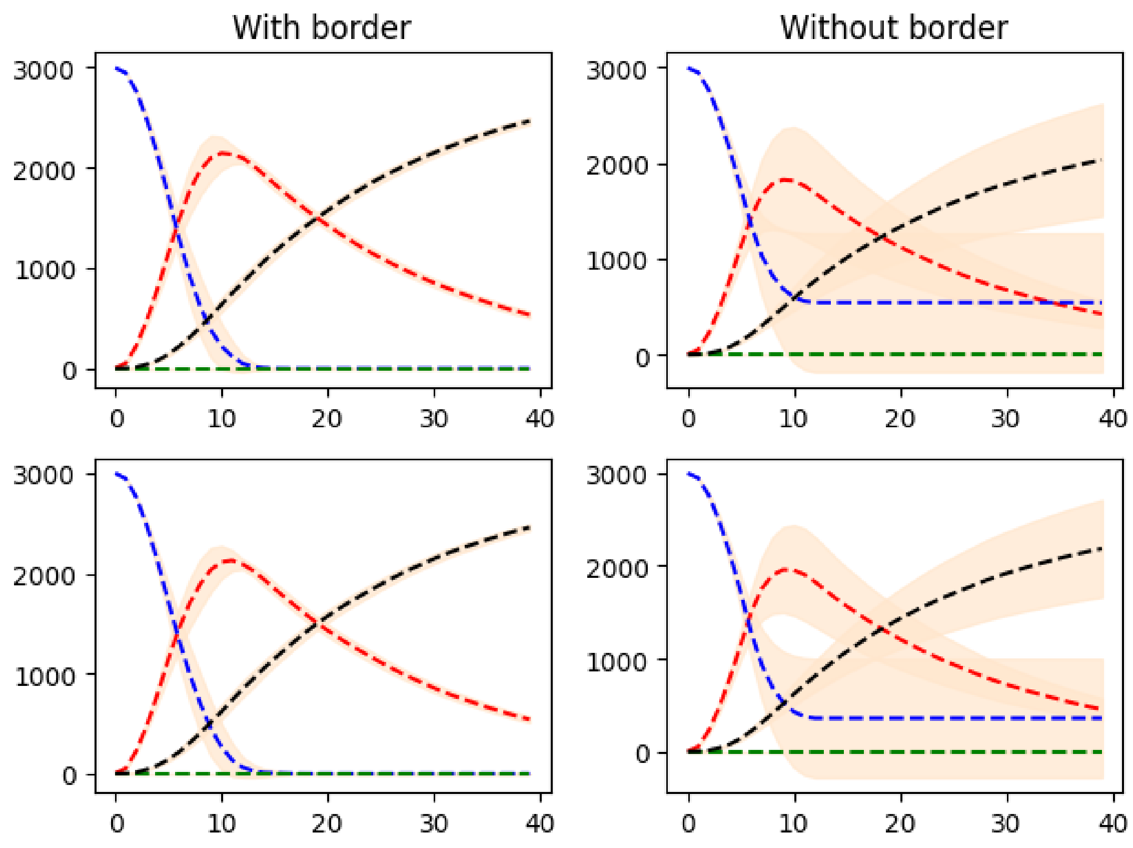
Since the results felt somewhat underwhelming, we decided to add some more feature even though we already had implemented two... cause why not.

We decided to simply add deaths, since that is a natural addition. This is shown below:

In [ ]:
```
size = 12
pop_size = 3000  # size*3
# math.ceil(size*(300*(size/10)))  # size = time_steps/(3*200)
time_steps = 40
no_init_infected = 3  # math.ceil(pop_size/100)
iterations = 25  # math.ceil(time_steps/pop_size)
seeds = [480, 720]  # np.arange(10*n, 10*(n+1))
recovery_probability = 0.00001e-2
infection_probability = 80e-2
death_rate = 5e-2
```

```
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed,
        death_rate=death_rate))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed,
        death_rate=death_rate))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect
print(
    f"death_rate={death_rate}")


numPlots = 2*len(seeds)
fig, axs = plt.subplots(int(numPlots/2), 2)
axs = axs.T.flatten()
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i])
    noBorder.plot_simulation(ax=axs[int(numPlots/2) + i])

axs[0].set_title('With border')
axs[int(numPlots/2)].set_title('Without border')
fig.tight_layout()
```

```
Current parameters:
size=12
time_steps=40
pop_size=3000
no_init_infected=3
iterations=25
seeds=[480, 720]
recovery_probability=1e-07
infection_probability=0.8
death_rate=0.05
```



Another addition which maybe was more interesting was a adaptive recovery rate, dependent on the amount of dead people in the population. Can call it a scared population effect or something. It uses the following equation to update the recovery rate:

$$r = r_i + \left[\frac{2d\lambda + i\lambda}{3N}\right]$$

Where;

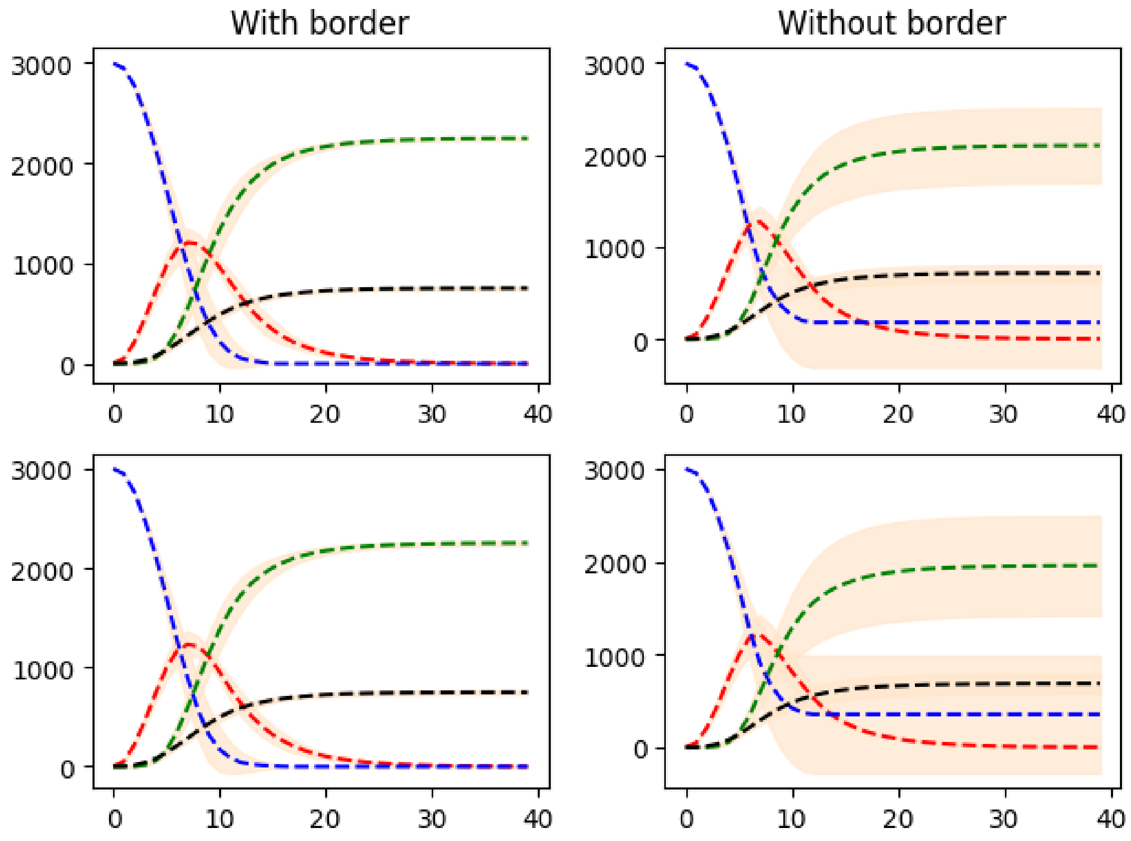
1. r is the recovery rate
2. d is the amount of deaths in the population
3. i is the amount of infected in the population
4. r_i is the initial/natural recovery rate
5. N is the total population number
6. λ is the scare factor / tuning parameter

This is done below, and the parameters for the model is printed.

In [ ]:
```
size = 12
pop_size = 3000  # size*3
# math.ceil(size*(300*(size/10)))  # size = time_steps/(3*200)
time_steps = 40
no_init_infected = 3  # math.ceil(pop_size/100)
iterations = 25  # math.ceil(time_steps/pop_size)
seeds = [480, 720]  # np.arange(10*n, 10*(n+1))
recovery_probability = 0.00001e-2
infection_probability = 80e-2
death_rate = 5e-2
```

```python
# recov = init + (2*death_perc*reac + infect_perc*reac)/3
desired_recov=0.4
infect_perc=0.5
death_perc=0.4

adaptive_recovery_rate = (desired_recov - recovery_probability)*3/(2*death_perc
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect
print(
    f"death_rate={death_rate}\nadaptive_recovery_rate={adaptive_recovery_rate}")


numPlots = 2*len(seeds)
fig, axs = plt.subplots(int(numPlots/2), 2)
axs = axs.T.flatten()
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i])
    noBorder.plot_simulation(ax=axs[int(numPlots/2) + i])

axs[0].set_title('With border')
axs[int(numPlots/2)].set_title('Without border')
fig.tight_layout()
```

```
Current parameters:
size=12
time_steps=40
pop_size=3000
no_init_infected=3
iterations=25
seeds=[480, 720]
recovery_probability=1e-07
infection_probability=0.8
death_rate=0.05
adaptive_recovery_rate=0.9230766923076923
```



Observing the result above, and comparing it to previous ones, it is clear that the recovery is varying as expected. This is much more complex behaviour than anything previously.

Specifically by looking at the result just prior to this one, when deaths were just implemented, the difference can be seen very well. These two simulations used identical parameters, aside from the adaptability change to the recovery rate. In that previous one, everyone ended up dying. In this one most recovered.

Next, to test the effect of waves on this much more complex model:

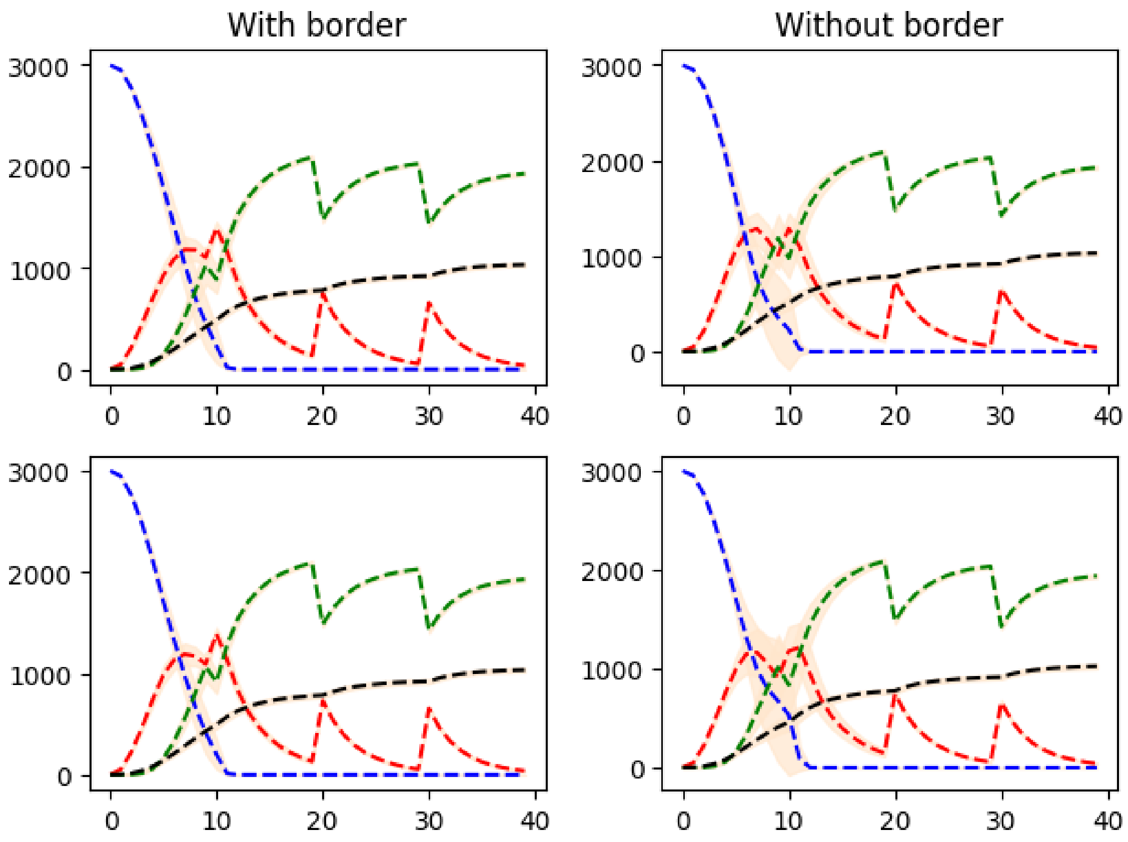
Will just use identical parameters, and have the same 3 waves as previously in this assignment:

```
In [ ]: size = 12
        pop_size = 3000  # size*3
        # math.ceil(size*(300*(size/10)))  # size = time_steps/(3*200)
        time_steps = 40
        no_init_infected = 3  # math.ceil(pop_size/100)
        iterations = 25  # math.ceil(time_steps/pop_size)
```

```
seeds = [480, 720]  # np.arange(10*n, 10*(n+1))
recovery_probability = 0.00001e-2
infection_probability = 80e-2
death_rate = 5e-2

# recov = init + (2*death_perc*reac + infect_perc*reac)/3
desired_recov=0.4
infect_perc=0.5
death_perc=0.4

adaptive_recovery_rate = (desired_recov - recovery_probability)*3/(2*death_perc
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate,
        wave_susc=0.1,
        wave_reco=0.3,
        num_waves=3,
        infectionWaves=True))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate,
        wave_susc=0.1,
        wave_reco=0.3,
        num_waves=3,
        infectionWaves=True))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect
print(
    f"death_rate={death_rate}\nadaptive_recovery_rate={adaptive_recovery_rate}")


numPlots = 2*len(seeds)
```

```
fig, axs = plt.subplots(int(numPlots/2), 2)
axs = axs.T.flatten()
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i])
    noBorder.plot_simulation(ax=axs[int(numPlots/2) + i])

axs[0].set_title('With border')
axs[int(numPlots/2)].set_title('Without border')
fig.tight_layout()
```

```
Current parameters:
size=12
time_steps=40
pop_size=3000
no_init_infected=3
iterations=25
seeds=[480, 720]
recovery_probability=1e-07
infection_probability=0.8
death_rate=0.05
adaptive_recovery_rate=0.9230769230769230769 23
```



This result is somewhat as expected, and shows that the wave effect changes the final number of deaths.

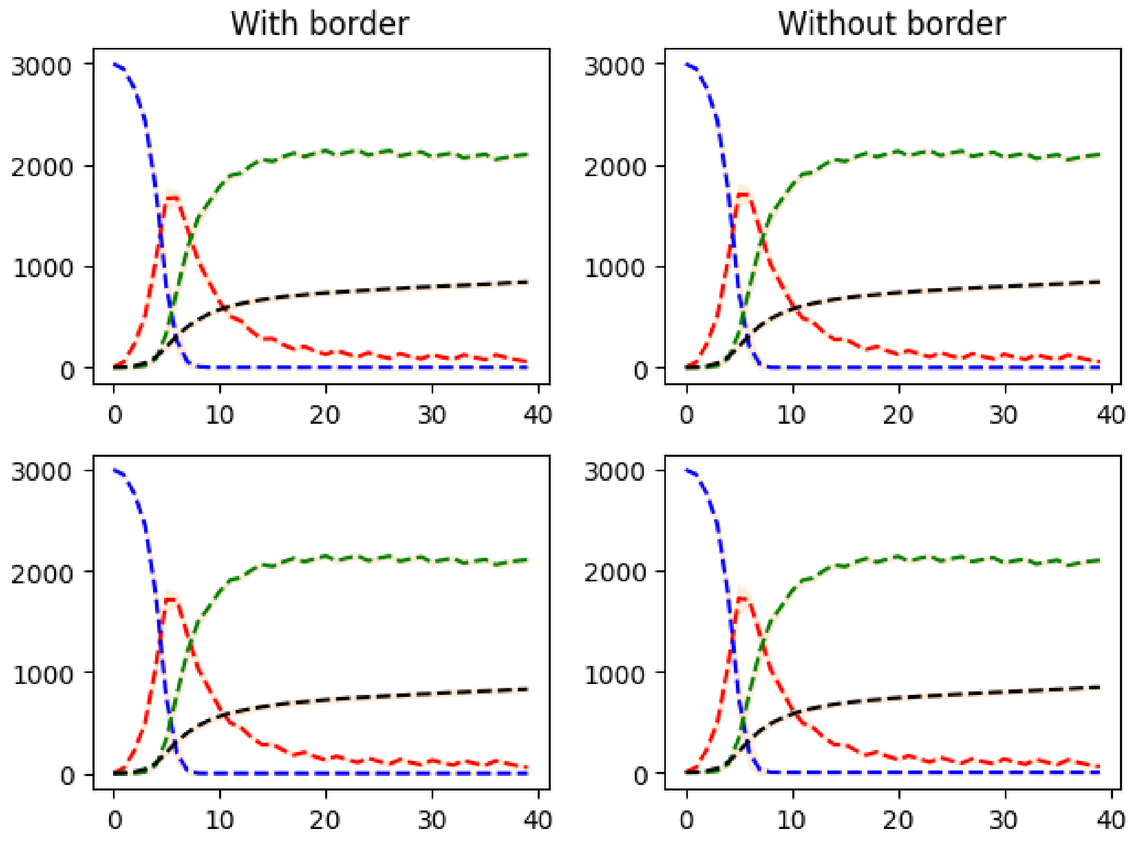
Then with the same 10 as before:

```
In [ ]: size = 12
pop_size = 3000  # size*3
# math.ceil(size*(300*(size/10)))  # size = time_steps/(3*200)
time_steps = 40
no_init_infected = 3  # math.ceil(pop_size/100)
iterations = 25  # math.ceil(time_steps/pop_size)
seeds = [480, 720]  # np.arange(10*n, 10*(n+1))
```

```
recovery_probability = 0.00001e-2
infection_probability = 80e-2
death_rate = 5e-2

# recov = init + (2*death_perc*reac + infect_perc*reac)/3
desired_recov=0.4
infect_perc=0.5
death_perc=0.4

adaptive_recovery_rate = (desired_recov - recovery_probability)*3/(2*death_perc
borders = []
noBorders = []
for seed in seeds:
    borders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate,
        wave_susc=0.01,
        wave_reco=0.03,
        num_waves=10,
        infectionWaves=True))
    noBorders.append(RandomlyViral(
        iterations=iterations,
        no_init_infected=no_init_infected,
        population_size=pop_size,
        nx=size,
        ny=size,
        time_steps=time_steps,
        recovery_probability=recovery_probability,
        infection_probability=infection_probability,
        Recovery="on",
        no_border=True,
        random_seed=seed,
        death_rate=death_rate,
        adaptive_recovery_rate=adaptive_recovery_rate,
        wave_susc=0.01,
        wave_reco=0.03,
        num_waves=10,
        infectionWaves=True))

print(f"Current parameters:\nsize={size}\ntime_steps={time_steps}")
print(f"pop_size={pop_size}\nno_init_infected={no_init_infected}")
print(f"iterations={iterations}\nseeds={seeds}")
print(
    f"recovery_probability={recovery_probability}\ninfection_probability={infect
print(
    f"death_rate={death_rate}\nadaptive_recovery_rate={adaptive_recovery_rate}")


numPlots = 2*len(seeds)
fig, axs = plt.subplots(int(numPlots/2), 2)
```

```
axs = axs.T.flatten()
for i, (border, noBorder) in enumerate(zip(borders, noBorders)):
    border.plot_simulation(ax=axs[i])
    noBorder.plot_simulation(ax=axs[int(numPlots/2) + i])

axs[0].set_title('With border')
axs[int(numPlots/2)].set_title('Without border')
fig.tight_layout()
```

```
Current parameters:
size=12
time_steps=40
pop_size=3000
no_init_infected=3
iterations=25
seeds=[480, 720]
recovery_probability=1e-07
infection_probability=0.8
death_rate=0.05
adaptive_recovery_rate=0.9230766923076923
```



Here we can clearly observe that the smaller waves had much less of a effect.

Conclusion

The main aim for this project was to model the spread of a disease through a population. The Monte Carlo principle was used for introducing randomness to the processes along the way. Movement of individuals, chance of infection, recovery, and even death - was all decided by the roll of the dice (or so to speak).

We observed that a deterministic ODE-model could be tuned to fit the Random Walk model quite well.

- Although it was not easy to grasp the exact meaning of the constant β, we observed the effectiveness of calibrating this constant.
- While calibrating τ we, unsurprisingly, found that if an infected individual has a 1% chance of recovering per time step, it will on average take 100 time steps for the individual to recover.

The strength of the Random walk approach is that its inherent modular framework relatively easy allows for adding-to, customizing and reworking the model. Through adding features and updating the model we had close to no issues with the stability of the code itself. It is clear that this approach is a rather blunt tool, which will not be used in situations where high orders of accuracy is needed. In situatuation like disease spreading, however, getting as close as a single order of magnitude may be close enough to guide decision making.

The limitations of the Random walk model became apparent while contemplating fitting it to real-world, big-scale data. The model shows the capability to model small sections of such datasets. We can imagine that we could gain insights to the effectiveness of regulations at different times or geographical locations by comparing how the curve-fitting constants would change between them. For the scope of this project we chose to not dive too deep into this rabbit hole.

All-in-all we have shown that chaotic behavior can be quite orderly when scaled up sufficiently. We must be careful not to misunderstand this, however, as the great Daniel Kahneman said: ' The idea that the future is unpredictable is undermined every day by the ease with which the past is explained. '

Personal Reflection

Andreas:

This project was incredibly open-ended, at least exercise 3 which I enjoyed. Even though our solution did not turn out to explore the most interesting dynamics possible, it nonetheless was decent practice at contemplating expected behaviours compared to actual. It was also pretty nice to find the practicality of borders in a experimental manner. The deaths was also kinda nice to implement, but what ended up giving the most interesting control over the result was the adaptable recovery rate. That was quite fun to tinker with.

Daniel:

Philosophically I think that it’s important to have goals, but that the things that ends up being precious to us are things we stumble upon on the way there. This project is like a miniature scale life in that sense. Technically I had the most fun exploring Boolean masking in this project. I like the game of trying to avoid for- and while-loops as much as you can, although I had to use them a few places where I think someone smarter would be able to still avoid them. As for previous projects it really helps for both my motivation and imagination to have such a concrete physical problem to tie the technical skills of coding into. For this last project it felt like “Group 4” really had our teamwork dialled in. The main strength of this project was the appeal to creativity by the open-ended Exercise 3. We had a lot of interesting discussions, not all documented in the report – along the way.

Sahar:

This project was very attractive for me since I could find an example in real world (Covid-19). Exercise 1, was interesing since moving of walker and the probability of getting infectious are completly randomly. It was informative due to some challenge in coding. Exercise 2 learned me how use from current model to extend it and adding parameter to it. And finally, exercise 3 helped me to how implement my idea in a model. I think this project was very intersting due to its similarity to real example. It helpes me imagine and underestanding much more better than previous ones.

References

[1] - Liliana Perez and Suzana Dragicevic. An agent-based approach for modeling dynamics of contagious disease spread. International Journal of Health Geographics, 8(1):1–17, 2009.

[2] - Chu, A., Huber, G., McGeever, A. et al. A random-walk-based epidemiological model. Sci Rep 11, 19308 (2021)

[3] - A. Hiorth, Nov 14. 2022, *Coronavirus Goes Randomly Viral*

[4]- Covid data by community of open-source, https://github.com/owid/covid-19-data

[5]- "Sigmoid function" by Wikipedia, https://en.wikipedia.org/wiki/Sigmoid_function

[6]- "Normal distribution" by Wikipedia, https://en.wikipedia.org/wiki/Normal_distribution