Analysis - Simulating disease spread through Cellular Automata and the SIR model DRAFT

Zebedee Marsh

January 4, 2021

Contents

1 Introduction

Since 1999, there have been 11 major disease outbreaks [1] across the globe, eight of which involved thousands of cases. To effectively control an outbreak and ultimately stop it in its tracks, it's current and past states must be analysed so it's spread can be predicted, which is essential in influencing how governments should deal with such a threat.

This is currently being seen in how the UK government is dealing with COVID-19, where a balance between a total lockdown and total freedom needs to be struck to control how the virus spreads. This balance is essential in allowing businesses to continue operating, keeping the economy afloat and allowing people their freedoms to do what they want.

My project aims to provide a starting point for where a disease outbreak can be modelled either by cellular automata or the SIR model, which can be used in an educational environment to teach people about the spread of diseases. A user will be able to input custom parameters according to the disease they are trying to simulate and be able to introduce special events such as a partial or permanent lockdown, to be able to see how the disease spread could change. A user should also be able to compare the two different ways of simulating a disease.

Source: https://sundial.csun.edu/156361/news/a-timeline-of-outbreaks-from-2000-to-present/

2 Initial objectives

These objects represent my initial aims of what this project should be able to do. These may change later on throughout this project.

- Simulate the spread of a disease using the SIR model
- Simulate the spread of a disease using the Cellular Automata model
- Input custom parameters or use default values (from a previous disease) including infection rate, incubation period and death rate for the SIR model
- Input custom parameters or use default values (from a previous disease) relating to the number and infectivity of cells for the CA model
- Set a quarantine period or introduce a vaccine (therefore limiting disease transmission)
- Click on a graph and show statistics from that point in time
- Export graphs produced as png
- Upload past simulation results into the program to generate a graph from that previous simulation
- Be able to compare graphs of two different diseases

3 Potential user

Mr Bliss - Biology teacher at Abingdon School As this model provides a starting point for modelling disease outbreak, it's most effective use would be in an education environment, where it is used to teach people about how disease can spread and how changes in parameters such as the infection rate, recovery rate, time of infection and immunity can affect how effectively a disease is transmitted throughout a cohort.

4 How the problem was researched

To first understand how I could make this project, I had to research the two main methods of modelling disease spread, the SIR model and Cellular Automata. Once I got the basic idea of how each model worked, I further researched how I could implement these in Python, as well as looking at how to use some Python libraries to help me.

- Find out how SIR and Cellular Automata work
- Research how to implement SIR and Cellular Automata in code
- Learn how to use the Matplotlib library to draw graphs for my disease models
- Learn how to use Pandas to export data to an external file
- Learn how to use Tkinter to create a GUI interface and multiple pages
- Learn how to create, use and modify an SQL database using Sqlite3

4.1 The SIR model

The SIR model is a simple disease spread model which uses differential equations to determine the number of people in three categories, the susceptible, the infected and the recovered. The susceptible population represent those who can become infected a disease, the infected population represent those currently infected with the disease who are able to spread it to other people and the recovered population represent those who are no longer infected. In some models, people in the recovered population can be infected again.

4.1.1 Equations for the SIR model

People in each group

$S = S(t)$ - number of susceptible individuals

$I = I(t)$ - number of infected individuals

$R = R(t)$ - number of recovered individuals

The Susceptible equation

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

where β is the rate at which infections occur (a positive constant)

The Infected equation

$$\frac{dI}{dt} = \frac{\beta S(t) I(t)}{N} - \gamma I(t)$$

where γ is the rate at which people recover

The Recovered equation

$$\frac{dR}{dt} = \gamma I(t)$$

Model without vital dynamics

$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = 0$$

As this model doesn't take into account vital dynamics ie. people aren't added through birth or removed through death, the sum of all the changes is constant and equal to 0

4.2 The Cellular Automata model

The basic idea of cellular automata that there there is a collection of cells with a finite set of states. Every new generation, the state of a cell can be changed depending on the state of it's neighbouring cells.

4.2.1 My implementation of CA

- Cells move randomly within a grid. Most cells will be susceptible but some will be infected
- If a susceptible cell comes in close range to an infected cell, there is a chance that that susceptible cell will be come infected
- Infected cells are only infected for a certain period of time, so after a certain number of generations the cell will no longer be infected and will be in the recovered category

5 Prototyping of code

5.1 The scientific models

5.1.1 Elementary cellular automata

My initial cellular automata model was very basic and only worked on a 1D line. The shape of each new line would be decided by looking at the previous line and applying a ruleset to it.

```python
class CA:
    def __init__(self, width, generations):
        self.ruleset = [0, 1, 1, 1, 1, 0, 1, 1]
        self.width = width
        self.cells = [0] * width
        self.new_cells = [None] * width
        self.cells[int((width / 2) - 1)] = 1
        self.generations = generations
        self.cell_timeline = []

    def rules(self, a, b, c, i):
        index = int((a + b + c), 2)
        ruleset = [0, 1, 0, 1, 1, 0, 1, 0]
        self.new_cells[i] = ruleset[index]

    def draw(self):
        output = ""
        for cell_state in self.cell_timeline:
            for element in cell_state:
                if element == 0:
                    output += " " + " "
                else:
                    output += "X" + " "
            print(output)
            time.sleep(0.1)
            output = ""

    def ca(self):
        for gen in range(self.generations):
            for i in range(self.width):
                a = str(self.cells[((i - 1) % self.width)])
                b = str(self.cells[i])
                c = str(self.cells[((i + 1) % self.width)])
                # print(a,b,c)
                self.rules(a, b, c, i)
            self.cells = self.new_cells
            self.cell_timeline.append(self.cells)
            self.new_cells = [None] * self.width
        self.draw()

ca = CA(100, 100)
ca.ca()
```

The result of elementary cellular automata

```
                              x x
                             x   x
                            x x x x
                           x       x
                          x x     x x
                         x   x   x   x
                        x x x x x x x x
                       x               x
                      x x             x x
                     x   x           x   x
                    x x x x         x x x x
                   x       x       x       x
                  x x     x x     x x     x x
                 x   x   x   x   x   x   x   x
                x x x x x x x x x x x x x x x x
               x                               x
              x x                             x x
             x   x                           x   x
            x x x x                         x x x x
           x       x                       x       x
          x x     x x                     x x     x x
         x   x   x   x                   x   x   x   x
        x x x x x x x x                 x x x x x x x x
       x               x               x               x
      x x             x x             x x             x x
     x   x           x   x           x   x           x   x
    x x x x         x x x x         x x x x         x x x x
   x       x       x       x       x       x       x       x
  x x     x x     x x     x x     x x     x x     x x     x x
 x   x   x   x   x   x   x   x   x   x   x   x   x   x   x   x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
```

5.1.2 Basic SIR model

My starting block for the SIR model was a function which calculated the rate of change of the susceptible, infected and recovered population.

```
def eqns(y0, t, beta, gamma):
S, I, R = y0
dsdt = -(beta * S * I) / N # rate of change of susceptible individuals
didt = ((beta * S * I) / N) - gamma * I # rate of change of infected individuals
drdt = gamma * I # rate of change of recovered individuals
return dsdt, didt, drdt
```

5.2 External library testing TBD

5.2.1 Tkinter

My intial prototying for tkinter was done with the help of a YouTube video which focused on having multiple tkinter pages in a single file. Link: https://youtu.be/jBUpjijYtCk

This prototype allowed there to be multiple Tkinter windows which could be switched between by a button click. The way this method worked was by removeing and adding widgets depending on which page should be shown. Although this method may have worked for me, in the final project I went with destroying a Tkinter window and starting a completely new one when a user wanted to switch, as the code for this was easier to understand and write.

```
import tkinter as tk


class Gui(tk.Tk):
   def __init__(self, *args,
                  **kwargs): # *args - pass through any number of variables; **kwargs
                      - pass through dictionaries
       tk.Tk.__init__(self, *args, **kwargs) # initialising tkinter
       container = tk.Frame(self) # the frame of the window
       container.pack(side="top", fill="both", expand=True)
       container.grid_rowconfigure(0, weight=1) # 0 sets minimum size, weight shows
           priority
       container.grid_columnconfigure(0, weight=1)
       self.frames = {} # allows application to open different types of pages easily

       for F in (StartPage, PageOne): # all pages need to be listed here
           frame = F(container, self)
           self.frames[F] = frame # saving classes to dictionary, "loading it in"
           frame.grid(row=0, column=0, sticky="nsew")

       self.show_frame(StartPage) # start page is shown first

   def show_frame(self, cont):
       frame = self.frames[cont] # looks for cont in dict
       frame.tkraise() # which is then raised


class StartPage(tk.Frame):

   def __init__(self, parent,
                  controller): # parent class is Gui, controller class is main class,
                      allowing show_frame to be called
       tk.Frame.__init__(self, parent)

       label = tk.Label(self, text="Start Page")
       label.pack(pady=10, padx=10)
       button1 = tk.Button(self, text="Page 1", command=lambda:
           controller.show_frame(PageOne))
       button1.pack()


class PageOne(tk.Frame):

   def __init__(self, parent, controller):
       tk.Frame.__init__(self, parent)

       label = tk.Label(self, text="This is page 1")
       label.pack(pady=10, padx=10)
       button2 = tk.Button(self, text="Start Page", command=lambda:
           controller.show_frame(StartPage))
       button2.pack()


app = Gui()
app.mainloop()
```

The result

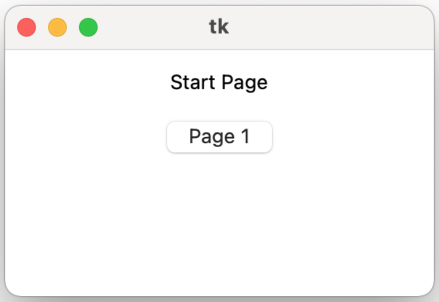


Figure 1: Start Tkinter window

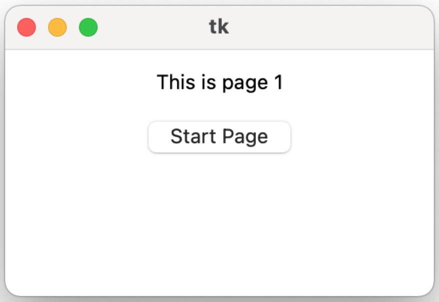


Figure 2: Page 1 Tkinter window

5.2.2 Matplotlib - FuncAnimation

The initial way I tried to update a graph with new values was by deleting the graph and plotting a new one. Funcanimation provided an easier way to do this, with the option of saving the animation as a video which was one of the objectives for my project. This prototype shows how a line graph works with funcanimate.

It works for both line graphs and scatter graphs

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from itertools import count
import random

x_full = [[229, 29, 52, 90, 78], [228, 29, 52, 90, 79], [229, 28, 53, 89, 79],
    [228, 27, 52, 90, 78], [227, 28, 51, 89, 79], [227, 27, 50, 88, 78], [226,
    28, 51, 88, 79], [227, 29, 51, 89, 79], [227, 30, 51, 90, 79], [228, 30, 51,
    90, 79], [229, 30, 50, 89, 78], [228, 30, 49, 90, 79], [228, 30, 50, 89, 78],
    [228, 31, 50, 88, 78], [229, 32, 49, 87, 78], [230, 33, 50, 87, 77], [230,
    34, 50, 88, 76], [231, 35, 51, 87, 75], [232, 36, 50, 87, 74], [232, 37, 51,
    86, 74]]
y_full = [[37, 23, 94, 195, 90], [37, 22, 93, 194, 89], [36, 22, 94, 195, 88],
    [36, 22, 95, 194, 88], [35, 22, 94, 195, 88], [36, 22, 95, 196, 89], [35, 21,
    94, 197, 90], [34, 20, 94, 196, 91], [34, 20, 94, 197, 90], [35, 19, 94, 196,
    90], [35, 18, 95, 197, 89], [35, 18, 94, 198, 88], [35, 19, 93, 197, 89],
    [35, 20, 94, 196, 90], [34, 20, 95, 196, 90], [33, 19, 95, 195, 90], [33, 19,
    96, 196, 91], [33, 20, 95, 196, 90], [32, 20, 94, 195, 90], [31, 19, 93, 194,
    91]]

x_vals = []
y_vals = []

index = count()

def animate(i):
    x_vals.append(next(index))
    y_vals.append(random.randint(0,5))
    plt.plot(x_vals, y_vals)
```

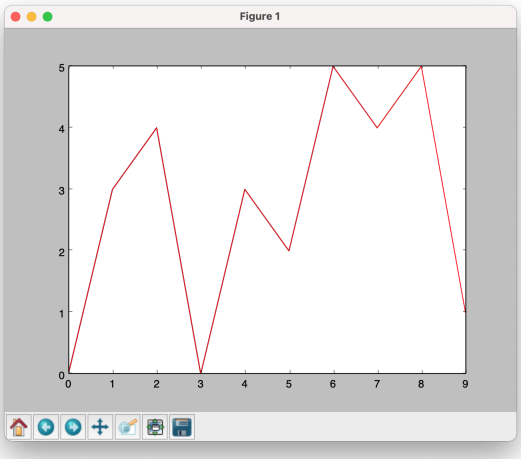


Figure 3: Graph 1

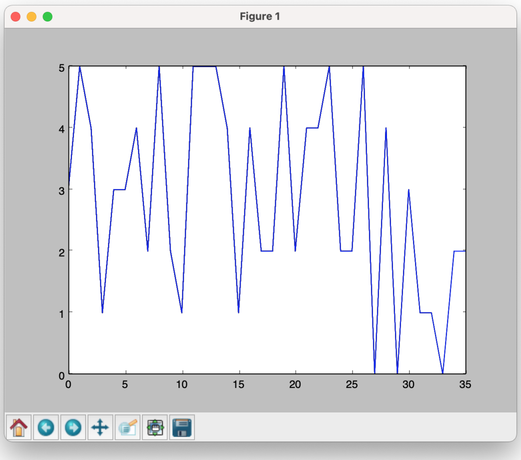


Figure 4: Graph 2

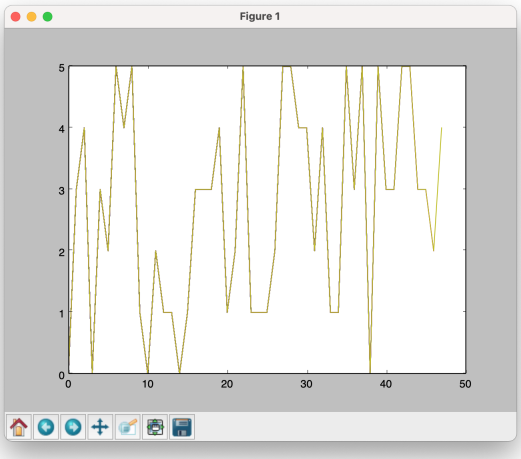


Figure 5: Graph 3

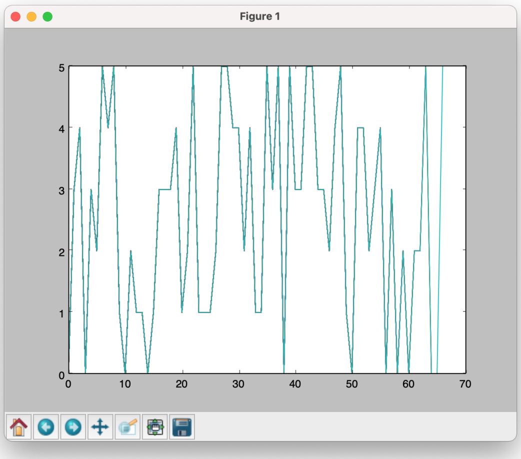


Figure 6: Graph 4

```
    ani = FuncAnimation(plt.gcf(), animate, interval=10) # get current figure,
        function, interval
    plt.show()
```

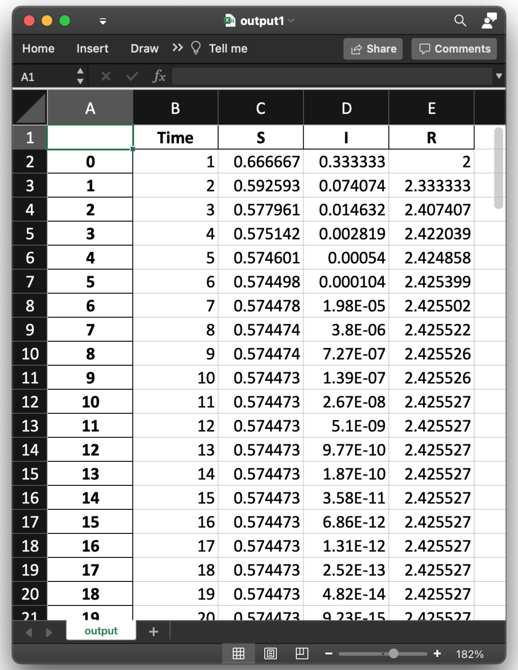
The result

5.2.3 Pandas

I used Pandas to help export the data generated using the SIR model. This allowed me to generate an excel file with appropriate column names, which could possibly be used later for data analysis

```
def export_to_excel():
data = {'Time': timearray,
        'S': solver_result[0],
        'I': solver_result[1],
        'R': solver_result[2]}

df = pd.DataFrame(data, columns=['Time', 'S', 'I', 'R'])
name = "output" + str(f) + ".xlsx"
df.to_excel(name, sheet_name='output')
```



	Time	S	I	R
0	1	0.666667	0.333333	2
1	2	0.592593	0.074074	2.333333
2	3	0.577961	0.014632	2.407407
3	4	0.575142	0.002819	2.422039
4	5	0.574601	0.00054	2.424858
5	6	0.574498	0.000104	2.425399
6	7	0.574478	1.98E-05	2.425502
7	8	0.574474	3.8E-06	2.425522
8	9	0.574474	7.27E-07	2.425526
9	10	0.574473	1.39E-07	2.425526
10	11	0.574473	2.67E-08	2.425527
11	12	0.574473	5.1E-09	2.425527
12	13	0.574473	9.77E-10	2.425527
13	14	0.574473	1.87E-10	2.425527
14	15	0.574473	3.58E-11	2.425527
15	16	0.574473	6.86E-12	2.425527
16	17	0.574473	1.31E-12	2.425527
17	18	0.574473	2.52E-13	2.425527
18	19	0.574473	4.82E-14	2.425527
19	20	0.574473	9.23E-15	2.425527

5.2.4 Pygame

One of the options I considered to simulate and show my Cellular Automata simulation was Pygame. With Pygame I would be able to easily create cells which could move randomly in a set space, and using Pygame's collision method I could detect if cells touched. In the end I decided against this method as although it may have been easier, using Matplotlib graphs would grant me extra flexibility, for example if I wanted to export position data of each cell I would already know how to do that with Matplotlib. The prototype I tested was made with the help of a YouTube video covering Pygame: https://youtu.be/NjvIooRpuH4

```
import pygame
import time

pygame.init()

black = (0,0,0)
white = (255,255,255)
red = (255,0,0)

gameDisplay = pygame.display.set_mode((800,600))
pygame.display.set_caption('Test')
```

```
clock = pygame.time.Clock()

carImg = pygame.image.load('unnamed.jpg')

def text_objects(text, font):
    textSurface = font.render(text, True, black) # what to render, anti-aliasing,
        colour
    return textSurface, textSurface

def message_display(text):
    largeText = pygame.font.Font('freesansbold.ttf', 115)
    TextSurf, TextRect = text_objects(text, largeText)
    TextRect.center = (800/2, 600/2)
    gameDisplay.blit(TextSurf, TextRect)

    time.sleep(2)

    game_loop()

def crash():
    message_display('You crashed')

def car(x,y): # car object
    gameDisplay.blit(carImg, (x,y)) # drawing carImg to x,y

def game_loop():

    x = (100)
    y = (100)
    x_change = 0

    gameExit = False

    while not gameExit: # user controls
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                gameExit = True

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x_change = -10
                elif event.key == pygame.K_RIGHT:
                    x_change = 10

            if event.type == pygame.KEYUP:
                if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                    x_change = 0

            print(event)

        x += x_change

        gameDisplay.fill(white)
        car(x,y)
        if x > 800 or x < 0: # boundaries
            crash()

        pygame.display.update()
        clock.tick(60)

game_loop()
pygame.quit()
```

5.3 SQL database TDB

6 Research of existing solutions TDB

TBD

7 Interview with target user

Q1. How could such a tool like this be used in the classroom?

A1. As a comparator of diseases; To be able to decide the best method to restrict a disease; To possibly predict the start of a second spike

E1. I will try to implement a feature where the parameters of two diseases could be entered, and a graph could be calculated and shown for both, which would allow a user to compare the spread of the two diseases.I will also add features that a government may try and implement to restrict a disease. This could include a lockdown feature or a vaccine introduction which would allow a user to see how the spread of a disease could change.

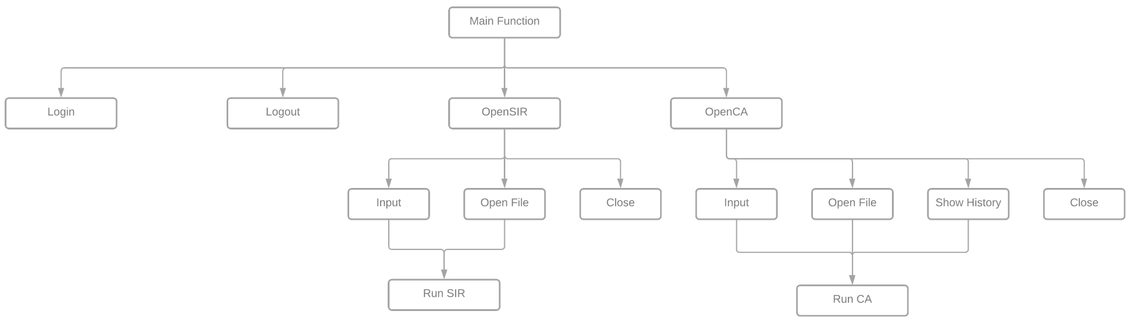
Q1. How could such a tool like this be used in the classroom?

A1. As a comparator of diseases; To be able to decide the best method to restrict a disease; To possibly predict the start of a second spike

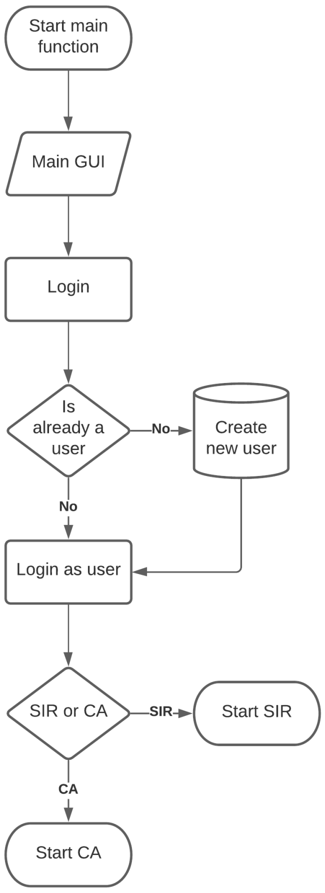
E1. I will try to implement a feature where the parameters of two diseases could be entered, and a graph could be calculated and shown for both, which would allow a user to compare the spread of the two diseases. I will also add features that a government may try and implement to restrict a disease. This could include a lockdown feature or a vaccine introduction which would allow a user to see how the spread of a disease could change.

8 Documented design COMMENTED OUT

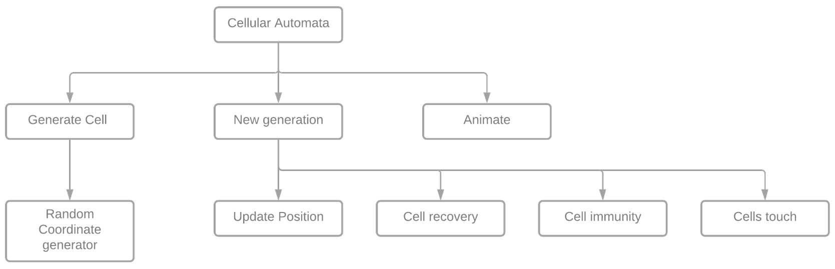
8.1 Main function structure diagram



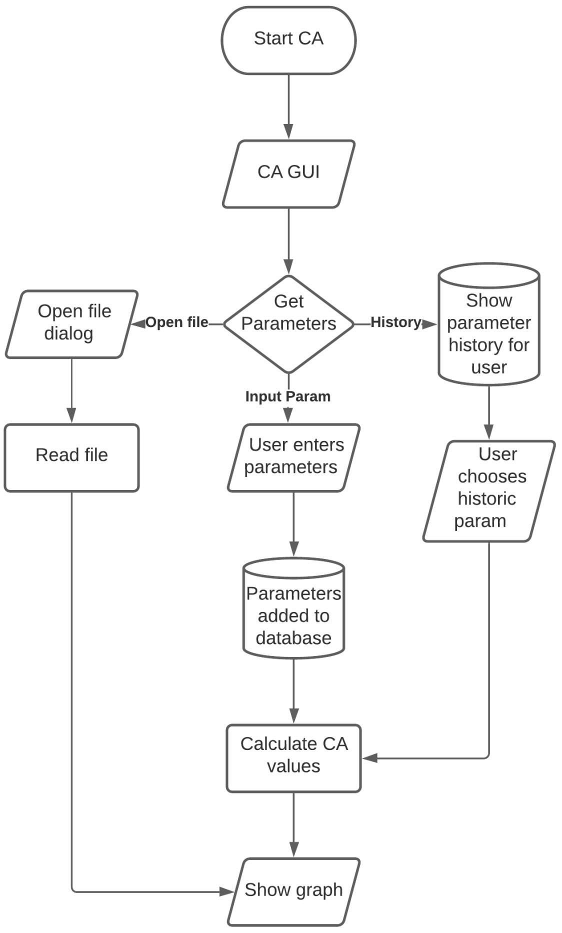
8.2 Main function flowchart



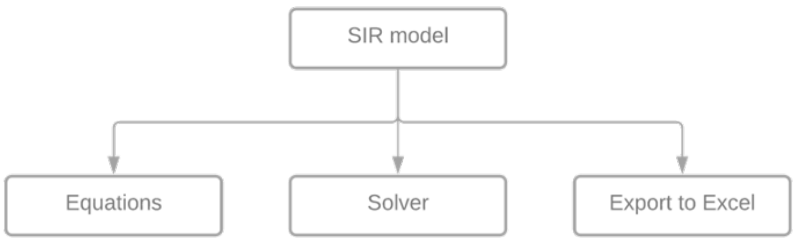
8.3 Cellular automata structure diagram



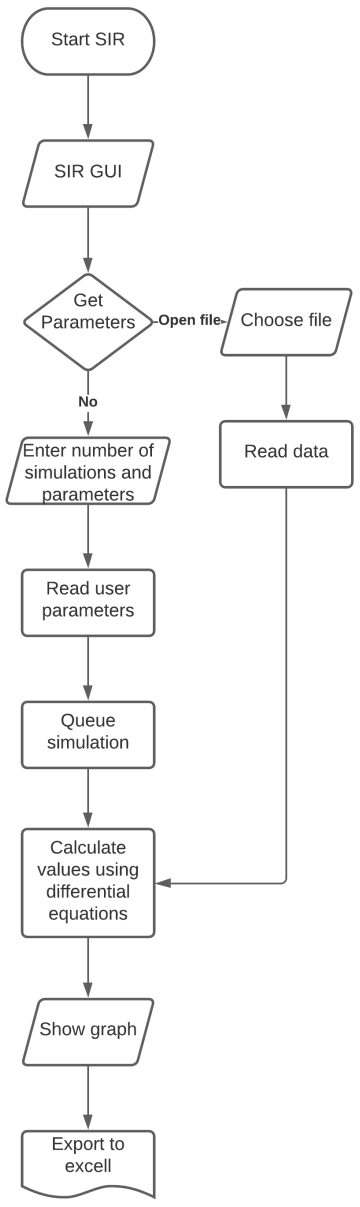
8.4 Cellular automata flowchart



8.5 SIR Model structure diagram



8.6 SIR model flowchart



9 The code COMMENTED OUT

References

[1] Gillian Moran-Perez. A timeline of outbreaks from 2000 to present, Feb 2020.