Semester Project

Shannon Cordoni

Shannon.Cordoni@Marist.edu

December 13, 2021

1 Problem One: Semester Project

1.1 The Data Structure

Given Professor Labouseur's Covid Simulation Articles, our job was to duplicate his Covid testing protocol in our own simulation.

1.2 Main Class

1.2.1 Description

This class is where most of our work is done, it takes in a parameter to be our population size and from there we infect the population at a 2% infection rate and simulate the testing protocol to find out how many of each case types their are in our population. The methods contained in this class preform all these operations and more.

```
1
2  /*
3   *
4   * Semester Project
5   * Due Date and Time: 12/15/21 before 12:00am
6   * Purpose: to develop a simulation testing program for covid screenings
7   * Input: The user will be inputting a population number to test the simulation on
8   * Output: The program will output the infection rate of the population
9   * @author Shannon Cordoni
10  *
11  */
12
13 import java.io.File;
14 import java.io.FileNotFoundException;
15 import java.util.*;
16
17
18 public class SimulationCordoni {
19
20     //Declare keyboard
```

```
21      static Scanner keyboard = new Scanner(System.in);
22
23      public static void main(String[] args) {
24
25
26          //output population
27          System.out.println("Population Simulation Number: " + args[0]);
28
29
30
31          //Declare and initialize variables
32
33          //start with population being 1000
34          int[] population = new int[Integer.parseInt(args[0])];
35
36          //set the population size
37          double populationSize = Double.parseDouble(args[0]);
38
39          int groupNumber = 0;
40
41          int groupSize = 8;
42
43          double infectionRate = 0.02;
44
45          double numberInfected = 0.0;
46
47          int index = 0;
48
49          int numberOfTests = 0;
50
51          int numberOf1Tests = 0;
52
53          int numberOf2Tests = 0;
54
55          int numberOf3Tests = 0;
56
57
58          int case1occurences = 0;
59
60          int case2occurences = 0;
61
62          int case3occurences = 0;
63
64          int popindex = 0;
65
66          //we are going to let 1 be infected and let 0 be healthy
67          //This method is going to randomly infect the population
68
69          numberInfected = populationSize * infectionRate;
70
71          int intNumberInfected = (int) numberInfected;
72
73          System.out.println("Number Infected: " + intNumberInfected);
74
75          for (int i = 0; i < intNumberInfected ; i++){
76
77              popindex = getRandomInfection(population);
78
79              //if the index is already set to 1 we do not want to reinfect
80              if(population[popindex] == 1){
81                  //System.out.println("Double Infection");
82
83                  //we keep calling random then until we find a non-infected person
84                  while(population[popindex] != 0){
85                      //System.out.println("new Infection");
```

```
                popindex = getRandomInfection(population);
            }//while

            population[popindex] = 1;

        }//if

        //otherwise we just infect
        else{
            population[popindex] = 1;
        }//else


    }//for


    //print out the population
    for (int i = 0; i < population.length; i++){

       //System.out.print(population[i]);

    }//for

    //split the population into groups of 8
    //want to make a 2 dimensional array so that we can go through each group of 8
    groupNumber = (Integer.parseInt(args[0])) / groupSize ;

    System.out.println("Group Number: " + groupNumber);

    //here we are going to make a copy of our group array to search through so that
    //we still have an original
    int[][] grouparray = new int [groupNumber][groupSize];

    int[][] groupArrayCopy = grouparray.clone();


    //go through the group array and input the population
    //then in each index of the group make an array the size of group size
    for (int i = 0; i < groupNumber; i++){

        for ( int j = 0; j < groupSize; j++){

           grouparray[i][j] = population[index];
           index++;
        }//for j
    }//for i

    //print out the group array to see each group of 8
    //System.out.println(Arrays.deepToString(grouparray));



    //now we test for infections!
    //testing 125 groups of 8
    for (int i = 0; i < groupNumber; i++){

        //go through and find each group sum to see if we need to individually test

        //find sum
        //System.out.println(findArraySum(groupArrayCopy[i]));

        //if the sum is equal to 0 then we dont have an infection
        if(findArraySum(groupArrayCopy[i]) == 0){

        //System.out.println("No infection in this group!");
```

```
151             numberOf1Tests++;
152             case1occurences++;
153
154
155             }//if
156
157             //else we have an infection
158             else{
159
160                 int subgroup1infection = 0;
161                 int subgroup2infection = 0 ;
162
163                     //now we split into 2 groups of 4 and retest
164                     //Here we test the first 4 in the group
165                     for(int k = 0; k < groupSize/2; k++){
166
167                         numberOf2Tests++;
168
169                         //if we find an infection then we set the variable to one
170                         //so that we can see later if we have a case 3 infection
171                         if(groupArrayCopy[i][k] == 1){
172
173                             subgroup1infection = 1;
174
175                         }//if
176
177                     }//for
178
179                     //here we test the second 4 in the group
180                     for(int k = 4; k < groupSize; k++){
181
182                         numberOf2Tests++;
183
184                         //if we find an infection then we set the variable to one
185                         //so that we can see later if we have a case 3 infection
186                         if(groupArrayCopy[i][k] == 1){
187
188                             subgroup2infection = 1;
189
190                         }//if
191
192                     }//for
193
194                     //If there is both a subgroup 1 and subgroup 2 infection then we
195                     //have a case 3
196                     if((subgroup1infection == 1) && (subgroup2infection == 1)){
197
198                         case3occurences++;
199
200                         //here we account for the original test of all 8, and the two
201                         //tests for each group of 4
202                         numberOf3Tests = numberOf3Tests + 3;
203
204                         //we loop through for testing all 8 in the group
205                         for(int j = 0; j < groupSize; j++){
206
207                             numberOf3Tests++;
208
209                         }//for
210
211                     }//if both subgroup
212
213
214                     //if we only have a subgroup 1 or subgroup 2 infection then we
215                     //have a case 2
```

```
                    else if(subgroup1infection == 1){

                        case2occurences++;

                    }//if subgroup 1

                    else if(subgroup2infection == 1){

                        case2occurences++;

                    }//if subgroup 2

            }//else

        }//for i


        //print out the results
        System.out.println("Case 1 occurences: " +  case1occurences);
        System.out.println("Case 2 occurences: " +  case2occurences);
        System.out.println("Case 3 occurences: " +  case3occurences);

        System.out.println("Number of case 1 tests: " +  numberOf1Tests);
        System.out.println("Number of case 2 tests: " +  numberOf2Tests);
        System.out.println("Number of case 3 tests: " +  numberOf3Tests);

        numberOfTests = numberOf1Tests + numberOf2Tests + numberOf3Tests;

        System.out.println("Number of total tests: " +  numberOfTests);

    }//main

    //This method randomly infects the population
    public static int getRandomInfection(int[] populationArray)
    {


        int randomIndex = 0;

        Random randomize = new Random();

        randomIndex = randomize.nextInt(populationArray.length);

        //System.out.println("random index" + randomIndex);

        return randomIndex;

    }//getRandomInfection

    //this method find the sum of the array passed in
    public static int findArraySum(int[] array) {

        int sum = 0;

        for (int i = 0; i < array.length; i++) {
            sum = sum + array[i];
        }//for

        return sum;
    }//find array sum

}//semester cordoni
```

1.2.2 Description of Main Code

The main class above consists of different methods to help simulate the Covid testing protocol. The good parts of the code first include taking in the population parameter and setting it to be the population size. Then we make an array the size of our population and "infect" it, calling our $getRandomInfection$ method to return random indexes to set as "1" to signify an infection. At each pass we check to see if our index is already infected before we possibly re-infect someone. We do this for 2% of whatever population was passed in as a parameter. Then we split this into groups of 8 and we create a 2 dimensional array with the first dimension being the group number and the second dimension being the group size. Along with passing each line of this 2 dimensional group array into our testing simulation to see how many occurences of each case there was and how many tests it took to test for each case. Case 1 being that there were 0 infections in the group, case 2 being that there was 1 infection in the group, and case 3 being that there was more than one infection in the group. Then to keep the non-simulation code out of the main method, different methods were used to help organize the code better. These methods include the $getRandomInfection$, and $findArraySum$ method.

To go into more detail of the actual simulation testing we go through each group of 8 and we take the sum of each group. If the group sum is 0 then we have a case 1 occurrence and 0 infections present. If the group sum is greater than 0 then we have an infection present and we must do further testing to see how many infections are present. First we loop through the first group of 4 and see if there are any infections present, during this time we increase the numbers of case 2 tests and if an infection is found then we set our $subgroup1infection$ variable to 1 for later. Then we do the same for the second group of 4 and if an infection is found then we set our $subgroup2infection$ variable to 1 for later. If both variables are set to 1 then we have a case 3 occurrence, of which there is more than 1 infection in the group. If only one of the variables is set to 1 then we have a case 2 occurrence. After testing we print out our results, and keep increasing the population size to see if their are any trends.

The $getRandomInfection$ method takes in an array representing the population, and gets a random index in this array to return to the main method for infection. The index returned then turns from 0 to 1 to represent an infection.

The $findArraySum$ method takes in an array, and returns the sum of it. This comes into play when initially determining if there is an infection in each group. If the array sum is 0 then there is no infection, however, if it is 1 then we need to do further testing to determine if the group falls under a case 2 occurrence or a case 3 occurrence.

1.3 Overall:

Overall, the simulation testing representation was successful in implementation. To go through each population possibility we can create a table for better data understanding, this table will show some population possibilities:

Populations		
1000	10000	100000
Population: 1000	Population: 10000	Population: 100000
Number Infected: 20	Number Infected: 200	Number Infected: 2000
Group Number: 125	Group Number: 1250	Group Number: 12500
Case 1 occurences: 106	Case 1 occurences: 1067	Case 1 occurences: 10655
Case 2 occurences: 18	Case 2 occurences: 177	Case 2 occurences: 1767
Case 3 occurences: 1	Case 3 occurences: 6	Case 3 occurences: 78
Number of case 1 tests: 106	Number of case 1 tests: 1067	Number of case 1 tests: 10655
Number of case 2 tests: 152	Number of case 2 tests: 1464	Number of case 2 tests: 14760
Number of case 3 tests: 11	Number of case 3 tests: 66	Number of case 3 tests: 858
Number of total tests: 269	Number of total tests: 2597	Number of total tests: 26273

The table above shows a quick understanding of the population possibilities. As the results get bigger for the increasing populations the results follow more of a hyper geometric distribution vs a binomial distribution. The difference between the two distributions is that the hyper geometric distribution is sampling without replacement, meaning the probabilities change at each turn and depend on the result of the previous turn. The binomial distribution is sampling with replacement meaning that each turn is independent of each other and have an equal chance of happening. For the future there are always many ways to improve ones code, for this simulation I am sure there is a better way to go about keeping track of how many tests are involved with each case occurrence, along with overall simplification of the code to be more effective. Overall, this simulation assignment went smoothly and was successful in implementation.