CS111 Final Project – Modeling Herd Immunity

Yoav Rabinovich

Disease propogation is a valuable subject of study that can directly save lives and huge amounts of resources. While we have discussed in class modeling of disease propogation, we have not considered the affects of immunization on the dynamics. In this paper I attempt to incorporate immunization into mathematical models of disease propogation.

Populations: Ordinary Differential Equations and Euler's Method

The first approch we look at is using systems of ODEs to model interactions between population. This approach is deterministic – there is no probability involved, and our model yields consistent predictions every time it is run with the same parameters. In the past, we have considered a system with three populations: healthy, infected and dead. The different populations can be modeled by differential equations over time. For instance, we model the healthy population over time with a birth rate proportional to the healthy population, a natural death rate proportional to the healthy population an infection rate proportional to the healthy and infected populations. The components are weighted with coeficients, and we can write the derivative of the healthy population over time:

$$dH/dt = C_{birth}*H - C_{nat\text{-}death}*H - C_{infection}*H*I$$

Similarly, the infected population can be modeled with the same infection term and a term for death proportional to the infected population, and the dead population can be modeled with the existing terms:

$$dI/dt = C_{infection}*H*I - C_{death}*I$$

$$dD/dt = C_{nat\text{-}death}*H + C_{death}*I$$

This system can now be solved using numerical methods, like the Euler method. With the Euler method, time is divided into distinct steps, and given initial populations and coefficients, the current population is computed by summing the population in the past step with the derivative times the time step:

$\mathbf{y_t = y_{t-1} + h * y'_{t-1}}$

For example, this solution was computed using python:

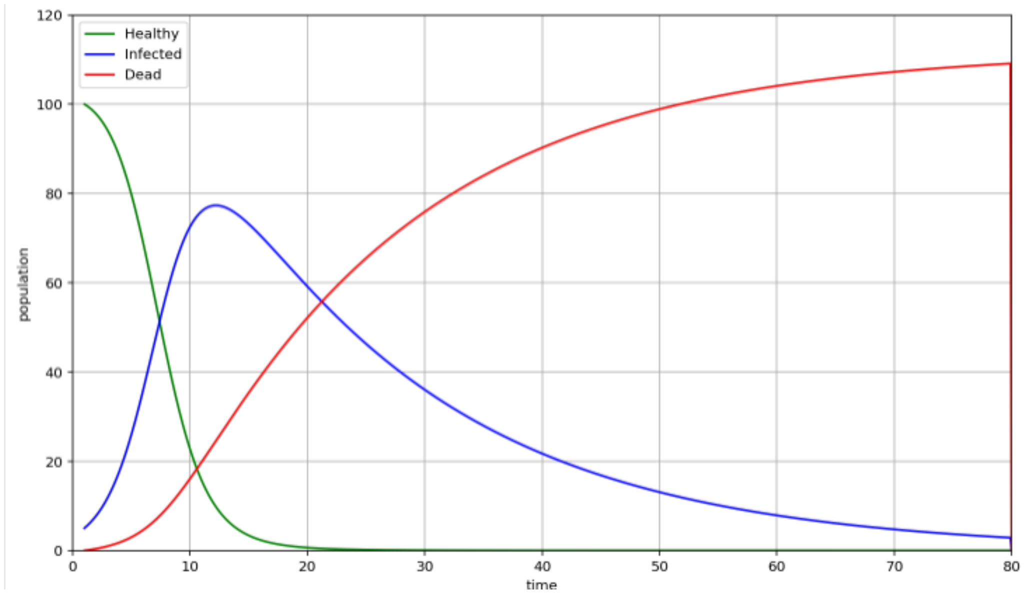


Figure 1. Population modeling using ODEs and Euler's Method. Input variables: birth = 0.01, natural death = 0.001, infection = 0.005, death = 0.05. Initlal populations: 100 health, 5 infected.

The example above reached a solution with 100% of the population dead, after which all derivatives result in 0, therefore the system doesn’t change. Using other parameters, we can reach non-trivial equilibria, with value fluctuating around certain values, as the population persists despite the disease. For example:

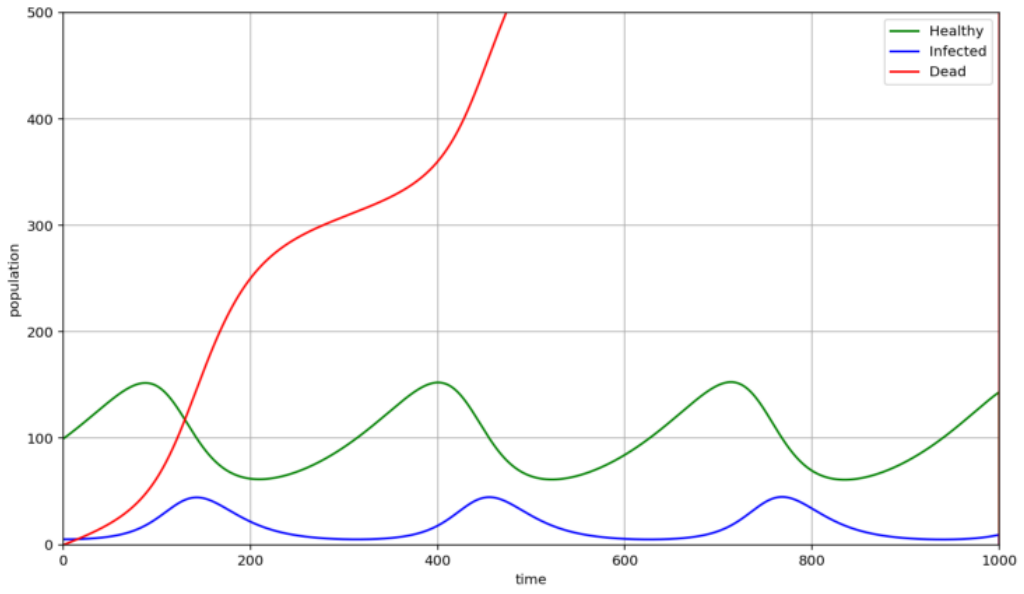


Figure 2. Fluctuating equilibrium. Input variables: birth = 0.01, natural death = 0.001, infection = 0.0005, death = 0.1. Initlal populations: 100 health, 5 infected.

To introduce immunization to the model, we can just introduce a population of people who cannot be infected, but without interaction with the healthy or infected, this population would not impact our results. Instead, we can speculate that the amount of people getting immunized is proportional to the healthy population and the infected population, since the more people infected the more fearful the healthy will become. So, we introduce population P, for "protected", and adjust the rest:

$$\mathbf{dP/dt = C_{imunization}*H*I - C_{nat\text{-}death}*P}$$

$$\mathbf{dH/dt = C_{birth}*H - C_{nat\text{-}death}*H - C_{infection}*H*I - C_{imunization}*H*I}$$

$$\mathbf{dD/dt = C_{nat\text{-}death}*H + C_{death}*I + C_{nat\text{-}death}*P}$$

Now, it's possible to eradicate the disease completely, without the entire population dying. For example:

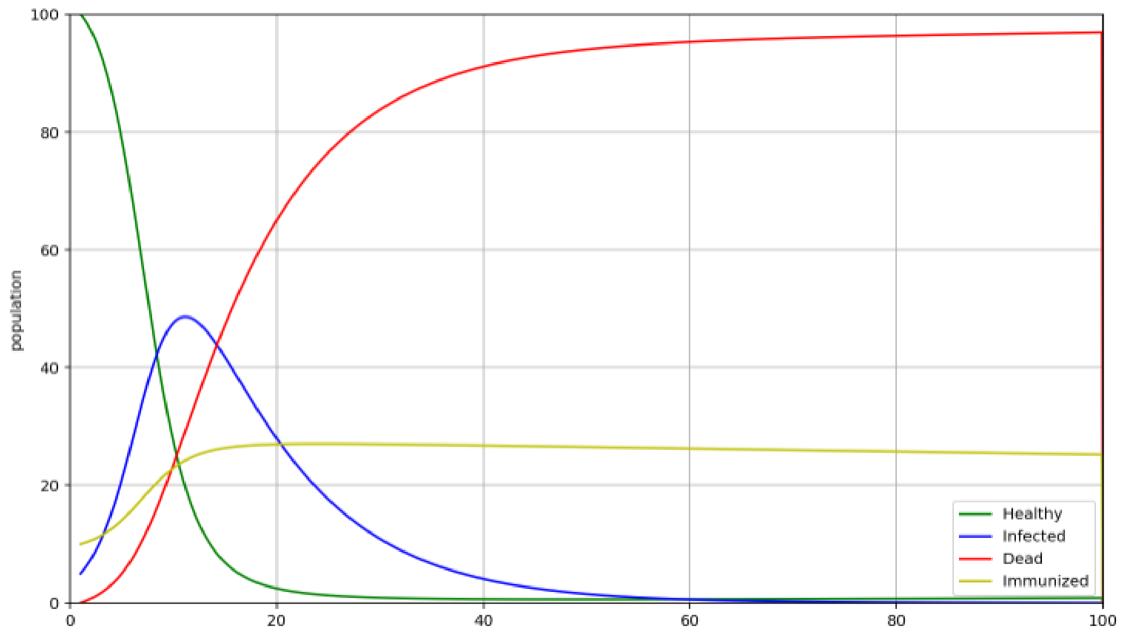


Figure 3. Elimination of disease by immunization. Input variables: birth = 0.01, natural death = 0.001, infection = 0.005, death = 0.1, immunization: 0.001. Initlal populations: 100 health, 5 infected, 10 immunized.

However, if we plot the same scenario for a longer time, we see an interesting phenomenon. A forgotten disease leads to less immunization, and as the unimmunized population grows, it sets the ground for a second outbreak. This is another form of oscillating equilibrium that is not a true solution.

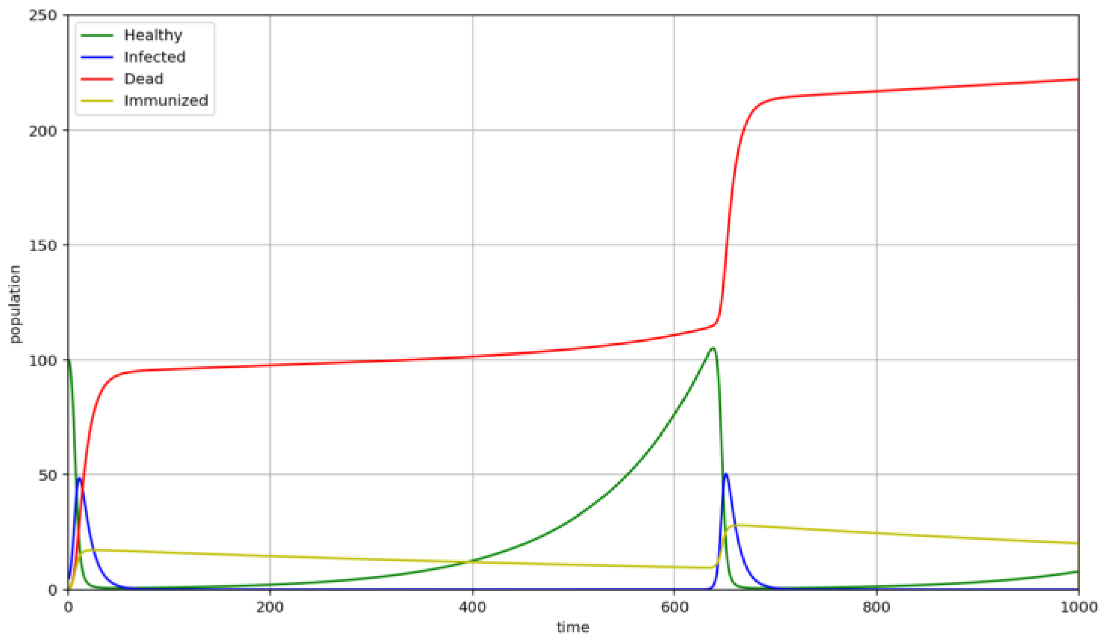


Figure 4. Repeated outbreaks. Input variables: birth = 0.01, natural death = 0.001, infection = 0.005, death = 0.1, immunization: 0.001. Initlal populations: 100 health, 5 infected, 10 immunized.

Herd Immunity: Markov Chains, Graphs and the Monte Carlo Method

One flaw of modeling populations with ODEs is that the simplification assumptions incorporated into the model don't allow us to examine emergent properties the same way less generalized models do. We know today that a key factor in immunization of populations is how the network properties of our natural society can give rise to "herd immunity", where when enough people are immunized in a population, the disease isn't able to spread even though there are infected – they just do not come in touch with susceptible people.

The deterministic representations in this sense are not modeling the system adequately. We can incorporate random elements in life mathematically using stochastic models. First, we examine Markov chains, that allow us to predict the state of our system after aggregating multiple instances of probabilistic state changes. To take into account how specific people can only infect people in their vicinity, we can look at a special case of Markov chains: the Stepping Stone model, where blocks are arranged in a grid, and for each time step, any block changes color to fit a random neighbor. We change the algorithm, to reflect a chance for the infected to infect the healthy, and the chance for any block to die. This Markov change encompasses a number of states equal to the amount of colors to the power of the number of squares, squared. This is impossible to represent in a traditional Markov tableau, but we can still solve the chain numerically to see the effect of immunity on the state the chain approaches over time. Below, we examine the effect of 100,000 steps, with an immunization probability of 10%, 30%, 50% and 80%, and other variables constant. We can see how immunized people can create safe enclosures into which the disease can't propagate. In the last example, the high mortality of the disease wipes it completely off the board, leaving an entirely healthy population behind.

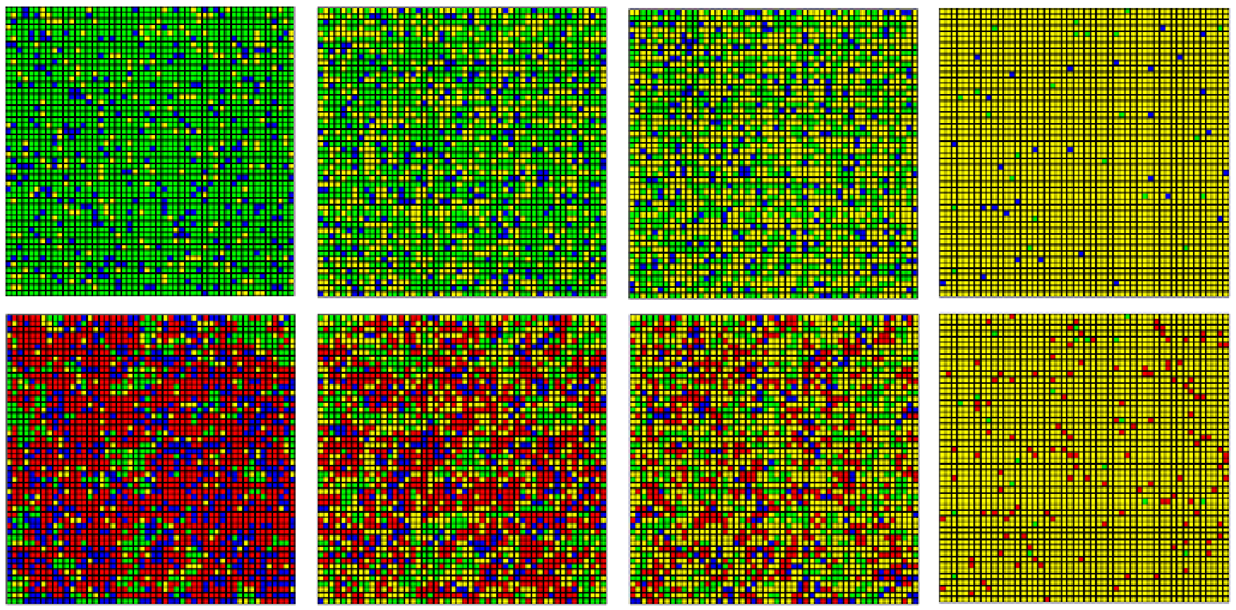
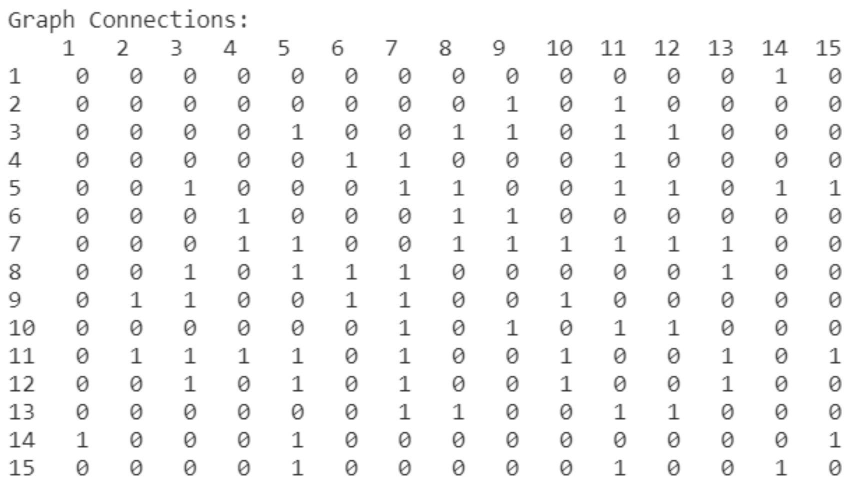


Figure 5. Simulated stepping stone Markov chain, with 10,30,50 and 80% immunized population. Green: healthy; Blue: infected; Red: dead; Yellow: Immunized. Population size = 250.

Since herd immunity is reliant on the specific configurations of relations between different people in a population, a useful tool for modeling this behavior is network graphs. We first devise an algorithm to generate an example population, with naturally occurring connections between each person. To simulate a functioning society, we distribute a "sociability" levels for the vertices in our graph, which determine their likelihood to form connections with one another. We can expect that some graphs would form with subgraphs who are not immunized but cut off completely from infected people, leading to herd immunity.

Here, in table form, we see all the edges connecting our randomly generated graph with 15 vertices. Rows and columns are vertices, intersections represent the existence of an edge between them:



```
Graph Connections:
    1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
2   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0
3   0   0   0   0   1   0   0   1   1   0   1   1   0   0   0
4   0   0   0   0   0   1   1   0   0   0   1   0   0   0   0
5   0   0   1   0   0   0   1   1   0   0   1   1   0   1   1
6   0   0   0   1   0   0   0   1   1   0   0   0   0   0   0
7   0   0   0   1   1   0   0   1   1   1   1   1   1   0   0
8   0   0   1   0   1   1   1   0   0   0   0   0   1   0   0
9   0   1   1   0   0   1   1   0   0   1   0   0   0   0   0
10  0   0   0   0   0   0   1   0   1   0   1   1   0   0   0
11  0   1   1   1   1   0   1   0   0   1   0   0   1   0   1
12  0   0   1   0   1   0   1   0   0   1   0   0   1   0   0
13  0   0   0   0   0   0   1   1   0   0   1   1   0   0   0
14  1   0   0   0   1   0   0   0   0   0   0   0   0   0   1
15  0   0   0   0   1   0   0   0   0   0   1   0   0   1   0
```

Figure 6. Randomly generated population graph. Input variables: pop_size = 15, p_l1 = 0.2, p_l2 = 0.3, p_l3 = 0.5, p2_l1 = 0.8, p2_l2 = 0.6, p2_l3 = 0.4.

Now, counting the number of transitions required to get from each edge to each other edge, can give us an idea of the vulnerability of this graph to disease. But running a path finding algorithm such as Dijkstra’s algorithm from each single vertex to every other can become computationally expensive, especially with large graphs. Instead, we can approximate our “vulnerability” quantity using stochastic methods, namely the Monte Carlo method. We choose a vertex at random to infect, a patient zero, and evaluate the impact of the effect by counting the number of vertex it propagates to in a set amount of time. If we average the performance of infections of many random vertices, we approximate the true value that would’ve been found in we were to infect every vertex. I ran the Monte Carlo algorithm with an increasing number of vertices, for populations with 10, 30, 50 and 80% immunization rate:

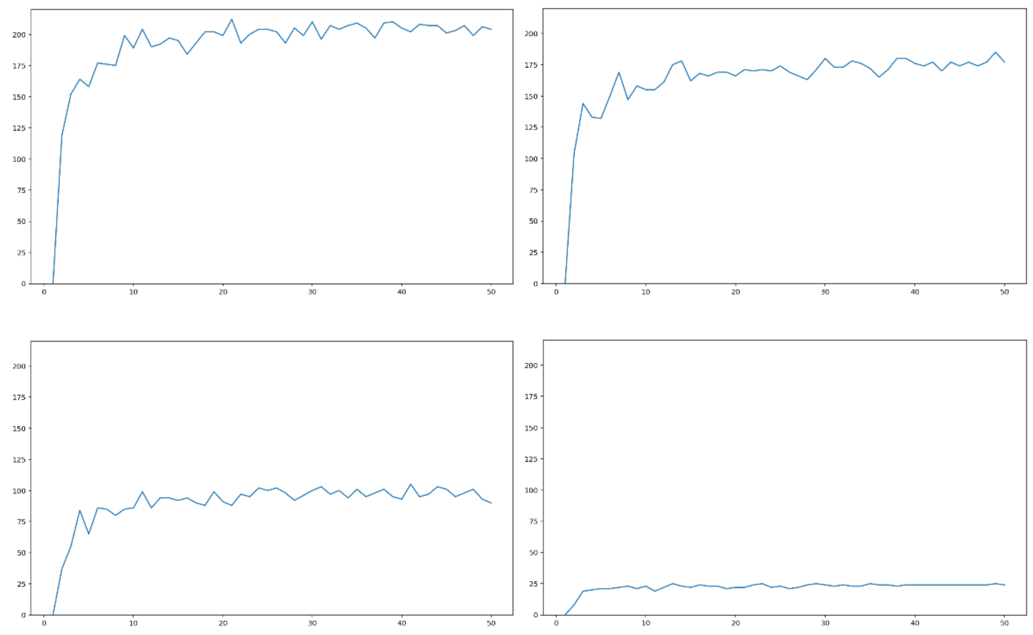


Figure 7. Conversion of Monte Carlo method on expected vulnerability of a population with 10, 30, 50 and 80% immunization. Y-axis represents infected people after 5 steps, and the X-axis represents the number of simulations aggregated. Population size = 250.

We can clearly see that Monte Carlo converges on a probabilistically expected factor that corresponds to the population vulnerability, even with as few as 20 vertices tested. It's also obvious that there's a nonlinear relationship between immunization rate and vulnerability, which is thanks to the herd immunity that arises from the network configuration.

Annex: Code

Python:

```
from numpy import *
from scipy import integrate
import pylab as plt
import random as rnd
import pandas as pd
#Euler's method
# Define parameters defining the behavior of the population
a = 0.01     # birth
b = 0.001     # natural death
c = 0.005     # infection
d = 0.1 # death
e = 0.001    # immunization
time = 1000
step = 0.1
stepN = int(time/step)+1
t = linspace(1, time, stepN)
h = zeros((1,stepN), dtype=float)
i = array(h)
x = array(h)
p = array(h)
h[0,0] = 100
i[0,0] = 5
p[0,0] = 0

for j in range(0,stepN-2):
    H_temp = h[0,j] + step*(a*h[0,j] - b*h[0,j] - c*i[0,j]*h[0,j]-
e*h[0,j]*i[0,j])
    I_temp = i[0,j] + step*(c*i[0,j]*h[0,j] - d*i[0,j])
    D_temp = x[0,j] + step*(b*h[0,j] + d*i[0,j] + b*p[0,j])
    P_temp = p[0,j] + step*(e*h[0,j]*i[0,j] - b*p[0,j])
    h[0,j+1] = H_temp
    i[0,j+1] = I_temp
    x[0,j+1] = D_temp
    p[0,j+1] = P_temp

plt.plot(t, h[0], 'g-', label='Healthy')
plt.plot(t, i[0], 'b-', label='Infected')
plt.plot(t, x[0], 'r-', label='Dead')
plt.plot(t, p[0], 'y-', label='Immunized')
plt.grid()
plt.legend(loc='best')
plt.xlabel('time')
plt.ylabel('population')
plt.xlim(0,time)
plt.ylim(0,250)
#random graph generation

pop_size = 15
p_l1 = 0.2
p_l2 = 0.3
p_l3 = 0.5
```

```
p2_l1 = 0.8
p2_l2 = 0.6
p2_l3 = 0.4

graph = zeros((pop_size,pop_size), dtype=int)

levels = zeros((pop_size))
for i in range (pop_size):
    rnd = rnd.random()
    if (rnd > p_l3):
        levels[i] = p2_l3
    elif (rnd > p_l2):
        levels[i] = p2_l2
    else:
        levels[i] = p2_l1

for i in range(pop_size):
    for j in range(i+1):
        if (i != j):
            rnd = rnd.random()
            if (rnd < levels[i]*levels[j]):
                graph[j,i] = 1
                graph[i,j] = 1

graph = pd.DataFrame(graph)
graph.index += 1
graph.columns += 1

print("Graph Connections:")
print(graph)

#vulnerability:
steps = 5
p0 = rnd.randint(1,pop_size)
infected = list()
infected.append(p0)
amount = 1
for step in range(1,steps):
    infected2 = list()
    for vert in infected:
        #print(vert)
        for edge in range(1,pop_size):
            #print(edge)
            if edge not in infected:
                if graph[vert,edge] == 1:
                    infected2.append(edge)
                    graph[vert,edge] = 0
    infected = infected + infected2

print(len(infected))
```

R:

N=50 #Size of the grid

```
NCols=4            #Number of Colors used

prob = c(0.1,0.1,0,08) #probability for initial population

interactions = c(0.001,0.3,0.05)

steps=100000

Mat=c()

cols=c("green","blue","red","yellow","blue","orange","purple","yellow","cyan","brown")

for(i in 1:(N*N)){

  Mat=append(Mat,sample(1:NCols,1,prob = prob)) #Randomly initialize the grid

}

M = matrix(Mat,nrow=N)

plot(x=NULL,y=NULL,xlim=c(0,N+1),ylim=c(0,N+1),axes=FALSE,xlab="",ylab="",asp=1)

for(i in 1:N){

  for(j in 1:N){

    symbols(i,j,squares=0.9,xlim=c(0,N+1),ylim=c(0,N+1),

            bg=cols[M[i,j]],add=TRUE,inches=FALSE)

  }} #Draw the grid initially

k = 0

while (k < steps){

  x=sample(1:N,1)

  y=sample(1:N,1) #Choose random cell
  v=sample(c(-1,0,1),1)

  h=sample(c(-1,0,1),1) #Choose random neighbor

  if(!(y==1 && v==-1)&&!(y==N && v==1)

     &&!(x==1 && h==-1)&&!(x==N && h==1)){

    if ((M[x+h,y+v]==2)&&(M[x,y]==1)&&(runif(1)<interactions[2])){

      M[x,y] <- 2

    }

    if (((M[x,y]==1)||(M[x,y]==4))&&(runif(1)<interactions[1])){

      M[x,y] <- 3

    }

    if ((M[x,y]==2)&&(runif(1)<interactions[3])){

      M[x,y] <- 3

    }

    symbols(x,y,squares=0.9,xlim=c(0,N+1),ylim=c(0,N+1),

            bg=cols[M[x,y]],add=TRUE,inches=FALSE)

  }

  k = k + 1

}
```