Modelling - Simulation Lab

Mandar Joshi - 970515-7494 - mandarj@kth.se
Albert Asratyan - 980308-6470 - asratyan@kth.se

Problem description

When a person gets sick, there is a chance that the sickness will spread to people around, if the disease is contagious. There may be a chance that the sick person could die, if the disease is serious enough. Having thousands of diseases, it may be useful to be able to model and monitor behaviour of infection spreading. This is the model that we have created. It takes multiple inputs that describe severity of the disease, such as probability that a sick individual infects a healthy neighbor who is not immune, length of time an individual is ill, probability that an individual dies because of the sickness and number of individuals that are ill initially and their coordinates. Having these many inputs allows to simulate different diseases, ranging from very basic and non-lethal diseases to plagues. The output of the model includes a wide range of statistical data that can be accessed at the end of the simulation, as well as a graphical representation of the infection spread. Specifically, we are interested in the turning point where disease becomes epidemic due to it spreading faster than people getting well.

Model reflection

The model can be said to be a simulation of diseases in real life. Scenarios where this could be useful can therefore be governments or various agencies who monitor the spread of disease in their respective populations, or even across the whole world. The benefits of such a model can be that one can learn about how to cure people depending on the density of the population. One can also see how the disease will spread depending on where in the country or where in the world the patient zero is located. The difficulties with this model are the same amount as the benefits. It can be hard to determine certain probabilities, for example the probability to spread the disease or the probability of dying. It is also much harder to calculate for the incubation period of the disease, which is the time in which the individual has been affected by the disease but is not showing any symptoms or spreading the disease yet. Another difficulty applied to the real world is people moving around, for example in the big city. In our simulation the people are static, and only have restricted access to the people next to them.

Our solution and validation of results

Before implementing the model, we had to model the structure of the code. We ended up having a simple state diagram, describing the change of the state for each of the humans, when a day passes.

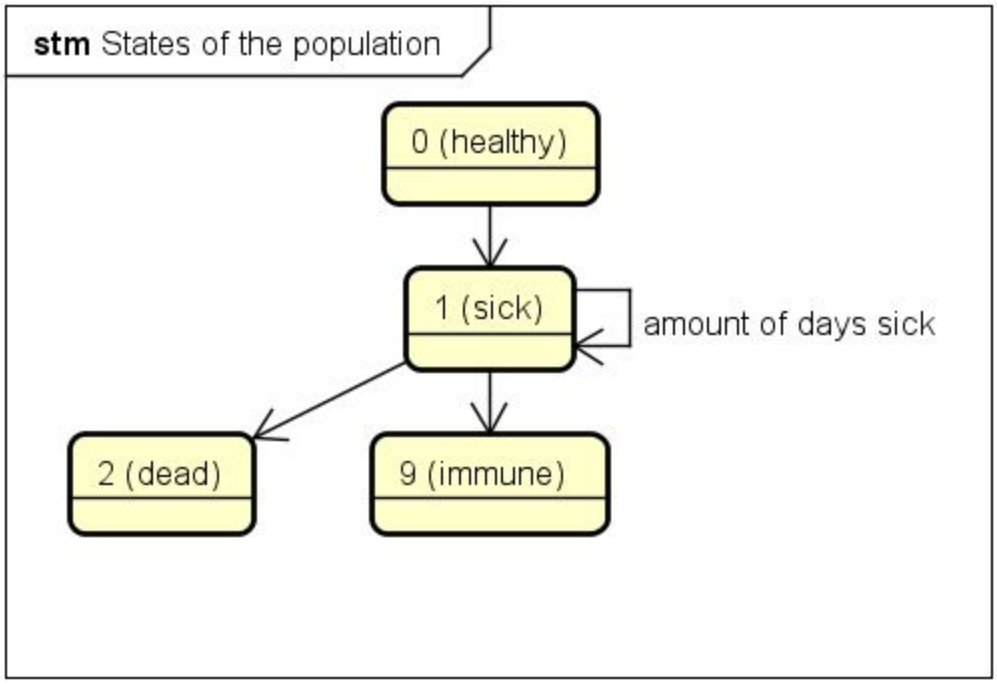

Diagram 1. State of each individual human

Any human can be in one of four states: healthy, sick, dead or immune. Their dependency on each other is described in the diagram above. It must be said in advance, that we adapted an incremental testing approach, which is to say that we have been testing the model block by block, then combining the blocks and testing again. With that said, the first step in the model creation after gathering all of the required inputs from the user is to create a population matrix. For ease of modeling we have decided to implement a graphical representation for the infection spread. The following snippet of code was used for testing results of every stage, including matrix creation:

```
for (int i = 0; i < pop.length; i++){
    for (int j = 0; j < pop[i].length; j++){
        System.out.print(pop[i][j].status + " ");
        if (j == pop[i].length-1)
            System.out.println();
    }
}
```

Picture 1. Code snippet that prints out the whole population

After the initial matrix has been created, we spawn infected humans in the population and verify it, by checking if they have appeared in the population (by using the code snippet above again). At this point, all of the preparations have been finished and the next step is to run the simulation. We are using a two dimensional matrix for population, so the next step would be that for every infected human, we have to check all of their neighbors, whether the random function has triggered the infection spread or no. The very same code snippet was used after simulating day by day to see the progression of the infection. However, in the final version of code, the excessive testing code was cut, so the user would not be

overwhelmed with data. After day by day spread was tested, simulation until no sick patients were left was tested.

From the requirements it was clear that the run should be terminated when there are no individuals who are infected. This led to a result remaining of only healthy people who have still not been infected, dead people and immune people. When it came to the individuals on the other edge of the matrix, our implementation made sure that they can still infect the people around them. We did not implement a population that is "connected", where for example the left edge is connected to the right edge of the matrix. We understood the problem in a way such that the population is in a city, where the people in one side of the city can not infect the people on the other side. In our implementation we took care of the incubation period with a boolean value, true meant that the person has been infected today, false meant the person has been infected on a previous day. Therefore, everyday the simulation ran, a person could only infect another person if their boolean value was false.

Results and conclusions

As it can be seen on picture 2, the simulation was successful, because no sick people were left (no "1" in the matrix). When the simulation modeling was done, the next step would be to gather statistics. Statistics monitoring was implemented within the code for triggering the infection spread for neighbors. To check whether it worked or no, at the end of the program statistics is displayed, and then compared against a graphical representation of a small sample (<10x10 matrix). This can be seen in picture 3.

Picture 2. An example of the final result with (30x30, 10%, 2 days, 5 days, 3%, (5,5), (18,18)) inputs

```
0 0 0 0 0 0 0 0 0 0  Total days: 10
0 0 0 0 0 0 0 0 0 0  Total deaths: 1
0 0 0 9 9 0 0 0 0 0  Total people getting sick excluding the initial sick: 6
0 0 9 9 2 0 0 0 0 0  Average deaths per day: 0.1
0 9 0 0 9 0 0 0 0 0  Average people getting sick per day: 0.6
0 0 0 0 0 0 0 0 0 0  Average recoveries per day: 0.7
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 9
```

Picture 3. Matrix final result against text final result

The example simulation above had the following input parameters: 10x10 matrix, 5% to spread, 2 min, 5 max, 2% to die, (3,3) and (9,9) initial spread points. If to compare data on the matrix to the data to the right of it, we can see that they match, thus making the model correct. At this point, the program has reached its final stage, as it can be seen on diagram 2 below.

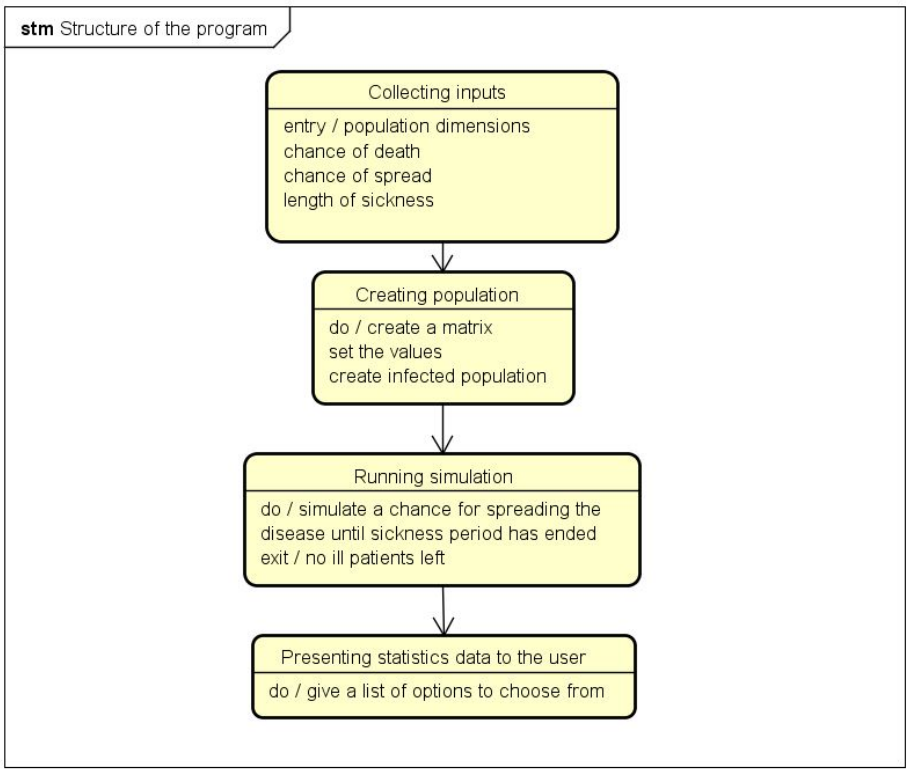

Diagram 2. Program's structure

As a result of running this model for all of the inputs specified on the website, the following results were recorded:

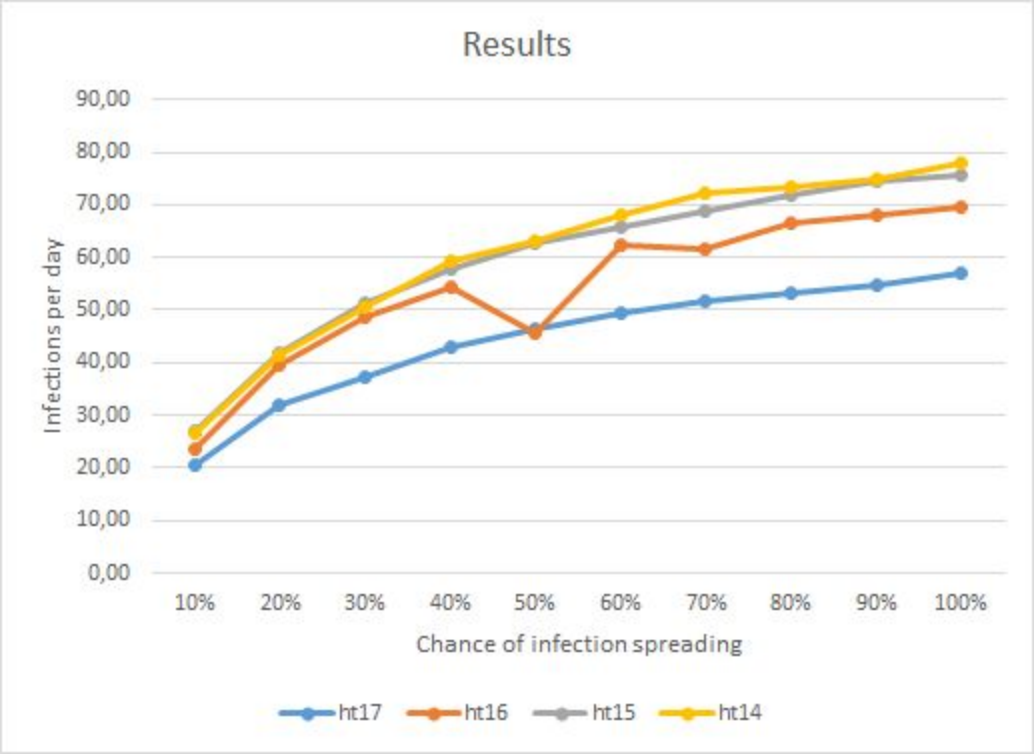

Diagram 3. Model's results for 10% to 100% disease spread chance

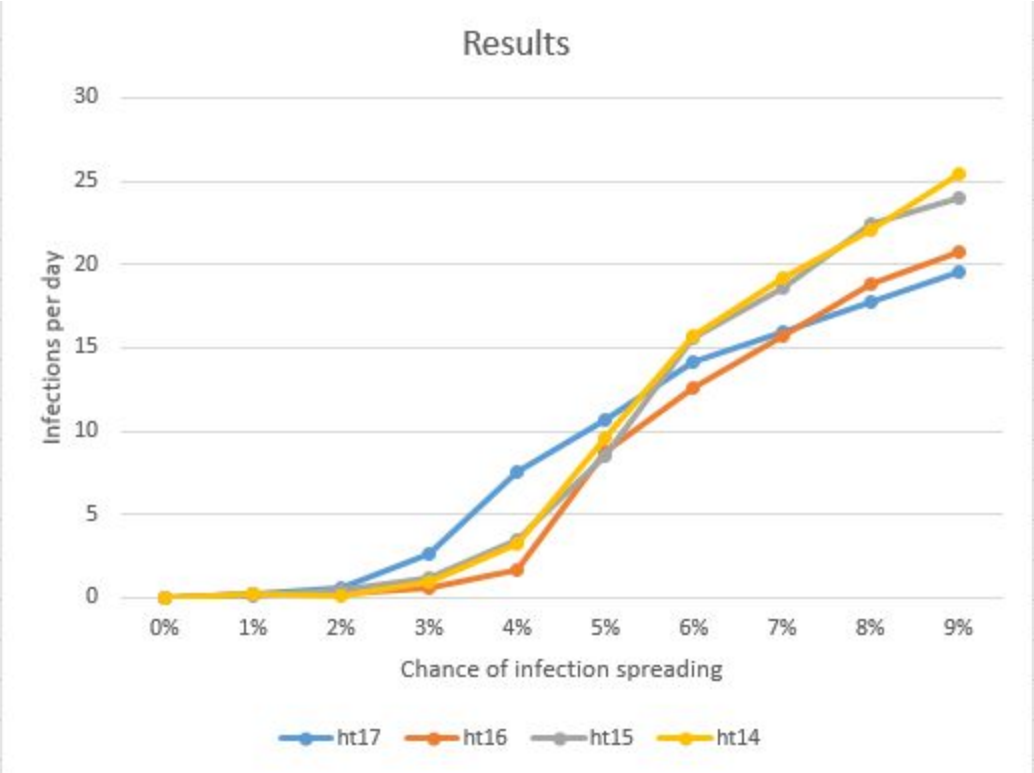

Diagram 4. Model's results for 0% to 9% disease spread chance

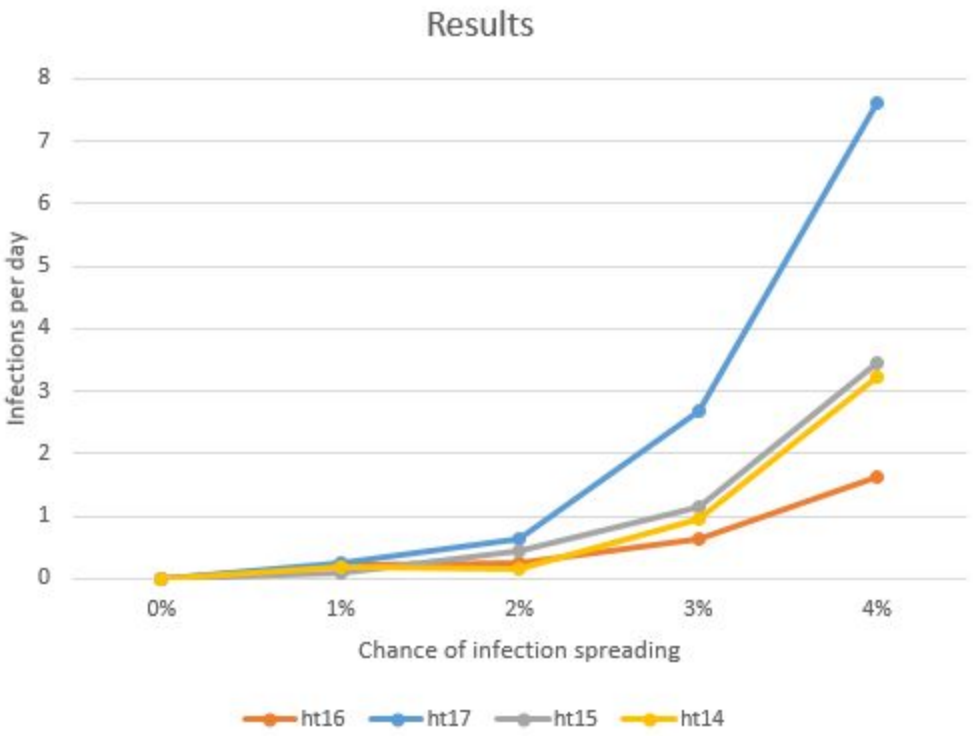

Diagram 5. Model's results for 0% to 4% disease spread chance

Raw data for these graphs can be found in the appendix. The graph shows how amount of infected people per day rises with the increasing chance of spreading the infection. It is a somewhat linear increase starting from 10% and up to 50%. However, starting at 60% and higher, a deviation from a common trend is noticed. It can be explained either by reaching the point where the infection is being spread faster than the days are passed, which means that the spread is limited by the maximum amount of neighbors being able to get sick on that day. Or another possible explanation is that the model does not handle high chances of spreading well, due to the random function that we are using for the simulation.

However, it is easy to notice that for all of the data with more than 10% disease spread chance the amount of infections per day is far greater than 1 infection/day. This number is important because this is what determines a epidemic. If infections per day is greater than 1, then disease will keep spreading. If the value is less than 1, then the disease will eventually die out. Because of the values are far too great starting at 10%, lower spread percentages must be examined to determine where the turning point for becoming an epidemic. If to look at the diagram 4 that shows values for spread chances from 0% to 9%, it can be seen that the turning point of exceeding 1 infection/day is present somewhere between 1% and 4%. However, it is not clear enough, and a closer look at those values should be taken. Diagram 5 shows data for disease spread chances from 0% to 4%. Here, turning points are obvious. For ht17, the disease becomes epidemic when exceeding slightly more than 2%. For ht16, the disease becomes epidemic somewhere between 3% and 4%. For ht15, the disease becomes epidemic around 2.5% and for ht14 the disease becomes epidemic at 3%. However, by looking at the raw data for ht16 and its

confidence intervals, it should be noted that the confidence intervals are quite large and it is unclear where exactly the turning point is, but it is definitely somewhere between 3% and 5%.

To sum up, all of the curves (ht17, ht16, ht15 and ht14) resemble roughly same trends. Diseases become epidemic at somewhere between 2% and 4%. However, at those percentages the values are barely over 1 (as can be seen in the appendix) and it takes some time to contaminate the whole population. However, starting at 10%, the diseases start spreading extremely fast by having at least 20 infections per day. Further than that, it starts growing even more, but this is only theoretical and thankfully it is very difficult to imagine that a disease with such a high chance of being spread might happen in real world.

Model testing

In order to test whether the model is reliable, we have analyzed the results presented on the graph above and then compared these results to a mathematical model of the spread.

First we look at the curves we have plotted. It is logical that ht17 has overall lower rate of spread, due to the fact that the matrix size (40x40) is smaller than the other 3 samples (50x50). Because of this, there are less infections overall, and thus, the average becomes smaller. All of the curves are concave down, because there is a limit to the amount of sick people. Because of this, higher percentages of disease spread do not increase the infections per day as much as the lower-to-mid change. It has a uniform growth, as well as the curves ht15 and ht14. Curves ht15 and ht14 are practically identical and overlap a lot, because they have the same parameters, but slightly different sickness period, where sometimes one is faster, and sometimes the other.

However, the ht16 is an interesting case. Since the sickness period ranges all the way from 1 to 12 days, there is a chance that the sickness will be either very short, or very long. It has happened by chance that during one of the simulations for 50%, the initial patient got well without spreading the disease, which is a very interesting case. Because of this, there exists a big dip in the graph. An explanation for this is the randomness of the sample, due to the extreme differences in the sickness period. Moreover, there is another dip at 70%, which happened to be lower than at 60%. There were only 5 tests performed per value, making it quite possibly not accurate enough for a sample of this random nature. Overall, the curves and trends fit within the conditions. Thus, the model can be considered successful.

After verifying our data, we can now compare it to the mathematical models of the same experiment. We have gathered the infections per day value from the test. From the mathematical model of the disease spread rate[1], rate of change of disease spread is calculated. In our case, infections per day is the same as rate of change of disease spread. By comparing two graphs, we can see that the curves have identical structures. Thus, we can say that our data follows the mathematical model trend.

Apart from this, standard library random generator was tested by being fed predetermined seeds. All of the tests have ended with the same results, as can be seen in the appendix. The random number generator is initialized at this line:

```
Random rand = new Random();
```

The appendix shows that the values for each of the seeds match. The input sequence was the same for every single run. The input arguments were: 20, 5, 4, 8, 0, (10,10). This validates the implementation, meaning that the code performs as we expected it to perform.

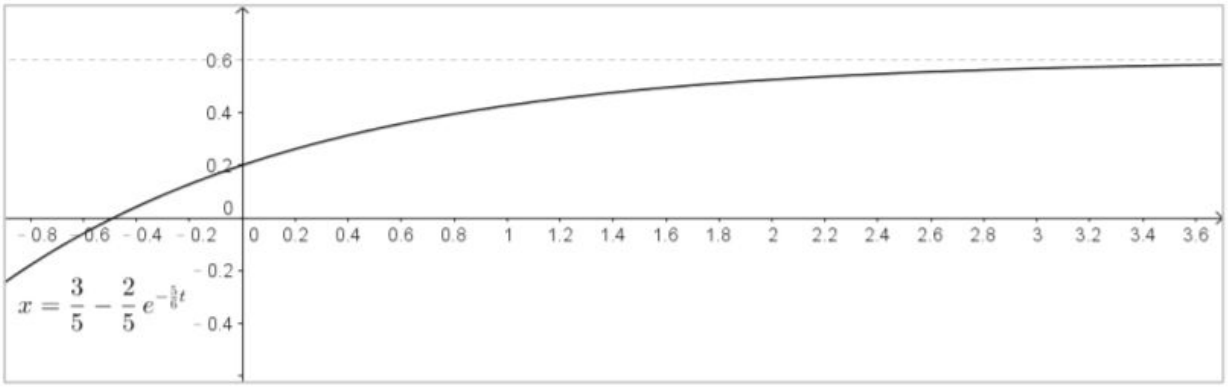

Diagram 6. Rate of change of disease spread (Education bureau of Hong Kong)

Reliability measures

Confidence intervals were used as a reliability measure for the data. 99% confidence level was used. The data with the confidence intervals is presented below. The raw data with all of the samples, means and standard deviations can be found in the appendix.

	data with intervals			
percentage	ht17	ht16	ht15	ht14
10%	20,65±1,02	23,71±1,98	27,06±1,45	26,60±0,74
20%	31,87±1,16	39,55±1,37	41,88±2,85	41,38±4,62
30%	37,43±4,96	48,66±1,97	51,48±2,38	50,61±1,58
40%	42,81±2,71	54,34±1,35	57,88±2,08	59,34±2,11
50%	46,55±2,43	45,48±29,36	62,82±1,96	63,11±1,00
60%	49,36±0,94	62,18±1,49	65,77±1,41	67,91±0,96
70%	51,36±1,36	61,63±3,00	68,68±1,98	72,05±3,48
80%	53,00±0,00	66,47±1,11	71,82±1,08	73,47±2,38
90%	54,76±0,94	67,93±1,77	74,38±1,40	74,83±1,40
100%	57,10±0,00	69,42±0,00	75,72±0,00	78,09±0,00

Diagram 7. Mean values with the confidence intervals of 99%

As it can be seen from the numbers, confidence intervals do not exceed 5% of the mean value most of the time. This means that the curves are relatively accurate, and the difference between ht15 and ht14 can be barely noticed (and potentially neglected) due to the CI overlaps.

As a side note, there was a dip in the 50% for ht16 that we talked about previously. It can be seen in the confidence interval of ±29,36. The smaller dip at 70% is at ±3,00. These two outstanding CIs exist, because of the observations described before.

How to compile and run/use the program

We tried installing OpenAFS and kerberos on windows 10, but it just did not work, this could be because OpenAFS does not support windows 10 or something unrelated, so instead of uploading our source code to KTH we will attach it as a zipped up java project for NetBeans IDE at this address (https://drive.google.com/open?id=1ah5rZ7ahIygIK60ppOLLaq2pxE65ez_j). It can be exported to the IDE

and run from there. The program will then ask the user to specify the population. Here one must enter a number, which will construct the matrix and show a graphical output of it.

After, the user must specify the probability of spreading the sickness, followed by the minimum and maximum length a person can be sick. This is followed by the probability of dying within the aforementioned period. Lastly, the user needs to enter the locations in the population where the first infected are located. This is in the form of (x,y) coordinates. After these inputs, the program will run and simulate until there is no one infected.

For example, a valid series if inputs would be:

20	(size of the matrix)
10	(chance to infect in %)
3	(min sick days)
6	(max sick days)
3	(chance to die in % during sickness period)
3	(x coordinate of the first sick person)
3	(y coordinate of the first sick person)
19	(x coordinate of the second sick person)
19	(y coordinate of the second sick person)
999	(command to start simulation)

At the end of the simulation, the user is presented with a list of options of output they can select. For example, entering (1) outputs total days. To exit the program, type (0). These commands are visible in the console.

Appendices

- Requirements compilation/analysis

Name/Description of each requirement	What is required for the requirement to be satisfied	To what degree has the requirement been met
Create matrix of population	A two dimensional array	Fully
Take care of the inputs	Size of population, 2 probabilities, interval, starting positions of infected	Fully
Code 1 person infecting its neighbours	The random function from the input of probability, and the areas where the matrix stops	Fully
Create a java constructor for the humans	position of the human, status, if they got sick today, how long period of time they will be sick for	Fully
For each people in the matrix, spread the infection	Reset the got_sick value, so a new day has began and they can start infecting.	Fully
Take care of dying probability	Check the random value at the end of the day after individual has infected, if they are dead or not	Fully
Stop the simulation if there is no individual infected	go through the matrix to see if there are any infected	Fully (the simulation ends on day 1)
Outputs	Have different options for the user	Fully
Testing the program	Trying values like 0 or 100 for probabilities, different intervals, different starting locations	Fully
Writing the report	All sections of the outline are covered	Fully

- Outline of the report
    - Problem description
    - A reflection on the pros and cons of the model. A discussion on for what scenarios this model could be useful and how it could be extended/modified to apply to more general scenarios for the spread of an infectious disease in a population.
    - Describe your solution
        - Describe the code/program in both text and as a model by using structured programming and/or UML.
        - A detailed description of how you have planned and performed the validation of each step in your modeling and implementation.
        - Description of how you have validated the results
    - Results and conclusions
    - Information on/description of how to compile and run/use the program
    - Location of the source code on the Linux servers of the school of ICT
    - Appendices
- Source code:

```
package infectionsimulation;

import java.io.*;
import static java.lang.System.exit;
import static java.lang.Thread.sleep;
import java.util.*;



public class InfectionSimulation {

    static int days = 0;                                                    // length of the
simulation
    static int deaths = 0;                                                  // amount of humans
dying to the sickness
    static int getting_sick = 0;                                            // amount of humans
becoming sick
    static int recovered = 0;                                               // amount of humans
recovered from the disease

    public static void main(String[] args) throws InterruptedException {

        System.out.println("Dimension of the population matrix N (where NxN is the matrix)");
        Scanner input = new Scanner(System.in);
        int in1 = input.nextInt();
        //int in1 = 50;

        Human [][] pop = new Human [in1][in1];
        // Creating the matrix
        for (int i = 0; i < pop.length; i++){
            for (int j = 0; j < pop[i].length; j++){
```

```
                pop[i][j] = new Human(i, j, 0);
                //pop[i][j] = 0;
                System.out.print(pop[i][j].status + " ");
                if (j == pop[i].length-1)
                    System.out.println();
            }
        }

        //inputs
        /*
        Random rand = new Random();
        int in2 = 100;
        int in3min = 4;
        int in3max = 8;
        int in4 = 0;
        pop[25][25].status = 1;
        pop[25][25].sickness_period = in3min + rand.nextInt(in3max - in3min + 1);
        */


        System.out.println("Probability of getting sick in %");
        int in2 = input.nextInt();
        Random rand = new Random();
        System.out.println("Length of sickness: min days");
        int in3min = input.nextInt();
        System.out.println("Length of sickness: max days");
        int in3max = input.nextInt();
        System.out.println("Probability of death in %");
        int in4 = input.nextInt();

        System.out.println("Sick people:");
        // Input of initially sick people
        while (true) {
            try {
                System.out.println("Input X coordinate or type 999 to simulate");
                int x = input.nextInt();
                if (x == 999) {
                    break;
                }
                else {
                    System.out.println("Input Y coordinate");
                    int y = input.nextInt();
                    pop[x][y].status = 1;
                    pop[x][y].sickness_period = in3min + rand.nextInt(in3max - in3min + 1);         //
random length based on min and max values
                }
            }
            catch (Exception e) {
                System.out.println("Wrong input! Try again!");
            }
        }


        // Display the matrix with infected humans
        for (int i = 0; i < pop.length; i++){
            for (int j = 0; j < pop[i].length; j++){
```

```
                System.out.print(pop[i][j].status + " ");
                if (j == pop[i].length-1)
                    System.out.println();
            }
        }
        System.out.println();

        // Simulation begins
        while (true) {
            days++;

            // Reset got_sick status at the beginning of the next day
            for (int i = 0; i < pop.length; i++){
                for (int j = 0; j < pop[i].length; j++){
                    pop[i][j].got_sick = false;
                }
            }

            // for every sick human, check the neighbors
            for (int i = 0; i < pop.length; i++){
                for (int j = 0; j < pop[i].length; j++){
                    if ((pop[i][j].status == 1) && (pop[i][j].got_sick == false)
&&(pop[i][j].sickness_period > 0)) {
                        try { //top left neighbor
                            if (pop[i-1][j-1].status == 0) {
                                int val = rand.nextInt(100);
                                if(val <= in2) {
                                    // if random event happens, change status to sick and assign the
following variables
                                    pop[i-1][j-1].status = 1;
                                    pop[i-1][j-1].got_sick = true;
                                    getting_sick++;
                                    pop[i-1][j-1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                                }
                            }
                        }
                        catch(Exception e) {
                        }
                        try { //top
                            if (pop[i-1][j].status == 0) {
                                int val = rand.nextInt(100);
                                if(val <= in2) {
                                    pop[i-1][j].status = 1;
                                    pop[i-1][j].got_sick = true;
                                    getting_sick++;
                                    pop[i-1][j].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                                }
                            }
                        }
                        catch(Exception e) {
                        }
                        try { //top right
                            if (pop[i-1][j+1].status == 0) {
                                int val = rand.nextInt(100);
```

```
                            if(val <= in2) {
                                pop[i-1][j+1].status = 1;
                                pop[i-1][j+1].got_sick = true;
                                getting_sick++;
                                pop[i-1][j+1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                            }
                        }
                    }
                    catch(Exception e) {
                    }
                    try { // left
                        if (pop[i][j-1].status == 0) {
                            int val = rand.nextInt(100);
                            if(val <= in2) {
                                pop[i][j-1].status = 1;
                                pop[i][j-1].got_sick = true;
                                getting_sick++;
                                pop[i][j-1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                            }
                        }
                    }
                    catch(Exception e) {
                    }
                    try { // right
                        if (pop[i][j+1].status == 0) {
                            int val = rand.nextInt(100);
                            if(val <= in2) {
                                pop[i][j+1].status = 1;
                                pop[i][j+1].got_sick = true;
                                getting_sick++;
                                pop[i][j+1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                            }
                        }
                    }
                    catch(Exception e) {
                    }
                    try { // bottom left
                        if (pop[i+1][j-1].status == 0) {
                            int val = rand.nextInt(100);
                            if(val <= in2) {
                                pop[i+1][j-1].status = 1;
                                pop[i+1][j-1].got_sick = true;
                                getting_sick++;
                                pop[i+1][j-1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                            }
                        }
                    }
                    catch(Exception e) {
                    }
                    try { // bottom
                        if (pop[i+1][j].status == 0) {
                            int val = rand.nextInt(100);
```

```
                                if(val <= in2) {
                                    pop[i+1][j].status = 1;
                                    pop[i+1][j].got_sick = true;
                                    getting_sick++;
                                    pop[i+1][j].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                                }
                            }
                        }
                        catch(Exception e) {
                        }
                        try { // bottom right
                            if (pop[i+1][j+1].status == 0) {
                                int val = rand.nextInt(100);
                                if(val <= in2) {
                                    pop[i+1][j+1].status = 1;
                                    pop[i+1][j+1].got_sick = true;
                                    getting_sick++;
                                    pop[i+1][j+1].sickness_period = in3min + rand.nextInt(in3max -
in3min);
                                }
                            }
                        }
                        catch(Exception e) {
                        }
                        // if death random even occurs, change status to 2 (dead)
                        if (in4 != 0) {
                            boolean val = rand.nextInt(100/in4)==0;
                            if(val == true) {
                                pop[i][j].status = 2;
                                deaths++;
                            }
                        }

                        // at the end of the day, reduce sickness period of the current human
                        pop[i][j].sickness_period--;
                        if (pop[i][j].sickness_period == 0) {
                            pop[i][j].status = 9;
                            recovered++;
                        }
                    }
                }
            }

            // the loop continues until there are no sick left
            boolean continue_app = false;

            for (int i = 0; i < pop.length; i++){
                for (int j = 0; j < pop[i].length; j++){
                    if (pop[i][j].status == 1)
                        continue_app = true;
                }
            }

            if (continue_app == false) {
                for (int i = 0; i < pop.length; i++){
```

```
                    for (int j = 0; j < pop[i].length; j++){
                        System.out.print(pop[i][j].status + " ");
                        if (j == pop[i].length-1)
                            System.out.println();
                    }
                }
                break;
            }
        }

        // Output interface
        System.out.println("What statistics do you want to see? Enter the according number:");
        System.out.println("1. Total days");
        System.out.println("2. Total deaths");
        System.out.println("3. Total people getting sick excluding patient(s) zero");
        System.out.println("4. Average deaths per day");
        System.out.println("5. Average people getting sick per day");
        System.out.println("6. Average recoveries per day");
        System.out.println("7. All from the above");
        System.out.println("");
        System.out.println("0. Exit");

        while (true) {
            //int query = 5;
            int query = input.nextInt();

            if (query == 1){
                System.out.println("Total days: " + days);
            }
            if (query == 2){
                System.out.println("Total deaths: " + deaths);
            }
            if (query == 3){
                System.out.println("Total people getting sick excluding the initial sick: " +
getting_sick);
            }
            if (query == 4){
                double deaths_per_day = (double)deaths/(double)days;
                System.out.println("Average deaths per day: " + deaths_per_day);
            }
            if (query == 5){
                double sick_per_day = (double)getting_sick/(double)days;
                System.out.println("Average people getting sick per day: " + sick_per_day);
                //exit(0);
            }
            if (query == 6){
                double recovered_per_day = (double)recovered/(double)days;
                System.out.println("Average recoveries per day: " + recovered_per_day);
            }
            if (query == 7){
                System.out.println("Total days: " + days);
                System.out.println("Total deaths: " + deaths);
                System.out.println("Total people getting sick excluding the initial sick: " +
getting_sick);
                double deaths_per_day = (double)deaths/(double)days;
                System.out.println("Average deaths per day: " + deaths_per_day);
```

```
                double sick_per_day = (double)getting_sick/(double)days;
                System.out.println("Average people getting sick per day: " + sick_per_day);
                double recovered_per_day = (double)recovered/(double)days;
                System.out.println("Average recoveries per day: " + recovered_per_day);
            }
            if (query == 0){
                exit(0);
            }
            sleep(50);
        }
    }// Human class constructor
    public static class Human {
        int x;                                                          // x coord
        int y;                                                          // y coord
        int status = 0;                                                 // 0 - healthy; 1 -
sick; 2 - dead; 9 - recovered
        boolean got_sick = false;                                       // got sick today
        int sickness_period = 0;
        Human(int i){
            status = i;
        }
        Human(int i, boolean j){
            status = i;
            got_sick = j;
        }
        Human(int i, int j, int k){
            x = i;
            y = j;
            status = k;
        }
    }
}
```

- Raw data

	number of infected indiduals per day			
	ht17	ht16	ht15	ht14
10%	20,65	23,71	27,06	26,60
20%	31,87	39,55	41,88	41,38
30%	37,43	48,66	51,48	50,61
40%	42,81	54,34	57,88	59,34
50%	46,55	45,48	62,82	63,11
60%	49,36	62,18	65,77	67,91
70%	51,60	61,63	68,68	72,05
80%	53,30	66,48	71,82	73,47
90%	54,76	67,93	74,38	74,83
100%	57,11	69,42	75,72	78,09

	number of ifected individuals per day									
	0%	1%	2%	3%	4%	5%	6%	7%	8%	9%
ht17	0	0,26636	0,63526	2,6786	7,60904	10,64458	14,20061	16,007	17,809	19,532
ht16	0	0,21392	0,26598	0,631	1,63376	8,73328	12,62268	15,6789	18,837	20,73306
ht15	0	0,0946	0,45068	1,15842	3,452	8,52802	15,576	18,5862	22,4498	23,99
ht14	0	0,17576	0,16968	0,94714	3,21512	9,544	15,704	19,218	22,12	25,40736

Means for each of the measurements

	data with intervals			
percentage	ht17	ht16	ht15	ht14
10%	20,65±1,02	23,71±1,98	27,06±1,45	26,60±0,74
20%	31,87±1,16	39,55±1,37	41,88±2,85	41,38±4,62
30%	37,43±4,96	48,66±1,97	51,48±2,38	50,61±1,58
40%	42,81±2,71	54,34±1,35	57,88±2,08	59,34±2,11
50%	46,55±2,43	45,48±29,36	62,82±1,96	63,11±1,00
60%	49,36±0,94	62,18±1,49	65,77±1,41	67,91±0,96
70%	51,36±1,36	61,63±3,00	68,68±1,98	72,05±3,48
80%	53,00±0,00	66,47±1,11	71,82±1,08	73,47±2,38
90%	54,76±0,94	67,93±1,77	74,38±1,40	74,83±1,40
100%	57,10±0,00	69,42±0,00	75,72±0,00	78,09±0,00

Means with confidence intervals

99% confidence interval										
17	10	20	30	40	50	60	70	80	90	100
1	20,88	32,63	38,07	43,02	44,42	48,46	53,3	53,3	55,13	57,1
2	20,44	31,98	39,97	45,68	45,69	49,96	51,58	53,3	53,3	57,1
3	19,4	32,63	30,07	41	47,02	49,96	51,58	53,3	55,13	57,1
4	21,87	31,98	41	39,97	45,69	49,96	49,96	53,3	55,13	57,1
5	20,67	30,17	38,07	44,42	49,97	48,46	51,58	53,3	55,13	57,1
mean	20,652	31,878	37,436	42,818	46,558	49,36	51,6	53,3	54,764	57,1
std dev	0,88717	1,008598	4,306539	2,356612	2,117373	0,821584	1,181186	0	0,818401	0
conf	1,021971	1,16185	4,9609	2,714689	2,4391	0,94642	1,360662	#NUM!	0,942754	#NUM!
16										
1	23,15	40,29	49	53,17	54,32	62,47	65,76	67,54	69,42	69,42
2	25,03	39,04	46,28	53,17	0	60,95	60,96	65,77	67,54	69,42
3	24,67	39,63	49	54,32	56,79	60,95	59,5	67,54	69,42	69,42
4	24,8	40,96	51	55,53	59,5	62,47	62,47	65,77	67,54	69,42
5	20,94	37,86	48,06	55,53	56,79	64,07	59,5	65,77	65,76	69,42
mean	23,718	39,556	48,668	54,344	45,48	62,182	61,638	66,478	67,936	69,42
std dev	1,72034	1,18934	1,712519	1,180076	25,49004	1,300584	2,611747	0,969469	1,537296	0
conf	1,981739	1,370055	1,972729	1,359384	29,36315	1,498203	3,008591	1,116776	1,770882	#NUM!

15										
1	28,05	41,65	49	58,11	60,96	65,76	69,42	71,4	73,5	75,72
2	27,86	39,03	49,98	56,79	60,96	67,54	71,4	71,4	75,72	75,72
3	28,02	43,07	54,32	56,79	64,07	65,76	67,54	71,4	75,72	75,72
4	25,55	40,25	52,06	56,79	64,07	65,76	67,54	71,4	73,5	75,72
5	25,82	45,4	52,06	60,96	64,07	64,07	67,54	73,5	73,5	75,72
mean	27,06	41,88	51,484	57,888	62,826	65,778	68,688	71,82	74,388	75,72
std dev	1,260893	2,481874	2,069947	1,809923	1,703417	1,227078	1,720791	0,939149	1,215944	0
conf	1,45248	2,858985	2,384467	2,084933	1,962244	1,413527	1,982258	1,081848	1,400702	#NUM!
14										
1	27,43	39,65	49,98	60,95	62,48	67,54	73,5	75,34	75,73	78,09
2	27,14	37,28	52,06	56,79	62,47	69,42	69,42	71,4	73,5	78,09
3	26,07	39,03	49	60,95	64,07	67,54	69,42	73,5	73,5	78,09
4	26,02	47,15	49,98	59,9	64,07	67,54	71,4	75,72	75,72	78,09
5	26,35	43,82	52,06	58,11	62,47	67,54	76,54	71,4	75,72	78,09
mean	26,602	41,386	50,616	59,34	63,112	67,916	72,056	73,472	74,834	78,09
std dev	0,644259	4,017827	1,377563	1,838015	0,87454	0,840762	3,021536	2,069425	1,217777	0
conf	0,742152	4,628319	1,586878	2,117294	1,007423	0,968512	3,480646	2,383866	1,402813	#NUM!

Raw data. 5 measurements per disease spread %. Mean, standard deviation and confidence intervals calculated at the bottom

17	0	1	2	3	4	5	6	7	8	9
1	0	0,3077	0	5,27	9,5374	11,4772	13,9818	16,385	18,72	19,62
2	0	0,2	0,5961	1,39	7,62	10,69	12,275	16,88	17,93	20,1
3	0	0,4167	0,7727	3,033	6,6048	10,29	14,819	16,39	15,17	20,42
4	0	0,4074	1,3306	1,95	7,1237	9,4551	14,839	15,29	19,35	18,9
5	0	0	0,4769	1,75	7,1593	11,3106	15,08823	15,09	17,875	18,62
mean	0	0,26636	0,63526	2,6786	7,60904	10,64458	14,20061	16,007	17,809	19,532
std dev	0	0,172883	0,482844	1,572784	1,136333	0,818458	1,154737	0,775674	1,595926	0,766499
conf	0	0,199152	0,55621	1,811762	1,308994	0,942819	1,330194	0,893534	1,83842	0,882965
16										
1	0	0,3846	0	0,6388	0,8292	8,1607	14,5034	15,85	20,86	21,8303
2	0	0	0,8571	1,6125	2,9356	10,645	9,98	12,52	18,41	24,64
3	0	0	0,125	0,7037	1,0408	8,9126	14,5	18,0625	19,96	18,14
4	0	0,375	0	0	3,3632	7,6481	11,87	15,812	14,005	20,024
5	0	0,31	0,3478	0,2	0	8,3	12,26	16,15	20,95	19,031
mean	0	0,21392	0,26598	0,631	1,63376	8,73328	12,62268	15,6789	18,837	20,73306
std dev	0	0,197379	0,359682	0,623019	1,445151	1,159744	1,919848	1,995565	2,887429	2,578364
conf	0	0,22737	0,414334	0,717684	1,664735	1,335962	2,211561	2,298783	3,326162	2,970136
15										
1	0	0	0,7931	2,4574	2,36	8,6161	17,23	18,84	22,78	23,82
2	0	0	0,4117	1,5217	2,18	10,8465	16,07	18,061	22,109	23,25
3	0	0	0,2307	0,7317	5,63	8,4493	14,58	19,53	22,98	22,93
4	0	0,238	0,6111	0	2,41	7,7722	14,32	20,21	21,12	26,76
5	0	0,235	0,2068	1,0813	4,68	6,956	15,68	16,29	23,26	23,19
mean	0	0,0946	0,45068	1,15842	3,452	8,52802	15,576	18,5862	22,4498	23,99
std dev	0	0,129541	0,251184	0,915054	1,592787	1,451879	1,178995	1,511644	0,85602	1,582166
conf	0	0,149224	0,289351	1,054093	1,834804	1,672487	1,358139	1,741332	0,986089	1,82257
14										
1	0	0	0,25	0,5333	7,6886	8,68	16,91	19,68	23,77	27,73
2	0	0,4375	0	0,5714	2,093	10,57	16,27	19,12	22,9	24,62
3	0	0,111	0,4166	1,6969	0,25	8,6	14,35	18,05	21,41	25,87
4	0	0,1875	0	1,4341	2,97	9,31	14,21	19,29	23,85	24,029
5	0	0,1428	0,1818	0,5	3,074	10,56	16,78	19,95	18,67	24,7878
mean	0	0,17576	0,16968	0,94714	3,21512	9,544	15,704	19,218	22,12	25,40736
std dev	0	0,161878	0,176883	0,572636	2,745137	0,971766	1,322679	0,729431	2,163932	1,458797
conf	0	0,186474	0,20376	0,659646	3,162249	1,119422	1,523655	0,840265	2,492732	1,680456

Raw data. 5 measurements per disease spread %. Mean, standard deviation and confidence intervals calculated at the bottom

seed	100	3	123456
attempt 1	1,6568	1,8467	4,4743
attempt 2	1,6568	1,8467	4,4743
attempt 3	1,6568	1,8467	4,4743
attempt 4	1,6568	1,8467	4,4743
attempt 5	1,6568	1,8467	4,4743
attempt 6	1,6568	1,8467	4,4743
attempt 7	1,6568	1,8467	4,4743
attempt 8	1,6568	1,8467	4,4743
attempt 9	1,6568	1,8467	4,4743
attempt 10	1,6568	1,8467	4,4743

Reliability test. Results for input (20, 5, 4, 8, 0, (10,10))

References

[1] Modelling the spread of the disease. Education Bureau of Hong Kong, 2018
https://www.edb.gov.hk/attachment/en/curriculum-development/kla/ma/res/example_SS_2_modelling_eng.pdf

Mandar Joshi - Reflection

We started the assignment by creating the basic things we needed in the program. like the matrix and the inputs. After this we discussed together how we should continue with the next part of the assignment, using paint and some very basic state diagrams. From this we simply implemented, and while we were implementing, everytime we tried something new we tested our implementation. This we did using different kinds of values, for example for the probability of getting sick we tested 0 and 100. This effort was duplicated throughout the implementation, with constantly adding new features and testing them on an iterative basis. These tests formed the main validation and verification for the model.

These general skills of modelling and thinking about the project before actually implementation helps a lot, even the simplest state diagram goes a long way to help someone get started with their implementations. As for hose these skills can be applied to other assignments, one basically starts to get the mindset of thinking about what to do, and checking the facts before actually attempting to do what one set out to. Since we did this in a group, using these models made communication easier between us since we both understood what we had to do in a simple and effective manner. These communication skills can also be applied to other areas and assignments.

Albert Asratyan - Reflection

Modeling is the connection between thinking about the problem and implementing the problem. It allowed us to split the problem into many sub parts and focus on them one by one. In the end we have achieved a full working model, that is able to monitor and simulate infection spread among the population. Implementing the model was done step by step. Each part was tested before proceeding to the next one. When blocks were developed fully and it was time to combine them, testing was always performed afterwards. Tests were used for verification and validation.

These skills are the most important general skills for us as future engineers, because these skills can be applied to solving any problem, no matter the field. Being able to model correctly also develops a certain way of thinking and approaching technical problems. Having these skills, anyone can make it easier to understand the core of the problem, making it possible, for example, to explain the problem to someone who does not have much knowledge in the area, given logical explanations.