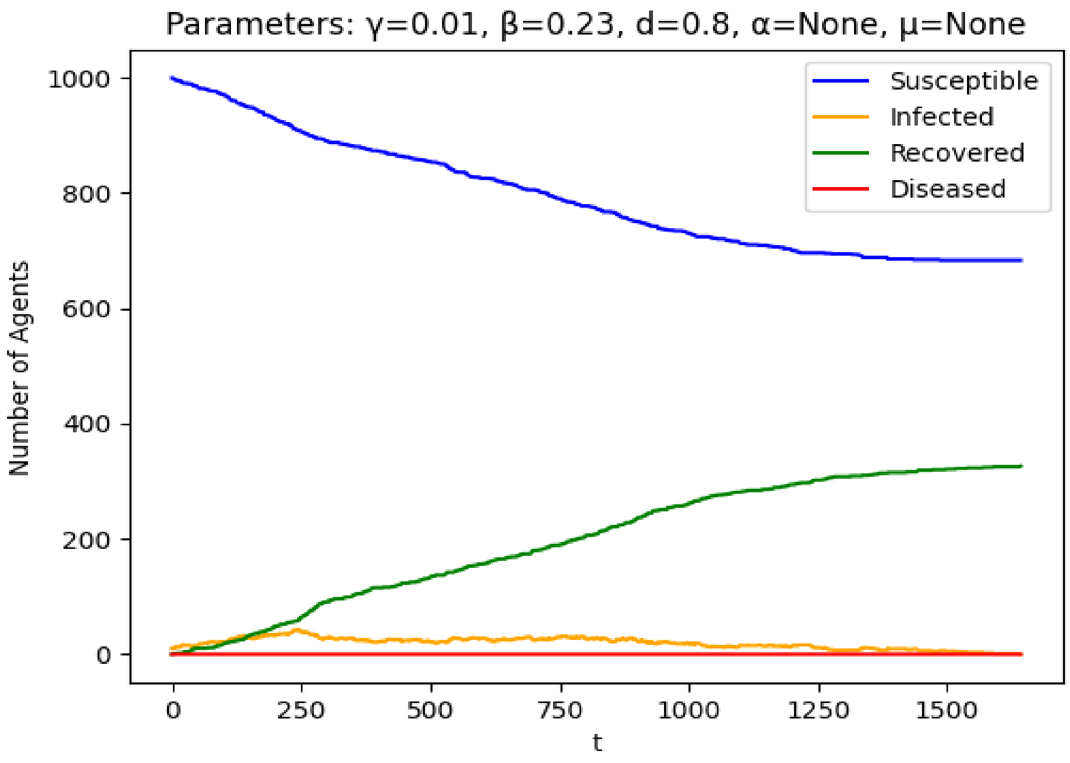
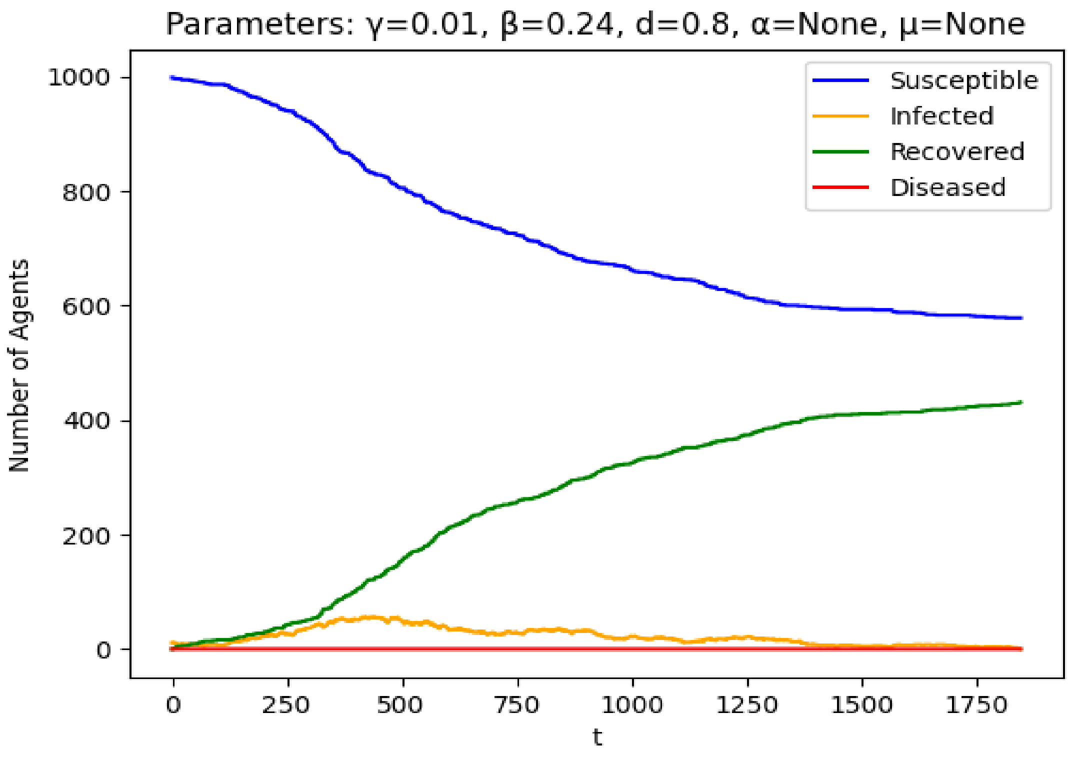
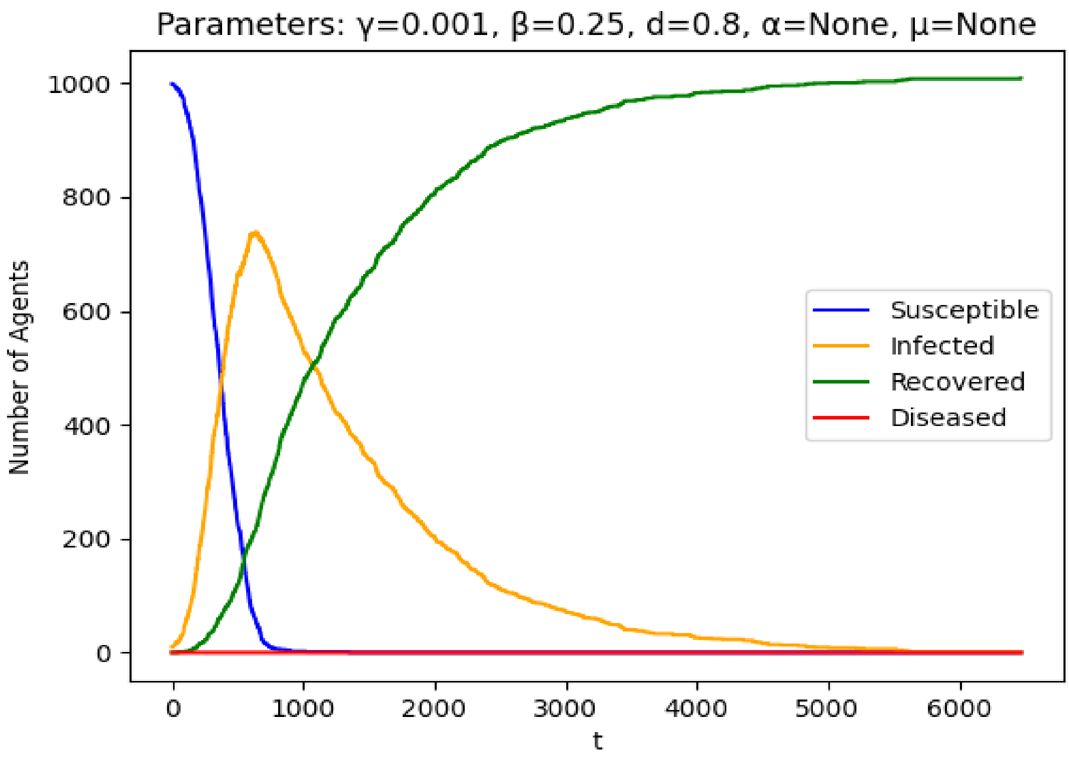
11.1

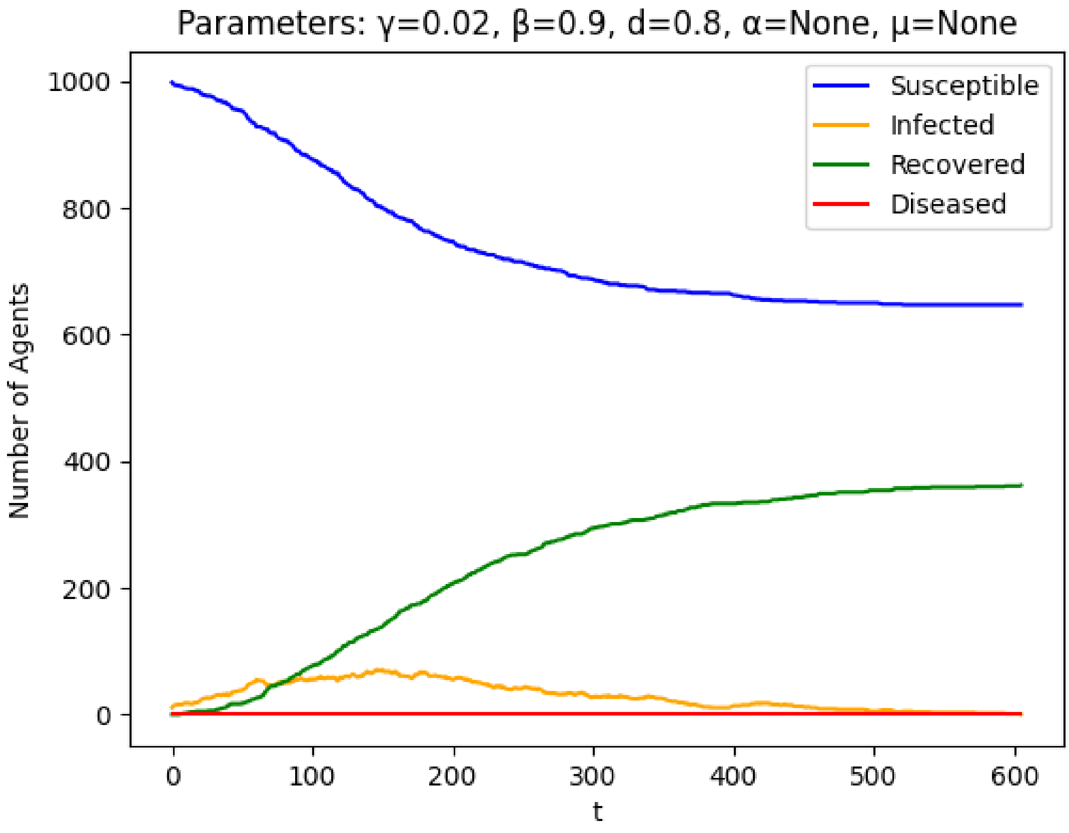


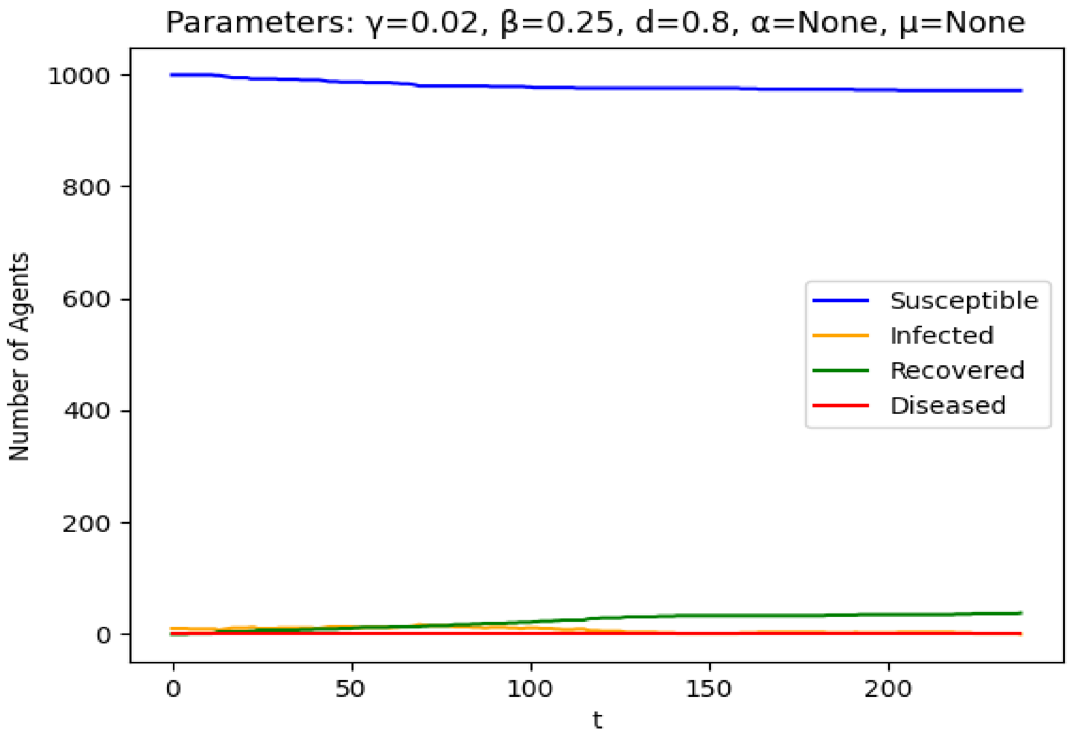
Parameters: γ=0.01, β=0.23, d=0.8, α=None, μ=None
Number of Agents
t
Susceptible
Infected
Recovered
Diseased



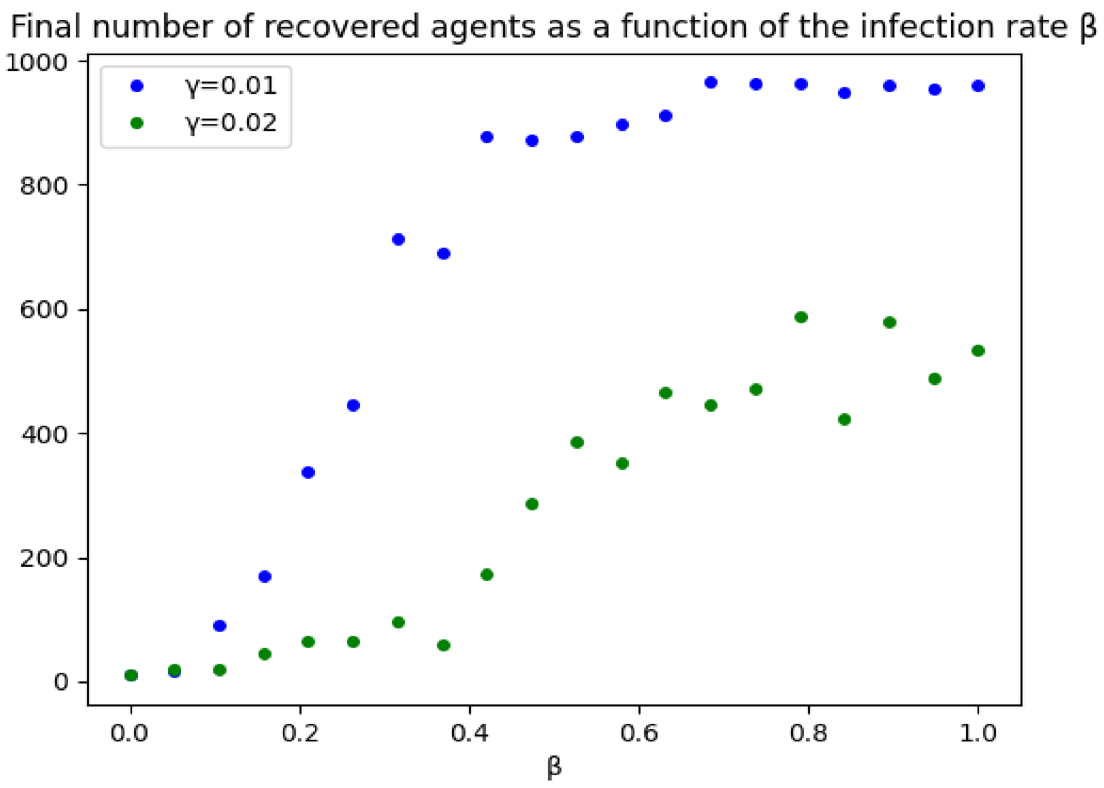
Parameters: γ=0.01, β=0.24, d=0.8, α=None, μ=None
Number of Agents
t
Susceptible
Infected
Recovered
Diseased

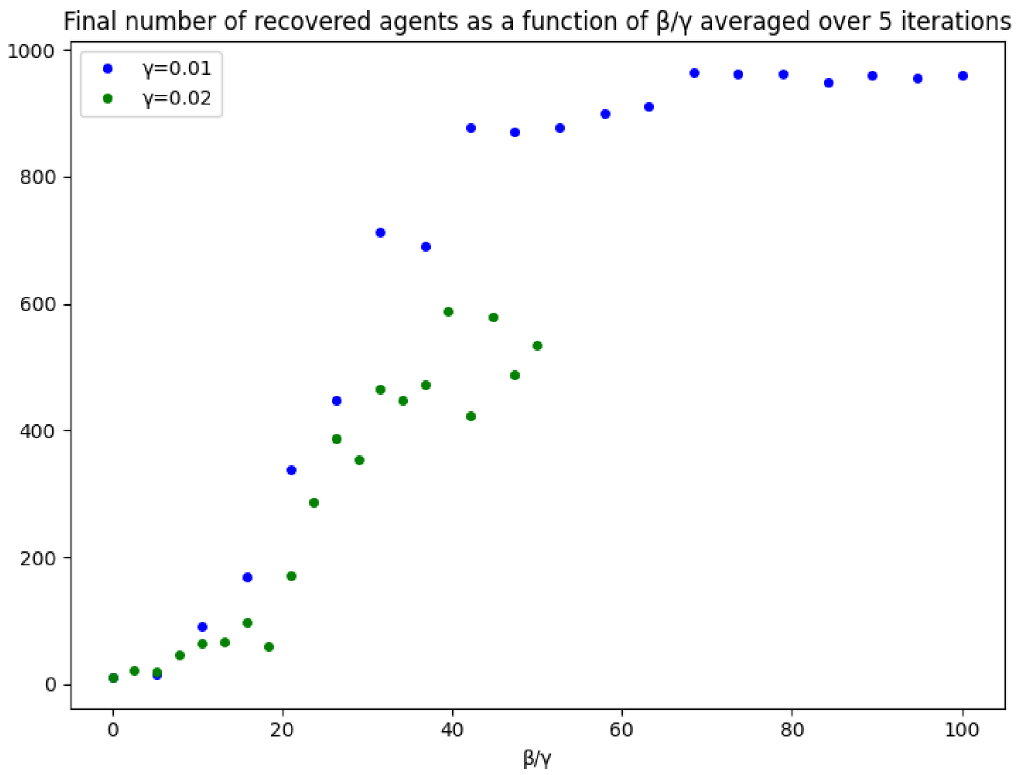

Parameters: γ=0.001, β=0.25, d=0.8, α=None, μ=None
Number of Agents
1000
800
600
400
200
0
0
1000
2000
3000
4000
5000
6000
t
Susceptible
Infected
Recovered
Diseased

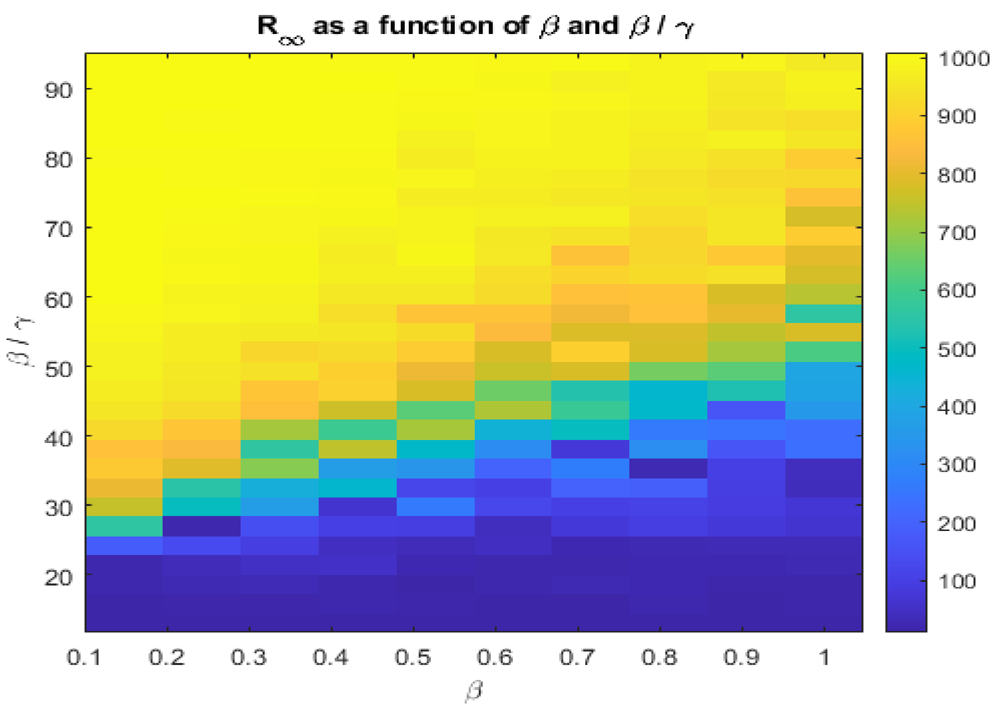

Parameters: γ=0.02, β=0.9, d=0.8, α=None, μ=None
Number of Agents
1000
800
600
400
200
0
0
100
200
300
400
500
600
t
Susceptible
Infected
Recovered
Diseased


Parameters: γ=0.02, β=0.25, d=0.8, α=None, μ=None
Number of Agents
t
0
50
100
150
200
0
200
400
600
800
1000
Susceptible
Infected
Recovered
Diseased

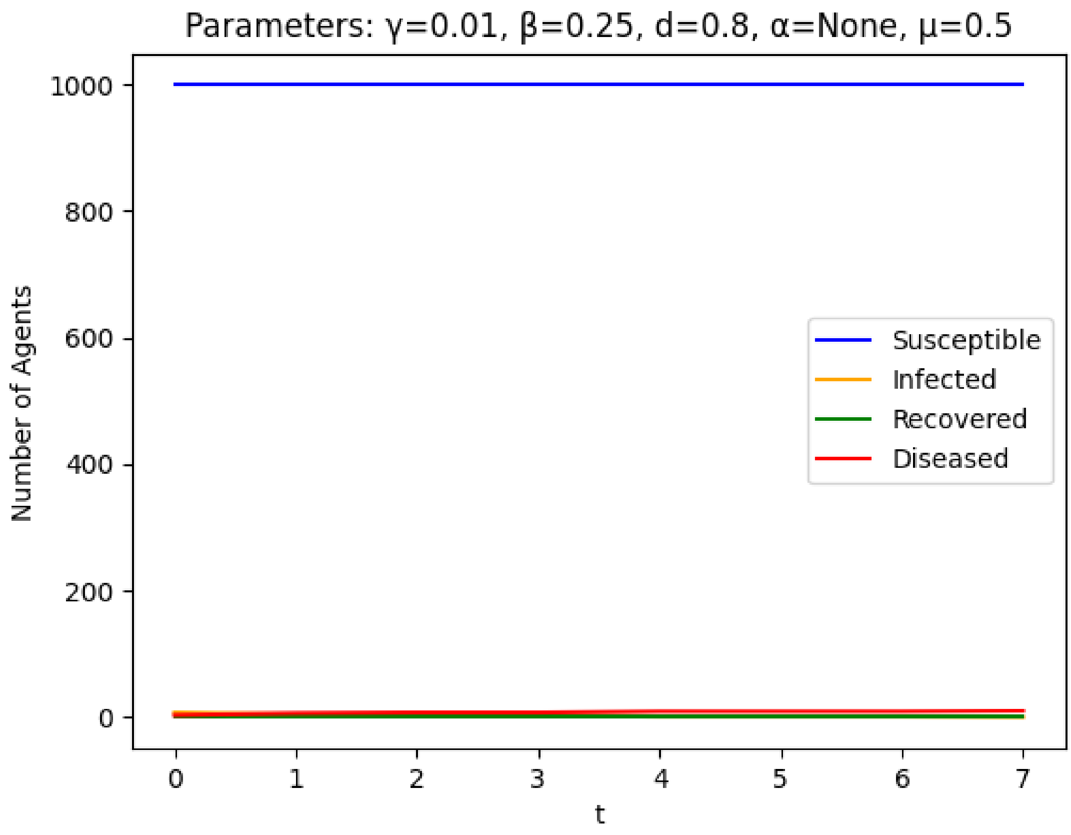
11.2

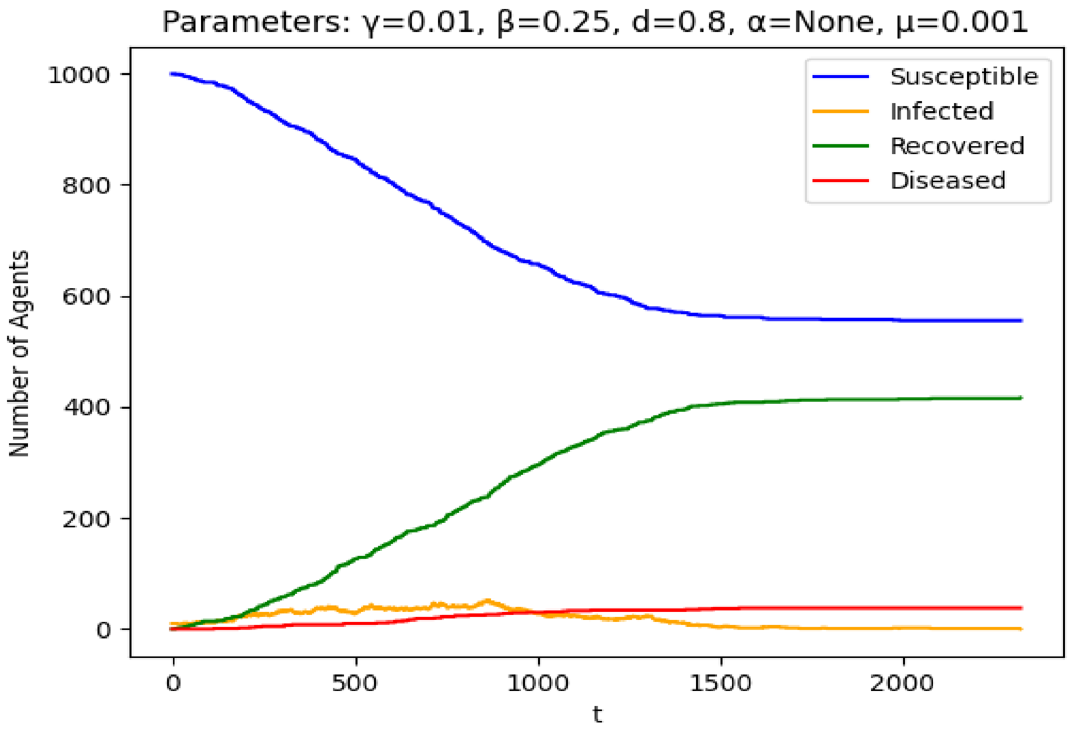

Final number of recovered agents as a function of the infection rate β
γ=0.01
γ=0.02
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0
β

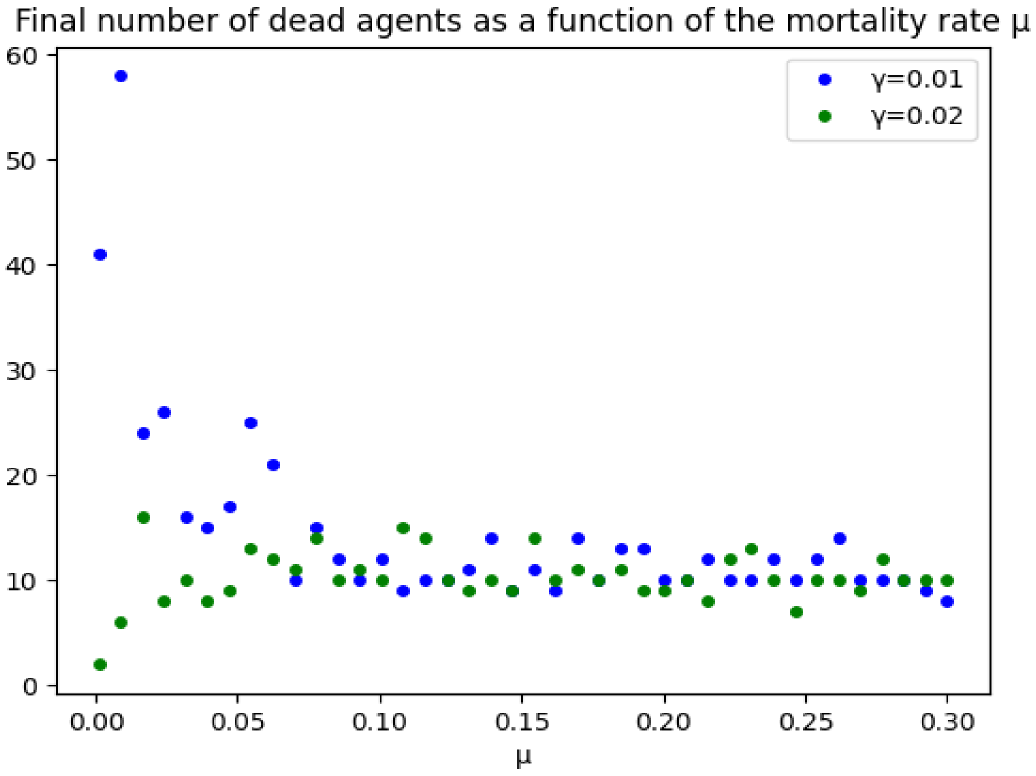

Final number of recovered agents as a function of β/γ averaged over 5 iterations
γ=0.01
γ=0.02
1000
800
600
400
200
0
0
20
40
60
80
100
β/γ

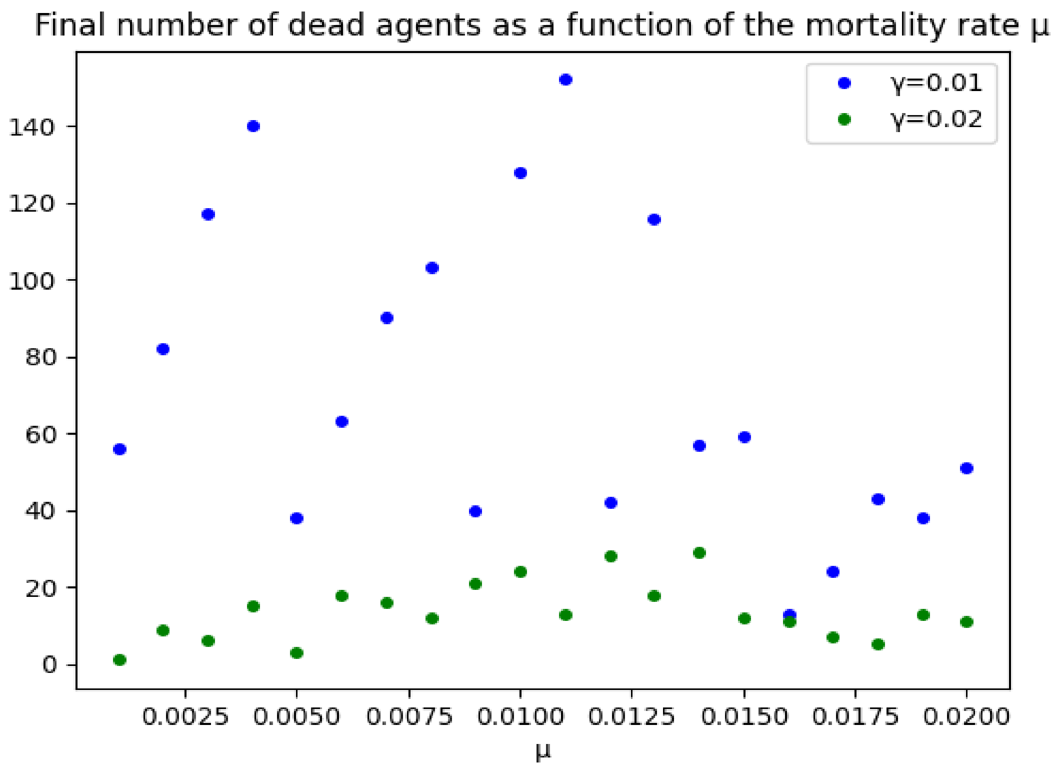

R_∞ as a function of β and β / γ
β / γ
β
90
80
70
60
50
40
30
20
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1000
900
800
700
600
500
400
300
200
100

11.3

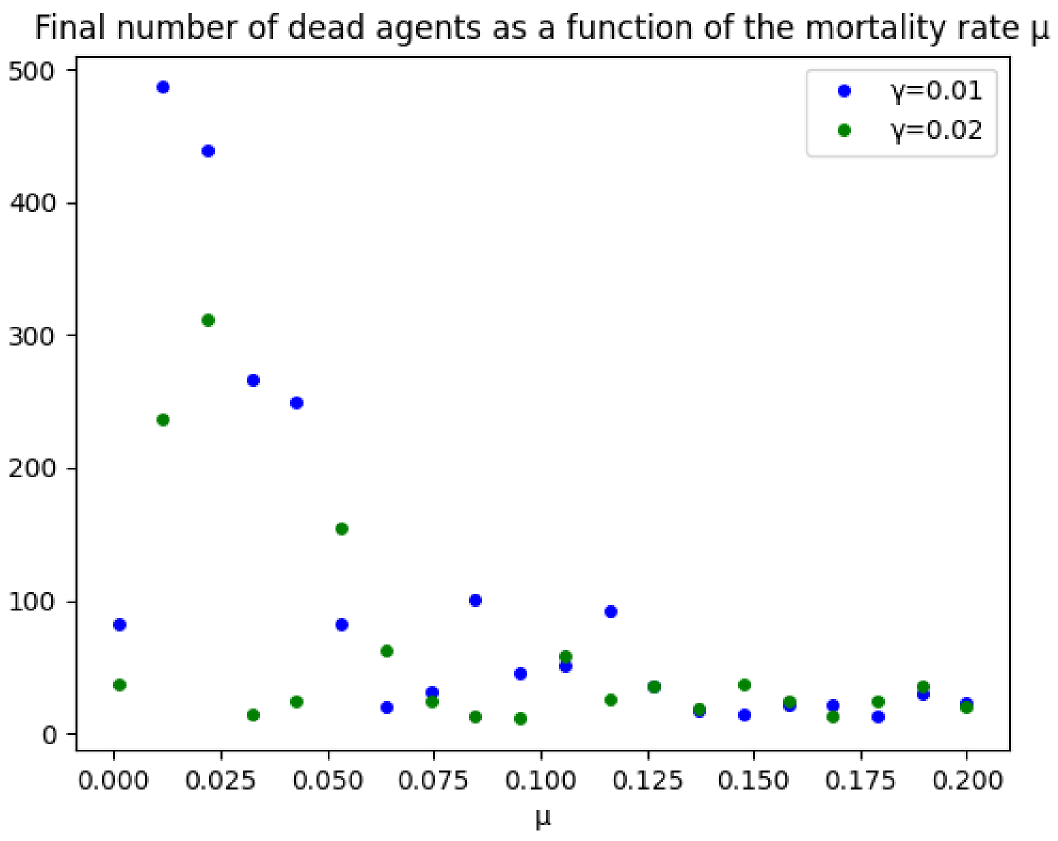

Parameters: γ=0.01, β=0.25, d=0.8, α=None, μ=0.5
Number of Agents
t
Susceptible
Infected
Recovered
Diseased
1000
800
600
400
200
0
0
1
2
3
4
5
6
7


Parameters: γ=0.01, β=0.25, d=0.8, α=None, μ=0.001
Susceptible
Infected
Recovered
Diseased
Number of Agents
t

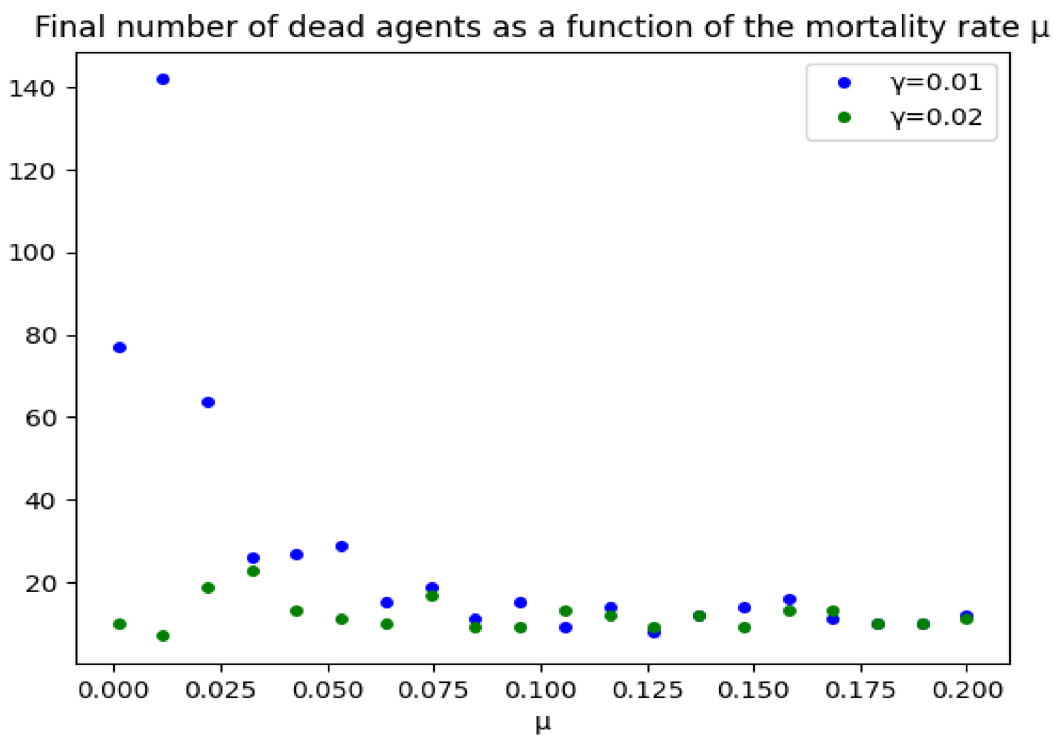

Final number of dead agents as a function of the mortality rate μ
γ=0.01
γ=0.02
μ

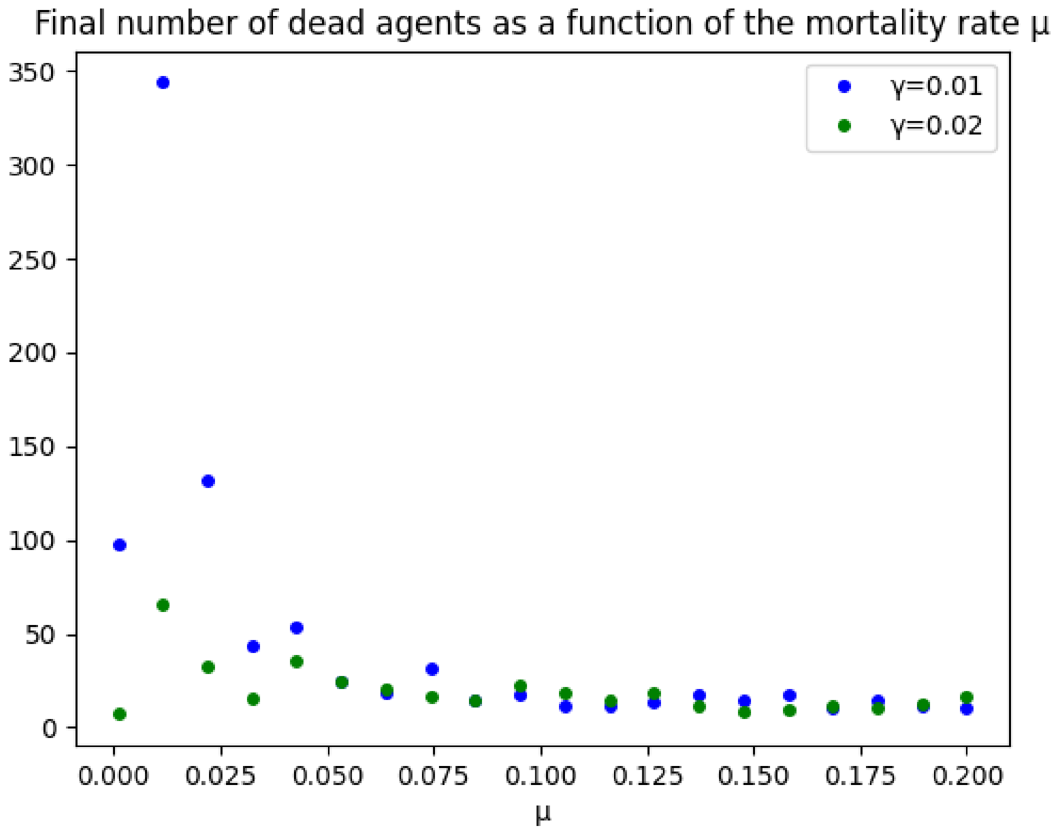


Final number of dead agents as a function of the mortality rate μ
γ=0.01
γ=0.02
μ

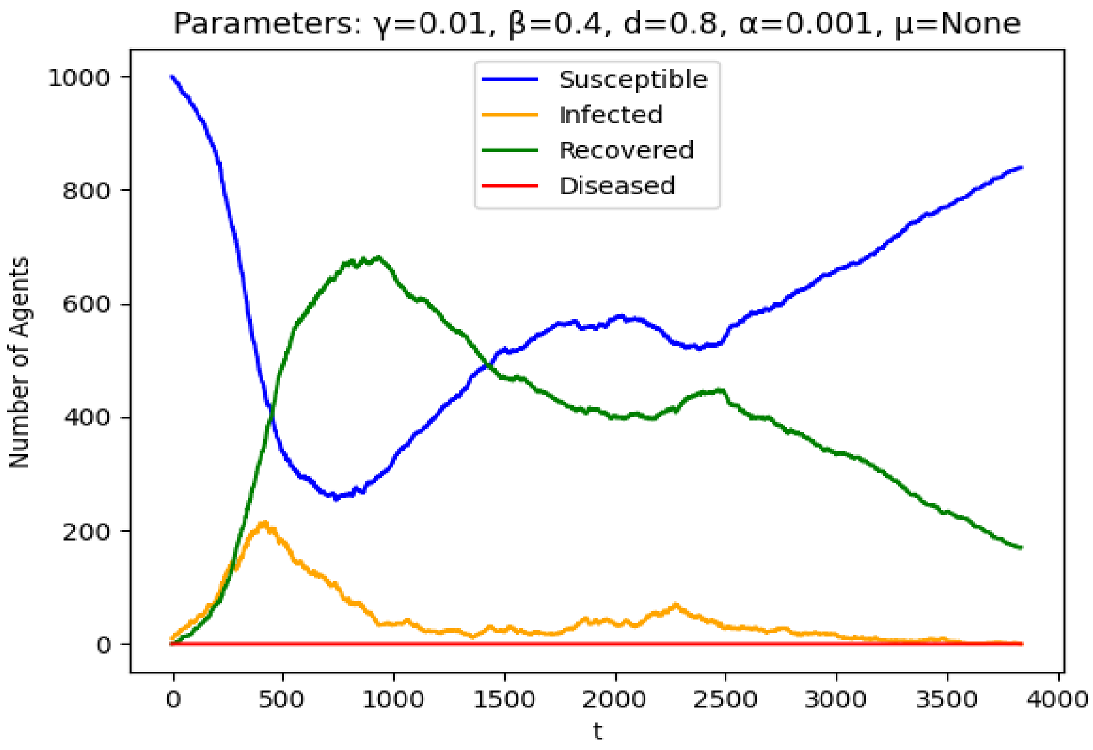


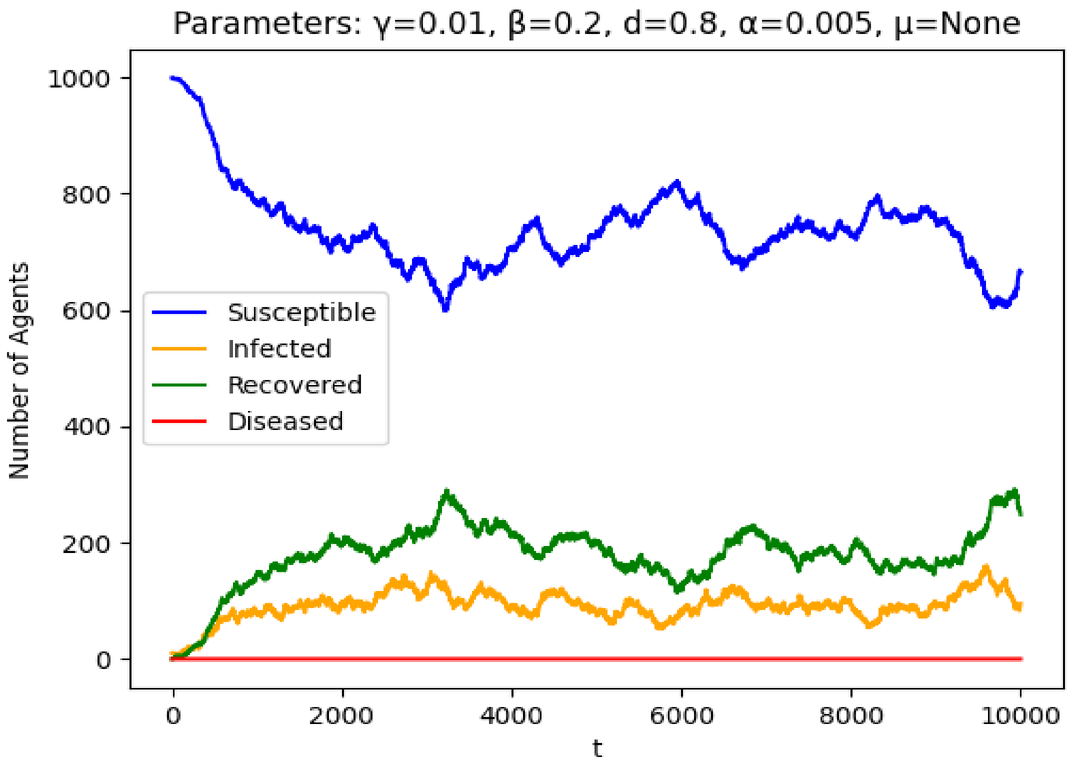
Final number of dead agents as a function of the mortality rate μ
γ=0.01
γ=0.02
μ

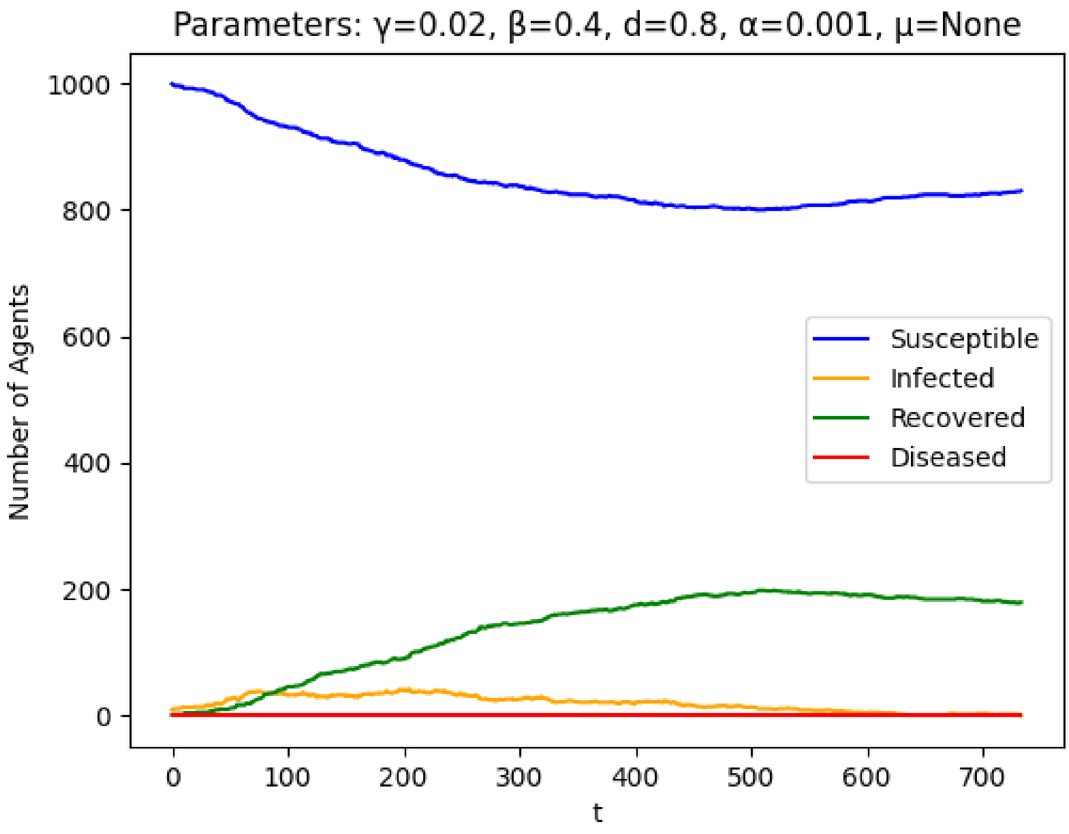

Final number of dead agents as a function of the mortality rate μ
γ=0.01
γ=0.02
μ
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
20
40
60
80
100
120
140


Final number of dead agents as a function of the mortality rate μ
γ=0.01
γ=0.02
μ
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
50
100
150
200
250
300
350

11.4


Parameters: γ=0.01, β=0.4, d=0.8, α=0.001, μ=None
Susceptible
Infected
Recovered
Diseased
Number of Agents
t
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
3500
4000


Parameters: γ=0.01, β=0.2, d=0.8, α=0.005, μ=None
Susceptible
Infected
Recovered
Diseased
Number of Agents
t
0
200
400
600
800
1000
0
2000
4000
6000
8000
10000


Parameters: γ=0.02, β=0.4, d=0.8, α=0.001, μ=None
Number of Agents
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
t
Susceptible
Infected
Recovered
Diseased

Individual.py

```
import numpy as np


class Individual:
    def __init__(self, state: str, lattice: int):
        if isinstance(lattice+1, int):
            self.position = np.random.randint(lattice, size=2)
        else:
            raise ValueError('Lattice not valid')

        if state == 'recovered':
            self.state = 'recovered'
        elif state == 'infected':
            self.state = 'infected'
        elif state == 'diseased':
            self.state = 'diseased'
        elif state == 'susceptible':
            self.state = 'susceptible'
        else:
            raise TypeError('State not valid')

    def update_state(self, state: str):
        if state == 'recovered':
            self.state = 'recovered'
        elif state == 'infected':
            self.state = 'infected'
        elif state == 'diseased':
            self.state = 'diseased'
        elif state == 'susceptible':
            self.state = 'susceptible'
        else:
            raise TypeError('State not valid')

    def move(self, direction, lattice):
        if direction == 'up' and self.position[1] < lattice:
            self.position[1] += 1
        if direction == 'down' and self.position[1] > 0:
            self.position[1] -= 1
        if direction == 'right' and self.position[0] < lattice:
            self.position[0] += 1
        if direction == 'left' and self.position[0] > 0:
            self.position[0] -= 1

    def update_position(self, position: np.ndarray):
        if isinstance(position, np.ndarray):
            self.position = position
        else:
            raise TypeError('Has to be ndarray')
```

SIR.py

```
from Individual import Individual
import matplotlib.pyplot as plt
import numpy as np


def initialize_world(lattice, nAgents, infection_rate):
    nInfected = int(nAgents * infection_rate)
    nSusceptible = int(nAgents * 1-infection_rate)
    susceptible = [Individual("susceptible", lattice) for _ in range(nSusceptible)]
    infected = [Individual("infected", lattice) for _ in range(nInfected)]

    return susceptible, infected


def random_walk(individual, move_prob, lattice):
    r = np.random.rand()
    if r < move_prob:
        d = np.random.randint(4)
        if d == 0:
            individual.move('up', lattice)
        if d == 1:
            individual.move('down', lattice)
        if d == 2:
            individual.move('right', lattice)
        if d == 3:
            individual.move('left', lattice)


def walk(object_list, prob, lattice):
    for obj in object_list:
        random_walk(obj, prob, lattice)


def check_infected(infected, susceptible, beta):
    for infectant in infected:
        r = np.random.rand()
        if r < beta:
            for s in susceptible:
                if np.array_equal(s.position, infectant.position):
                    s.update_state("infected")
                    infected.append(s)
                    susceptible.remove(s)
    return infected, susceptible


def recovery(infected, recovered, gamma):
    for i in infected:
        r = np.random.rand()
        if r < gamma:
            i.update_state('recovered')
            recovered.append(i)
            infected.remove(i)
    return recovered, infected


def death(diseased, infected, mu):
    for i in infected:
        r = np.random.rand()
        if r < mu:
            i.update_state('diseased')
            diseased.append(i)
            infected.remove(i)
    return diseased, infected


def re_susceptible(susceptible, recovered, alpha):
    for rec in recovered:
        r = np.random.rand()
        if r < alpha:
            rec.update_state("susceptible")
            susceptible.append(rec)
            recovered.remove(rec)
    return susceptible, recovered


def plot_sir(susceptible, infected, recovered, diseased, gamma, d, beta, mu=None, alpha=None
    alpha_unicode = "\u03B1"
    gamma_unicode = "\u03B3"
    beta_unicode = "\u03B2"
    mu_unicode = "\u03BC"
    plt.plot(susceptible, color='blue')
    plt.plot(infected, color='orange')
    plt.plot(recovered, color='green')
    plt.plot(diseased, color='red')
    plt.legend(
        ["Susceptible", "Infected", "Recovered", "Diseased"])
    plt.ylabel('Number of Agents')
    plt.xlabel('t')
    plt.title(f'Parameters: {gamma_unicode}={gamma}, {beta_unicode}={beta}, d={d}, {alpha_un


def plot_lattice(infected, susceptible, recovered, diseased):
    x_infected = [obj.position[0] for obj in infected]
    y_infected = [obj.position[1] for obj in infected]

    x_susceptible = [obj.position[0] for obj in susceptible]
    y_susceptible = [obj.position[1] for obj in susceptible]

    x_recovered = [obj.position[0] for obj in recovered]
    y_recovered = [obj.position[1] for obj in recovered]

    x_diseased = [obj.position[0] for obj in diseased]
    y_diseased = [obj.position[1] for obj in diseased]

    plt.plot(x_infected, y_infected, "o", color="orange", markersize=4)
    plt.plot(x_susceptible, y_susceptible, "o", color="blue", markersize=4)
    plt.plot(x_recovered, y_recovered, "o", color="green", markersize=4)
    plt.plot(x_diseased, y_diseased, "x", color="red", markersize=4)
    plt.legend(
        ["Infected", "Susceptible", "Recovered", "Diseased"])
    plt.title(f"#Inf = {len(infected)} #Sus = {len(susceptible)} #Rec = {len(recovered)} #Di
```

sir_standard.py

```
import matplotlib.pyplot as plt
import SIR as sir

# Parameters
lattice = 100
nAgents = 1000
infection_rate = 0.01   # Initial infection_rate
move_prob = 0.8
beta = 0.25            # Infection probability
gamma = 0.01         # Recover probability
mu = 0.5             # Mortality probability
alpha = None          # Re-susceptible probability

# Initialization
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
infected_overtime = []
recovered_overtime = []
susceptible_overtime = []
diseased_overtime = []

timestep = 0
susceptible, infected = sir.initialize_world(lattice, nAgents, infection_rate)
recovered = []
diseased = []
infected_population = []


while len(infected) > 0:
    sir.walk(susceptible, move_prob, lattice)
    sir.walk(infected, move_prob, lattice)
    sir.walk(recovered, move_prob, lattice)

    infected, susceptible = sir.check_infected(infected, susceptible, beta)
    recovered, infected = sir.recovery(infected, recovered, gamma)
    diseased, infected = sir.death(diseased, infected, mu)

    susceptible_overtime.append(len(susceptible))
    infected_overtime.append(len(infected))
    recovered_overtime.append(len(recovered))
    diseased_overtime.append(len(diseased))

    timestep += 1
    print(timestep)
    print(f'infected: {len(infected)}')

sir.plot_sir(susceptible_overtime, infected_overtime, recovered_overtime, diseased_overtime
plt.show()
```

sir_final_number_recovered_agents.py

```
from tqdm import tqdm

# Parameters
lattice = 100
nAgents = 1000
infection_rate = 0.01               # Initial infection_rate
move_prob = 0.8
Beta = np.linspace(0, 1, 20)        # Infection probability
Gamma = [0.01, 0.02]                # Recover probability

# Initialization
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
infected_overtime = []
recovered_overtime = []
susceptible_overtime = []
diseased_overtime = []


for i, gamma in enumerate(tqdm(Gamma)):
    nrRecovered_list = []
    for beta in tqdm(Beta):
        timestep = 0
        susceptible, infected = sir.initialize_world(lattice, nAgents, infection_rate)
        recovered = []
        diseased = []
        while len(infected) > 0:
            sir.walk(susceptible, move_prob, lattice)
            sir.walk(infected, move_prob, lattice)
            sir.walk(recovered, move_prob, lattice)

            infected, susceptible = sir.check_infected(infected, susceptible, beta)
            recovered, infected = sir.recovery(infected, recovered, gamma)

            susceptible_overtime.append(len(susceptible))
            infected_overtime.append(len(infected))
            recovered_overtime.append(len(recovered))
            diseased_overtime.append(len(diseased))
        nrRecovered_list.append(len(recovered))
    if gamma == 0.01:
        time_now = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
        nrRecovered_array = np.array(nrRecovered_list)
        np.save(time_now + f'R_inf_gamma={Gamma[0]}', nrRecovered_array)
        plt.plot(Beta, nrRecovered_list, "o", color="blue", markersize=4)
    if gamma == 0.02:
        time_now = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
        nrRecovered_array = np.array(nrRecovered_list)
        np.save(time_now + f'R_inf_gamma={Gamma[1]}', nrRecovered_array)
        plt.plot(Beta, nrRecovered_list, "o", color="green", markersize=4)

# Plot final recovered as a function of infection rate
plt.legend([f"{gamma_unicode}={Gamma[0]}", f"{gamma_unicode}={Gamma[1]}"])
plt.xlabel(f'{beta_unicode}')
plt.title(f"Final number of recovered agents as a function of the infection rate {beta_unic
plt.show()
```

average_recovered.py

```
import matplotlib.pyplot as plt

# Initialize
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
Beta = np.linspace(0.1, 0.9, 9)
Gamma = [0.01, 0.02]

# Load
gamma_1 = np.load('R_inf_gamma=0.01_1.npy') + np.load('R_inf_gamma=0.01_2.npy') + np.load('F
          np.load('R_inf_gamma=0.01_4.npy') + np.load('R_inf_gamma=0.01_5.npy')

gamma_2 = np.load('R_inf_gamma=0.02_1.npy') + np.load('R_inf_gamma=0.02_2.npy') + np.load('F
          np.load('R_inf_gamma=0.02_4.npy') + np.load('R_inf_gamma=0.02_5.npy')
average_1 = gamma_1/5
average_2 = gamma_2/5

beta_gamma_1 = Beta/Gamma[0]
beta_gamma_2 = Beta/Gamma[1]

"""
# Function of beta
plt.plot(Beta, average_1, "o", color="blue", markersize=4)
plt.plot(Beta, average_2, "o", color="green", markersize=4)
plt.legend([f"{gamma_unicode}={Gamma[0]}", f"{gamma_unicode}={Gamma[1]}"])
plt.xlabel(f'{beta_unicode}')
plt.title(f"Final number of recovered agents as a function of the infection rate {beta_unic
plt.show()
"""
"""
# Function of Beta / gamma
plt.plot(beta_gamma_1, average_1, "o", color="blue", markersize=4)
plt.plot(beta_gamma_2, average_2, "o", color="green", markersize=4)
plt.legend([f"{gamma_unicode}={Gamma[0]}", f"{gamma_unicode}={Gamma[1]}"])
plt.xlabel(f'{beta_unicode}/{gamma_unicode}')
plt.title(f"Final number of recovered agents as a function of {beta_unicode}/{gamma_unicode]
plt.show()
"""

# Phase diagram
# y_max = np.max(beta_gamma_1)
# y_min = np.min(beta_gamma_1)
# x_max = np.max(Beta)
# x_min = np.min(Beta)
# plt.imshow(average_1, extent=[x_min, x_max, y_min, y_max])
# plt.show()
```

beta_over_gamma.py

```
from tqdm import tqdm

# Parameters
lattice = 100
nAgents = 1000
infection_rate = 0.01               # Initial infection_rate
move_prob = 0.8
Beta = np.linspace(0.1, 1, 10)        # Infection probability
Q = np.linspace(1, 100, 30)

# Initialization
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
infected_overtime = []
recovered_overtime = []
susceptible_overtime = []
diseased_overtime = []

for q in tqdm(Q):
    nrRecovered_list = []
    for beta in tqdm(Beta):
        timestep = 0
        susceptible, infected = sir.initialize_world(lattice, nAgents, infection_rate)
        recovered = []
        diseased = []
        gamma = beta/q
        while len(infected) > 0:
            sir.walk(susceptible, move_prob, lattice)
            sir.walk(infected, move_prob, lattice)
            sir.walk(recovered, move_prob, lattice)

            infected, susceptible = sir.check_infected(infected, susceptible, beta)
            recovered, infected = sir.recovery(infected, recovered, gamma)

            susceptible_overtime.append(len(susceptible))
            infected_overtime.append(len(infected))
            recovered_overtime.append(len(recovered))
            diseased_overtime.append(len(diseased))
        nrRecovered_list.append(len(recovered))
    time_now = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
    nrRecovered_array = np.array(nrRecovered_list)
    #np.save('arrays/' + time_now + f' R_inf,beta_over_gamma,Q={q}', nrRecovered_array)
    np.savetxt('arrays/' + time_now + f' R_inf,beta_over_gamma,Q={q}.csv', nrRecovered_arra
```

sir_mortality.py

```
import numpy as np

# Parameters
lattice = 100
nAgents = 1000
infection_rate = 0.01
move_prob = 0.8
beta = 0.4                          # Infection probability
gamma = [0.01, 0.02]                # Recover probability
mu = np.linspace(0.001, 0.02, 20)    # Death probability

# Initialization
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
infected_overtime = []
recovered_overtime = []
susceptible_overtime = []
diseased_overtime = []
timestep = 0

for i, g in enumerate(gamma):
    death_list = []
    for m in mu:
        timestep = 0
        recovered = []
        diseased = []
        susceptible, infected = sir.initialize_world(lattice, nAgents, infection_rate)
        while len(infected) > 0:
            sir.walk(susceptible, move_prob, lattice)
            sir.walk(infected, move_prob, lattice)
            sir.walk(recovered, move_prob, lattice)

            infected, susceptible = sir.check_infected(infected, susceptible, beta)
            recovered, infected = sir.recovery(infected, recovered, g)
            diseased, infected = sir.death(diseased, infected, m)

            susceptible_overtime.append(len(susceptible))
            infected_overtime.append(len(infected))
            recovered_overtime.append(len(recovered))
            diseased_overtime.append(len(diseased))
            timestep += 1
            print(timestep)
            print(f'infected: {len(infected)}')
        death_list.append(len(diseased))
    if i == 0:
        plt.plot(mu, death_list, "o", color="blue", markersize=4)
    if i == 1:
        plt.plot(mu, death_list, "o", color="green", markersize=4)

# Plot final deaths as a function of mortality rate
plt.legend([f"{gamma_unicode}={gamma[0]}", f"{gamma_unicode}={gamma[1]}"])
plt.xlabel(f'{mu_unicode}')
plt.title(f"Final number of dead agents as a function of the mortality rate {mu_unicode}")
plt.savefig(f'Final number of dead agents as a function of the mortality rate {mu_unicode}.p
plt.show()
```

sir_temporary_immunity.py

,

```
import matplotlib.pyplot as plt
import SIR as sir
import numpy as np
from datetime import datetime

# Parameters
lattice = 100
nAgents = 1000
infection_rate = 0.01   # Initial infection_rate
move_prob = 0.8
beta = 0.2              # Infection probability
gamma = 0.01            # Recover probability
alpha = 0.005            # Re-susceptible probability
mu = None

# Initialization
alpha_unicode = "\u03B1"
gamma_unicode = "\u03B3"
beta_unicode = "\u03B2"
mu_unicode = "\u03BC"
infected_overtime = []
recovered_overtime = []
susceptible_overtime = []
diseased_overtime = []
timestep = 0

recovered = []
diseased = []
susceptible, infected = sir.initialize_world(lattice, nAgents, infection_rate)


while len(infected) > 0 and timestep < 10000:
    sir.walk(susceptible, move_prob, lattice)
    sir.walk(infected, move_prob, lattice)
    sir.walk(recovered, move_prob, lattice)

    infected, susceptible = sir.check_infected(infected, susceptible, beta)
    recovered, infected = sir.recovery(infected, recovered, gamma)
    susceptible, recovered = sir.re_susceptible(susceptible, recovered, alpha)

    susceptible_overtime.append(len(susceptible))
    infected_overtime.append(len(infected))
    recovered_overtime.append(len(recovered))
    diseased_overtime.append(len(diseased))

    timestep += 1
    print(timestep)
    print(f'infected: {len(infected)}')

sir.plot_sir(susceptible_overtime, infected_overtime, recovered_overtime, diseased_overtime
plt.show()
```

phase_diagram.m

```
clear all
close all
clc

myfiles = dir("arrays\");
filenames={myfiles(:).name}';
filefolders={myfiles(:).folder}';
csvfiles=filenames(endsWith(filenames,'.csv'));
csvfolders=filefolders(endsWith(filenames,'.csv'));
files=fullfile(csvfolders,csvfiles);
values = zeros(30,10);

for i = 1:length(values)
        f = files{i};
        ff = load(f)';
        values(i,:) = ff;
end



beta = linspace(0.1, 1, 20);
Q = linspace(10, 100, 30);


imagesc([beta(1), beta(end)], [Q(1), Q(end)], values)
colorbar
xlabel('\beta')
ylabel('\beta / \gamma')
title('R_\infty as a function of \beta and \beta / \gamma')
set(gca,'YDir','normal')
```