Project Report

ON

COVID SIMULATOR

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE

SUBMITTED BY

Vaishvik Tripathi 20070122512

Shantanu Bhattacharya 20070122514

Under the Guidance of

Prof. Kalyani Kadam

SYMBIOSIS INSTITUTE OF TECHNOLOGY

(A CONSTITUENT OF SYMBIOSIS INTERNATIONAL UNIVERSITY)

Pune - 412115

CS-C3

2019-23

CONTENTS

1. Introduction

1.1 Purpose

The novel coronavirus, named SARS-CoV-2, has been spreading throughout the world for several months now. It emerged in China, most likely in December 2019.

Many of us follow news about its progression and we observe the growing sick numbers with increasing anxiety. And we feel helpless. Most of us only see the epidemic's indirect effects and the rest is just media buzz. Our coronavirus simulation attempts to explain everyone, using simple means, what is happening across the world right now. It can show – in a limited way, as it is only a simulation – how our and our government's actions in the following weeks can influence the spread of the virus.

1.2 Intended audience and reading suggestions.

We attempt to simulate the dynamics of human-to-human interaction and the viral infections that occur because of them. While such processes are relatively easy to model mathematically, especially for very large numbers, they do not allow us to see the process itself.

In our simulation the subjects of the epidemic are balls in 2-dimensional space. They move at a constant speed and bounce off each other. Each bounce is contact and a possible spread of the coronavirus. While this model is simple, in a way it does a good job of representing the real life, where similarly, each human-to-human contact, random or not, may result in an infection.

1.3 References

[1] Oxford Coronavirus Government Response Tracker
https://www.bsg.ox.ac.uk/research/research-projects/coronavirus-government-response-tracker

[2] Mummert A, Weiss H, Long LP, Amigó JM, Wan XF (2013) A Perspective on Multiple Waves of Influenza Pandemics. PLOS ONE 8(4): e60343. https://doi.org/10.1371/journal.pone.0060343

[3] Viceconte, Giulio, and Nicola Petrosillo. "COVID-19 R0: Magic number or conundrum?." *Infectious disease reports* vol. 12,1 8516. 24 Feb. 2020, doi:10.4081/idr.2020.8516
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7073717/

[4] https://nyuscholars.nyu.edu/en/publications/automata-networks-and-artificial-intelligence)

2. Overall Description

2.1 Product Perspective

The Outbreak Simulator is designed to be a self-contained product that can act as a small model for a pandemic and show the user the gravity of the situation. It is simple in its workings and uses. The simulator shows the virus spread randomly and shows a real time graph of the number of infected, healthy, and recovered patients.

2.2 Product Functions

The Outbreak Simulator will ask the user for a population size and then represent the population

on the screen in the form of small dots of different colours depending on their health. Each person

on the screen will move randomly and collide with each other sometimes infecting the un-infected person.

The simulation will show the user what might happen if the situation is left uncontrolled.

2.3 User Classes and Characteristics

We believe that the Outbreak Simulator has a lot of uses, for example it is a perfect simulation to add to a news article to show the reader the situation, it can also be used by classrooms to help the students understand the spread of Viruses.

2.4 Operating Environment

Software requirements

Languages/Packages: java(swings)

Operating Systems: Any OS

Hardware requirements

RAM: 256 MB

Hard disk capacity: 4 GB

3. Other Non-functional requirements

3.1 Performance requirements

1. Platform: Any OS

2. Language: JAVA

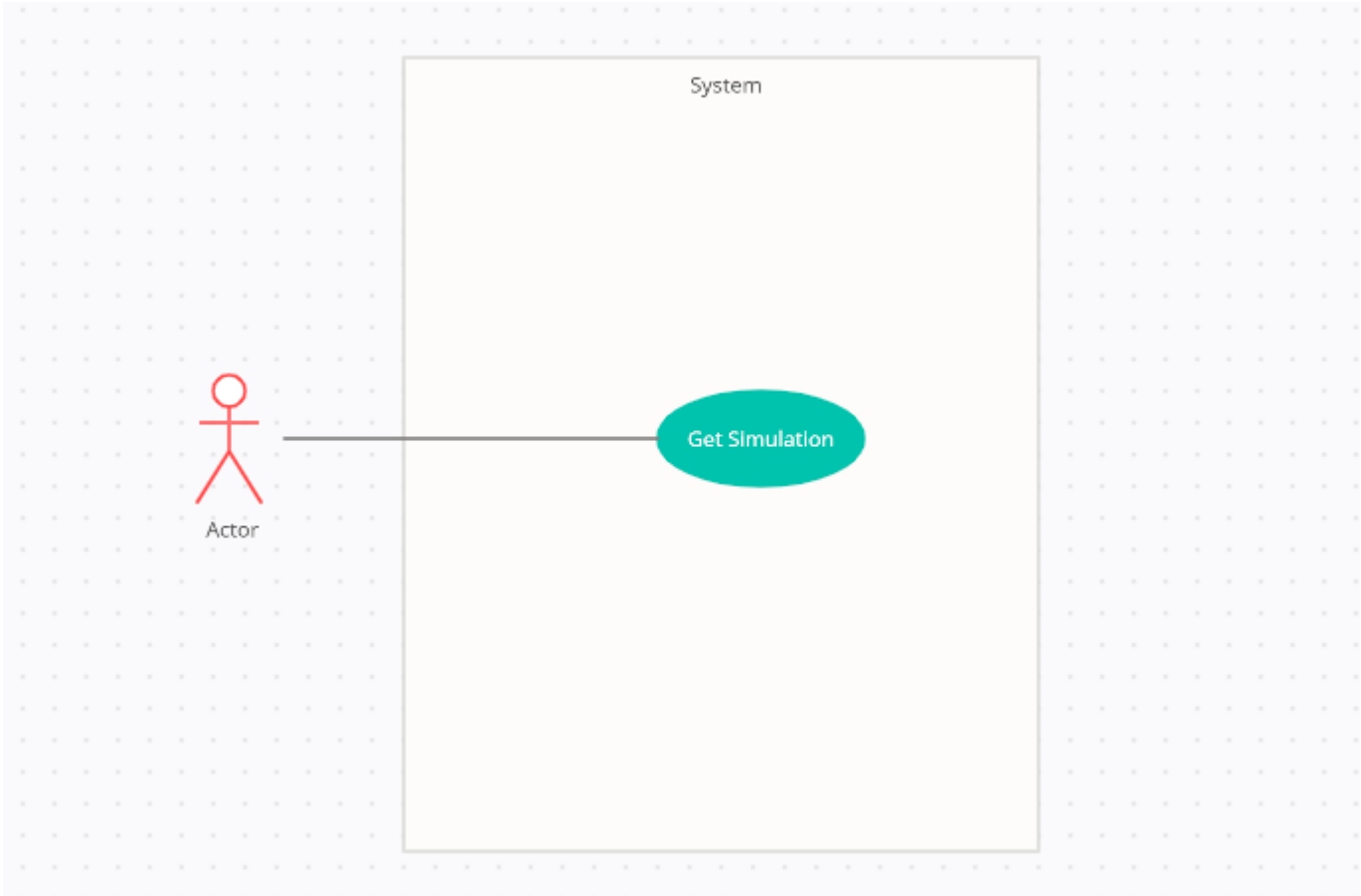
3. IDE: VISUAL STUDIO

3.2 Software quality attributes

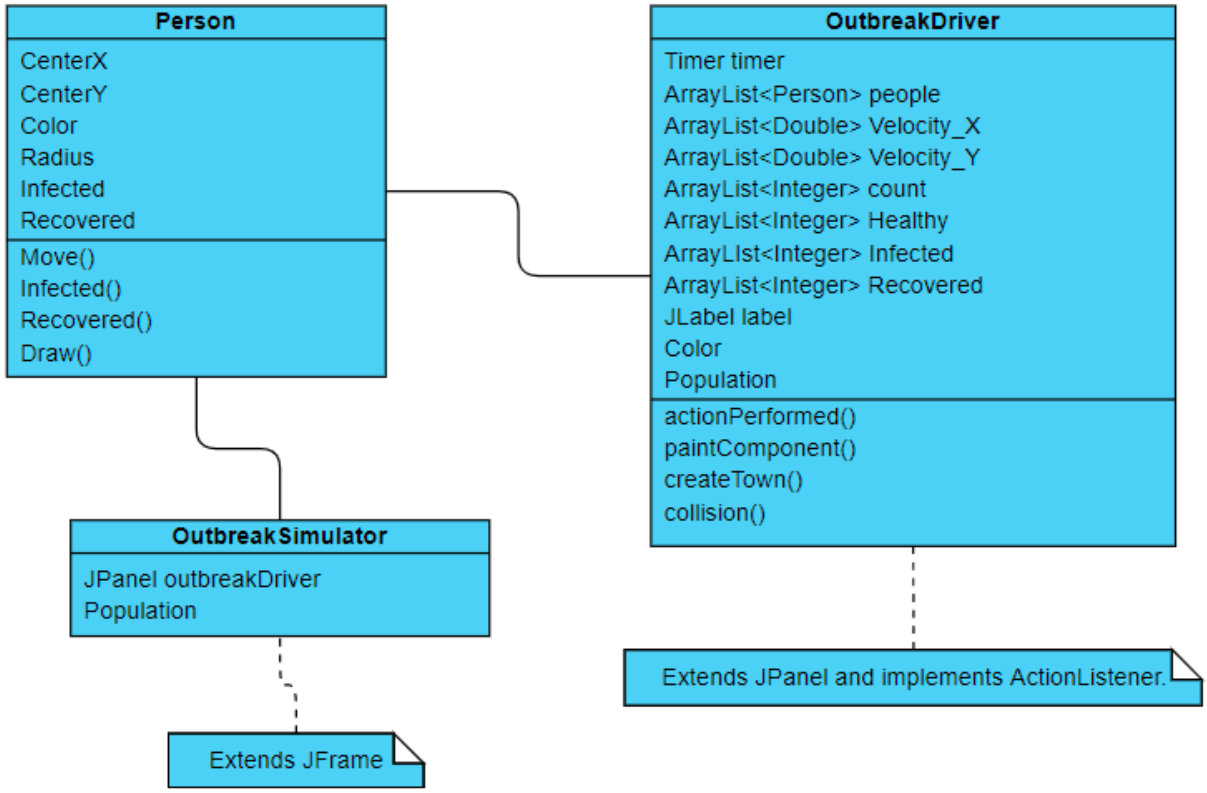
The quality of the software is maintained in such a way so that it can be user friendly and easy to understand.

4. Analysis Models

4.1 Use case diagram



4.2 Class diagram



5. Conclusion

Our simulation showed how the SARS-CoV-2 can spread across the population and how different countermeasures can affect how it spreads.

Social distancing and closing schools significantly slowed the spread, but eventually the whole population got infected.

Social distancing, self-isolation of the sick and sick contact tracing also slowed the virus. Despite that, most of the population got infected.

All sets of countermeasures had positive effects. However, only using them all at once allowed most of the population to remain healthy.

What can we do then? Both not much and a lot. How to stop the spread? By taking countermeasures, which, for a normal person, usually means to not do anything special – stay at home, wear mask or cover your face when you have to get out. And, most importantly, start social distancing, even isolate yourself.

The more we do, the more countermeasures we use, the better the results and the more people are saved.

Code Output

Classes:

- Person
- OutbreakDriver
- OutbreakSimulator

Code:

Person class:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.*;

public class Person 
{
   private Color fillColor;
   private final int RADIUS = 10;
   private double my_x;
   private double my_y;
   private double centerX;
   private double centerY;
   private boolean infected = false;
   private boolean recovered = false;

   public Person(Color givenFillColor, double x, double y)
   {
      fillColor = givenFillColor;
      my_x = x;
      my_y = y;
      centerX = (my_x + RADIUS);
      centerY = (my_y + RADIUS);
   }

   public Person(double x, double y)
   {
      my_x = x;
      my_y = y;
   }

   public double getCenterX()
   {
```

```
    return centerX;
  }

  public double getCenterY()
  {
    return centerY;
  }

  public void setFillColor(Color c)
  {
    fillColor = c;
  }

  public Color getFillColor()
  {
    return fillColor;
  }

  public double getX()
  {
    return this.my_x;
  }

  public void setX(double x)
  {
    this.my_x = x;
  }

  public double getY()
  {
    return this.my_y;
  }

  public void setY(double y)
  {
    this.my_y = y;
  }

  public void move(double dx, double dy)
  {
    my_x = my_x + dx;
    my_y = my_y + dy;

    centerX = (my_x + RADIUS);
    centerY = (my_y + RADIUS);
  }

  public boolean isInfected()
  {
    return this.infected;
  }
```

```
    public void setInfected()
    {
        infected = true;
    }

    public boolean isRecovered()
    {
        return this.recovered;
    }

    public void setRecovered()
    {
        recovered = true;
        infected = false;
    }

    public void draw(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
        Ellipse2D circle = new Ellipse2D.Double(this.my_x, this.my_y, this.RADIUS, this
.RADIUS);
        g2.setPaint(this.fillColor);
        g2.fill(circle);
    }

}
```

OutbreakDriver class:

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Random;
import java.awt.Color;
import java.awt.Graphics;

public class OutbreakDriver extends JPanel implements ActionListener
{

   private Timer timer;
   private ArrayList<Person> people;
   private ArrayList<Double> dX_Array;
   private ArrayList<Double> dY_Array;
   private ArrayList<Integer> count;
   private ArrayList<Integer> infectedCounts;
   private ArrayList<Integer> recoveredCounts;
   private ArrayList<Integer> healthyCounts;
   private ArrayList<Integer> timeCounts;
   private ArrayList<Integer> timeCountsRecovered;
   private ArrayList<Integer> timeCountsHealthy;
   private int time;
   private int actualTime;
   private int infectedCount;
   private int recoveredCount;
   private int graph_y;
   private JLabel label;
   private JLabel label2;
   private Color fillColor;
   private Random random;
   private int Population;

   public OutbreakDriver(int givenPopulation)
   {
      Population = givenPopulation;
      people = new ArrayList<Person>();
      dX_Array = new ArrayList<Double>();
      dY_Array = new ArrayList<Double>();
      count = new ArrayList<Integer>();
      infectedCounts = new ArrayList<Integer>();
      recoveredCounts = new ArrayList<Integer>();
      healthyCounts = new ArrayList<Integer>();
      timeCounts = new ArrayList<Integer>();
      timeCountsRecovered = new ArrayList<Integer>();
      timeCountsHealthy = new ArrayList<Integer>();
```

```
    graph_y = ((Population/2)+50);
    infectedCounts.add(graph_y - 1);
    recoveredCounts.add(graph_y);
    healthyCounts.add(graph_y - (Population/3));
    timeCounts.add(200);
    timeCountsRecovered.add(200);
    timeCountsHealthy.add(200);
    time = 0;
    actualTime = 0;
    infectedCount = 1;
    recoveredCount = 0;
    label = new JLabel();
    label.setBounds(30, 30, 150, 150);
    this.add(label);
    label2 = new JLabel();
    label2.setBounds(250, 0, 200, 30);
    this.add(label2);
    fillColor = Color.BLUE;
    random = new Random();

    this.createTown(Population);

    this.setFocusable(true);
    this.requestFocus();
    timer = new Timer(1000/60, this);
    timer.start();
  }

  @Override
  public void actionPerformed(ActionEvent arg0)
  {
    time++;

    if (time >= 3)
    {
      actualTime = actualTime + 1;
      time = 0;
    }

    for (int i = 0; i < people.size(); i++)
    {
      double cent_x = people.get(i).getCenterX();
      double cent_y = people.get(i).getCenterY();
      if (cent_x > 1250 || cent_x < 50)
      {
        dX_Array.set(i, -(dX_Array.get(i)));
      }
      if (cent_y > 650 || cent_y < 50)
      {
        dY_Array.set(i, -(dY_Array.get(i)));
      }
```

```
        double dx = dX_Array.get(i);
        double dy = dY_Array.get(i);
        people.get(i).move(dx, dy);

        if (people.get(i).isInfected())
        {
          count.set(i, count.get(i)+1);
        }

        if (count.get(i) == 559 && people.get(i).isInfected())
        {
          people.get(i).setFillColor(Color.MAGENTA);
          people.get(i).setRecovered();
          recoveredCount = recoveredCount + 1;
          timeCountsRecovered.add(200+actualTime);
          recoveredCounts.add(graph_y - (recoveredCount/3));
          if (infectedCount != 0)
          {
            infectedCount = infectedCount - 1;
            infectedCounts.add(graph_y-(infectedCount/3));
            timeCounts.add(200+actualTime);
          }
        }

        int healthy = (people.size() - infectedCount - recoveredCount);
        if (infectedCount != 0)
        {
          healthyCounts.add(graph_y - (healthy/3));
          timeCountsHealthy.add(200+actualTime);
        }
        label.setText(String.format("<html> <p><b><FONT COLOR=BLACK> Count </FONT></
b></p> <br> <p> %-9s <FONT COLOR=PURPLE> %5d </FONT></p> <br> <p> %-9s <FONT COLOR=
BLUE> %5d </FONT></p> <br> <p> %-9s <FONT COLOR=RED> %5d </FONT></p> </html>", "Rec
overed", recoveredCount, "Healthy", healthy, "Sick", infectedCount));

        for (int j = i+1; j < people.size(); j++)
        {
          double xDist = ((people.get(i).getCenterX()) - (people.get(j).getCenterX())
);
          double yDist = ((people.get(i).getCenterY()) - (people.get(j).getCenterY())
);
          double distBetweenPeople = Math.sqrt(Math.pow(xDist, 2) + Math.pow(yDist, 2
));
          int idealDist = 20;

          if (distBetweenPeople - idealDist < 0)
          {
            doCollision(people.get(i), people.get(j), dX_Array.get(i), dY_Array.get(
i), dX_Array.get(j), dY_Array.get(j), i, j);
```

```
            if (people.get(i).isInfected() && !people.get(j).isRecovered() && !
people.get(j).isInfected())
            {
                people.get(j).setFillColor(Color.RED);
                people.get(j).setInfected();
                infectedCount = infectedCount + 1;
                infectedCounts.add(graph_y-(infectedCount/3));
                timeCounts.add(200+actualTime);
            }
            else if (people.get(j).isInfected() && !people.get(i).isRecovered() && !
people.get(i).isInfected())
            {
                people.get(i).setFillColor(Color.RED);
                people.get(i).setInfected();
                infectedCount = infectedCount + 1;
                infectedCounts.add(graph_y-(infectedCount/3));
                timeCounts.add(200+actualTime);
            }
        }
    }


    this.repaint();
  }

  @Override
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    setOpaque(true);
    setBackground(Color.WHITE);
    label.setBounds(10, 0, 150, 150);
    this.add(label);
    label2.setText("<html> <p><b> Change over time </b></p> </html>");
    label2.setBounds(220, 20, 200, 30);
    this.add(label2);

    g.setColor(Color.BLACK);
    g.drawLine(200, (graph_y+3), 1270, (graph_y+3));
    g.drawLine(200, (graph_y+3), 200, 15);
    g.drawLine(200, 15, 1270, 15);
    g.drawLine(1270, (graph_y+3), 1270, 15);

    for (int l = 1; l < healthyCounts.size() && l < timeCountsHealthy.size(); l++)
    {
      g.setColor(Color.BLUE);
      g.fillRect(timeCountsHealthy.get(l-1), healthyCounts.get(l-1), (timeCountsHea
lthy.get(l) - timeCountsHealthy.get(l-1)), ((graph_y+3) - healthyCounts.get(l-1)));
      //g.drawLine(timeCountsHealthy.get(l-1), healthyCounts.get(l-1), timeCountsHe
althy.get(l), healthyCounts.get(l));
```

```
    }

    for (int k = 1; k < recoveredCounts.size() && k < timeCountsRecovered.size(); k
++)
    {
      g.setColor(Color.MAGENTA);
      g.fillRect(timeCountsRecovered.get(k-1), recoveredCounts.get(k-1), (timeCount
sRecovered.get(k) - timeCountsRecovered.get(k-1)), ((graph_y+3) - recoveredCount
s.get(k-1)));
      //g.drawLine(timeCountsRecovered.get(k-1), recoveredCounts.get(k-1), timeCoun
tsRecovered.get(k), recoveredCounts.get(k));
    }

    for (int j = 1; j < infectedCounts.size() && j < timeCounts.size(); j++)
    {
      g.setColor(Color.RED);
      g.fillRect(timeCounts.get(j-1), infectedCounts.get(j-1), (timeCounts.get(j) -
timeCounts.get(j-1)), ((graph_y+3) - infectedCounts.get(j-1)));
      //g.drawLine(timeCounts.get(j-1), infectedCounts.get(j-1), timeCounts.get(j),
infectedCounts.get(j));
    }

    for (int i = 0; i < people.size(); i++)
    {
      people.get(i).draw(g);
    }
  }

  public void createTown(int Population)
  {
    int i = Population;
    while(i > 0)
    {
      int x = random.nextInt(1170)+50;
      int y = random.nextInt(550)+50;
      int centerX = x + 10;
      int centerY = y + 10;

      if (i != Population)
      {
        for (int k = 0; k < people.size(); k++)
        {
          centerX = x + 10;
          centerY = y + 10;
          double xDist = (centerX - (people.get(k).getCenterX()));
          double yDist = (centerY - (people.get(k).getCenterY()));
          double distBetweenPeople = Math.sqrt(Math.pow(xDist, 2) + Math.pow(yDist
, 2));
          int idealDist = 20;

          if(distBetweenPeople - idealDist < 0)
```

```
            {
                x = random.nextInt(1170)+50;
                y = random.nextInt(550)+50;

                k = -1;
            }
        }
    }

    Person person = new Person(fillColor, x, y);
    people.add(person);

    int randomNumber = random.nextInt(2);
    double dX = 0;
    double dY = 0;

    if (randomNumber == 0)
    {
        dX = -1;
    }
    if (randomNumber == 1)
    {
        dX = 1;
    }

    randomNumber = random.nextInt(2);

    if (randomNumber == 0)
    {
        dY = 1;
    }
    if (randomNumber == 1)
    {
        dY = -1;
    }

    dX_Array.add(dX);
    dY_Array.add(dY);
    count.add(0);

    i--;
  }

  int randomNumber = random.nextInt(Population);
  Person infected = new Person(Color.RED, people.get(randomNumber).getX(), people
.get(randomNumber).getY());
  infected.setInfected();
  people.set(randomNumber, infected);
 }
```

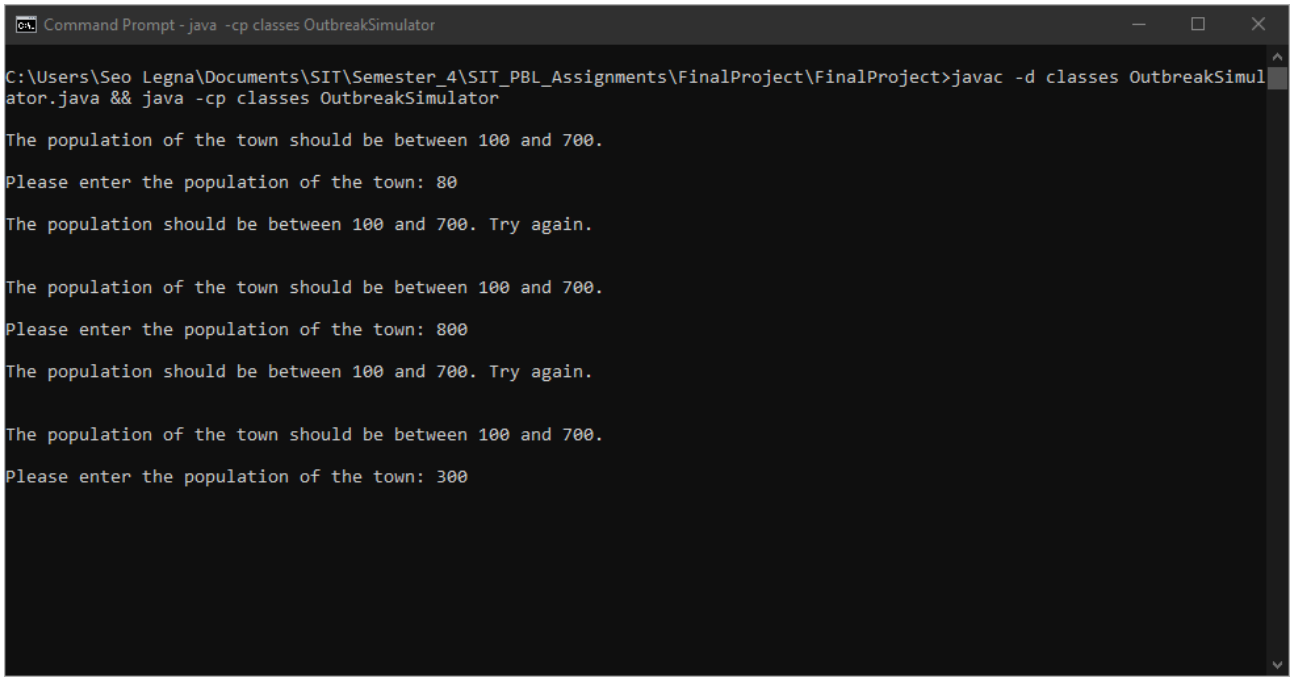
```
  public void doCollision(Person one, Person two, double dx_one, double dy_one, dou
ble dx_two, double dy_two, int i, int j)
  {
    double xVelocityDiff = dx_one - dx_two;
    double yVelocityDiff = dy_one - dy_two;

    double xDist = two.getCenterX() - one.getCenterX();
    double yDist = two.getCenterY() - one.getCenterY();

    if ((xVelocityDiff * xDist) + (yVelocityDiff * yDist) >= 0)
    {
      double angle = -Math.atan2(two.getCenterY() - one.getCenterY(), tw
o.getCenterX() - one.getCenterX());
      int m_one = 1;
      int m_two = 1;

      double u_one_x = (dx_one * Math.cos(angle)) - (dy_one * Math.sin(angle));
      double u_one_y = (dx_one * Math.sin(angle)) + (dy_one * Math.cos(angle));
      double u_two_x = (dx_two * Math.cos(angle)) - (dy_two * Math.sin(angle));
      double u_two_y = (dx_two * Math.sin(angle)) + (dy_two * Math.cos(angle));

      double v_x_one_temp = ((u_one_x * (m_one - m_two))/(m_one + m_two)) + ((u_two
_x * 2 * m_two)/(m_one + m_two));
      double v_x_two_temp = ((u_two_x * (m_one - m_two))/(m_one + m_two)) + ((u_one
_x * 2 * m_two)/(m_one + m_two));
      double v_y_one_temp = u_one_y;
      double v_y_two_temp = u_two_y;

      double v_x_one_final = (v_x_one_temp * Math.cos(-angle)) - (v_y_one_temp * Ma
th.sin(-angle));
      double v_x_two_final = (v_x_two_temp * Math.cos(-angle)) - (v_y_two_temp * Ma
th.sin(-angle));
      double v_y_one_final = (v_x_one_temp * Math.sin(-angle)) + (v_y_one_temp * Ma
th.cos(-angle));
      double v_y_two_final = (v_x_two_temp * Math.sin(-angle)) + (v_y_two_temp * Ma
th.cos(-angle));

      dX_Array.set(i, v_x_one_final);
      dY_Array.set(i, v_y_one_final);
      dX_Array.set(j, v_x_two_final);
      dY_Array.set(j, v_y_two_final);
    }
  }

}
```

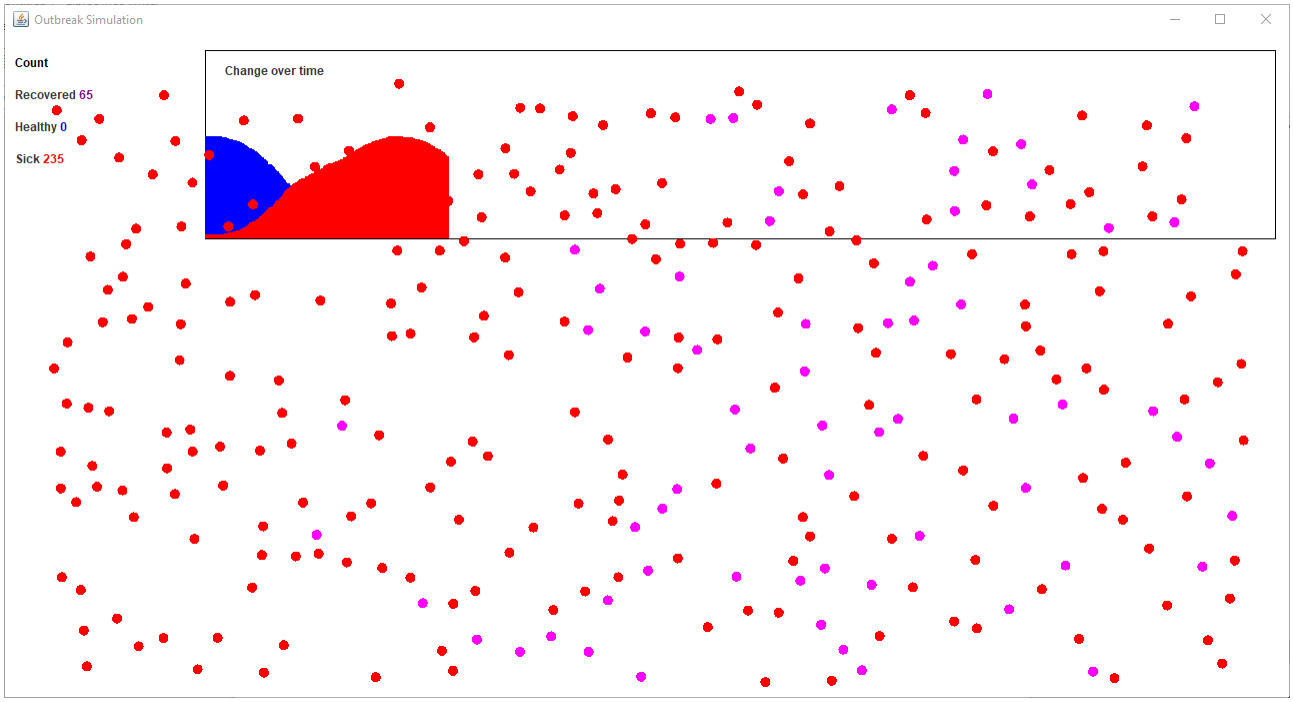
OutbreakSimulator class:

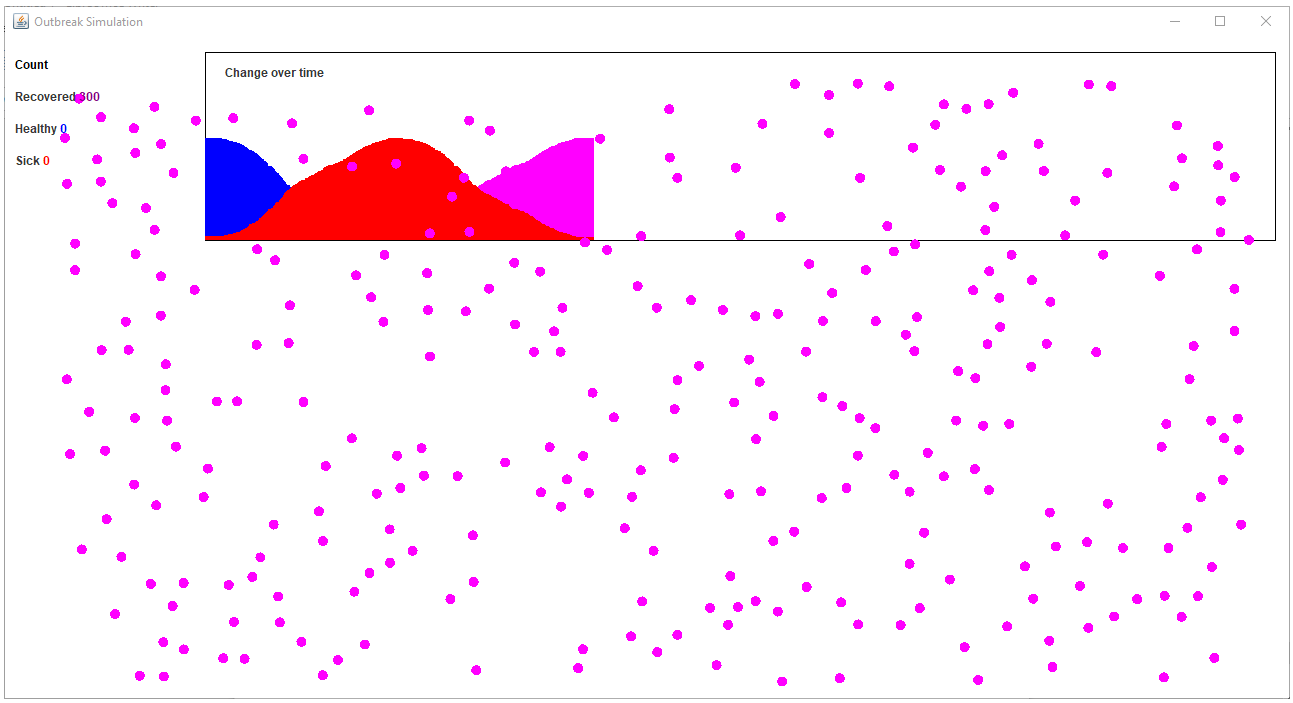
```
import javax.swing.*;
import java.util.*;
import javax.swing.JPanel;

public class OutbreakSimulator extends JFrame
{

  private JPanel outbreak;
  private static int population = 0;

  public OutbreakSimulator(int givenPopulation)
  {
    population = givenPopulation;
    outbreak = new OutbreakDriver(population);
    this.getContentPane().add(outbreak);
  }

  public static void main(String[] args)
  {
    try (Scanner in = new Scanner(System.in);) {
      boolean done = false;
      while (!done) {
        System.out.println("\nThe population of the town should be between 100 and
700.\n");
        System.out.print("Please enter the population of the town: ");
        if (in.hasNextInt()) {
          population = Math.abs(in.nextInt());
          if (population >= 100 && population <= 700) {
            done = true;
          } else {
            System.out.println("\nThe population should be between 100 and 700. Tr
y again.\n");
          }
        } else {
          System.out.println("Please enter a valid integer as the population.");
          System.out.println();
        }
      }

      OutbreakSimulator frame = new OutbreakSimulator(population);
      frame.setSize(1300, 700);
      frame.setTitle("Outbreak Simulation");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
    } catch (Exception e) {
      e.printStackTrace();
      System.out.println("\nSomething went wrong!\n");
```

```
    }
}

}
```

Output:


Command Prompt - java -cp classes OutbreakSimulator
C:\Users\Seo Legna\Documents\SIT\Semester_4\SIT_PBL_Assignments\FinalProject\FinalProject>javac -d classes OutbreakSimulator.java && java -cp classes OutbreakSimulator
The population of the town should be between 100 and 700.
Please enter the population of the town: 80
The population should be between 100 and 700. Try again.
The population of the town should be between 100 and 700.
Please enter the population of the town: 800
The population should be between 100 and 700. Try again.
The population of the town should be between 100 and 700.
Please enter the population of the town: 300


Outbreak Simulation
Count
Recovered 65
Healthy 0
Sick 235
Change over time


Outbreak Simulation
Count
Recovered 300
Healthy 0
Sick 0
Change over time