Modeling the Spread of Influenza and the Common Cold in Schools

Ryan D. Torok

[11rytorok3900@gmail.com](mailto:11rytorok3900@gmail.com)

(281)-475-6297 (C)

*Foreword*

The following analysis originates from the SIR method presented by David Smith and Lang Moore in 2004 for the MAA. An algorithm that will be introduced to accurately model spread of infection in a classroom setting. The algorithm models a common intermediate or high school setting ranging from five to fifty classrooms, with up to forty desks each. However, both Smith and Moore’s SIR model and the following algorithm may be applied to larger sets of people, a city, or even across several continents.

Contents

[Introduction 3](#_Toc471121445)

[Methods and Design 4](#_Toc471121446)

[A Practical Example: The Common Cold 5](#_Toc471121447)

[Data across many iterations 10](#_Toc471121448)

[Discussion 16](#_Toc471121449)

[The Economics of Disease 18](#_Toc471121450)

[Conclusion 20](#_Toc471121451)

[Works Cited 21](#_Toc471121452)

[Appendix 22](#_Toc471121453)

[Inputs.txt 22](#_Toc471121454)

[Actor.java 22](#_Toc471121455)

[Student.java 27](#_Toc471121456)

[Class.java 29](#_Toc471121457)

[School.java 30](#_Toc471121458)

# Introduction

Catching colds and influenza has proved to be the norm rather than the exception in schools packed to capacity with adolescents. Students are further impacted by the intense levels of stress and anxiety attributed to classes, social relations, and sleep deprivation. The colloquialism “Cover your Cough” is largely ineffective with students surrounded by hundreds of attractive hosts for a hungry virus. High school students place their hopes on falling ill on the rare light work weeks. Unfortunately, little can be done to abate the aggression of pathogens and therefore the difficulty of avoiding spreading disease to peers.

A more feasible problem to solve is the issue of the incentive to attend school despite illness. Many school districts fail to draw a distinction between sick days and unexcused absences. Staying home from school due to illness results in students receiving make-up work and missing lessons, negatively impacting academic performance. In addition, many school districts provide incentives to not stay home ill because they are compensated by state education boards based on how many students are present each day. The following algorithm and statistical process will illustrate how students who stay home when ill will reduce the risk of peers from being infected.

# Methods and Design

The algorithm models student contact using a set of variable parameters, using the following steps and conditions.

1. A school consists of a fixed number of classrooms, .
2. A classroom consists of a fixed number of rows and columns of desks, , respectively.
3. The total number of students, , equals , such that the total number of students equals the total number of desks in the school. There are no empty seats in the initial state.
4. A day consists of a fixed number of class periods, .
5. Each student is randomly assigned a rigid schedule, which assigns him/her a room and seat for each class period. Every day, all students traverse their schedule, visiting the same classes in the same order and sitting in the same seat.
6. It is possible for a student to visit the same room more than once on his/her schedule, and he/she may or may not use the same seat among visits.
7. A student’s health is defined using four states:
   * State 0: Unaffected
   * State 1: Infected and moderately contagious but not displaying symptoms
   * State 2: Infected and peak contagiousness
   * State 3: Recovered and no longer contagious
8. The algorithm begins with one student initially infected in State 1.
9. A period consists of all students in each class sitting in their assigned seats. An infected student’s chance of infecting each unaffected student in the room is given by the formula , where p is the probability of infecting an unaffected student horizontally or vertically adjacent to the student, is a constant of exponential decay, and is the Pythagorean distance between the students, with two students horizontally or vertically adjacent representing .
10. Once a student is infected, he/she enters state 1.
11. A student remains in state 1 for a variable number of days, modeled by a normal distribution with mean and standard deviation .
12. Once a student enters state 2, he remains in state 2 for a variable number of days, modeled by a normal distribution with mean and standard deviation .
13. Note that while a student enters state 1 immediately upon infection, he/she will only change to state 2 or 3 between days, e.g. a student whom the normal distribution would change from state 1 to 2 on day 2.6 would change at the start of day 3.
14. A student upon entering state 2 has a fixed probability of remaining absent while sick. Any such student will abstain from attending classes until entering state 3 and thus cannot infect other students.
15. The algorithm terminates when all students are either in state 0 or 3

# A Practical Example: The Common Cold

According to Dr. Kimberly Dowdell at Tufts Medical Center, in an interview with *The Boston Globe,* it usually takes between one and three days before noticeable symptoms arise after infection, and symptoms tend to last three to ten days (Humphries 1). Dr. Sharon Bergquist of Emory University claims the typical incubation time to be two to five days, and the average length of symptoms to be about seven days (Landau 1).

To cover a wide range of tolerances and levels of immunity, the following model sets the time to shift from state 1 to state 2 as a normal distribution with mean 3.5 days and standard deviation 1.75 days. This places approximately 95% of cases within the range of one to six days of incubation, per the empirical rule or normal distributions. To maintain symmetry in the distribution, any student spending six days in state 1 will certainly enter state 2 on the seventh day, since it is impossible to enter state 2 on the zeroth day (the day the student catches the disease).

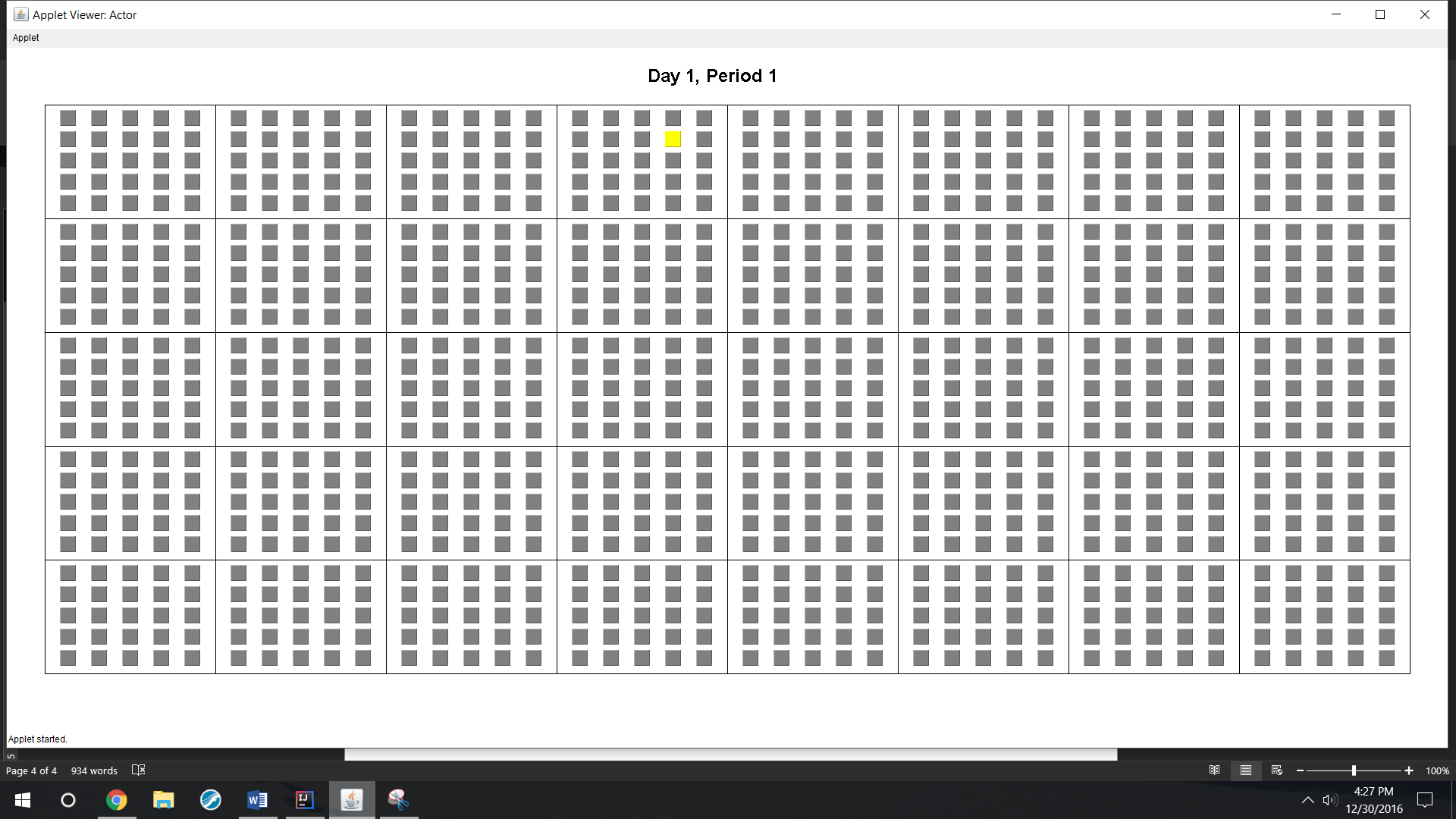
By the same logic, the time in state 2 is modeled by a normal distribution with mean 6.5 days and standard deviation 1 day.

This example shows this algorithm running in a school of 1000 students, separated into 40 rooms, each with 25 students in 5 x 5 matrices.

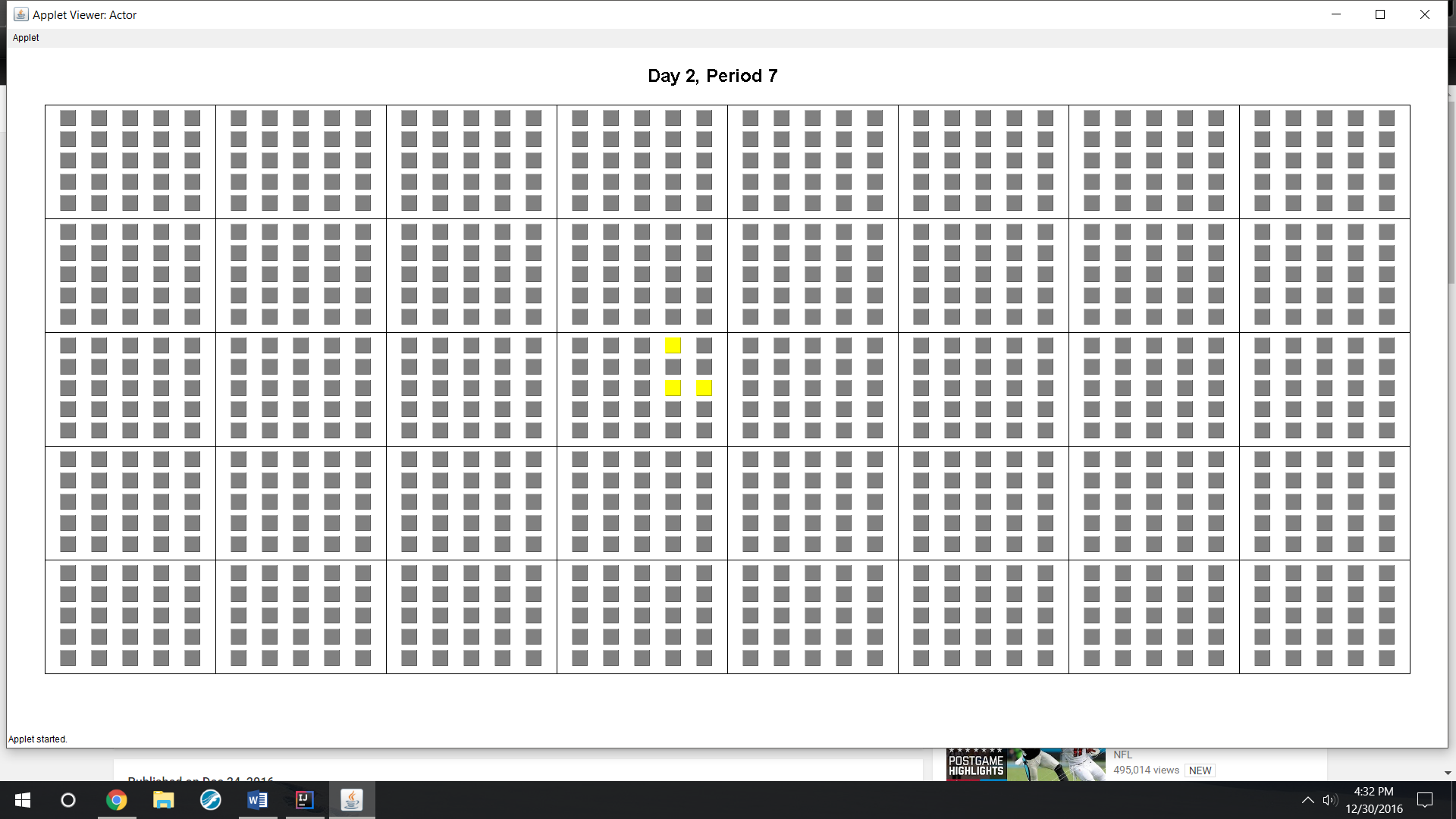
Variables:

* (probability of infecting adjacent student while in state 1)
* (probability of infecting adjacent student while in state 2)

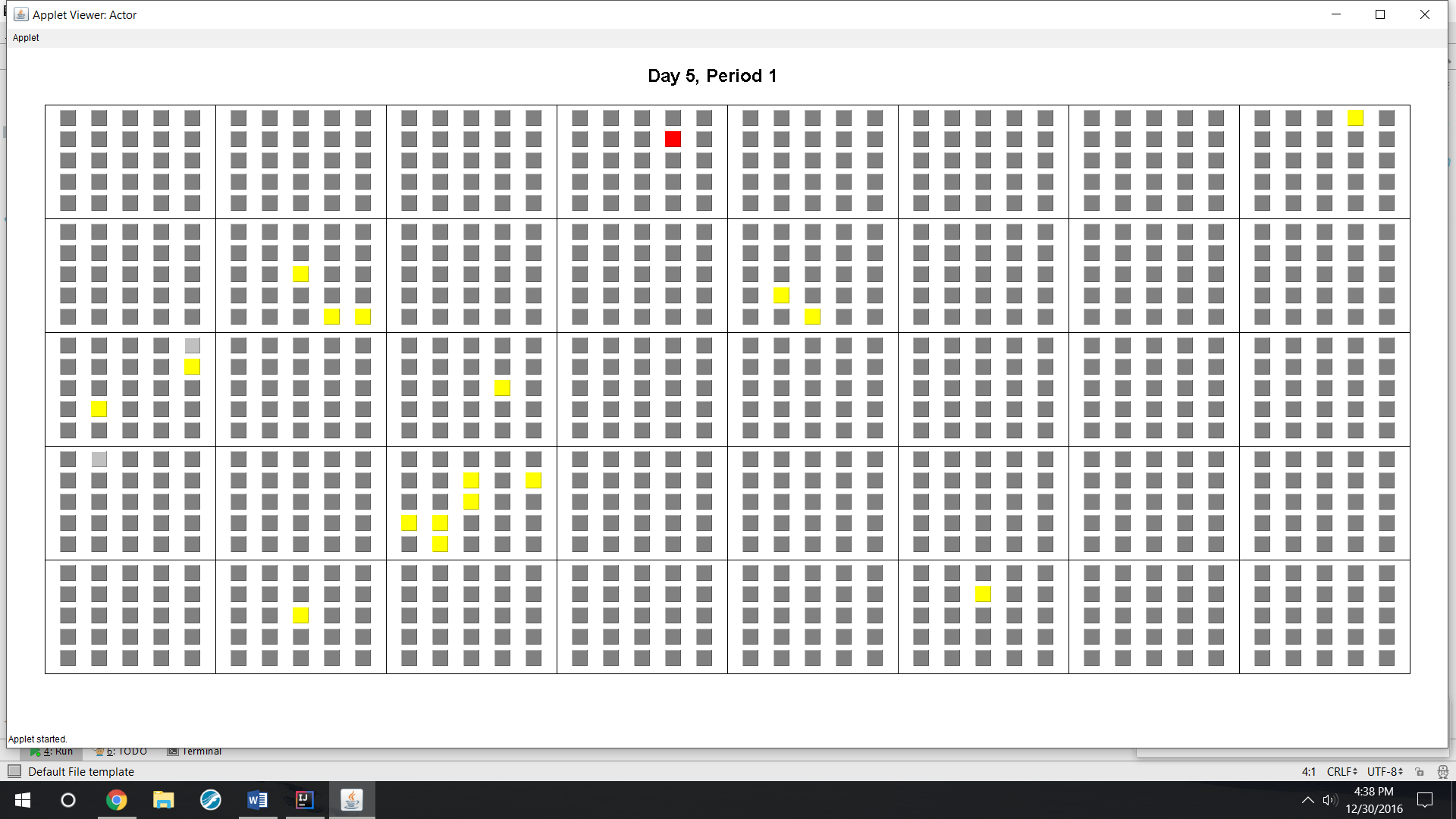
Initial State: One student enters field in state 1, infected but asymptomatic, (yellow), all others in state 0, unaffected (dark gray).



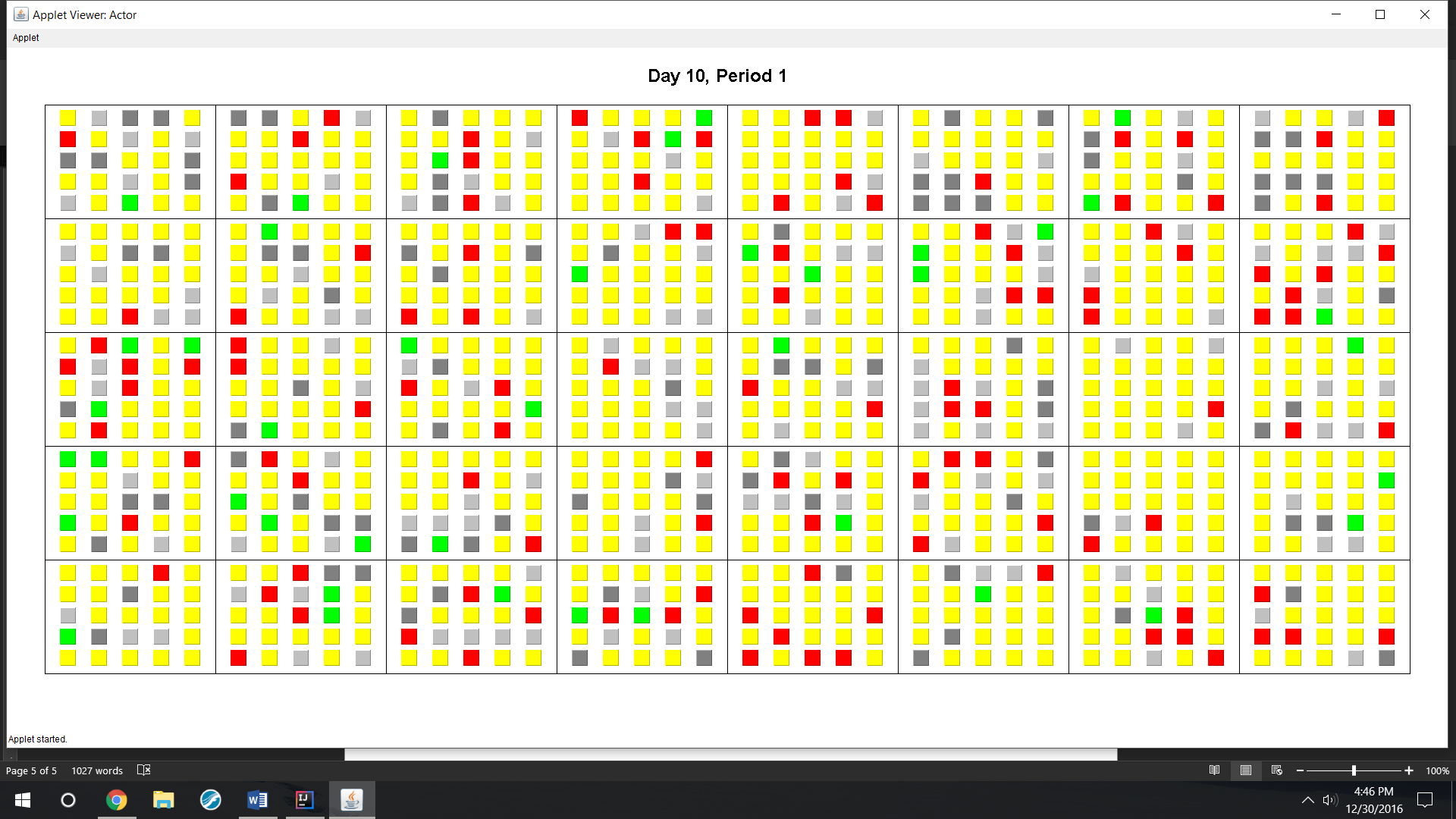
Day 2, Period 7: The initially infected student infects two students while still in state 1.



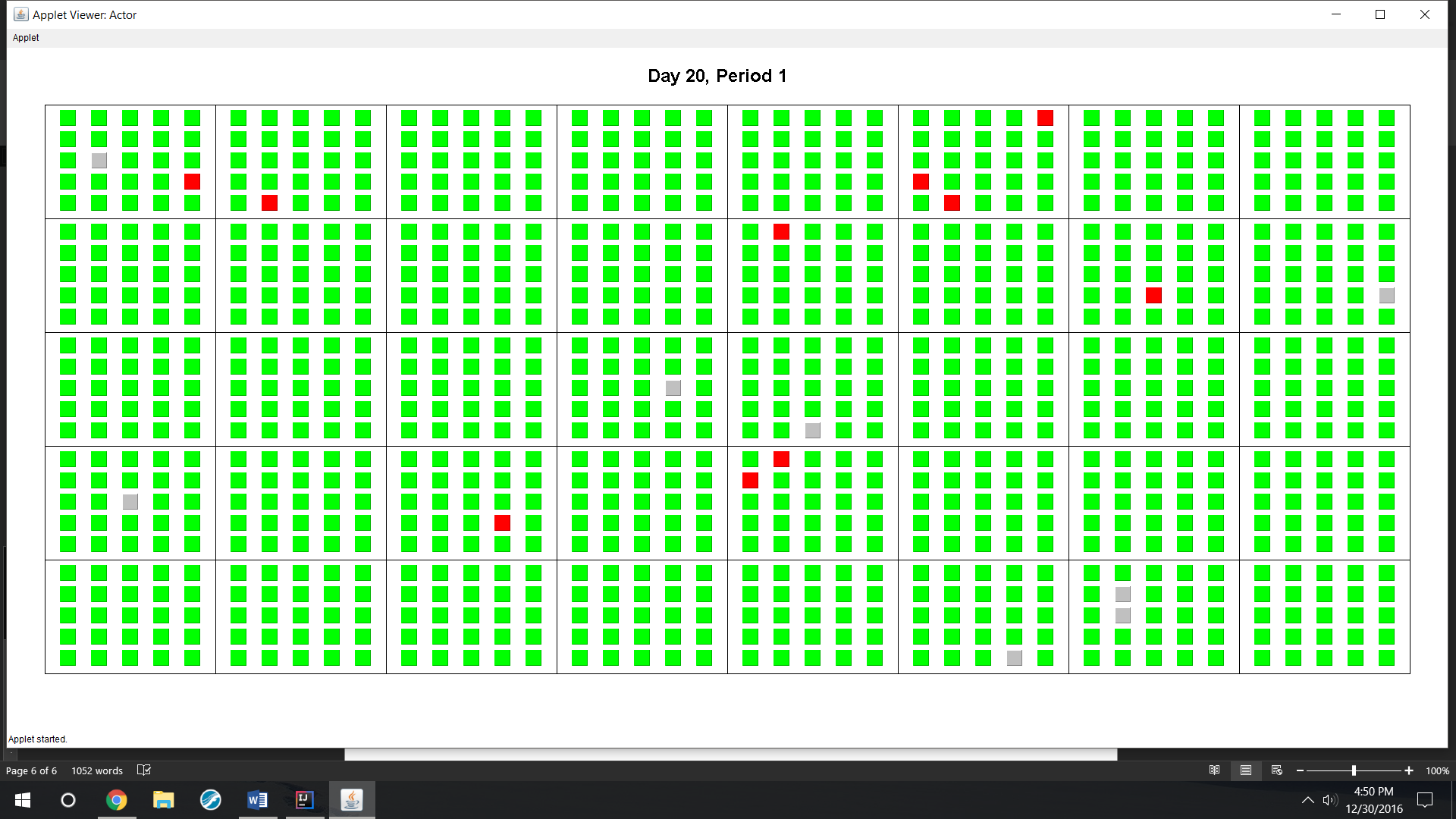
Day 5, Period 1: After Day 4, the initially infected student (red) enters state 2, increasing the likelihood of him infecting students in classes on his schedule. Two students on the left column (light gray) also enter state 2 and decide to abstain from attending school until entering state 3 and will not infect others.



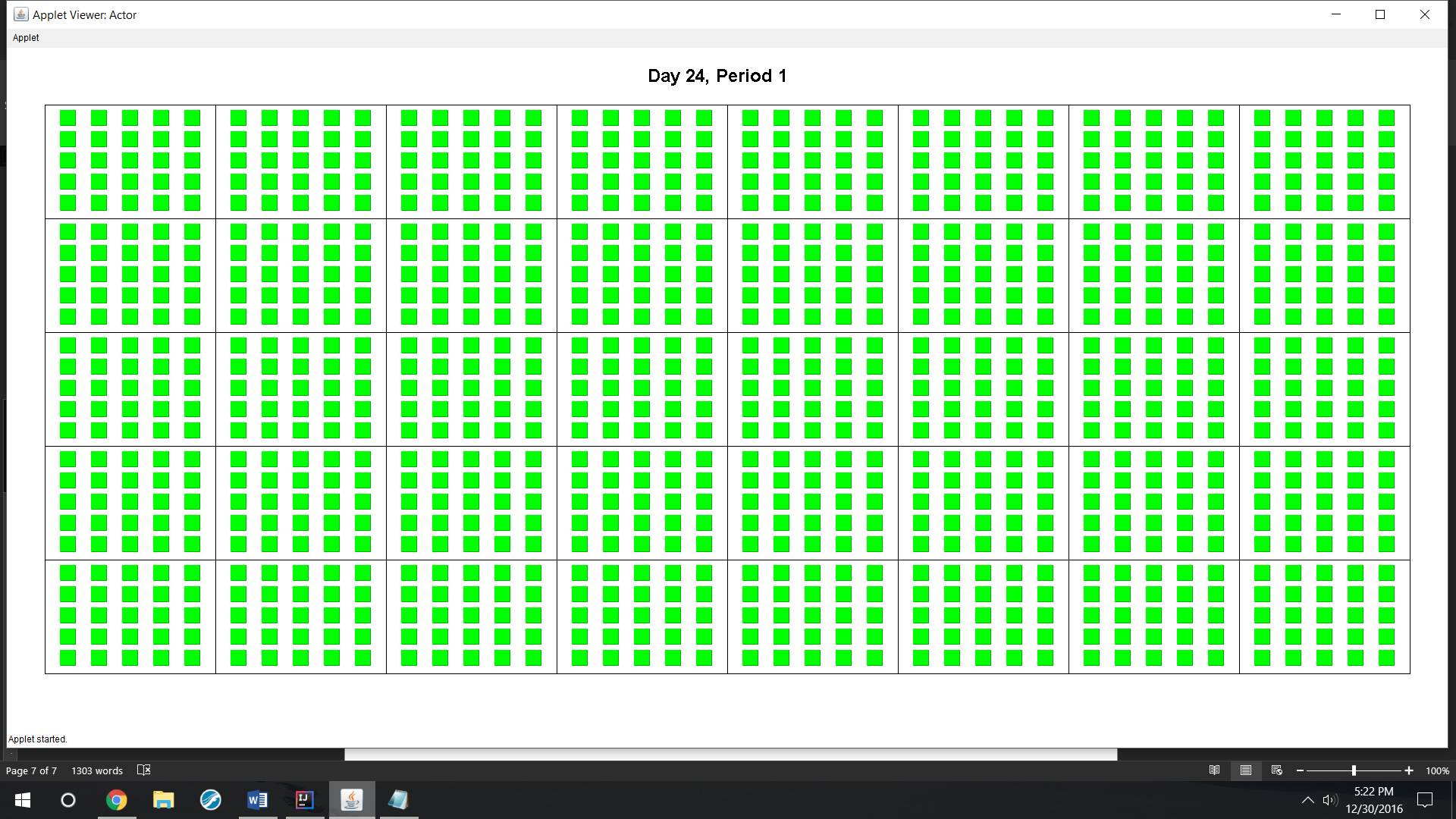
Day 10, Period 1: Most students have been exposed, students in green are healed (state 3) and can no longer catch nor spread the disease.



Day 20, Period 1: The overwhelming majority of students are healed at this point. All spreading has ceased at this point, only healed and infected students remain.



Day 24, Period 1: All students are healed. The algorithm terminates.



# Data across many iterations

Table 1 illustrates the relationship between proportion of students absent when in state 2 and total number of students infected across several trials.

Table 1: Number of Students Evading Common Cold

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Trial 1 | Trial 2 | Trial 3 | Trial 4 | Trial 5 | Average |
| 0 | 6 | 6 | 7 | 2 | 5 | 5.2 |
| 0.1 | 7 | 5 | 4 | 6 | 3 | 5 |
| 0.2 | 7 | 10 | 10 | 10 | 8 | 9 |
| 0.3 | 19 | 18 | 25 | 9 | 20 | 18.2 |
| 0.4 | 27 | 29 | 27 | 32 | 27 | 28.4 |
| 0.5 | 39 | 43 | 46 | 46 | 42 | 43.2 |
| 0.6 | 63 | 79 | 101 | 73 | 97 | 82.6 |
| 0.7 | 109 | 137 | 127 | 157 | 118 | 129.6 |
| 0.8 | 278 | 258 | 214 | 204 | 232 | 237.2 |
| 0.9 | 414 | 575 | 503 | 433 | 437 | 472.4 |

Table 1 and Figure 1 show the simulation with the given parameters may be estimated exponentially by the function , where is the number of students infected over the course of a simulation, with no bounds on time.

To achieve discernable results in low values of , it was necessary to significantly lower the probability of spreading disease. The previous data originates from and . The side effect of this change was significant bipolar results, in that several trials were eliminated due to few or in some cases one student catching the disease. Including these occurrences in the above analysis would skew the conclusion to depend almost entirely on when these A tally of the count of these occurrences shown in Table 1.

Table 2: Unpropagated Disease

|  |  |
| --- | --- |
|  | Trials eliminated with fewer than five students infected |
| 0 | 0 |
| 0.1 | 0 |
| 0.2 | 1 |
| 0.3 | 2 |
| 0.4 | 2 |
| 0.5 | 6 |
| 0.6 | 1 |
| 0.7 | 1 |
| 0.8 | 8 |
| 0.9 | 18 |

The above results indicate a positive relationship between the likelihood of the disease not propagating and the proportion of students absent when sick. This appears reasonable due to the higher chance the initial student has of remaining out of school when in state 2 with increasing values of . The rate of increase, however, appears to be inconsistent and may not be entirely visible without an exceedingly large number of trials.

During the previous experiment, it became evident that the number of simulated days required to reach the terminal state of the algorithm was significantly higher than in the previous illustrated simulation. Direct measurement is required to discern whether this is due to the lower probability of infecting others, that is, lower and , or the change in

Table 3: Days to Reach Terminal State vs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trial 1 | Trial 2 | Trial 3 | Average |
| 0 | 39 | 39 | 36 | 38 |
| 0.2 | 37 | 44 | 41 | 40.66667 |
| 0.4 | 54 | 44 | 45 | 47.66667 |
| 0.6 | 62 | 62 | 56 | 60 |
| 0.8 | 73 | 62 | 68 | 67.66667 |

Table 4: Days to Reach Terminal State vs and ,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Trial 1 | Trial 2 | Trial 3 | Average |
| 0.005 | 0.05 | 35 | 34 | 36 | 35 |
| 0.01 | 0.1 | 28 | 25 | 24 | 25.66667 |
| 0.015 | 0.15 | 21 | 21 | 19 | 20.33333 |
| 0.02 | 0.2 | 20 | 19 | 20 | 19.66667 |
| 0.025 | 0.25 | 18 | 18 | 18 | 18 |

In the above simulations, all variables were kept the same as in the Common Cold experiment, except in the experiment in Table 4 and Figure 3, where it was kept constant at .

Table 3 and Figure 2 show that a linear increase in causes the longevity of the disease to increase at an increasing rate, although it cannot be determined whether this model is exponential or polynomial without an exceedingly large number of trials. If is the longevity of the disease, the better correlation between those models is , from .

Table 4 and Figure 3 shows an inverse relationship between the probability of spreading a disease and its longevity. The relationship is best modeled by the function

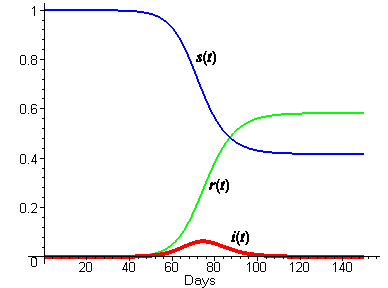
.

# Discussion

The above results in Figure 1 illustrate a significant reduction in probability of contracting a disease when peers who are sick do not attend classes. As the proportion of absent students increases, the number of students avoiding the disease increases, though the longevity of the disease also increases. Additionally, strongly contagious diseases such as influenza and streptococcus last longer in the school than weaker viruses such as the common cold, though it is unknown what effect the random chances of mutation would have on lengthening the time the virus lasts.

Once this algorithm infects more than 5-10 people, it is nearly impossible to prevent a massive propagation that infects half to nearly all the members of the school. A possible further experiment would be to determine the exact tipping point number of infected students where propagation may no longer be prevented, and whether any of the parameters for the algorithm shift this tipping point.

This algorithm originates from David Smith and Lang Moore’s differential equations for the SIR model published for the MAA in 2004, which employs three functions of time. measures the number of susceptible (unaffected) individuals, measures the number of infected individuals, and measures the number of recovered individuals. Smith and Moore published a graph of these functions for an indefinite population to be the following:



Smith, David, and Lang Moore. SIR Model. Digital image. MAA Press. Mathematical Association of America, Dec. 2004. Web. 2 Jan. 2017.

A useful further investigation would be to experimentally compare this model the output of this algorithm over many trials, and analyze the deviation from these normal and logistic curves with different sets of parameters. It would also be useful to calculate the expected values of these three quantities for a given number of days for the algorithm, and compare them to the experimental results.

# The Economics of Disease

The incentives to attending school despite illness may be modeled by treating illness of others students as a negative externality.

Figure 4

D = MB

S = MPC

MSC

Students Attending School Ill

Cost of Attending School Ill

Qs Qp

In the same manner as a negative externality in a product results in its overproduction to reach social allocative efficiency, factoring in the spread of disease to others into the cost of attending school reveals that too many students are attending school. However, according to the graph above, reducing the marginal benefit (demand) for attending school (created by incentives like perfect attendance awards and the like) would maintain the same disparity between social and private optimal attendance quantities. The improvement, therefore, lies in the shape of the Marginal Social Cost curve. The MSC curve in Figure 4 above ignores the bipolar nature of disease revealed in the simulation, that shows that linearly increasing numbers of students attending school ill will cause the Marginal Social Cost to increase at an increasing rate. Figures 5 and 6 show this.

MSC

D = MB

S = MPC

Students Attending School Ill

Cost of Attending School Ill

Qs Qp1

Figure 5: Propagation of Social Cost

In this case, a reduction in incentives would shift the Marginal Benefit curve to the left, reducing the distance between total social cost and total private cost, given by

MSC

D1

S = MPC

Students Attending School Ill

Cost of Attending School Ill

D2

Qs Qp2

Figure 6: Shift of Marginal Benefit

# Conclusion

The simulation shows that the number of people in a school infected with a disease and the length of time the disease remains in the school increase at an increasing rate with respect to the proportion of students who elect to miss class when at peak level of illness. Additionally, it was shown that the length of time the disease lingered in a school decreases in an inverse relationship with respect to the probability of spreading the disease to others.

There are multiple ways the algorithm may be improved to better fit the actual spread of disease. First, it would make sense to decrease the probability of a state 2 individual infecting others the longer he/she is in state 2, to more closely align to the trend that sick individuals are most contagious the first few days of displaying symptoms (Humphries 1). It would be difficult, however, without biological experimentation, to determine the model of decrease, that is, whether contagiousness decreases linearly, exponentially, logarithmically, or following a power model. Making this change may slightly alleviate the bipolar results of the algorithm; that is, iterations where close to half of the population contracts the disease are more likely, as opposed to the current model, which, except at very small values of and , results in either only one to five students contracting the virus, or all but a small percentage falling ill at some point.

Second, it may make sense to set a minimum value of chance of catching a virus from an infected student, since an infected student’s chance of spreading the virus to a student five or more seats away by coughing or similar is almost zero, with the real danger occurring from touching common classroom objects, such as staplers and pencil sharpeners, which carries a risk of infection independent of the seating arrangement of the classroom. This change may cause the disease to propagate more quickly in the simulation, resulting in more infected students, but a shorter time of risk of falling ill.

# Works Cited

Humphries, Courtney. "When and for How Long Is a Cold Contagious?" BostonGlobe.com. The

Boston Globe, 16 Feb. 2014. Web. 16 Dec. 2016.

Landau, Elizabeth. "The Truth about Colds." CNN. Cable News Network, 16 Feb. 2011. Web. 09 Dec.

2016.

Smith, David, and Lang Moore. "The SIR Model for Spread of Disease - The Differential Equation

Model." MAA Press. Mathematical Association of America, Dec. 2004. Web. 22 Dec. 2016.

# Appendix

The program written for this algorithm is shown below. All code was written in the Java Programming Language created by Sun Microsystems in 1995, now operated by Oracle, Inc.

## Inputs.txt

Welcome to School Disease Modeler! Please specify inputs below.

Chance of spreading disease to direct neighbor before contraction: .025

Chance of spreading disease to direct neighbor at contraction: .25

Average time to display symptoms in days: 3.5

Range of times to display symptoms in days: 6

Average length of disease in days: 6.5

Range of lengths of disease in days: 4

Number of classrooms (up to 50): 40

Number of class periods per day (1-10): 7

Number of rows of desks in a classroom (2-10): 5

Number of columns of desks in a classroom (2-10): 5

Proportion of students who stay home when at peak (decimal between 0 and 1): 0.5

Note that the number of desks per classroom may not exceed 40.

## Actor.java

**import** java.applet.Applet;  
**import** java.awt.\*;  
**import** java.awt.event.KeyEvent;  
**import** java.awt.event.KeyListener;  
**import** java.awt.event.MouseEvent;  
**import** java.awt.event.MouseListener;  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Created by 11ryt on 12/12/2016.  
 \*/***public class** Actor **extends** Applet {  
 **public static final int *matrixVerticalPadding*** = 75;  
 **public static final int *matrixHorizontalPadding*** = 50;  
 **public static double** *CHANCE\_CATCH\_PREMATURE*;  
 **public static double** *CHANCE\_CATCH*;  
 **public static double** *meanState*[];  
 **public static int** *rangeState*[];  
 **protected static int** *num\_Rooms*;  
 **protected static int** *num\_Students*;  
 **protected static int** *num\_Periods*;  
 **protected static int** *rows*;  
 **protected static int** *cols*;  
 **static double** *porportionAbsent*;  
 **static** Scanner *s*;  
 **static** School *inst*;  
 **private boolean initial** = **true**;  
 **protected static int** *dayNo* = 1;  
 **protected static int** *pdNo* = 1;  
  
 **public void** initialSetup() {  
 **try** {  
 *s* = **new** Scanner(**new** File(**"inputs.txt"**));  
 } **catch** (FileNotFoundException e) {  
 System.***out***.println(**"Cannot find file. Please try again."**);  
 System.*exit*(0);  
 }  
 *s*.nextLine();  
 *s*.nextLine();  
 *meanState* = **new double**[2];  
 *rangeState* = **new int**[2];  
 *CHANCE\_CATCH\_PREMATURE* = Double.*parseDouble*(*s*.nextLine().split(**":"**)[1].trim()); *//usurps the number following the prompt colon.  
 CHANCE\_CATCH* = Double.*parseDouble*(*s*.nextLine().split(**":"**)[1].trim());  
 *meanState*[0] = Double.*parseDouble*(*s*.nextLine().split(**":"**)[1].trim());  
 *rangeState*[0] = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *meanState*[1] = Double.*parseDouble*(*s*.nextLine().split(**":"**)[1].trim());  
 *rangeState*[1] = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *num\_Rooms* = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *num\_Periods* = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *rows* = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *cols* = Integer.*parseInt*(*s*.nextLine().split(**":"**)[1].trim());  
 *porportionAbsent* = Double.*parseDouble*(*s*.nextLine().split(**":"**)[1].trim());  
 **if** (*rows* \* *cols* > 40) {  
 *rows* = 5;  
 *cols* = 8;  
 }  
 *num\_Students* = *rows* \* *cols* \* *num\_Rooms*;  
 Student.*chanceAdvanceState12* = **new double**[*rangeState*[0] + 1];  
 Student.*chanceAdvanceState23* = **new double**[*rangeState*[1] + 1];  
 **double** sd12 = getStDev(*rangeState*[0]);  
 **for** (**int** i = 0; i < Student.*chanceAdvanceState12*.**length**; i++) {  
 Student.*chanceAdvanceState12*[i] = normalArea(*meanState*[0], sd12, i, i + 1);  
 Student.*chanceAdvanceState12*[0] = .05;  
 Student.*chanceAdvanceState12*[Student.*chanceAdvanceState12*.**length** - 1] = .05;  
 }  
 */\* At this point, each value stored in the probability array is the initial chance of the day the student gets sick on a particular day.  
 The next loop changes the values to be the chance of getting sick on a particular day given he/she did not get sick on a previous day. \*/* **for** (**int** i = 0; i < Student.*chanceAdvanceState12*.**length**; i++) {  
 **double** sum = 0;  
 **for** (**int** j = i; j < Student.*chanceAdvanceState12*.**length**; j++) {  
 sum += Student.*chanceAdvanceState12*[j];  
 }  
 Student.*chanceAdvanceState12*[i] = Student.*chanceAdvanceState12*[i] / sum;  
 }  
 **double** sd23 = getStDev(*rangeState*[1]);  
 **for** (**int** i = 0; i < Student.*chanceAdvanceState23*.**length**; i++) {  
 Student.*chanceAdvanceState23*[i] = normalArea(*meanState*[1], sd23, i, i + 1);  
 Student.*chanceAdvanceState23*[0] = .025;  
 }  
 *//same status as above comment, except for the 2 to 3 state change probabilities.* **for** (**int** i = 0; i < Student.*chanceAdvanceState23*.**length**; i++) {  
 **double** sum = 0;  
 **for** (**int** j = i; j < Student.*chanceAdvanceState23*.**length**; j++) {  
 sum += Student.*chanceAdvanceState23*[j];  
 }  
 Student.*chanceAdvanceState23*[i] = Student.*chanceAdvanceState23*[i] / sum;  
 }  
 *inst* = **new** School();  
 *inst*.**masterStudentList**.get(0).catchDisease();  
 **initial** = **false**;  
 **this**.addKeyListener(**new** KeyListener() {  
 @Override  
 **public void** keyTyped(KeyEvent e) {  
 advancePeriod();  
 }  
  
 @Override  
 **public void** keyPressed(KeyEvent e) {  
  
 }  
  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
  
 }  
 });  
 **this**.addMouseListener(**new** MouseListener() {  
 @Override  
 **public void** mouseClicked(MouseEvent e) {  
 advancePeriod();  
 }  
  
 @Override  
 **public void** mousePressed(MouseEvent e) {  
  
 }  
  
 @Override  
 **public void** mouseReleased(MouseEvent e) {  
  
 }  
  
 @Override  
 **public void** mouseEntered(MouseEvent e) {  
  
 }  
  
 @Override  
 **public void** mouseExited(MouseEvent e) {  
  
 }  
 });  
 }  
  
 **public void** paint(Graphics g) {  
 **if** (**initial**) {  
 initialSetup();  
 }  
 **this**.setSize(1900, 900);  
 g.setFont(**new** Font(**"Arial"**, Font.***BOLD***, 24));  
 g.drawString(**"Day "** + *dayNo* + **", Period "** + *pdNo*, **this**.getBounds().**width** / 2 - (**this**.getBounds().**width** / 18), **this**.getBounds().**height** / 20);  
 paintbordersAndStudents(g);  
 }  
  
 **private void** paintbordersAndStudents(Graphics g) {  
 **int** numRoomsPerRow = 2 \* (**int**) Math.*sqrt*(*num\_Rooms* / 2); *//sets matrix length to at most twice its height* **int** roomLength = (**this**.getBounds().**width** - 2 \* ***matrixHorizontalPadding***) / numRoomsPerRow;  
 **int** numRoomsPerCol = (*num\_Rooms* % numRoomsPerRow == 0) ? *num\_Rooms* / numRoomsPerRow : *num\_Rooms* / numRoomsPerRow + 1; *//accounts for possible incomplete row at bottom of matrix* **int** roomHeight = (**this**.getBounds().**height** - 2 \* ***matrixVerticalPadding***) / numRoomsPerCol;  
 **for** (**int** i = 0; i < numRoomsPerCol; i++) {  
 **for** (**int** j = 0; j < numRoomsPerRow; j++) {  
 **if** (i \* numRoomsPerRow + j < *num\_Rooms*)  
 g.drawRect(***matrixHorizontalPadding*** + j \* roomLength, ***matrixVerticalPadding*** + i \* roomHeight, roomLength, roomHeight);  
 }  
 }  
 **int** studentSquareSide = (**int**) Math.*min*(.7 \* roomLength / *cols*, .7 \* roomHeight / *rows*);  
 **int** rowPadding = (roomHeight - *rows* \* studentSquareSide) / (*rows* + 1);  
 **int** columnPadding = (roomLength - *cols* \* studentSquareSide) / (*cols* + 1);  
 **for** (Student s : *inst*.**masterStudentList**) {  
 **int** roomIndex = *inst*.**assignmentOrder**.get(*pdNo* - 1).indexOf(s) / (Actor.*rows* \* Actor.*cols*); *//index in list of rooms of class student has at this time.* **int** seatIndex = *inst*.**assignmentOrder**.get(*pdNo* - 1).indexOf(s) % (Actor.*rows* \* Actor.*cols*); *//seat index of student in class.* **int** classOrigin[] = {roomLength \* (roomIndex % numRoomsPerRow) + ***matrixHorizontalPadding***, roomHeight \* (roomIndex / numRoomsPerRow) + ***matrixVerticalPadding***}; *//coordinate of top-left corner of specified class* **int** horizontalOffset = (seatIndex % *cols*) \* studentSquareSide + (seatIndex % *cols* + 1) \* columnPadding;  
 **int** verticalOffset = (seatIndex / *cols*) \* studentSquareSide + (seatIndex / *cols* + 1) \* rowPadding;  
 **if** (s.isAbsent())  
 g.setColor(Color.***LIGHT\_GRAY***);  
 **else** {  
 **switch** (s.getState()) {  
 **case** 0:  
 g.setColor(Color.***GRAY***);  
 **break**;  
 **case** 1:  
 g.setColor(Color.***YELLOW***);  
 **break**;  
 **case** 2:  
 g.setColor(Color.***RED***);  
 **break**;  
 **case** 3:  
 g.setColor(Color.***GREEN***);  
  
 }  
 }  
 g.fill3DRect(classOrigin[0] + horizontalOffset, classOrigin[1] + verticalOffset, studentSquareSide, studentSquareSide, **true**);  
 }  
 **int** sum = 0;  
 **for**(Student s: *inst*.**masterStudentList**){  
 **if** (s.getState() == 0){  
 sum++;  
 }  
 }  
 g.setColor(Color.***BLACK***);  
 g.drawString(**"Unaffected Students: "** + sum, **this**.getBounds().**width**/2 -**this**.getBounds().**width**/9, ***matrixVerticalPadding*** + numRoomsPerCol\*roomHeight + 30);  
 }  
  
 **void** advancePeriod() {  
 **if** (*pdNo* == *num\_Periods*) {  
 *pdNo* = 1;  
 **for** (Class c : *inst*.**roomList**) {  
 c.runClass(*pdNo*);  
 }  
 *dayNo*++;  
 **for** (Student s : *inst*.**masterStudentList**) {  
 s.advanceDay();  
 }  
 } **else** {  
 *pdNo*++;  
 **for** (Class c : *inst*.**roomList**) {  
 c.runClass(*pdNo*);  
 }  
 }  
 repaint();  
 }  
  
 **double** normalArea(**double** mean, **double** stDev, **double** lowerBound, **double** upperBound) { *//performs a midpoint Riemann Sum with 10000 subdivisions* **int** subDiv = 10000;  
 **double** width = (upperBound - lowerBound) / subDiv;  
 **double** totalArea = 0.0;  
 **for** (**double** i = lowerBound; i < upperBound; i += width) {  
 totalArea += (1 / (stDev \* Math.*sqrt*(2 \* Math.***PI***))) \* Math.*pow*(Math.***E***, -0.5 \* (Math.*pow*((i + (width \* .5) - mean) / stDev, 2))) \* width;  
 }  
 **return** totalArea;  
 }  
 **double** getStDev(**int** range) { *//95% of normal distribution area is within two standard deviations of the mean, so the standard deviation is the range of 95% of the data divided by four.* **return** range / 4.0;  
 }  
}

## Student.java

*/\*\*  
 \* Created by 11ryt on 12/12/2016.  
 \*/***public class** Student {  
 **private int id**;  
 **private** Class[] **classList**;  
 **private byte state**; *//0 for unaffected, 1 for caught but not contracted, 2 for sick, 3 for healed* **private byte delayShift**; *//counts the number of days in particular state* **private boolean absent**;  
 **public static double**[] *chanceAdvanceState12*;  
 **public static double**[] *chanceAdvanceState23*;  
 **public boolean**[] **used**;  
  
 **public** Student(**int** id) {  
 **classList** = **new** Class[Actor.*num\_Periods*];  
 **used** = **new boolean**[Actor.*num\_Periods*];  
 **this**.setId(id);  
 **state** = 0;  
 **absent** = **false**;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** Class[] getClassList() {  
 **return classList**;  
 }  
  
 **public void** setClassList(Class[] classList) {  
 **this**.**classList** = classList;  
 }  
  
 **public byte** getState() {  
 **return state**;  
 }  
  
 **public void** setState(**byte** state) {  
 **this**.**state** = state;  
 }  
  
 **public boolean** isAbsent() {  
 **return absent**;  
 }  
  
 **void** catchDisease() {  
 **if** (**state** == 0) {  
 **state** = 1;  
 **delayShift** = 0;  
 }  
 }  
  
 **void** advanceDay() {  
 **switch** (**state**) {  
 **case** 0:  
 **break**;  
 **case** 1:  
 **if** (Math.*random*() < *chanceAdvanceState12*[**delayShift**]) {  
 **delayShift** = 0;  
 **state**++;  
 **if** (Math.*random*() < Actor.*porportionAbsent*) {  
 **absent** = **true**;  
 }  
 } **else** {  
 **delayShift**++;  
 }  
 **break**;  
 **case** 2: {  
 **if** (Math.*random*() < *chanceAdvanceState23*[**delayShift**]) {  
 **delayShift** = 0;  
 **state**++;  
 **absent** = **false**;  
 } **else** {  
 **delayShift**++;  
 }  
 **break**;  
 }  
 }  
 }  
  
}

## Class.java

**import** java.util.ArrayList;

*// Created by 11ryt on 12/12/2016.***public class** Class {  
  
 **protected** ArrayList<ArrayList<Student>> **studentList**;  
  
 Class() {  
 **studentList** = **new** ArrayList();  
 **for** (**int** i = 0; i < Actor.*num\_Periods*; i++) {  
 **studentList**.add(**new** ArrayList<Student>());  
 }  
 }  
  
 **void** runClass(**int** pdNo) {  
 **int** countStudents = 0;  
 Student[][] studentMatrix = **new** Student[Actor.*rows*][Actor.*cols*];  
 **for** (**int** i = 0; i < Actor.*rows*; i++) {  
 **for** (**int** j = 0; j < Actor.*cols*; j++) {  
 studentMatrix[i][j] = **studentList**.get(pdNo - 1).get(countStudents);  
 countStudents++;  
 }  
 }  
 **for** (**int** i = 0; i < studentMatrix.**length**; i++) {  
 **for** (**int** j = 0; j < studentMatrix[i].**length**; j++) {  
 **if** (!studentMatrix[i][j].isAbsent()) {  
 **switch** (studentMatrix[i][j].getState()) {  
 **case** 0:  
 **break**;  
 **case** 1:  
 **case** 2: {  
 **for** (**int** k = 0; k < studentMatrix.**length**; k++) {  
 **for** (**int** l = 0; l < studentMatrix[k].**length**; l++) {  
 **if** (i == k && j == l) { *//skips a student possibly infecting him/herself* **continue**;  
 }  
 **double** distance = Math.*sqrt*(Math.*pow*(Math.*abs*((**double**) k - (**double**) i), 2) + Math.*pow*(Math.*abs*((**double**) l - (**double**) j), 2));  
 **double** probabilityExposure = (studentMatrix[i][j].getState() == 1) ? Actor.*CHANCE\_CATCH\_PREMATURE* : Actor.*CHANCE\_CATCH*; *//direct chance ignoring distance* probabilityExposure = probabilityExposure \* Math.*pow*(0.4, distance - 1);  
 **if** (Math.*random*() < probabilityExposure) {  
 studentMatrix[k][l].catchDisease();  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
}

## School.java

**import** java.util.ArrayList;  
**import** java.util.Collections;  
  
*/\*\*  
 \* Created by 11ryt on 12/12/2016.  
 \*/***public class** School {  
 ArrayList<Student> **masterStudentList**;  
 ArrayList<Class> **roomList**;  
 ArrayList<ArrayList<Student>> **assignmentOrder**;  
  
 School() {  
 **masterStudentList** = **new** ArrayList<>();  
 **for** (**int** i = 0; i < Actor.*num\_Students*; i++) {  
 **masterStudentList**.add(**new** Student(i + 1));  
 }  
 **roomList** = **new** ArrayList();  
 **for** (**int** i = 0; i < Actor.*num\_Rooms*; i++) {  
 **roomList**.add(**new** Class());  
 }  
 **assignmentOrder** = **new** ArrayList();  
 **for** (**int** per = 0; per < Actor.*num\_Periods*; per++) {  
 **assignmentOrder**.add((ArrayList) **masterStudentList**.clone());  
 Collections.*shuffle*(**assignmentOrder**.get(per));  
 }  
 **for** (**int** roomNo = 0; roomNo < Actor.*num\_Rooms*; roomNo++) {  
 Class c = **roomList**.get(roomNo);  
 **int** studentOffset = roomNo \* Actor.*rows* \* Actor.*cols*;  
 **for** (**int** period = 0; period < Actor.*num\_Periods*; period++) {  
 **for** (**int** seat = 0; seat < Actor.*rows* \* Actor.*cols*; seat++) {  
 c.**studentList**.get(period).add(**assignmentOrder**.get(period).get(studentOffset + seat));  
 }  
 }  
 }  
 }  
}