Contents

[Analysis: 2](#_Toc36818712)

[Background of the problem: 2](#_Toc36818713)

[Interview: 2](#_Toc36818714)

[What was found out in the interview: 3](#_Toc36818715)

[My thoughts of the needs of my client: 3](#_Toc36818716)

[Investigation: 3](#_Toc36818717)

[What problems will need to be solved in my code: 5](#_Toc36818718)

[Objectives: 5](#_Toc36818719)

[IPSO Chart: 7](#_Toc36818720)

[Documented design: 7](#_Toc36818721)

[Structure of the software: 8](#_Toc36818722)

[Top down diagram: 9](#_Toc36818723)

[Interface design: 9](#_Toc36818724)

[Data dictionary: 10](#_Toc36818725)

[IPSO Chart: 11](#_Toc36818726)

[Classes: 12](#_Toc36818727)

[Algorithms: 13](#_Toc36818728)

[Euler’s method: 13](#_Toc36818729)

[CheckStatus: 14](#_Toc36818730)

[Move: 14](#_Toc36818731)

[Merge sort: 14](#_Toc36818732)

[Main window cycle: 15](#_Toc36818733)

[Technical solution: 15](#_Toc36818734)

[Procedure listings: 15](#_Toc36818735)

[Interface design: 17](#_Toc36818736)

[Parameter entry window: 17](#_Toc36818737)

[Invalid data window: 17](#_Toc36818738)

[Simulation: 17](#_Toc36818739)

[Graph: 18](#_Toc36818740)

[Statistics page: 18](#_Toc36818741)

[Code: 19](#_Toc36818742)

[Testing: 25](#_Toc36818743)

[Evaluation: 29](#_Toc36818744)

[How did I meet my objectives? 29](#_Toc36818745)

[Feedback from user: 31](#_Toc36818746)

[Analysis of feedback: 32](#_Toc36818747)

# Analysis:

## Background of the problem:

The client Mrs Stern is interested in modelling the spread of disease but can’t find a suitable solution on the internet. Any simulation she can find is a compromise in some way. What she needs is a simulation which can take user input, show the spread of a disease on a population and graph it simultaneously.

## Interview:

An interview was conducted:

Q1: Can you outline the main things you want the program to do?

A1: “I would like to be able to model the spread of disease in a visual way, I would like a graph of how the population varies over time. I want to be able to compare a theoretical disease model to a computer simulation”

Q2: Why do you need this program?

A2: “I am interested in modelling the spread of disease but couldn’t manage to find, any nice simulations on the internet”.

Q3: What did you not like in any existing software you found online?

A3: I could only find 2 simulations, and the population in them were stationary, and I couldn’t control things such as the population size. A lot of features I wanted were missing, and they didn’t look very nice. I couldn’t find any that predicted the results mathematically”

Q4: What kind of user input would you like the program to have?

A4: “I would like to be able to be able to determine things such as the initial population, recovery rate, and likelihood of infection. I want to be able to set my own initial conditions for the simulation and then have it run based on those conditions.”

Q5: What would you want the simulation to look like?

A5: “I would want a way to see all the population, and how they move and interact with each other.”

Q6: What kind of interaction do you want the population to have with each other?

A6: “I would like them to be able to move around and encounter each other, so that way they will be able to infect other people or get infected themselves. Now that you mention it, I would also like to be able to have some sort of control over how much the people con move around.”

Q7: You mentioned that you would like a graph to be produced, can you elaborate on that?

A7: “I want to have a graph in the same window of the simulation that plots in real time how many people are infected, that way I can see a trend more clearly. At the end I want to be able to compare the graph produced with a mathematical prediction”

Q8: Is there anything else you would want the program to do?

A8: “I would like the simulation to end once the last infected person recovers. A window with statistics at the end of the simulation would be nice. “

## What was found out in the interview:

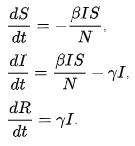
During this interview my client made it clear that she could not find a suitable program for her needs. Any solution she could find was missing some features that are essential to her, and that’s why she wants this program to be written for her. The program must except user input that will influence the simulation, and that it should have a graph updating in real time alongside the simulation. A mathematical prediction must also be implemented. The needs of the client are very clear.

## My thoughts of the needs of my client:

My client would benefit from having a window just made for entry of parameters. Having a graph update on the same window of the simulation will help to visualise what is happening in the simulation. My client seems to have made their needs clear apart from that.

## Investigation:

In order to represent a disease in an abstract way online research was conducted to see how other simulations have done this. It seems that all of them have used a 3 or 4 compartment model. This means that any person can be represented by one of 3 (or 4) states. In this case the states are susceptible, infected and recovered (and dead). Since the client hasn’t mentioned her desire for the simulation to be of a lethal disease, the program will use a 3-compartment model. This particular 3-compartment model is known as the SIR model. This is a common model for the spread of disease and can be modelled by 3 ordinary differential equations:



This series of equations rely on 2 inputted values: beta and gamma.

Beta - The average number of contacts per person times the probability of infection in a contact between a susceptible and infected person.

Gamma – The rate of recovery or the rate of death, if we call the duration of the infection D, then gamma = 1/D.

The simulation will use this model. When the simulation runs it won’t use these equations, however they will be used in order to create a prediction. Since an exact analytical solution can’t be found to these equations, a numerical method will be needed to approximate the solution.

In order to solve these equations numerically, Euler’s method will be used. For a differential equation:

Using this method, these formulas can be derived to solve the set of equations numerically:

In order to calculate , all that is needed is to set it to

Calculating , is a little more difficult as

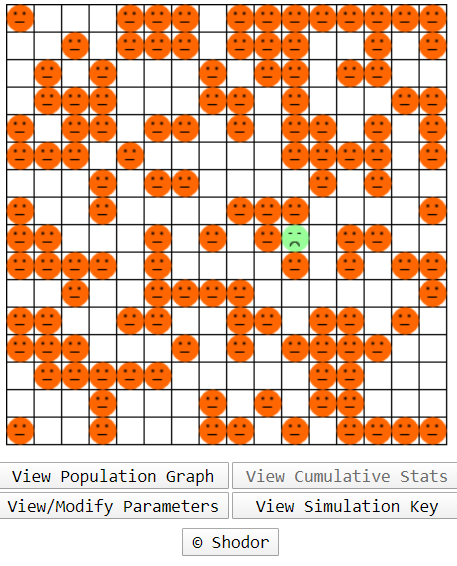
Probability of infection will be entered by the user, however contact rate will have to be calculated.

The equation that was created is:

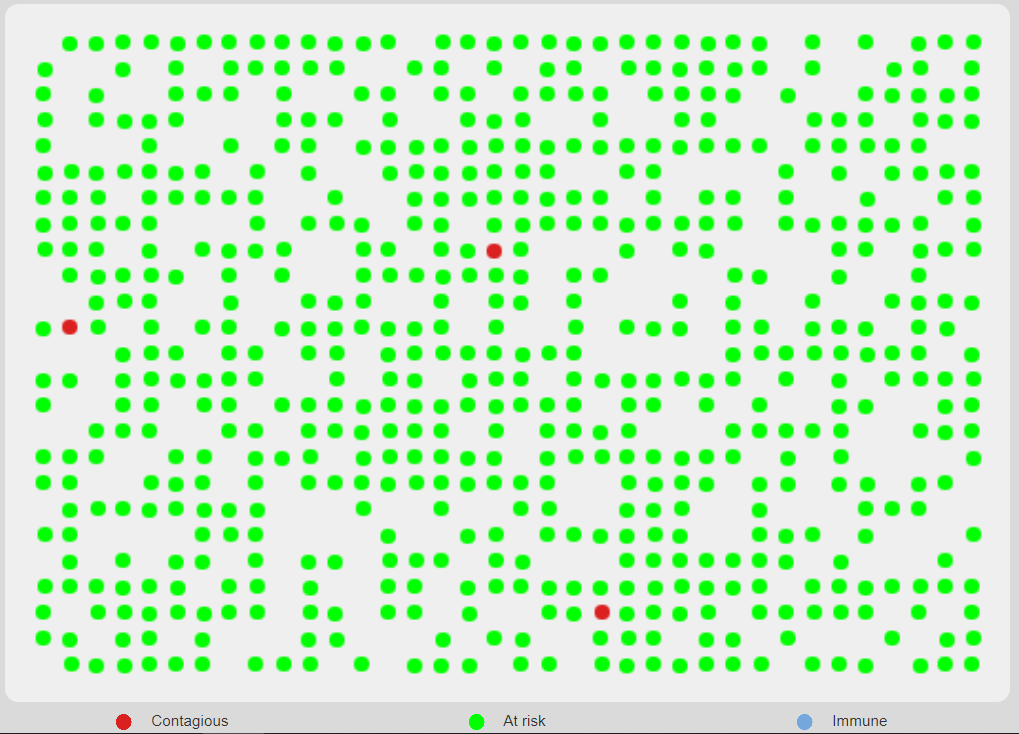
I will be setting

Research was also done on approaches of existing simulations.

These are the simulation windows of simulations that were found online:



In this simulation the population is in a grid, so their movement is somewhat restricted, but people are represented in a simple and easy to understand way.



In this simulation the population is stationary but is also represented in a similar way as above.

This seems like a good way to represent the compartments of the SIR model.

The program will represent a person as a circle which can be in 1 of 3 colours which represent the compartments of the SIR model:

Green- susceptible

Red- infected

Blue- recovered

## What problems will need to be solved in my code:

In order to make the program, some key problems need to be solved:

- Be able to populate a window with people who will be generated.

- Have the population move at random, but relative to their previous position.

-Make sure the population stay within confined boundaries.

- Check distances between all people on screen every time someone moves

-Make sure that people recover with immunity and can’t be infected with the disease for a second time. This will probably mean changing an attribute of a person.

-People need to recover after a certain number of days, I will have to keep track of how many days a person is infected.

- Create a parameter entry window and have a button to save and run.

-Have a graph alongside the simulation update in real time.

- Implement Euler’s method.

-Know when to stop the simulation.

## Objectives:

1. Have a window with people moving and encountering each other.

* Population must be moving and not static
  + Where and by how much they move must be random, but within a certain range inputted by the user.
  + There must be borders to the simulation, so no person can leave the confined borders.

1. Each person can be in one of 3 states: susceptible, infected or recovered.

* Must follow the 3 compartments of the SIR model
* Each state must be represented by a colour on a person so you can tell who’s in what state.
* Once a person changes state their colour must change on screen.

1. The user can input parameters that will influence the outcome of the simulation.

* Have a ‘setup’ window for the user to enter all the parameters before the simulation starts.
* The user should be able to input 5 parameters:
  + Population size
  + Radius of infection
  + How much a person can move every cycle
  + Probability of infection
  + Rate of recovery
* Validate the data being inputted
  + Population, infection radius, movement range and rate of recovery – integer
  + Probability of infection – float
  + Have an invalid data window

1. Create a prediction of the outcome of the simulation

* Implement Euler’s method to solve the differential equations.
* Calculate a suitable beta and gamma value from the users input.

1. Generate the population requested by the user.

* Initial X and Y coordinates of every person should be randomly generated.
* Everyone should be susceptible once they are created.
* Once the population is generated, infect one person.

1. A person can catch the disease from a nearby person.

* The radius of infection will be the maximum distance 2 people must be apart in order to have a chance of getting infected.
* If a person is a is close enough to get infected, a random number will be compared to the probability of infection, and if it is lower than the person will be infected.

1. Graph the number of susceptible, infected and recovered individuals against time.
   * The 3 different curves will be displayed on the same set of axes.
   * Display a key that shows which curve represents witch state.
     + Each curve will have a different colour.
     + The colour of each curve will be the same as the colour coding in the simulation.

* Plot the number of people in each state as they get infected/recover.
* The graph and the simulation should be displayed on the same page.
* The simulation and the graph must run simultaneously, and the graph must update in real time.

1. The simulation should stop once the infection is eradicated.

* Once there are no more infected people, the main loop should terminate.
* There should be some variables to keep track of the number of people in each state.

1. An infected person should recover in the number of days specified by the user.

* Keep track of how many days a person has been infected for.
* Check for recovery every cycle.

1. Display a statistics page at the end of the simulation.

* Should be displayed once the simulation stops.
* It should display the parameters entered.
* It should display the peak number of infected, total number infected and days passed since the start.
* Display both predicted and actual graph side by side.

## IPSO Chart:

|  |  |  |
| --- | --- | --- |
| IPSO | Information | Evidence |
| Input | User inputted parameters:  - Population size  - Infection radius  -Probability of infection  -Movement range  - Rate of recovery | Interview |
| Process | Generate population with the number given by the user. | Interview |
| Process | Move the population based on the movement range given by the user. | Interview |
| Process | Calculate whether a person is getting infected or recovering based on rate of recovery, probability of infection and infection radius. | Interview |
| Process | Use Euler’s to solve the differential equations and predict the outcome of the simulation | Interview |
| Output | Display the people on screen moving and changing state based on the user inputted parameters | Interview |
| Output | Plot a graph of the states of the people in the simulation | Interview |
| Output | Display statistics page at the end of the simulation with statistics from the simulation, and a side by side comparison of the 2 graphs produced | Interview |

# Documented design:

## Structure of the software:

Objective 3

Parameter entry window

No

Invalid data window

Is data valid?

Yes

Objective 3

Generate population

Objectives 2 and 5

Create environment

Objective 8

Is at least one person infected?

Yes

No

Calculate prediction with Euler’s method

Move the population

Objective 4

Objective 1

Display statistics page

Objective 10

Objectives 2,6 and 9

Check for infection and recovery

Objective 7

Plot number of people in each state

## Top down diagram:

Parameter entry window

Simulation window

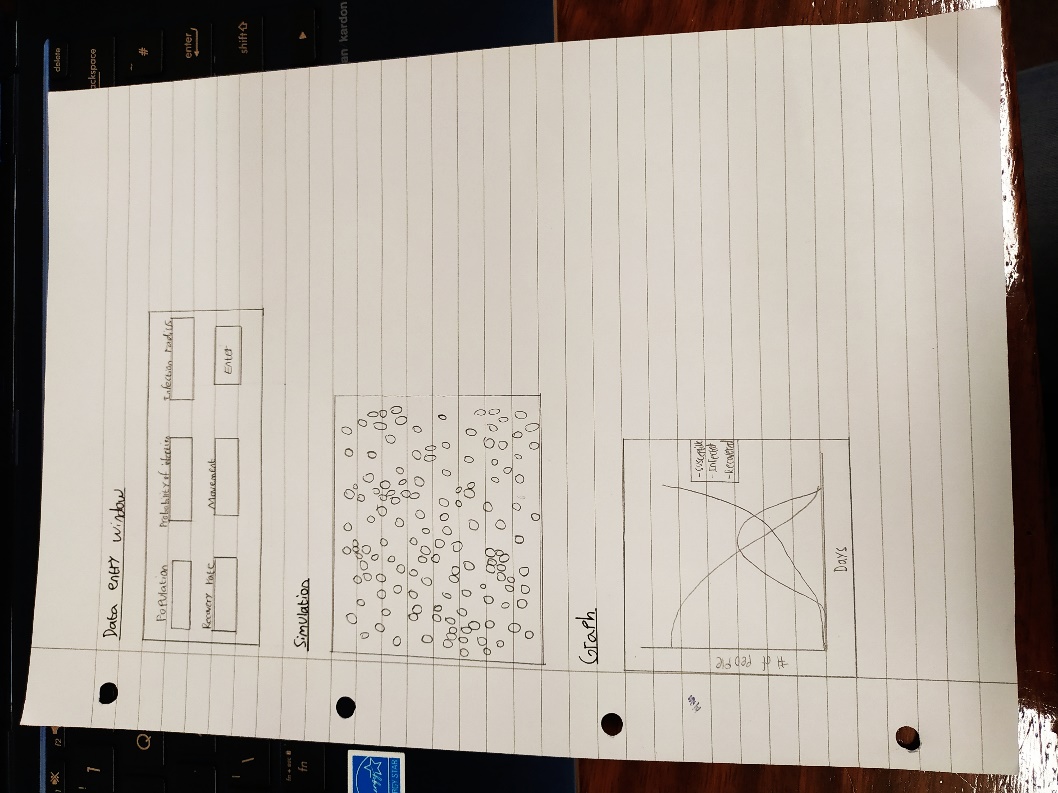
Invalid data type

Simulation

Graph

Statistics page

## Interface design:

****

## Data dictionary:

|  |  |  |  |
| --- | --- | --- | --- |
| Data item | Data type | Validation | Sample data |
| Population number (global) | Integer | Try:  natural(populationnumber) | 100 |
| Infection radius (global) | Integer | Try:  natural (infectionradius) | 50 |
| Probability of infection  (global) | Float | Try:  Float(probability)  If not 0 <= float(probability) <= 1:  Raise ValueError | 0.5 |
| Movement radius (global) | Integer | Try:  If not natural (movementradius) <100 | 30 |
| Recovery rate  (global) | Integer | Try:  natural (recoveryrate) | 10 |
| Person co-ordinates  (top left and bottom right) | 1D array |  | [10,45,30,65] |
| State | String |  | ‘Infected’ |
| Move in X direction | Integer |  | -23 |
| Move in Y direction | Integer |  | 54 |
| Distance between 2 people | Float |  | 36.543 |
| Random number between 1 and 100 for probability of infection | Integer |  | 79 |
| Day counter | Integer |  | 25 |
| Number of susceptible people (global) | Integer |  | 70 |
| Number of infected people (global) | Integer |  | 15 |
| Number of recovered people (global) | Integer |  | 5 |
| Number of susceptible people after each cycle | 1D array |  | [100,98,94,..2] |
| Number of infected people after each cycle | 1D array |  | [0,2,6,..0] |
| Number of recovered people after each cycle | 1D array |  | [0,0,0,..98] |
| Number of days passed after each cycle | 1D array |  | [1,2,3,..80] |
| Simulation complete | Boolean |  | False |

## IPSO Chart:

|  |  |  |
| --- | --- | --- |
| IPSO | Program section | Item |
| Input | Parameter entry window | Population number  Infection radius  Probability of infection  Rate of recovery  Movement range |
| Processing | Validate input data | Check if the inputted data is the correct data type |
| Generate population | Create a person with random coordinates, who is susceptible and has a probability of infection of what the user inputted.  Repeat for the population size entered by the user. |
| Move the population | Go through all the people in the population and move them by a random amount that will be generated by the user inputted movement range. |
| Restrict borders of simulation | If a person is about to move out of the dimensions of the simulation window, change its direction for the next move. |
| Check for infection | Check the distance between all the people on screen. If the distance between 2 people s less then or equal to the infection radius inputted by the user and one of them is infected, then infect the other person. |
| Check for recovery | If the number of days infected is equal to the rate of recovery entered by the user, then recover the person. |
| Infect person | Change the persons state from susceptible to infected. Increment total number of infected people. Decrement number of susceptible people. |
| Recover a person | Change the persons state from infected to recovered. Increment total number of recovered people. Decrement number of infected people. |
| Animate graph | Clear the graph and re-plot the updated numbers for the number of people in each state. |
| Sort list | Sort a list that will be used in the statistics window. |
| Euler’s method | Solve the differential equations numerically with inputted beta and gamma values |
| Storage | Save current number of people in each state | Append the total number of people in each state to separate lists.  Append the number of days to a different list. |
| Output | Parameter entry window | Display a window with labels and entry fields for each parameter to be inputted, plus a button to save and run. |
| Validate input data | If data is not valid a window will pop up and say “invalid data type, make sure the data types you have used are as following…” |
| Generate population | Show all the people in the simulation window. |
| Move population | Show the population moving around in the simulation window. |
| Infect Person | If a person is infected their colour will change from green to red. |
| Recover person | If a person recovers their colour will change from red to blue. |
| Animate graph | Show the graph plotting a curve in real time next to the simulation. |
| Statistics page | Display the statistics of the simulation once it terminates and the predicted vs actual graph. |

## Classes:

For the simulation a ‘Shape’ class was created. It would contain the geometrical properties of a circle and the way it moves. It will be a parent class to a child ‘human’ class which will inherit all its attributes and methods and add properties of a human. The class diagram looks like this:

|  |
| --- |
| **CShape** |
| xcoord: float  ycoord: float  Shape – Canvas(tkinter) |
| move(self, movement) |

|  |
| --- |
| **CHuman** |
| State – string  ProbabilityOfInfection -Float  DaysInfected - Integer |
| CheckStatus (self, Neighbour, LocalInfectionRadius, RecoveryTime)  Infected(self)  Recover(self)  UpdateInfectionStatus(self) |

In order to make a GUI for the entry window it would make the most sense to make a class for it as well:

|  |
| --- |
| **CWindow** |
| Master: Tkinter window  Labels: Tkinter label  Entries: Tkinter entry  Buttons: Tkinter button |
| ButtonClick(self) |

A class for the prediction was also made:

|  |
| --- |
| **CPrediction** |
| Population: integer  susceptible: integer  infected: integer  recovered: integer  beta:float  gamma:float |
| Eulers(self, t, deltat)  Susceptiblefunction()  Infectedfunction()  Recoveredfunction() |

## Algorithms:

### Euler’s method:

counter = 0

end = false

Repeat until end = true:

Susceptible = susceptible + (deltat \* S’(t))

Infected = infected + (deltat \* I’(t))

Recovered = recovered + (deltat \* R’(t))

Time = time + deltat

Save Susceptible, infected, recovered and time into a list

If infected < population/100 and number of infected are decreasing:

End = true

Increment counter by 1

Return Susceptible, infected, recovered and time lists

### CheckStatus:

If days infected = recovery time:

Recover

Days infected = 0

Else if state = infected or recovered:

Ignore

Else if distance between 2 people <= infection radius and neighbour is infected and random number < probability:

Infect

### Move:

Move x = random number

Move y = random number

If x position of top left corner < 10:

Move x = positive random number

If y position of top left corner < 10:

Move y = positive random number

If x position of bottom right corner > 690:

Move x = negative random number

If y position of bottom right corner > 690:

Move y = negative random number

### Merge sort:

If length of array <= 1:

Return array

Right array = merge sort of the right side of the array

Left array = merge sort of the left side of the array

Repeat until the end of either left or right array is reached:

If the current element of the right array is bigger than the left:

Add current element from left array to the result array

Move to the next element in the left array

Else:

Add current element from right array into result array

Move to the next element in the right array

Add the remaining elements from the remaining array into the result array

Return sorted array

### Main window cycle:

Repeat until the simulation is complete:

Increment day counter

For every person in the population:

Call move method

Call update infection status method

Repeat the number of people population:

Check status between current person and every other person on screen

Plot data on graph

If number of recovered and susceptible people is equal to the total population:

Simulation is complete

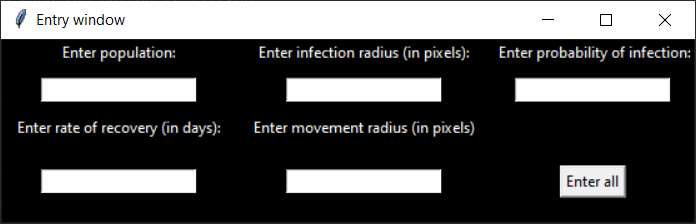
# Technical solution:

## Procedure listings:

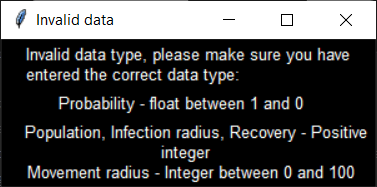
|  |  |  |
| --- | --- | --- |
| Procedures | Purpose | Variables |
| Method – move (Class Shape) | Generate new co-ordinates and move the person to the new location. | xmove  ymove  position  movment (Passed as parameter) |
| Method – infected (Class CHuman) | Change the state of a person to infected | Inumber  Snumber |
| Method – recover (Class CHuman) | Change the state of a person to recovered | Inumber  Rnumber |
| Method – UpdateInfectionStatus (Class CHuman) | Increment the days infected on a person |  |
| Method – checkstatus (Class CHuman) | Check for infetion and recovery | Neighbour (Passed as parameter)  Localinfectionradius (Passed as parameter)  Recoverytime (Passed as parameter)  Randomnumber  Distance |
| Method – buttonclick (Class CWindow) | Validate and save data entered | Population  Infectionradius  Recoveryrate  Movementradius  Probability |
| Method – Eulers (Class CPrediction) | Numerical solution to differential equations | t (Passed as parameter)  deltat (Passed as parameter)  end  counter  slist, ilist, rlist |
| Procedure – animate | Refresh and plot new data on the graph | Dayspassed  S,I,R  Snumber, Inumber, Rnumber  Handles  Labels |
| Function - mergesort | Use the merge sort algorithm with recursion to sort a list | Arr (Passed as parameter)  Middle, left , right  Result  Leftpointer , rightpointer |
| Function - getpeak | Find the position of largest value in a list | Maximum  Maximumposition |
| Function - natural | Checks if a number is a positive integer | Entrynumber (passed as parameter) |

## Interface design:

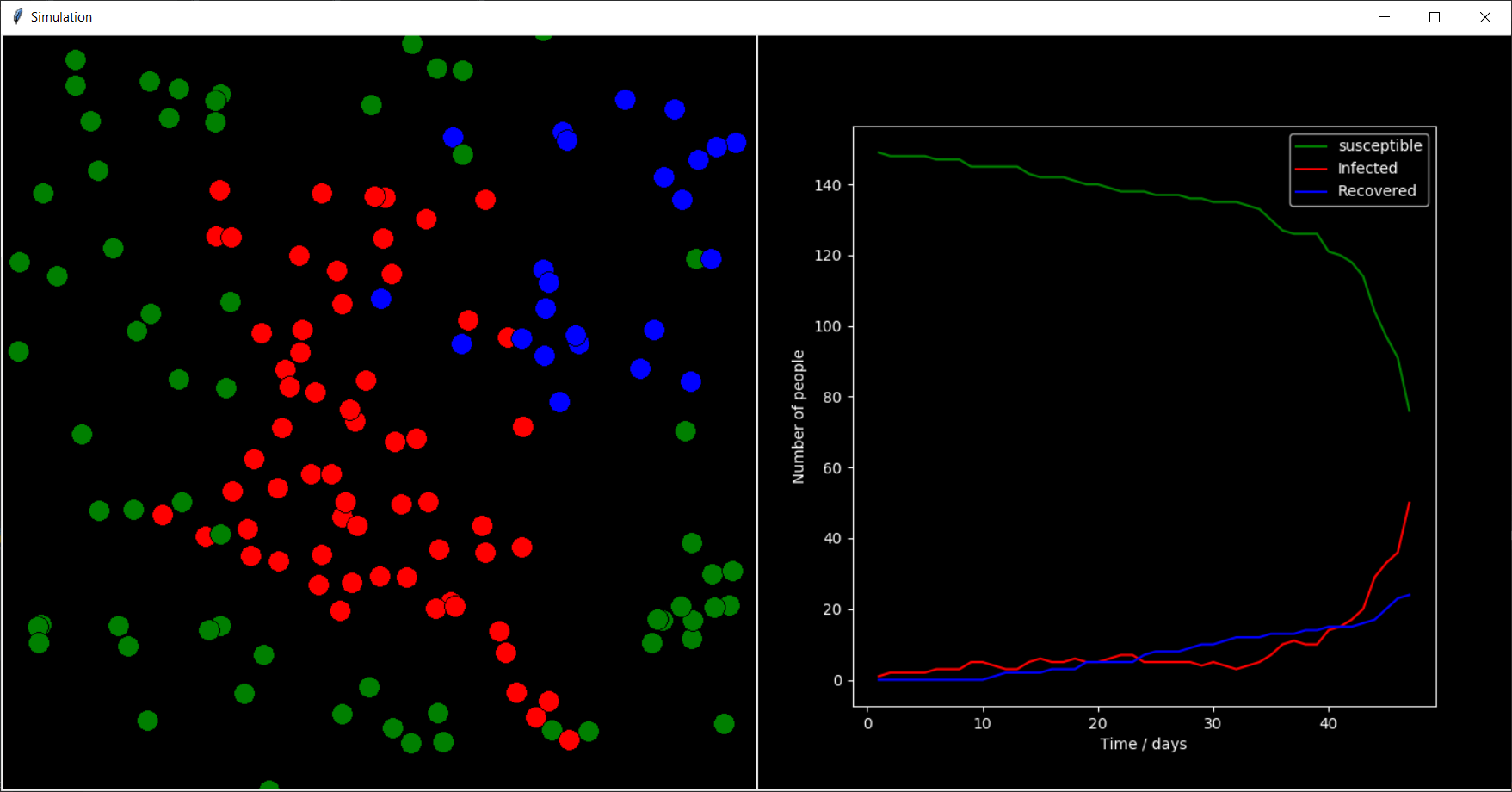
### Parameter entry window:



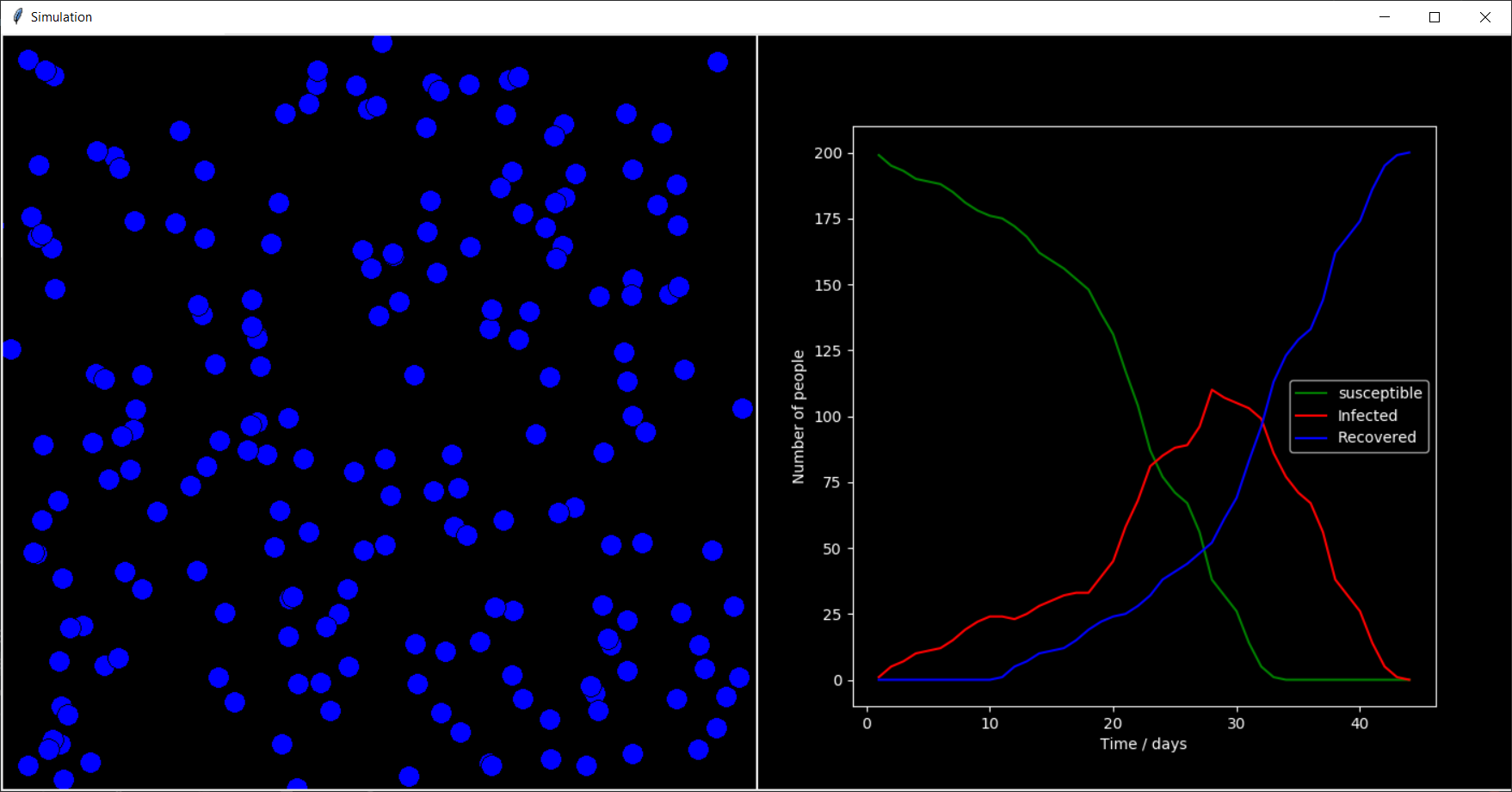
### Invalid data window:



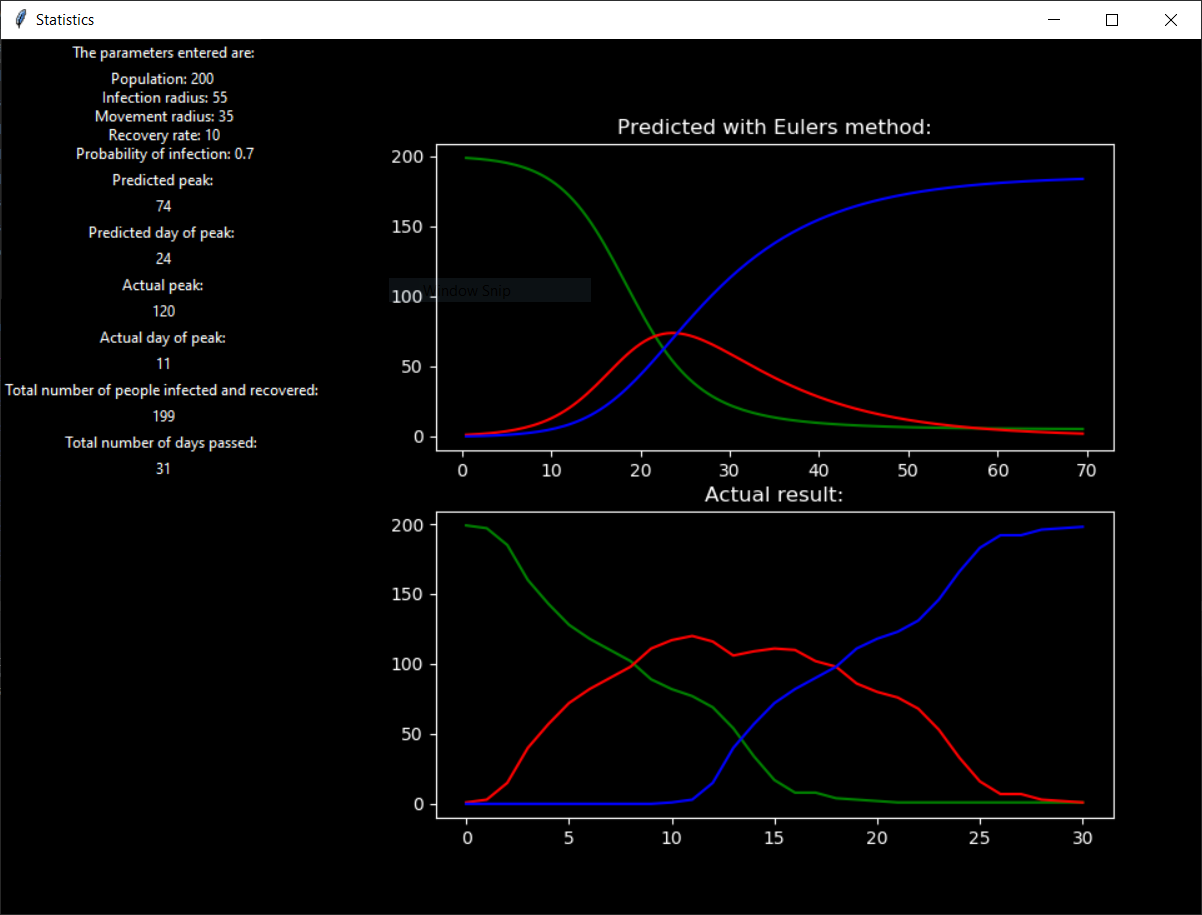
### Simulation:



### Graph:



### Statistics page:



## Code:

import tkinter # Import a graphics library  
import matplotlib.pyplot as plt # Import graph plotting library  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg # Import a function to make the graph run on a tkinter window  
from matplotlib import style # Library to change the look of the graph  
from matplotlib import figure # For graph implementation in tkinter  
import random # For random functionality  
import math # For the square root function  
style.use('dark\_background') # Set the theme of the graph to be dark  
  
  
class CPrediction: # Prediction  
 def \_\_init\_\_(self, predictionpopulation, beta, gamma): # Set attributes  
 self.population = predictionpopulation # Set population  
 self.s0 = predictionpopulation - 1 # Initial susceptible  
 self.i0 = 1 # Initial infected  
 self.r0 = 0 # Initial recovered  
 self.susceptiblefunc = lambda b, s, i, p: (-b\*s\*i)/p # The 3 differential equations  
 self.infectedfunc = lambda b, g, s, i, p: (b\*s\*i)/p - g\*i  
 self.recoveredfunc = lambda g, i: g\*i  
 self.beta = beta # Contact rate \* probability of infection  
 self.gamma = gamma # Rate of recovery  
  
 def eulers(self, t, deltat): # Numerical solution using eulers method  
 # Lists containing data to be plotted  
 slist = []  
 ilist = []  
 rlist = []  
 tlist = []  
 end = False  
 counter = 0  
  
 while not end: # Repeat until condition 'end' is met  
 # Iterative formulas  
 self.s0 = self.s0 + (self.susceptiblefunc(self.beta, self.s0, self.i0, self.population) \* deltat)  
 self.i0 = self.i0 + (self.infectedfunc(self.beta, self.gamma, self.s0, self.i0, self.population) \* deltat)  
 self.r0 = self.r0 + (self.recoveredfunc(self.gamma, self.i0) \* deltat)  
 t = t+deltat  
 # Add the calculated values to their lists  
 slist.append(self.s0)  
 ilist.append(self.i0)  
 rlist.append(self.r0)  
 tlist.append(t)  
 if self.i0 < self.population/100 and ilist[counter] < ilist[counter - 1]: # If number of infected is decreasing and less then 1/100 then population, end calculation  
 end = True  
 counter += 1 # increment counter  
 return tlist, slist, ilist, rlist # Return the lists  
  
  
class CShape: # Contains geometrical properties and methods of a circle  
 def \_\_init\_\_(self, xcoord, ycoord): # Set attributes  
 self.xcoord = xcoord # Set x coordinate  
 self.ycoord = ycoord # Set y coordinate  
 self.shape = canvas.create\_oval(self.xcoord, self.ycoord, self.xcoord + 20, self.ycoord + 20, fill='green') # Define the shape  
  
 def move(self, movement): # Method to allow the person to move  
 xmove = random.randint(-movement, movement) # Generate random distance for each person to move (within a range given from the user)  
 ymove = random.randint(-movement, movement)  
 position = canvas.coords(self.shape) # Get the current coordinates of position  
 # Set boundaries  
 if position[0] < 10: # If the new position is put of the boudries, chooose a different one in the oposite direction  
 xmove = random.randint(0, movement)  
 if position[1] < 10:  
 ymove = random.randint(0, movement)  
 if position[2] > 690:  
 xmove = random.randint(-movement, 0)  
 if position[3] > 690:  
 ymove = random.randint(-movement, 0)  
  
 canvas.move(self.shape, xmove, ymove) # canvas.move moves the object by the value of the new coordinate  
 canvas.update() # Update the window  
 self.xcoord = self.xcoord + xmove # Update coordinates of object  
 self.ycoord = self.ycoord + ymove  
  
  
class CHuman(CShape): # Contains properties and methods of a human  
 def \_\_init\_\_(self, xcoord, ycoord, state, probabilityofinfection): # Set attributes of human  
 super().\_\_init\_\_(xcoord, ycoord) # Inherit x and y coordinates and all methods of CGeometry  
 self.probabilityofinfection = probabilityofinfection # Set the probability of infection  
 self.state = state # Set the SIR state  
 self.daysinfected = 0 # Set the number of days infected  
  
 def infected(self): # Method to infect a person  
 canvas.itemconfig(self.shape, fill='red') # Change colour of person to red  
 self.state = 'infected' # Change the state  
 global Inumber, Snumber  
 Inumber += 1 # Increment by 1 the total number of people infected  
 Snumber -= 1 # Decrement by 1 the total number of people susceptible  
  
 def recover(self): # Method to make a person recover  
 canvas.itemconfig(self.shape, fill='blue') # Change colour of person to blue  
 self.state = 'recovered' # Change the state  
 global Rnumber, Inumber  
 Rnumber += 1 # Increment by 1 the total number of people recovered  
 Inumber -= 1 # Decrement by 1 the total number of people Infected  
  
 def updateinfectionstatus(self): # Update the number of days infected  
 if self.state == 'infected': # Increment the number of days a person has been infected for  
 self.daysinfected += 1  
  
 def checkstatus(self, neighbour, localinfectionradius, recoverytime): # Check for infection or recovery  
 randomnumber = random.randint(0, 100)  
 distance = math.sqrt(((self.xcoord-neighbour.xcoord)\*\*2)+((self.ycoord-neighbour.ycoord)\*\*2))  
 if self.daysinfected == recoverytime: # If you have been infected for the time the user has entered  
 self.recover() # Recover  
 self.daysinfected = 0 # To make sure the function no longer gets called  
 elif self.state == 'infected' or self.state == 'recovered': # If self is not susceptible then ignore  
 pass  
 elif distance <= localinfectionradius and neighbour.state == 'infected' and randomnumber < (self.probabilityofinfection \* 100):  
 self.infected() # If the distance between between self and an infected person is less then the radius given by the user and probability of infection is met, call the infect method  
  
  
class CWindow: # Class for the entry window  
 def \_\_init\_\_(self, master):  
 self.master = master # Set attributes: Labels, entries and buttons  
 self.master.title("Entry window")  
 self.master.configure(bg='black')  
 self.populationlbl = tkinter.Label(master, text='Enter population:', bg='black', fg='white')  
 self.populationlbl.grid(row=1)  
 self.populationentry = tkinter.Entry(master)  
 self.populationentry.grid(row=2, padx=10, pady=10)  
 self.enterbutton = tkinter.Button(master, text='Enter all', command=self.buttonclick)  
 self.enterbutton.grid(column=3, row=4, pady=20)  
 self.infectiousnesslbl = tkinter.Label(master, text='Enter infection radius (in pixels):', bg='black', fg='white')  
 self.infectiousnesslbl.grid(row=1, column=2)  
 self.infectiousnessentry = tkinter.Entry(master)  
 self.infectiousnessentry.grid(row=2, column=2, padx=10, pady=10)  
 self.recoverylbl = tkinter.Label(master, text='Enter rate of recovery (in days):', bg='black', fg='white')  
 self.recoverylbl.grid(row=3, column=0, padx=10)  
 self.recoveryentry = tkinter.Entry(master)  
 self.recoveryentry.grid(row=4, column=0, padx=10, pady=10)  
 self.speedlbl = tkinter.Label(master, text='Enter movement radius (in pixels)', bg='black', fg='white')  
 self.speedlbl.grid(row=3, column=2, padx=10)  
 self.speedentry = tkinter.Entry(master)  
 self.speedentry.grid(row=4, column=2, padx=10, pady=10)  
 self.probabilitylbl = tkinter.Label(master, text=' Enter probability of infection:', bg='black', fg='white')  
 self.probabilitylbl.grid(row=1, column=3)  
 self.probabilityentry = tkinter.Entry(master)  
 self.probabilityentry.grid(row=2, column=3)  
  
 def buttonclick(self): # Method for when the button is clicked  
 global population, infectionradius, recoveryrate, movementradius, probability  
 try: # Check for valid data  
 natural(self.populationentry.get())  
 natural(self.infectiousnessentry.get())  
 natural(self.recoveryentry.get())  
 if not 0 <= float(self.probabilityentry.get()) <= 1 or int(self.speedentry.get()) >= 100:  
 raise ValueError  
 except ValueError: # If data is not valid  
 errorwindow = tkinter.Tk() # Show invalid data window  
 errorwindow.title('Invalid data')  
 errorwindow.config(bg='black')  
 errorlabel1 = tkinter.Message(errorwindow, text='Invalid data type, please make sure you have entered the correct data type:', bg='black', fg='white', font='Ariel, 10', width=307)  
 errorlabel2 = tkinter.Label(errorwindow, text=' Probability - float between 1 and 0', bg='black', fg='white', font='Ariel, 10', width=34, anchor='w')  
 errorlabel3 = tkinter.Message(errorwindow, text=' Population, Infection radius, Recovery - Positive integer \n Movement radius - Integer between 0 and 100', bg='black', fg='white', font='Ariel, 10', width=330, justify='center')  
 errorlabel1.pack(), errorlabel2.pack(), errorlabel3.pack()  
 errorwindow.mainloop()  
 else: # If data is valid  
 population = int(self.populationentry.get()) # Save the inputted data  
 infectionradius = int(self.infectiousnessentry.get())  
 recoveryrate = int(self.recoveryentry.get())  
 movementradius = int(self.speedentry.get())  
 probability = float(self.probabilityentry.get())  
 self.master.destroy() # Close the window  
  
  
def natural(entrynumber): # Check if input is a natural number  
 if int(entrynumber) < 0:  
 raise ValueError  
  
  
def animate(): # Plot data  
 DaysPassed.append(DayCounter) # Update x values  
 S.append(Snumber) # Update the list containing number of susceptible people for each cycle  
 I.append(Inumber) # Update the list containing number of infected people for each cycle  
 R.append(Rnumber) # Update the list containing number of recovered people for each cycle  
 plt.cla() # Clear the axes  
 plt.xlabel('Time / days') # X-axis label  
 plt.ylabel('Number of people') # Y-axis label  
 ax.plot(DaysPassed, S, 'g', label='Susceptible') # Plot the points  
 ax.plot(DaysPassed, I, 'r', label='Infected')  
 ax.plot(DaysPassed, R, 'b', label='Recovered')  
 handles, labels = ax.get\_legend\_handles\_labels() # Show labels  
 ax.legend(labels)  
 Environmentwindow.update\_idletasks() # To make the animation a little smoother  
  
  
def mergesort(arr): # Merge sort algorithm  
 if len(arr) <= 1: # Base case  
 return arr  
 middle = int(len(arr)/2) # Split array into 2  
 left = mergesort(arr[:middle]) # Call mergesort for the left and right arrays  
 right = mergesort(arr[middle:])  
 result = []  
 leftpointer, rightpointer = 0, 0  
 while leftpointer < len(left) and rightpointer < len(right): # While you havent reached the end of an array  
 if left[leftpointer] <= right[rightpointer]: # If an element is bigger then the other  
 result.append(left[leftpointer]) # Add elemnt to result  
 leftpointer += 1 # Increment pointer  
 else:  
 result.append(right[rightpointer]) # Add elemnt to result  
 rightpointer += 1 # Increment pointer  
  
 result += left[leftpointer:] # Add remaining values to the result array  
 result += right[rightpointer:]  
 return result  
  
  
def getpeak(array): # Get the position of the largest number in an array  
 maximum = array[0]  
 maxposition = 0  
 for i in range(1, len(array)): # Search through the whole array  
 if array[i] > maximum: # If current element is greater then maximum  
 maximum = array[i] # Set a new maximum  
 maxposition = i # Get the position of the maximum  
 return maxposition  
  
  
# initialise global variables  
population, infectionradius, recoveryrate, movementradius, probability = 0, 0, 0, 0, 0  
  
# create the entry window  
root = tkinter.Tk()  
EntryWindowMain = CWindow(root)  
root.mainloop()  
  
# create environment  
Environmentwindow = tkinter.Tk()  
Environmentwindow.title('Simulation')  
canvas = tkinter.Canvas(Environmentwindow, height='700', width='700', bg='black')  
  
# Setting up the graph data  
Snumber = population  
Inumber = 0  
Rnumber = 0  
DaysPassed = []  
S = []  
I = []  
R = []  
  
# to make the people move  
populationList = []  
for x in range(population): # Create the requested number of people and add them to a list  
 person = CHuman(random.randint(0, 700), random.randint(0, 700), 'susceptible', probability)  
 populationList.append(person)  
  
populationList[0].infected() # infect the first person  
  
fig = plt.figure(figsize=(7, 7)) # Create the graph  
ax = fig.add\_subplot(1, 1, 1)  
plt.ion() # Activate interactive mode  
dataPlot = FigureCanvasTkAgg(fig, master=Environmentwindow) # To enable the graph to be drawn on a tkinter window  
dataPlot.draw() # Draw graph  
dataPlot.get\_tk\_widget().pack(side=tkinter.RIGHT) # Draw graph on the right side of the window  
  
# Initialise simulation variables  
SimulationComplete = False  
DayCounter = 0  
  
while not SimulationComplete: # Repeat until no one is infected anymore  
 animate() # Plot the data  
 DayCounter += 1 # Increment the number of days passed since the start of the simulation  
 for person in populationList: # For every person in the list  
 person.move(movementradius) # Call the move method  
 person.updateinfectionstatus() # Call updateinfectionstatus method  
 for x in range(population):  
 person.checkstatus(populationList[x], infectionradius, recoveryrate) # Check for infection or recovery  
 canvas.pack(side=tkinter.LEFT) # Display the simulation on the right side of the window  
 if Snumber + Rnumber == population: # If no one is infected  
 SimulationComplete = True # End the simulation  
 Environmentwindow.destroy() # Destroy the simulation window  
  
collisionrate = (population/(700\*700)) \* (movementradius/2) \* infectionradius # Calculate collision rate  
betavalue = collisionrate \* probability # Calculate beta value  
gammavalue = 1/recoveryrate # Calculate gamma vAlue  
  
Isorted = mergesort(I) # Sort the I array  
# Stats page  
message = ('Population: ', str(population), '\n Infection radius: ', str(infectionradius), '\n Movement radius: ', str(movementradius), '\n Recovery rate: ', str(recoveryrate), '\n Probability of infection: ', str(probability))  
StatsPage = tkinter.Tk() # Create statistics page  
StatsPage.title('Statistics')  
StatsPage.config(bg='black')  
prediction = CPrediction(population, betavalue, gammavalue) # Create prediction  
time, psusceptible, pinfected, precovered = prediction.eulers(0, 0.5)  
Parameters1 = tkinter.Label(StatsPage, text='The parameters entered are:', bg='black', fg='white') # Set labels  
Parameters2 = tkinter.Label(StatsPage, text="".join(message), bg='black', fg='white')  
PredictedPeak1 = tkinter.Label(StatsPage, text='Predicted peak:', bg='black', fg='white')  
PredictedPeak2 = tkinter.Label(StatsPage, text=math.ceil(mergesort(pinfected)[len(pinfected)-1]), bg='black', fg='white') # Find peak value of pinfected  
PeakInfected1 = tkinter.Label(StatsPage, text='Actual peak:', bg='black', fg='white')  
PeakInfected2 = tkinter.Label(StatsPage, text=Isorted[len(I)-1], bg='black', fg='white') # Largest number in the I array  
PeakInfected3 = tkinter.Label(StatsPage, text='Actual day of peak:', bg='black', fg='white')  
PeakInfected4 = tkinter.Label(StatsPage, text=math.ceil(DaysPassed[getpeak(I)]), bg='black', fg='white') # Find the day which the peak is reached  
Recovered1 = tkinter.Label(StatsPage, text='Total number of people infected and recovered: ', bg='black', fg='white')  
Recovered2 = tkinter.Label(StatsPage, text=Rnumber, bg='black', fg='white')  
Peakday1 = tkinter.Label(StatsPage, text='Predicted day of peak: ', bg='black', fg='white')  
Peakday2 = tkinter.Label(StatsPage, text=math.ceil(time[getpeak(pinfected)]), bg='black', fg='white') # Find the day which the peak is reached  
Days1 = tkinter.Label(StatsPage, text='Total number of days passed: ', bg='black', fg='white')  
Days2 = tkinter.Label(StatsPage, text=DayCounter, bg='black', fg='white')  
fig2 = figure.Figure(figsize=(7, 7)) # Create a graph figure  
axs = fig2.subplots(2) # split the figure into 2  
dataPlot = FigureCanvasTkAgg(fig2, master=StatsPage) # To enable the graph to be drawn on a tkinter window  
axs[0].plot(time, psusceptible, 'g') # Plot predicted graph  
axs[0].plot(time, pinfected, 'r')  
axs[0].plot(time, precovered, 'b')  
axs[0].set\_title('Predicted with Eulers method:')  
axs[1].plot(DaysPassed, S, 'g', label='Susceptible') # Plot graph of actual results  
axs[1].plot(DaysPassed, I, 'r', label='Infected')  
axs[1].plot(DaysPassed, R, 'b', label='Recovered')  
axs[1].set\_title('Actual result:')  
dataPlot.draw() # Draw graph  
dataPlot.get\_tk\_widget().pack(side=tkinter.RIGHT)  
Parameters1.pack(), Parameters2.pack(), PredictedPeak1.pack(), PredictedPeak2.pack(), Peakday1.pack(), Peakday2.pack(), PeakInfected1.pack()  
PeakInfected2.pack(), PeakInfected3.pack(), PeakInfected4.pack(), Recovered1.pack(), Recovered2.pack(), Days1.pack(), Days2.pack() # Display lables  
StatsPage.mainloop()  
  
Environmentwindow.mainloop() # Run the window

# Testing:

Due to the nature of the project, most individual objectives can’t be tested. I will test different types of data being entered into the simulation, and the outcome it produces.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Description | Test data | Expected result | Actual result |
| 1 | Data validation | Typical – Positive integers and float  Erroneous – non integer type and float out of range for probability  Extreme – big integers | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – simulation runs | Typical:  <https://www.youtube.com/watch?v=LH5CncEH_KI> (Simulation runs as expected)  Erroneous:  <https://www.youtube.com/watch?v=M1OVUuJT9gw>  (Invalid data window opens)  Extreme:  <https://www.youtube.com/watch?v=ZNKwSSExi3A>  (Simulation runs) |
| 2 | Statistics window | Typical – Positive integers and float  Erroneous – non integer type and float out of range for probability  Extreme – big integers | Typical, Extreme – Window pops up at end of simulation  Erroneous – simulation doesn’t run | Typical:    Extreme:    Erroneous: |
| 3 | Population entry | Typical – Positive integers  Erroneous – non integer  Extreme – big integer (over 500) | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – simulation runs with a very big population | Typical – simulation runs as expected  Erroneous – invalid data window appears  Extreme – simulation runs, but very slowly |
| 4 | Infection- radius entry | Typical – Positive integers  Erroneous – non integer  Extreme – big integer (over 100) | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – infection spreads very quickly | Typical – simulation runs as expected  Erroneous – invalid data window appears  Extreme – simulation runs as expected |
| 5 | Probability of infection entry | Typical – float between 0 and 1  Erroneous – non float or float out of range  Extreme – 1.0 | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – infection can spread more easily | Typical – simulation runs as expected  Erroneous – invalid data window appears  Extreme – simulation runs as expected |
| 6 | Rate of recovery entry | Typical – Positive integers  Erroneous – non integer  Extreme – big integer (over 40) | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – people start to recover after a long time | Typical – simulation runs as expected  Erroneous – invalid data window appears  Extreme – simulation runs as expected |
| 7 | Movement radius entry | Typical – Positive integers  Erroneous – non integer  Extreme – big integer (over 100) | Typical – simulation runs  Erroneous – invalid data window appears  Extreme – People move greater distances. | Typical – simulation runs as expected  Erroneous – invalid data window appears  Extreme – simulation runs as expected |
| 8 | Objective 4 | Different values of typical data | The shape of the 2 graphs is similar | a)  b)    c) |

# Evaluation:

## How did I meet my objectives?

|  |  |
| --- | --- |
| Objective | Evaluation |
| 1- Have a window with people moving and encountering each other.   * Population must be moving and not static   + Where and by how much they move must be random, but within a certain range inputted by the user.   + There must be borders to the simulation, so no person can leave the confined borders. | -COMPLETELY ACHIEVED-  A class was created in order to allow an object ‘person’ to be created which has the method self.move, which allows it to generate new x and y coordinates based on the movement radius entered by the user. If the new coordinates lie outside of the borders of the simulation, new coordinates are generated, in the opposite direction. |
| 2- Each person can be in one of 3 states: susceptible, infected or recovered.   * Must follow the 3 compartments of the SIR model * Each state must be represented by a colour on a person so you can tell who’s in what state. * Once a person changes state their colour must change on screen. | -COMPLETELY ACHIEVED-  A person object can be in 1 of the 3 states of the SIR compartmental model, where a state is represented as a colour. As people get infected and recover you can see them changing state with their colour. |
| 3 - The user can input parameters that will influence the outcome of the simulation.   * Have a ‘setup’ window for the user to enter all the parameters before the simulation starts. * The user should be able to input 5 parameters:   + Population size   + Radius of infection   + How much a person can move every cycle   + Probability of infection   + Rate of recovery * Validate the data being inputted   + Population, infection radius, movement range and rate of recovery – integer   + Probability of infection – float   + Have an invalid data window | -COMPLETELY ACHIEVED-  The first thing to come up once you run the simulation is the data entry window, with the parameters: population, infection radius, movement radius, probability of infection and recovery rate. If you enter an incorrect data type, an invalid data window appears telling you what data is accepted by the simulation. |
| 4- Create a prediction of the outcome of the simulation   * Implement Euler’s method to solve the differential equations. * Calculate a suitable beta and gamma value from the users’ input. | -PARTIALLY ACHIEVED-  The implementation of Euler’s method works as intended, and beta and gamma values are appropriately predicted. However, since the Euler’s method solution to the differential equations provides an expected average, the 2 graphs don’t always look the same because the simulation is fundamentally random, so different outcomes can occur with the same parameters. |
| 5 - Generate the population requested by the user.   * Initial X and Y coordinates of every person should be randomly generated. * Everyone should be susceptible once they are created. * Once the population is generated, infect one person. | -COMPLETELY ACHIEVED-  Once the user has entered the desired population size, the program generates people with random x and y coordinates within the borders of the simulation, and everyone’s initial state is susceptible. Once the whole population is generated, one person gets infected. |
| 6 - A person can catch the disease from a nearby person.   * The radius of infection will be the maximum distance 2 people must be apart in order to have a chance of getting infected. * If a person is a is close enough to get infected, a random number will be compared to the probability of infection, and if it is lower than the person will be infected. | -COMPLETELY ACHIEVED-  The class CHuman has a method self.checkstatus which checks the distance between 2 people. If the distance is closer then the infection radius, then the probability of infection is compared to a random number which then determines whether the person will be infected. |
| 7 - Graph the number of susceptible, infected and recovered individuals against time.   * + The 3 different curves will be displayed on the same set of axes.   + Display a key that shows which curve represents witch state.     - Each curve will have a different colour.     - The colour of each curve will be the same as the colour coding in the simulation. * Plot the number of people in each state as they get infected/recover. * The graph and the simulation should be displayed on the same page. * The simulation and the graph must run simultaneously, and the graph must update in real time. | -COMPLETELY ACHIEVED-  Once the simulation begins, the graph is displayed alongside the simulation window, and updates after a cycle of the main loop passes.  A key is displayed on the side to show which graph represents what state. |
| 8 - The simulation should stop once the infection is eradicated.   * Once there are no more infected people, the main loop should terminate. * There should be some variables to keep track of the number of people in each state. | -COMPLETELY ACHIEVED-  Once the number of recovered and susceptible people are equal to the population size, the simulation stops. |
| 9 - An infected person should recover in the number of days specified by the user.   * Keep track of how many days a person has been infected for. * Check for recovery every cycle. | -COMPLETELY ACHIEVED-  Person object has an attribute of days infected which gets updated every cycle, and the self.checkstatus method checks if the person should recover. |
| 10 - Display a statistics page at the end of the simulation.   * Should be displayed once the simulation stops. * It should display the parameters entered. * It should display the peak number of infected, total number infected and days passed since the start. * Display both predicted and actual graph side by side. | -COMPLETELY ACHIEVED-  Once the program has detected that no one is infected, a window pops up with their entered parameters and additional statistics, along with the 2 graphs |

## Feedback from user:

‘Hi Yoav, I’m very content with the outcome of this program. All the promised features work as expected and the simulation looks great. The data validation system works, however if I enter a big population, the program struggles to handle it. The graph provides a nice visualisation to the data. The prediction is a bit hit or miss, sometimes the trend is predicted accurately and sometimes not as much. As you told me, the prediction predicts an average and the simulation is based on one case and not an average and that’s the reason for this. It would be nice if in the future you could add make the simulation run multiple times and then compare the 2 graphs for a more accurate comparison. Overall, I’m happy with the result. Thanks for helping me out!’

## Analysis of feedback:

My client was satisfied with the program, and all the requested features were present. Improvements need to be made either to the data validation system in order to restrict the population size or find a way to make the simulation run faster when a big population is entered. Also, an additional feature was requested to average out a few runs of a simulation for a more accurate comparison of the 2 graphs.