Simulation of Disease Spread

New Mexico Adventures in Supercomputing Challenge

Final Report

April 7, 2004

Team #033

Goddard High School

Team Members:

Jason Archer

Jerrick Morsey

Russell Tabor

Eric Willhelm

Teacher:

Mr. Greg Anderson

Table of Contents

Executive Summary 3

Introduction 4

Project Description 5

- **Scope** 5
- **Methods** 5

Results 7

Conclusions 8

Recommendations 9

References 10

Appendix A - Graphical Data 11

Appendix B - Code 12

- **DiseaseSim6b.java** 12
- **Attractor.java** 20
- **Group.java** 21
- **Locator.java** 25
- **Person.java** 26
- **Rng.java** 32

Executive Summary

In response to the recent threat of diseases such as Severe Acute Respiratory Syndrome (SARS), we have created a program that replicates the transmission of a virulent SARS-like disease among the members of a community that has been warned of the presence of such a disease. The purpose of this program is to simulate the effects of various disease control measures on the general population's susceptibility to the disease.

The program allows the variation of several parameters to determine the effect of such variation on the spread of the disease. It records the percentage of people sick for each run of the simulation, thus allowing us to compare percentages from different runs. We can therefore analyze the percentage data to determine what kind of actions should be taken in the event of a new worldwide epidemic.

In a series of 20 runs of the simulation, we determined an effective method of considerably reducing the percentage of people sick. We discovered that certain types of groupings of people that arise within normal communities must be shut down while the disease remains a threat. Without shutting down certain types of groups, the percentage of people sick increases by a factor of roughly six.

Introduction

Our project is the simulation of the spread of a disease within a community that has been warned of the disease. In recent years, the spread of infectious diseases has become a major fear due to new diseases, such as the SARS epidemic. Therefore, the intent of our simulation is to provide information regarding how to deal with such diseases by dealing with them experimentally.

Since many diseases that people fear spread through the air, we chose to model droplet transmission with our simulation. Transmission by droplet contact involves droplets from an infected person's cough or sneeze contacting surfaces of another person's nose or mouth; these droplets quickly settle out of the air. These droplets are generally greater than 5 microns in size and usually move up to 3 feet before settling. [1] This method of transmission is the method that is used by the pathogen that causes SARS. [5]

Our program uses stochastic modeling to simulate droplet transmission of a disease among a population of individuals within a town. The program moves people within the community and organizes them into various groups while the disease progresses. It allows the modification of several parameters to quantify those parameters' effect on the rate of disease spread. Some of these parameters are population size, number of groups, size of groups, and length of contagiousness.

Project Description

Scope

The program was initially intended to determine the point at which an airborne disease would reach epidemic levels. However, the definition of "epidemic" used by the Centers for Disease Control and Prevention (an unusual spike in the number of cases of a given disease in a given region) [6] told us that we were already modeling an epidemic, regardless of its behavior. Therefore, the focus of the project shifted to the determination of methods to help control such an epidemic on a municipal scale.

Another problem was the nature of "airborne transmission"; a disease that uses airborne transmission can be carried in the air for long distances. [2] However, SARS, the disease whose dangers inspired this program, does not use airborne transmission. It uses "droplet-contact transmission", in which the disease is carried in droplets briefly suspended in the air from an infected person's cough or sneeze. [4] Our simulation has therefore been created to determine how to control a SARS-like epidemic of which a community has been warned by altering various parameters within our simulation to determine those parameters' effect on the percentage of infected people within a community.

Methods

The simulation was designed to represent a community. As in a real community, numerous people move about randomly. Although in reality people move in ordered and purposeful ways, the combined movements of all people can seem random when taken as a whole.

People do not move with complete randomness, however. In reality, people congregate in various ways. We assume that people congregate in two basic ways: the "boxed group" and the "attractor". The boxed group involves a group of people that periodically meet within a given area separate from the general public. This would be similar to a school, church, stadium, or theater,

since sets of people generally come and go in unison at specific times. The attractor involves a set location where individual people come and go, but there is generally a group of people open to the public. The attractor would be similar to a store, airport, or office, where there are constantly people but not the same set of people.

Within the simulation, for each type of group, a specific number of people from the population are assigned to populate each group. Each boxed group has the same number of people, and each attractor has the same number of people. This allows us to see whether group size has any effect on the spread of the disease. Although in reality similar groups within a single community can be of many different sizes, this size assignment scheme was incorporated due to simplicity and time constraints.

For the simulation to be relevant, people needed to be able to get sick, not just move realistically. Regardless of the initial conditions, one person is always randomly determined to be the first person infected. This person then spreads the disease by coming within a certain radius of another person; that other person automatically becomes infected. Although this is not entirely accurate since briefly coming in close proximity to a sick person once is not a guaranteed way to get sick, this inaccuracy could not be fixed due to time constraints.

The simulation requires the user to define several parameters before beginning the simulation. These parameters are population, number of attractors, number of people to be assigned to each attractor, number of boxed groups, number of people to be assigned to each boxed group, maximum contamination distance of each infected person, and length of time that the disease takes before the infected person becomes well again.

The result of the simulation is a percentage of sick people in relation to the total number of people. This data is recorded for each run of the simulation in order to compare multiple sets of data to reach conclusions regarding how to control disease spread.

Results

We ran control runs with the arbitrarily established initial conditions of 500 for population, 6 for number of attractors, 40 for number of people per attractor, 3 for number of boxed groups, 40 for number of people per boxed group, 60 for the length of illness, and 3 for the contamination distance. After we ran these control runs with the program, we established 4 sets of experimental runs: one with the number of attractors reduced to 0, one with the number of people per attractor lowered to 10, one with the number of boxed groups reduced to 0, and one with the number of people per boxed group lowered to 10. Each set of program runs required us to run the program 4 times, to verify our results. The unabridged data for these control runs is available in Appendix A.

Using the initial conditions of the simulation, the percentage of people sick reached roughly 80% throughout all the runs. When the number of attractors was reduced to 0, that percentage reached roughly 70%. By reducing the number of people per attractor to 10, roughly 70% of the people got sick. By lowering the number of boxed groups to 0, 13% of the people got sick. When the number of people per boxed group was lowered to 10, roughly 60% of the people caught the disease.

Conclusions

In light of the evidence presented by the simulation, the simplest way to control a future SARS-like epidemic (if it could be caught early enough with some advance warning from already infected communities) would be to shut down all businesses, churches, schools, and any other buildings where people gather periodically. Shutting down other types of locations such as stores, airports, or offices where people continuously gather would be less helpful, but not wasteful as a measure to combat disease spread. In short, total shutdown of any official meeting place of any kind is the best way to solve the problem of disease spread, but shutting down places of periodic gathering is most effective at reducing the percentage of people sick.

Simply encouraging people to stay home does not seem to be enough to make a major difference, since reducing the size of either type of group within the simulation only made comparatively minor reductions in the overall percentage of people sick. However, such a measure would provide a small level of safety for the population and could be considered if a forced shutdown of all of a community's grouping places was not possible for any reason.

Recommendations

In order to gain more insight into the problem of disease spread with our simulation, we could run more sets of simulations. There are also several modifications to the program code (shown in Appendix B) that could make the program more realistic.

Given more time, we could run more simulations. We could explore the behavior of diseases that require different amounts of time to remain contagious, and we could examine the disease's behavior among communities with populations of different sizes. We could also alter multiple parameters to determine better solutions to the problem of disease spread. We could have run such simulations, but the deadline did not allow us to do so.

We could also alter the simulation itself. A major problem that we could never solve was how to deal with differences in gender, age, and race with regard to susceptibility to disease; we simply treated all people the same within the simulation. Another problem was that we could never deal with "fomites", or inanimate objects, such as tables or chairs, that become briefly contaminated when infected droplets from a person's cough or sneeze land on them after they settle out of the air; they are capable of carrying a droplet-transmitted disease. [3] However, we were not able to deal with any of these problems due to time constraints.

References

[1] http://www.cdc.gov/ncidod/hip/isolat/isopart2.htm - Part II. Recommendations for Isolation Precautions in Hospitals

[2] http://www.drgreene.com/21_1022.html - Airborne Transmission

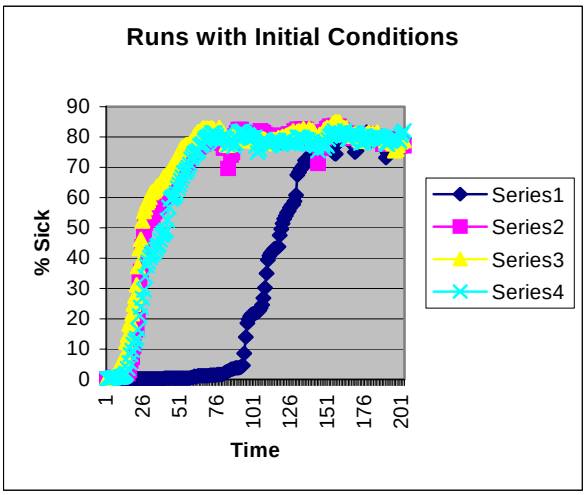
[3] http://www.drgreene.com/21_1073.html - Droplet Transmission

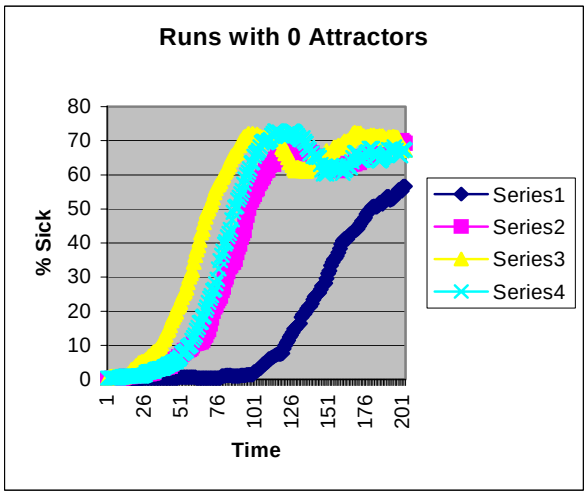
[4] http://microbiology.mtsinai.on.ca/faq/transmission.shtml - Methods of Disease Transmission

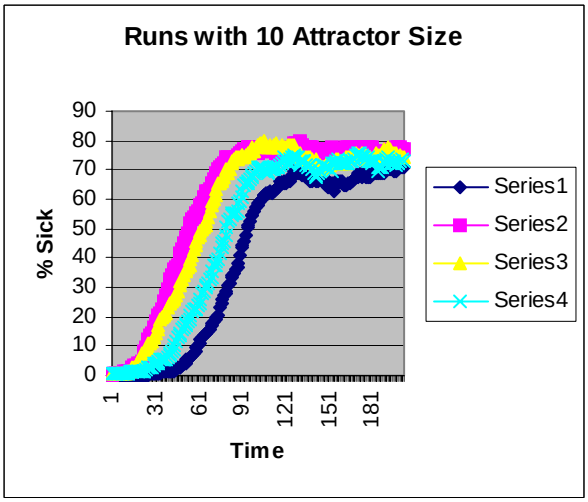
[5] http://www.salem-cumberlandhealth.org/Cumberland/FAQ%20Sheets/SARS%20%28Public%20Fact%20Sheet%29.pdf - What is SARS (Severe Acute Respiratory Syndrome)

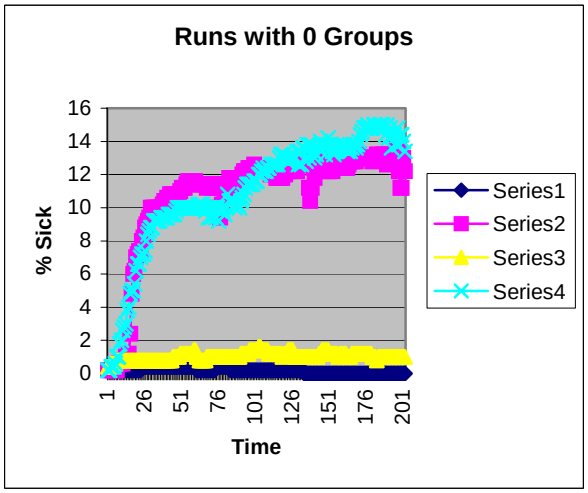
[6] http://slate.msn.com/id/2092969/ - Outbreaks vs. Epidemics – Whether it's time to freak about the flu.

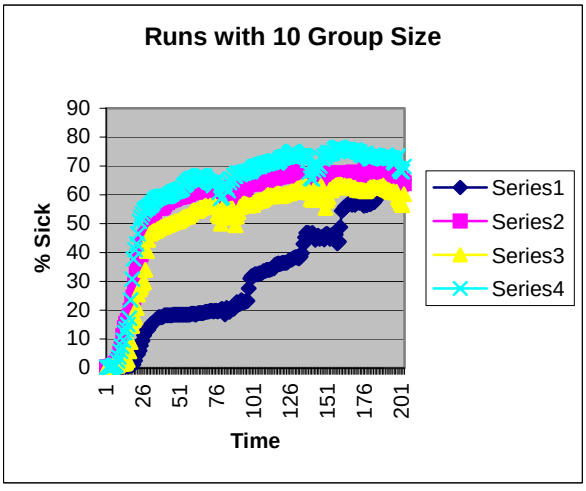
Appendix A – Graphical Data











Appendix B – Code

DiseaseSim6b.java

```
// Programmer:          AISC AM Programming class
// Jan. 22, 2004
// DiseaseSim2 driver to test out People class

  import javax.swing.*;
  import java.awt.*;
  import java.awt.event.*;
  import java.io.*;

   public class DiseaseSim6b extends JApplet implements ActionListener
   {
     JButton submit = new JButton("Submit");
     JButton cancel = new JButton("Cancel");
     JButton start = new JButton("Start Program");
     static int numPeople;
     JLabel peopleLabel = new JLabel("Amount of People: ");
     JTextField peopleField = new JTextField("500",10);
     static int numPerAttractor;
     JLabel perAttractorLabel = new JLabel("# People per Attractor: ");
     JTextField perAttractorField = new JTextField("40",10);
     static int numAttractors;
     JLabel attractorsLabel = new JLabel("# of Attractors: ");
     JTextField attractorsField = new JTextField("6",10);
     static int numPerGroup;
     JLabel perGroupLabel = new JLabel("# of People per Group: ");
     JTextField perGroupField = new JTextField("40",10);
     static int numGroups;
     JLabel groupsLabel = new JLabel("# of Groups: ");
     JTextField groupsField = new JTextField("3",10);
     static int numLengthOfIllness;
     JLabel lengthOfIllnessLabel = new JLabel("Length of Illness: ");
     JTextField lengthOfIllnessField = new JTextField("60",10);
     static int numContaminationDistance;
     JLabel contaminationDistanceLabel = new JLabel("Contamination Distance: ");
     JTextField contaminationDistanceField = new JTextField("3",10);


      public void init()
      {
        Container frame = getContentPane();
        frame = getContentPane();
        frame.setLayout(new FlowLayout(FlowLayout.CENTER));
        frame.add(peopleLabel);
```

```
  frame.add(peopleField);
  frame.add(attractorsLabel);
  frame.add(attractorsField);
  frame.add(perAttractorLabel);
  frame.add(perAttractorField);
  frame.add(groupsLabel);
  frame.add(groupsField);
  frame.add(perGroupLabel);
  frame.add(perGroupField);
  frame.add(lengthOfIllnessLabel);
  frame.add(lengthOfIllnessField);
  frame.add(contaminationDistanceLabel);
  frame.add(contaminationDistanceField);
  frame.add(submit);
  frame.add(cancel);
  frame.add(start);
  submit.addActionListener(this);
  cancel.addActionListener(this);
  start.addActionListener(this);
}

 public void actionPerformed(ActionEvent ae)
{
  if(ae.getSource() == submit)
  {
    String strPeople = peopleField.getText();
    numPeople = Integer.parseInt(strPeople);

    String strAttractors = attractorsField.getText();
    numAttractors = Integer.parseInt(strAttractors);

    String strPerAttractor = perAttractorField.getText();
    numPerAttractor = Integer.parseInt(strPerAttractor);

    String strGroups = groupsField.getText();
    numGroups = Integer.parseInt(strGroups);

    String strPerGroup = perGroupField.getText();
    numPerGroup = Integer.parseInt(strPerGroup);

    String strLengthOfIllness = lengthOfIllnessField.getText();
    numLengthOfIllness = Integer.parseInt(strLengthOfIllness);

    String strContaminationDistance = contaminationDistanceField.getText();
    numContaminationDistance = Integer.parseInt(strContaminationDistance);

    System.out.println("Submitted");
```

```
        if(numPeople < ((numAttractors * numPerAttractor) + (numGroups * numPerGroup)))
        {
          new ErrorFrame().setVisible(true);
        }

      }

      if(ae.getSource() == start)
      {
        new DiseaseSimFrame().setVisible(true);
        this.setVisible(false);
      }
      if(ae.getSource() == cancel)
      {
      //    this.dispose();
        System.out.println("Cancel Button Clicked");
      }

    }

     public void paint(Graphics g)
    {
      setBackground(Color.blue);
      start.repaint();
      peopleField.repaint();
      peopleLabel.repaint();
      groupsField.repaint();
      groupsLabel.repaint();
      perGroupField.repaint();
      perGroupLabel.repaint();
      attractorsField.repaint();
      attractorsLabel.repaint();
      perAttractorField.repaint();
      perAttractorLabel.repaint();
      lengthOfIllnessField.repaint();
      lengthOfIllnessLabel.repaint();
      contaminationDistanceField.repaint();
      contaminationDistanceLabel.repaint();
      cancel.repaint();
      submit.repaint();

    }

}

 class DiseaseSimFrame extends JFrame implements ActionListener
{
```

```
    int count = 0;
    int i,j,r;
    int numPeople= DiseaseSim6b.numPeople;
    int diameter= 5;
    int univTime=1;
    int numPeoplePerAttractor= DiseaseSim6b.numPerAttractor;
                int numAttractors= DiseaseSim6b.numAttractors;
    int numPeoplePerGroup= DiseaseSim6b.numPerGroup;
                int numGroups=DiseaseSim6b.numGroups;
    int mx,my;
    int lengthOfIllness = DiseaseSim6b.numLengthOfIllness;
    int contaminationDistance = DiseaseSim6b.numContaminationDistance;
    int univNorth=40,univEast=500,univSouth=500,univWest=100;
    int startX,endX,startY,endY;
    boolean sick;
    Person[] people  = new Person[numPeople];
    Rng randn = new Rng();
    Locator datastore = new Locator(contaminationDistance);
    Group[] group = new Group[numGroups];
    Attractor[] attractor = new Attractor[numAttractors];
    JButton pressButton = new JButton("Press For One Iteration");

//********************BUG RACE CODE*********************************
                double percent;
                Font bigFont = new Font("Helvetica",Font.BOLD,24);
                JTextField  percentField = new JTextField("0",8);
//******************************************************************

     public DiseaseSimFrame()
    {super("Disease Frame");
      setSize(500,500);
      //setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLocationRelativeTo(null);
      Container con = getContentPane();
      con.setLayout (new FlowLayout());
      con.add(pressButton);
//***************BUG RACE CODE*************************************
                        con.add(percentField);
//*****************************************************************
      pressButton.addActionListener(this);

      startX = univWest +  (int) (.1 * (double) univWest);
      endX   = univEast -  (int) (.1 * (double) univEast);
      startY = univNorth + (int) (.1 * (double) univNorth);
      endY   = univSouth - (int) (.1 * (double) univSouth);

      for(i=0;i<numPeople;i++)
      {
```

```
        //use constructor that sets NESW bounds,sets initial x y randomly, and passes the randn
object
        people[i] = new
Person(univNorth,univEast,univSouth,univWest,randn.getR(startX,endX),randn.getR(startY,endY),r
andn);

      }

      for(i=0;i<numGroups;i++)
      {
              //(north,east,south,west,whenToMeet,howLongToMeet,groupID)
        group[i] = new Group(75+(100*i),75,150+(100*i),0,randn.getR(15,25),randn.getR(5,10),i+1);
        group[i].setUnivBoundaries(univNorth,univEast,univSouth,univWest);
      }


      for(i=0;i<numAttractors;i++)
      {
        //use constructor that sets up attractors
        attractor[i] = new Attractor(randn);
      }

      r=randn.getR(0,numPeople-1);
      people[r].setSick(true);

      //ASSIGN PERSON TO A GROUP
      for(j=0;j<numGroups;j++)
      {
        for(i=0;i<numPeoplePerGroup;i++)
        {
          r=randn.getR(0,numPeople-1);
          if(people[r].getInGroup()==0){
            people[r].setInGroup(j+1);
          }
          else{
            i=i-1;
          }
        }
      }
      //ASSIGN PERSON TO AN ATTRACTOR
      for(j=0;j<numAttractors;j++)
      {
        for(i=0;i<numPeoplePerAttractor;i++)
        {
          r=randn.getR(0,numPeople-1);
          if(people[r].getInGroup()==0)
          {
            if(people[r].getInAttractor()==0){
```

```
                people[r].setInAttractor(j+1);
              }
              else{
                i=i-1;
              }
            }
            else{
              i=i-1;
            }
          }
        }

      }

      public void actionPerformed(ActionEvent e)
      {
        Object source = e.getSource();
        if(source == pressButton)
        {
          //INCREMENT UNIVERSAL TIME
          univTime=univTime + 1;
          repaint();
        }
      }

      public void paint(Graphics gr)
      {

        setBackground(Color.black);
        pressButton.repaint();
//*****************************************
                          percentField.repaint();




        for(i=0;i<numPeople;i++)
        {
          gr.clearRect(people[i].getx(),people[i].gety(),diameter,diameter+1);
        }
        gr.setColor(Color.yellow);
        for(i=0;i<numGroups;i++)
        {
          gr.drawRect(0,75+(100*i),75,75);
        }

        datastore.setNumPeople(numPeople);
        datastore.contaminate(people);
```

```
      for(i=0;i<numGroups;i++)
      {
        group[i].setNumPeople(numPeople);
        group[i].setUnivTime(univTime);
        group[i].grouping(people,randn);

      }

      for(i=0;i<numPeople;i++)
      {

//***Move people toward attactor  ACTIVE CODE
        if(people[i].getInAttractor()>0)
        {
          attractor[people[i].getInAttractor()-1].movePeople(people[i]);
        }
//***End Move people toward attractor ACTIVE CODE
        else
        {
          mx = randn.getR(-10,10);
          my = randn.getR(-10,10);
          people[i].move(mx,my);
        }

        sick = people[i].getSick();
        if(sick)
        {
          gr.setColor(Color.red);
          people[i].countDaySick();
        }

        else if(people[i].getInGroup()==1)
        {
          gr.setColor(Color.blue);
        }

        else if(people[i].getInGroup()==2)
        {
          gr.setColor(Color.yellow);
  //    System.out.println("yellow x="+people[i].getxcoord()+" yellow y="+people[i].getycoord());
        }

        else if(people[i].getInGroup()==3)
        {
          gr.setColor(Color.white);
        }
        else if(people[i].getInAttractor()!=0)
        {
```

```
            gr.setColor(Color.cyan);
          }

          else
          {
            gr.setColor(Color.green);
          }


          gr.fillOval(people[i].getx(),people[i].gety(),diameter,diameter);

          if(sick && people[i].getDaySick() > lengthOfIllness)
          {
            people[i].setSick(false);
            people[i].setDaySick(0);
          }
        }

        gr.setColor(Color.white);
        for(i=0;i<numAttractors;i++)
        {
          gr.fillRect(attractor[i].getXCoord(),attractor[i].getYCoord(),diameter+10,diameter+10);
        }

//*****************************BUG RACE CODE**********************
                        double percentNum=0;
                        for(i=0; i<numPeople; i++)
                        {

                                if(people[i].getSick() == true)
                                {
                                        percentNum = percentNum + 1;
                                }
                        }
                percent = ((int)((percentNum/numPeople) * 1000))/10.;
                String strPercent = Double.toString(percent);
                percentField.setText(strPercent+"% Sick");
//*************************************************************
            System.out.print(percent+",");
    }

     void pause(int time)
    {
      try{ Thread.sleep(time);}
         catch(InterruptedException e) { }
    }

  }
```

```
 class ErrorFrame extends JFrame implements ActionListener
{
  JLabel error1 = new JLabel("Number of People must be greater than or equal to 120");
  JButton close = new JButton("Close");
   ErrorFrame()
  {
    setSize(350,120);
    Container con = getContentPane();
    con.setLayout (new FlowLayout());
    con.add(error1);
    con.add(close);
    close.addActionListener(this);
  }

   public void actionPerformed(ActionEvent ae)
  {
    if(ae.getSource() == close)
    {
      this.dispose();
    }
  }


}
```

Attractor.java

```
/*
AISC AM Programming class
2/10/04
Attractor for Continuous Groups
*/

import java.util.Random;

public class Attractor
{
        private int attractorXCoord;
        private int attractorYCoord;


        //constructing attractor w/ coordinates
        Attractor(Rng randn){
        attractorXCoord=(int)(randn.getR(100,450));
        attractorYCoord=(int)(randn.getR(100,450));
        }
```

```
            public void movePeople(Person p){
            if(p.getx()<=attractorXCoord && p.gety()<=attractorYCoord)
            p.moveInAttractor(0,0,20,20);
            if(p.getx()>=attractorXCoord && p.gety()<=attractorYCoord)
            p.moveInAttractor(-20,0,0,20);
            if(p.getx()<=attractorXCoord && p.gety()>=attractorYCoord)
            p.moveInAttractor(0,-20,20,0);
            if(p.getx()>=attractorXCoord && p.gety()>=attractorYCoord)
            p.moveInAttractor(-20,-20,0,0);
            }

            public void setXCoord(int x){
            attractorXCoord=x;
            }

            public int getXCoord(){
            return attractorXCoord;
            }

            public void setYCoord(int y){
            attractorYCoord=y;
            }

            public int getYCoord(){
            return attractorYCoord;
            }


    void pause(int time)
    {
     try{ Thread.sleep(time);}
     catch(InterruptedException e) { }
    }
}
```

Group.java

```
public class Group
{
            private int lengthGroup=500, numPeople, i, mx, my;
      private int groupNorth, groupEast, groupWest, groupSouth;
      private int univEast,univSouth, univNorth, univWest;
      private int startX, endX, startY, endY;
      private int groupID;
      private int univTime, whenToMeet, howLongToMeet,meetingCount=0;
      private boolean currentlyMeeting=false;
//          Rng randn = new Rng();
```

```
    Group(int n,int e,int s, int w, int wh, int hl, int g)
    {
        groupNorth=n; groupEast=e; groupSouth=s; groupWest=w;
        whenToMeet=wh; howLongToMeet=hl; groupID=g;
//         System.out.println("whenToMeet="+whenToMeet+"
howLongToMeet="+howLongToMeet);
    }


    public void setNumPeople(int p)
    {
                numPeople = p;
    }

    public void setUnivTime(int u)
    {
       univTime = u;
    }

    public void setUnivBoundaries(int n, int e, int s, int w)
    {
        univNorth=n; univEast=e; univSouth=s; univWest=w;
//System.out.println("Group univNorth="+univNorth+" univEast="+univEast+"
univSouth="+univSouth+" univWest="+univWest);
    }

    public void setGroupID(int i)
    {
       groupID = i;
    }

    public void setBoundaries(int n, int e, int s, int w)
    {
                groupNorth = n;
                groupEast = e;
                groupSouth = s;
                groupWest = w;
    }

    public void setgroupNorth(int n)
    {
                    groupNorth = n;
    }

    public void setgroupEast(int e)
    {
```

```
                groupEast = e;
  }

  public void setgroupSouth(int s)
  {
                groupSouth = s;
  }

  public void setgroupWest(int w)
  {
                groupWest = w;
  }

  public int getGroupID()
  {
                return groupID;
  }

            public int getgroupNorth()
  {
                return groupNorth;
  }

  public int getgroupEast()
  {
                return groupEast;
  }

  public int getgroupSouth()
  {
                return groupSouth;
  }

  public int getgroupWest()
  {
                return groupWest;
  }

public void grouping(Person[] people,Rng randn)
{
 if(univTime % whenToMeet==0)
 {
   currentlyMeeting=true;
   for(i=0;i<numPeople;i++)
   {
              if(people[i].getInGroup()==groupID)
              {
                        people[i].setNorth(groupNorth);
```

```
                    people[i].setEast(groupEast);
                    people[i].setSouth(groupSouth);
                    people[i].setWest(groupWest);
                people[i].setxcoord(randn.getR(groupWest,groupEast));
                people[i].setycoord(randn.getR(groupNorth,groupSouth));
                people[i].setCurrentlyMeeting(true);
/*
  System.out.println("people["+i+"].currentlyMeeting="+people[i].getCurrentlyMeeting());
  System.out.println("groupNorth="+people[i].getNorth());
  System.out.println("groupEast="+people[i].getEast());
  System.out.println("groupSouth="+people[i].getSouth());
  System.out.println("groupWest="+people[i].getWest());
*/
        }
     }//end for

   }//end if

   if(currentlyMeeting)
   {
          meetingCount = meetingCount + 1;
   }

   if(meetingCount> howLongToMeet)
   {
                meetingCount = 0;
          currentlyMeeting = false;

     for(i=0;i<numPeople;i++)
     {
        if(people[i].getInGroup()==groupID)
        {

              people[i].setNorth(univNorth);
              people[i].setEast(univSouth);
              people[i].setSouth(univEast);
              people[i].setWest(univWest);

             startX = univWest +  (int) (.1 * (double) univWest);
             endX   = univEast -  (int) (.1 * (double) univEast);
             startY = univNorth + (int) (.1 * (double) univNorth);
             endY   = univSouth - (int) (.1 * (double) univSouth);

             people[i].setxcoord(randn.getR(startX,endX));
             people[i].setycoord(randn.getR(startY,endY));

              people[i].setGroupDay(0);
        }
```

```
    }
   }

 }
}
```

Locator.java

```
// Programmer:          AISC AM Programming class
//Date:    January 27, 2004
//Program:     Locator Class is used to check if sick people
//                  are near well people thereby making them sick by contact.
//                  It "scans" the people array

public class Locator
{
 private int i,j,totalSick,numPeople,contaminationDistance;

 Locator(int d)
 {
  contaminationDistance = d;
 }

 public void setContaminationDistance(int d)
 {
  contaminationDistance = d;
 }

 public int getContaminationDistance()
 {
  return contaminationDistance;
 }


 public void setNumPeople(int n)
 {
  numPeople = n;
 }

 public void contaminate(Person[] people)
 {
  //Search for sick person
  for(i=0;i<numPeople;i++)
  {

   if(people[i].getSick())
   {
    for(j=0;j<numPeople;j++)
```

```
      {
       //calculate proximity of others to sick person
       if(Math.abs(people[i].getx()-people[j].getx())<=contaminationDistance &&
                          Math.abs(people[i].gety()-people[j].gety())<=contaminationDistance &&
                  !people[j].getSick())
       {
         people[j].setSick(true);
       }
      }
    }
  }
 }
}
```

Person.java

```
// Programmer:          AISC AM Programming class
//Program:     Person.java
//Description: This program is a class that acts like a Person that can
//             move horizontally and vertically in random units
//                  The person can also be sick or well
//Date:          November 4, 2003

public class Person{
  //private data members
  private int east,west,north,south;
  private int xcoord, ycoord;
  private int xmove, ymove;
  private int start, end;
  private int numDaysSick;
  private int lengthGroup;
  private String type;
  private boolean sick;
  private Rng randn = new Rng();
  private int inAttractor;
  private int inGroup;
  private boolean currentlyMeeting=false;

  //constructor default
  Person()
  {
    xcoord=0;
    ycoord=0;
    type = "generic";
    sick=false;
    numDaysSick=0;
  }
```

```
//constructor set NESW boundaries and set x and y positions
Person(int n,int e,int s, int w, int x, int y, Rng randn)
{
  north=n; east=e; south=s; west=w;
  xcoord = x;    ycoord = y;
  type = "generic";
  sick = false;
}

//constructor set x and y positions
Person(int x,int y)
{
  xcoord = x;
  ycoord = y;
  type = "generic";
  sick = false;
}

//constructor set x,y,type,sick
Person(int x,int y,String t,boolean s )
{
  xcoord = x;
  ycoord = y;
  type = t;
  sick = s;
}

//set methods
public void setCurrentlyMeeting(boolean i)
{
     currentlyMeeting = i;
}

public void setxcoord(int x)
{
     xcoord=x;
}

public void setycoord(int y)
{
     ycoord=y;
}

public void setNorth(int b)
{
              north = b;
}
```

```
  public void setSouth(int b)
{
        south = b;
}

  public void setEast(int b)
{
        east = b;
}

  public void setWest(int b)
{
        west = b;
}

//get methods
public boolean getCurrentlyMeeting()
{
    return currentlyMeeting;
}

public int getxcoord()
{
    return xcoord;
}

public int getycoord()
{
    return ycoord;
}

public int getNorth()
{
        return north;
}

  public int getEast()
{
        return east;
}

  public int getSouth()
{
        return south;
}

  public int getWest()
{
```

```
            return west;
}

public void move(int s, int e)
{
 start = s;
 end = e;
 xmove = randn.getR(start,end);
 ymove = randn.getR(start,end);
 xcoord = xcoord + xmove;
 ycoord = ycoord + ymove;
 if(xcoord > east)
   {
    xcoord = xcoord - east;
   }
 if(ycoord < north)
   {
    ycoord = ycoord + south;
   }
 if(xcoord < west)
   {
    xcoord = xcoord + east;
   }
 if(ycoord > south)
   {
    ycoord = ycoord - south;
   }

}

public void moveInAttractor(int sx, int sy, int ex, int ey)
{
 xmove = randn.getR(sx,ex);
 ymove = randn.getR(sy,ey);
 xcoord = xcoord + xmove;
 ycoord = ycoord + ymove;
 if(xcoord > east)
   {
    xcoord = xcoord - east;
   }
 if(ycoord < north)
   {
    ycoord = ycoord + south;
   }
 if(xcoord < west)
   {
    xcoord = xcoord + east;
   }
```

```
 if(ycoord > south)
   {
    ycoord = ycoord - south;
   }

}

public void countDaySick()
{
 numDaysSick = numDaysSick +1;
}

public void setDaySick(int n)
{
 numDaysSick = n;
}

public void countGroupDay()
{
        lengthGroup = lengthGroup + 1;
}

public void setGroupDay(int l)
{
        lengthGroup = l;
}

public int getGroupDay()
{
        return lengthGroup;
}

public int getDaySick()
{
 return numDaysSick;
}

public void setx(int x)
{
 xcoord=x;
}

public int getx()
{
 return xcoord;
}

public void sety(int y)
```

```
{
  ycoord=y;
}

public int gety()
{
  return ycoord;
}


public void setType(String t)
{
   type = t;
}

public String getType()
{
  return type;
}

public void setSick(boolean s)
{
  sick = s;
}

public boolean getSick()
{
  return sick;
}

public void setInAttractor(int ia)
{
  inAttractor = ia;
}

public int getInAttractor()
{
  return inAttractor;
}

public void setInGroup(int ig)
{
        inGroup = ig;
}

public int getInGroup()
{
        return inGroup;
```

```
  }
}//end class
```

Rng.java

```
// Programmer:          AISC AM Programming class
//Program:      Rng.java (Random Number Generator
//Description: This program is a class that generates and returns
//              random numbers.
//                   It receives two arguments, start and end numbers.
//Date:          January 26, 2004

import java.util.Random;

public class Rng{
  //private data members
  private int start;
  private int end;
  private int range;
  private int randomNumber;

  private Random generator = new Random();
  //constructor default
  Rng()
  {
      start = 0;
      end = 10;
      range=11;
  }

  public int getR(int s,int e)
  {
      start = s;
      end = e;
      range = end - start + 1;
      randomNumber = (int) (generator.nextDouble() * range + start);
      return randomNumber;
  }
}
```