Technical details of Clusters program

Java (version J2SE 5.0) source code

BackgroundColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
public class BackgroundColorView extends ColorView implements ChangeListener {
    public void set(){
        ((Clusters)page).setBackgroundColor(getColor());
    };
    public void get(){
        setColor(((Clusters)page).getBackgroundColor());
    };
    public BackgroundColorView(Clusters page)
    {   super(page); }
}
```

BorderColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
public class BorderColorView extends ColorView implements ChangeListener {
    public void set(){
        ((Clusters)page).setBorderColor(getColor());
    };
    public void get(){
        setColor(((Clusters)page).getBorderColor());
    };
    public BorderColorView(Clusters page)
    {   super(page); }
}
```

Clusters.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import java.awt.Graphics;
import java.text.DecimalFormat;
import java.util.Collections;
import java.util.Vector;
import javax.imageio.ImageIO;
import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.image.*;
import java.awt.geom.*;
import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.Rectangle;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.io.*;
import java.awt.geom.Rectangle2D;
import java.awt.AlphaComposite;
import java.io.File;
import java.io.FilenameFilter;
import java.lang.*;
import java.awt.*;
import java.io.*;
public class Clusters implements ActionListener, ItemListener {
   Color startcolor = Color.GREEN; //seed colour
   Color endcolor = Color.RED; //trap colour
   Color bordercolor = Color.BLACK; //particle colour
   Color backgroundcolor = Color.WHITE; //background
   Color trapneighbourbordercolor = Color.BLUE; //trap neighbor colour
   Color trapneighbourstartcolor = Color.GRAY; //trap neightbour particle colour
   public Color getStartColor() { return startcolor; }
   public Color getTrapNeighbourStartColor() { return trapneighbourstartcolor; }
   public Color getEndColor() { return endcolor; }
   public Color getBorderColor() { return bordercolor; }
   public Color getTrapNeighbourBorderColor() { return trapneighbourbordercolor; }
   public Color getBackgroundColor() { return backgroundcolor; }
   public void setStartColor(Color c) { startcolor = c; grid.repaint(); }
   public void setTrapNeighbourStartColor(Color c) { trapneighbourstartcolor = c; grid.repaint(); }
   public void setEndColor(Color c) { endcolor = c; grid.repaint(); }
   public void setBorderColor(Color c) { bordercolor = c; grid.repaint(); }
   public void setTrapNeighbourBorderColor(Color c) { trapneighbourbordercolor = c; grid.repaint(); }
   public void setBackgroundColor(Color c) { backgroundcolor = c; grid.repaint(); }
   boolean highlight = false;
   boolean showseeds = true;
   boolean showtraps = true;
   boolean calculated = false;
   boolean spreadaccordingtoneighbours = false;
   boolean restricttobinary = false;
   int simulationlength = 1000;
   int displayeveryxsteps = 1;
   int step = 0;
   public float fix(float d) { //function to restrict a float to the closed interval [0-1]
      if (d < 0) return 0f;
      if (d > 1) return 1f;
      return d;
   }
   class MyJPanel extends JPanel { //our graphics panel
      public void paintComponent(Graphics g) { //in swing you always override paintComponent
         super.paintComponent(g); //you must always call the super when overriding paint
         gridPaintProcedure(g);
      }
      public Dimension getMinimumSize() {
         return new Dimension(400,400);
      }
      public Dimension getPreferredSize() {
         return new Dimension(400,400);
      }
      public void gridPaintProcedure(Graphics g) {
         int n =
            nx > ny ? nx :
               ny;
         if (n > 400)
         {  g.clearRect(0,0,400,400);
            return;
         }
         int cellwidth =
            n > 0 ? 400/n:
               400;
```

```
int borderwidth =
   cellwidth < 3 ? 0 : 1;
g.clearRect(0,0,400,400);
g.setColor(backgroundcolor);
g.fillRect(0,0,nx*cellwidth,ny*cellwidth);
g.setColor(bordercolor);
for (int i = 0; i < nx; i++)
   for (int j = 0; j < ny; j++)
      if (rectangle[i][j] ) {//!= null)
         if (trapneighbours[i][j]==0) {
            g.setColor(bordercolor);
            g.fillRect(i*cellwidth,j*cellwidth,cellwidth-borderwidth,cellwidth-borderwidth);
         } else {
            g.setColor(trapneighbourbordercolor);
            g.fillRect(i*cellwidth,j*cellwidth,cellwidth-borderwidth,cellwidth-borderwidth);
         }
      }
if (showseeds)
{  for (int i = 0; i < nx; i++)
      for (int j = 0; j < ny; j++) {
         if (seeds[i][j] > 0 && trapneighbours[i][j]==0 && traps[i][j]==0)
         {  double multiplier = Math.min(1f,(float)seeds[i][j]);
            g.setColor(new Color(
               fix((float)(
               startcolor.getRed()*multiplier/255 + bordercolor.getRed()*(1-multiplier)/255
               )),
               fix((float)(
               startcolor.getGreen()*multiplier/255 + bordercolor.getGreen()*(1-multiplier)/255
               )),
               fix((float)(
               startcolor.getBlue()*multiplier/255 + bordercolor.getBlue()*(1-multiplier)/255
               ))
            )
            );
            g.fillRect(i*cellwidth,j*cellwidth,cellwidth-borderwidth,cellwidth-borderwidth);
         }
         if (seeds[i][j] > 0 && trapneighbours[i][j] > 0 && traps[i][j]==0) {
            double multiplier = Math.min(1f,(float)seeds[i][j]);
            g.setColor(new Color(
               fix((float)(
               trapneighbourstartcolor.getRed()*multiplier/255 + trapneighbourbordercolor.getRed()*(1-multiplier)/255
               )),
               fix((float)(
               trapneighbourstartcolor.getGreen()*multiplier/255 + trapneighbourbordercolor.getGreen()*(1-multiplier)/255
               )),
               fix((float)(
               trapneighbourstartcolor.getBlue()*multiplier/255 + trapneighbourbordercolor.getBlue()*(1-multiplier)/255
               ))
            )
            );
            g.fillRect(i*cellwidth,j*cellwidth,cellwidth-borderwidth,cellwidth-borderwidth);

         }
      }
}
if (showtraps) {
   g.setColor(endcolor);
   for (int i = 0; i < nx; i++)
      for (int j = 0; j < ny; j++) {
         if (traps[i][j] > 0)
            g.fillRect(i*cellwidth,j*cellwidth,cellwidth-borderwidth,cellwidth-borderwidth);
      }
}
if (showseeds) {
   for (int i = 0; i < nx; i++)
      for (int j = 0; j < ny; j++)
      {  DecimalFormat df = new DecimalFormat();
```

```
                df.setMaximumFractionDigits(2);
                g.setColor(Color.GRAY);
                g.setFont(oursmallfont);
                if (cellwidth>10) g.drawString(df.format(seeds[i][j]), i*cellwidth, j*cellwidth+cellwidth);
             }
         }
     }
 }
MyJPanel grid = new MyJPanel();
int nx = 10, ny = 10; //cube edge length
double p = 0.50; //probability of cell being filled
double e = 0.50; //probability of hole being punched
double q = 0.50; //probability of cell being seeded
double r = 0.50; //probability of cell being trapped
int trapparts = 1;
int trappartsgaps = 1;
int nontrapparts = 1;
int nontrappartsgaps = 1;
double redistributetocentrefactor = 0;
double decayfactor = 0;
int particlecount;
int holecount;
double seedcount; int onecount;
double seedcountlines = 0;
double seedcountgaps = 0;
double fraction_given_to_non_trap_part = 0.25;
double fraction_given_to_non_trap_part_gap = 0.25;
double fraction_given_to_trap_neighbour = 0.25;
double fraction_given_to_trap_neighbour_gap = 0.25;
double fraction_given_to_non_trap_part_FROMGAP = 0.25;
double fraction_given_to_non_trap_part_gap_FROMGAP = 0.25;
double fraction_given_to_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_trap_neighbour_gap_FROMGAP = 0.25;
double fraction_given_to_trap_part_from_trap_neighbour = 0.25;
double fraction_given_to_trap_part_gap_from_trap_neighbour = 0.25;
double fraction_given_to_non_trap_part_from_trap_neighbour = 0.25;
double fraction_given_to_non_trap_part_gap_from_trap_neighbour = 0.25;
double fraction_given_to_trap_neighbour_from_trap_neighbour = 0.25;
double fraction_given_to_trap_neighbour_gap_from_trap_neighbour = 0.25;
double fraction_given_to_trap_part_from_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP = 0.25;
double fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP = 0.25;
double percentagetospread = 100;
int trapcount;
boolean[][] rectangle = new boolean[nx][ny];
double[][] seeds = new double[nx][ny];
double[][] seeds2 = new double[nx][ny];
int[][] traps = new int[nx][ny];
int[][] trapneighbours = new int[nx][ny];
private JFrame f = new JFrame("Clusters 2 - Rectangles smaller than 400x400 will be drawn in the rendering panel - Limit 1,000,000,000 cells");  //the program window and display
JTextArea display = new JTextArea("");
JButton //interaction techniques (i.e. units of user interface)
   filltechnique = new JButton("Fill the rectangle X"),
   fillstripeytechnique = new JButton("Fill the rectangle"),
   showseedstechnique = new JButton("Show/hide seeds"),
   showtrapstechnique = new JButton("Show/hide traps"),
   startsimulationtechnique = new JButton("Start simulation"),
   stopsimulationtechnique = new JButton("Stop simulation"),
   clearthetexttechnique = new JButton("Clear the text"),
   savetheimagetechnique = new JButton("Save the image"),
   reseedtechnique = new JButton("Seed centre"),
   reseedalltechnique = new JButton("Seed all");
JTextField
   percentagetospreadtechnique = new JTextField(""+percentagetospread, 10);
```

```
JCheckBox
   spreadaccordingtoneighbourstechnique = new JCheckBox("spread according to neighbours");
JCheckBox
   restricttobinarytechnique = new JCheckBox("restrict to binary");
JTextField
   nxtechnique = new JTextField(""+nx,10),
   nytechnique = new JTextField(""+ny,10),
   simulationlengthtechnique = new JTextField(""+simulationlength,10),
   displayeveryxstepstechnique = new JTextField(""+displayeveryxsteps,10),
   ptechnique = new JTextField(""+p,10),
   etechnique = new JTextField(""+p,10),
   qtechnique = new JTextField(""+q,10),
   rtechnique = new JTextField(""+r,10),
   fraction_given_to_non_trap_part_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part,10),
   fraction_given_to_non_trap_part_gap_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_gap,10),
   fraction_given_to_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour,10),
   fraction_given_to_trap_neighbour_gap_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_gap,10),
   fraction_given_to_non_trap_part_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_FROMGAP,10),
   fraction_given_to_non_trap_part_gap_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_gap_FROMGAP,10),
   fraction_given_to_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_FROMGAP,10),
   fraction_given_to_trap_neighbour_gap_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_gap_FROMGAP,10),
   fraction_given_to_trap_part_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_trap_part_from_trap_neighbour,10),
   fraction_given_to_trap_part_gap_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_trap_part_gap_from_trap_neighbour,10),
   fraction_given_to_non_trap_part_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_from_trap_neighbour,10),
   fraction_given_to_non_trap_part_gap_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_gap_from_trap_neighbour,10),
   fraction_given_to_trap_neighbour_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_from_trap_neighbour,10),
   fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_gap_from_trap_neighbour,10),
   fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_part_from_trap_neighbour_FROMGAP,10),
   fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP,10),
   fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP,10),
   fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP,10),
   fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP,10),
   fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TECHNIQUE = new JTextField(""+fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP,10),
   decayfactortechnique = new JTextField(""+decayfactor,10),
   redistributetocentrefactortechnique = new JTextField(""+redistributetocentrefactor,10);
public void actionPerformed(ActionEvent e) { //listener method
   if (e.getSource() == stopsimulationtechnique) {
      simulationrunning = false; //the thread will see this and stop
      updateSundryVariables(); //sundry variables are everything but size
   }
   if (simulationrunning) return;
   if (e.getSource() == fillstripeytechnique) {
      updateSundryVariables();
      fillStripeyProcedure();
   }
   if (e.getSource() == showseedstechnique) {
      updateSundryVariables();
      showseedsProcedure();
   }
   if (e.getSource() == showtrapstechnique) {
      updateSundryVariables();
      showtrapsProcedure();
   }
   if (e.getSource() == startsimulationtechnique) {
      updateSundryVariables();
      startsimulationProcedure();
   }
   if (e.getSource() == clearthetexttechnique) {
      updateSundryVariables();
      clearthetextProcedure();
   }
   if (e.getSource() == savetheimagetechnique) {
      updateSundryVariables();
      savetheimageProcedure();
   }
   if (e.getSource() == reseedtechnique) {
      updateSundryVariables();
```

```
         reseedProcedure();
      }
      if (e.getSource() == reseedalltechnique) {
         updateSundryVariables();
         reseedallProcedure();
      }
      updateTextFields(); //we must always repaint the control panel and the renderframe at the same time
      f.repaint();
      renderframe.repaint();
   }
   public void clearthetextProcedure() { display.setText(""); }
   public void savetheimageProcedure() { //the following lines are copied form gridPaintProcedure
      int n =
            nx > ny ? nx :
               ny;
         if (n > 400)
         {  return;
         }
         int cellwidth =
            n > 0 ? 400/n:
               400;
         int borderwidth =
            cellwidth < 3 ? 0 : 1;
      int width = nx*cellwidth, height = ny*cellwidth;
      BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
      Graphics2D g = image.createGraphics();
         grid.gridPaintProcedure(g);
      g.dispose(); //now pop a file selector up:
      JFileChooser fc = new JFileChooser();
      fc.setSelectedFile(
         new File(
            fc.getCurrentDirectory().getPath() + File.separator + "image.png"
         )
      );
      javax.swing.filechooser.FileFilter fileFilter = new javax.swing.filechooser.FileFilter() {
         public boolean accept(File file) {
               if (file.isDirectory()) return true;
               String name = file.getName();
               String extension = "";
               //put in braces so that the the variable i is only in scope within the braces
               { int i = name.length()-1; while (!(i==0||name.charAt(i)=='.')) i--;
               extension = name.substring(i+1, name.length());
               }
               return extension.toLowerCase().equals("png");
         }
         public String description = "Portable Network Graphics (*.png)";
            public String getDescription() {
               return description;
            }
            public void setDescription(String description) {
               this.description = description;
            }
      };
      fc.setFileFilter(fileFilter);
      renderframe.setAlwaysOnTop(false);
      colorframe.setAlwaysOnTop(false);
      int result =fc.showSaveDialog(f);
      File file;
      boolean safetosave = false;
      if (result == JFileChooser.CANCEL_OPTION) {
         colorframe.setAlwaysOnTop(true);
         renderframe.setAlwaysOnTop(true);
         colorframe.show();
         renderframe.show();
         return;
      }
      else if (result == JFileChooser.APPROVE_OPTION) {
        file = fc.getSelectedFile();
        if (file.isDirectory()) {
           JOptionPane.showMessageDialog(f, "Must choose a file, not a directory.");
           colorframe.setAlwaysOnTop(true);
           renderframe.setAlwaysOnTop(true);
           colorframe.show();
           renderframe.show();
           return;
        }
        if (file.exists()) {
           int response = JOptionPane.showConfirmDialog(
              null,
              "Overwrite existing file?","Confirm Overwrite",
              JOptionPane.OK_CANCEL_OPTION,
              JOptionPane.QUESTION_MESSAGE
           );
           if(response == JOptionPane.CANCEL_OPTION || response == JOptionPane.CLOSED_OPTION ) {
              colorframe.setAlwaysOnTop(true);
              renderframe.setAlwaysOnTop(true);
              colorframe.show();
              renderframe.show();
              return;
           }

        } //all is ok to save:
        try {
           ImageIO.write(image,"png", file);
           colorframe.setAlwaysOnTop(true);
           renderframe.setAlwaysOnTop(true);
           colorframe.show();
           renderframe.show();
           return;
        }
        catch (Exception e) { JOptionPane.showMessageDialog(f, "Was unable to save file. The problem was: "+e);
           colorframe.setAlwaysOnTop(true);
           renderframe.setAlwaysOnTop(true);
           colorframe.show();
           renderframe.show();
           return;
        }
     }
     else {
        colorframe.setAlwaysOnTop(true);
        renderframe.setAlwaysOnTop(true);
        colorframe.show();
        renderframe.show();
        return;
     }
  }
  public void itemStateChanged(ItemEvent e) {
     Object source = e.getItemSelectable();
     if (source == spreadaccordingtoneighbourstechnique) {
        boolean spreadaccordingtoneighbourstechniqueclicked = false;
        boolean spreadaccordingtoneighbourstechniquevalue = true;
        spreadaccordingtoneighbourstechniqueclicked = true;
        if (e.getStateChange() == ItemEvent.DESELECTED) {
           spreadaccordingtoneighbourstechniquevalue = false;
        }
        spreadaccordingtoneighbours = spreadaccordingtoneighbourstechniquevalue;
     }

     if (source == restricttobinarytechnique) {
        boolean restricttobinarytechniqueclicked = false;
        boolean restricttobinarytechniquevalue = true;
        restricttobinarytechniqueclicked = true;
        if (e.getStateChange() == ItemEvent.DESELECTED) {
           restricttobinarytechniquevalue = false;
```

```
            }
            restricttobinary = restricttobinarytechniquevalue;
        }
    }
    void reseedProcedure() {
        seedcount = 0; onecount = 0;
        seedcountlines = 0;
        seedcountgaps = 0;
        simseedcount = 0; simonecount = 0;
        simseedcountlines = 0;
        simseedcountgaps = 0;
        simtrapenergy = 0;
        simtrapenergylines = 0;
        simtrapenergygaps = 0;
        step = 0;
        for (int i = 0; i < nx; i++)
            for (int j = 0; j < ny; j++)
                if (rectangle[i][j] )
                {   if (
                        (   // this is meant to achieve seeding of central ninth exactly when nx and ny are multiples of 3. the behaviour when nx and ny are not multiples of 3 was not 
considered a design goal
                            (nx/3)<=i && i<((2*nx)/3))
                            &&
                            ((ny/3)<=j && j<((2*ny)/3)
                            &&
                            traps[i][j]==0
                        )
                    )
                    {   seeds[i][j] = 1;
                        seedcount++; onecount++;
                        if (i%2==0) seedcountlines++;
                        if (i%2==1) seedcountgaps++;
                    }
                    else seeds[i][j] =0;
                }
                else {
                    seeds[i][j] = 0;
                }
        display.append(
            "CENTRAL NINTH SEEDED:\t"+
            "NUMBER OF SEEDS:\t"+
            "NUMBER OF ONES:\t"+
            "SEEDS IN LINES (a):\t"+
            "SEEDS IN GAPS (b):\t"+
            "a+b:\n"+
            "\t"+seedcount+"\t"+onecount+"\t"+seedcountlines+"\t"+seedcountgaps+"\t"+(seedcountlines + seedcountgaps)+"\n");
        f.repaint();
        renderframe.repaint();
    }
    void reseedallProcedure() {
        seedcount = 0; onecount = 0;
        seedcountlines = 0;
        seedcountgaps = 0;
        simseedcount = 0; simonecount = 0;
        simseedcountlines = 0;
        simseedcountgaps = 0;
        simtrapenergy = 0;
        simtrapenergylines = 0;
        simtrapenergygaps = 0;
        step = 0;
        for (int i = 0; i < nx; i++)
            for (int j = 0; j < ny; j++)
                if (rectangle[i][j] )
                {   if (
                        (
                            traps[i][j]==0
                        )
                    )
                    {   seeds[i][j] = 1;
                        seedcount++; onecount++;
                        if (i%2==0) seedcountlines++;
                        if (i%2==1) seedcountgaps++;
                    }
                    else seeds[i][j] =0;
                }
                else {
                    seeds[i][j] = 0;
                }
        display.append(
            "ALL SEEDED:\t"+
            "NUMBER OF SEEDS:\t"+
            "NUMBER OF ONES:\t"+
            "SEEDS IN LINES (a):\t"+
            "SEEDS IN GAPS (b):\t"+
            "a+b:\n"+
            "\t"+seedcount+"\t"+onecount+"\t"+seedcountlines+"\t"+seedcountgaps+"\t"+(seedcountlines + seedcountgaps)+"\n");
        f.repaint();
        renderframe.repaint();
    }
    boolean simulationrunning = false; //indicates permission to run
    SwingWorker simulationthread;
    void startsimulationProcedure() {
        simulationthread = new SwingWorker() {
            public Object construct() {
                        step = 0;
                        simseedcount = seedcount; simonecount = onecount;
                        simseedcountlines = seedcountlines;
                        simseedcountgaps = seedcountgaps;
                        display.append(
                            "STARTING SIMULATION:\n"
                        );
                        display.append(
                            "STEP:\tSEEDS LEFT (s):\t"+
                            "ONES LEFT:\t"+
                            "SEEDS IN LINES (a):\t"+
                            "SEEDS IN GAPS (b):\t"+
                            "TRAPPED SEEDS (t):\t"+
                            "TRA. SEE. IN LINES (l):\t"+
                            "TRA. SEE. IN GAPS (g):\t"+
                            "s+t:\ta+b:\tl+g:\n"
                        );


display.append((step)+"\t"+simseedcount+"\t"+simonecount+"\t"+simseedcountlines+"\t"+simseedcountgaps+"\t"+simtrapenergy+"\t"+simtrapenergylines+"\t"+simtrapenergygaps+"\t"+(simseedcount
+simtrapenergy)+"\t"+(simseedcountlines+simseedcountgaps)+"\t"+(simtrapenergylines+simtrapenergygaps)+"\n");

                        while (step < simulationlength && simseedcount != 0 && simulationrunning)
                        {   stepsim();
                            if ((step+1)%displayeveryxsteps==0) {


display.append((step+1)+"\t"+simseedcount+"\t"+simonecount+"\t"+simseedcountlines+"\t"+simseedcountgaps+"\t"+simtrapenergy+"\t"+simtrapenergylines+"\t"+simtrapenergygaps+"\t"+(simseedcou
nt+simtrapenergy)+"\t"+(simseedcountlines+simseedcountgaps)+"\t"+(simtrapenergylines+simtrapenergygaps)+"\n");

                            display.setCaretPosition(display.getText().length());
                            }
                            step++;
                        }
                        f.repaint();
                        renderframe.repaint(); //this used to be before the step++, but logically it should be moved out of the loop since we only need to call it at the end
                        seedcount = simseedcount; onecount = simonecount;
                        seedcountlines = simseedcountlines;
                        seedcountgaps = simseedcountgaps;
                        simulationrunning = false;
                    return null;
            }
        };
        simulationrunning = true;
        simulationthread.start();
    }
    double simseedcount = 0; int simonecount = 0; //number of cells that have value 1
    double simseedcountlines = 0;
    double simseedcountgaps = 0;
    double simtrapenergy = 0;
    double simtrapenergylines = 0;
    double simtrapenergygaps = 0;
    void stepsim()
    {   simseedcount = 0; simonecount = 0;
        simseedcountlines = 0;
        simseedcountgaps = 0;
        simtrapenergy = 0;
        simtrapenergylines = 0;
        simtrapenergygaps = 0;
        for (int i = 0; i < nx; i++)
            for (int j = 0; j < ny; j++)
            {   if (traps[i][j]==0) seeds2[i][j] = 0;
                if (traps[i][j]==1) seeds2[i][j] = seeds[i][j];
            } //redistribute to centre
                    double pot = 0;
                    for (int i = 0; i < nx; i++)
                        for (int j = 0; j < ny; j++)
                        {   if (traps[i][j]==0) {
                                double take = seeds[i][j]*redistributetocentrefactor;
                                pot += take;
                                seeds[i][j] -= take;
                            }
                        }
                    int sitecount = 0;
                    for (int i = 0; i < nx; i++)
                        for (int j = 0; j < ny; j++)
                            if (rectangle[i][j] )//!= null)
                            {   if (((nx/3)<=i&&i<((2*nx)/3)) && ((ny/3)<=j&&j<((2*ny)/3))   )
                                {   sitecount++;
                                }
                            }
                    for (int i = 0; i < nx; i++)
                        for (int j = 0; j < ny; j++)
                            if (rectangle[i][j] )//!= null)
                            {   if (((nx/3)<=i&&i<((2*nx)/3)) && ((ny/3)<=j&&j<((2*ny)/3))   )
                                {   seeds2[i][j]+=pot/sitecount;
                                }
                            }
        for (int i = 0; i < nx; i++)
            for (int j = 0; j < ny; j++)
                if (traps[i][j]==0) {
                    double energytospread = seeds[i][j]*(1-decayfactor);
                    seeds[i][j] = 0;
                        double energygiventonontrappart = fraction_given_to_non_trap_part*energytospread;
                        double energygiventonontrappartgap = fraction_given_to_non_trap_part_gap*energytospread;
                        double energygiventotrapneighbour = fraction_given_to_trap_neighbour*energytospread;
                        double energygiventotrapneighbourgap = fraction_given_to_trap_neighbour_gap*energytospread;
                        double energygiventonontrappart_FROMGAP = fraction_given_to_non_trap_part_FROMGAP*energytospread;
                        double energygiventonontrappartgap_FROMGAP = fraction_given_to_non_trap_part_gap_FROMGAP*energytospread;
                        double energygiventotrapneighbour_FROMGAP = fraction_given_to_trap_neighbour_FROMGAP*energytospread;
                        double energygiventotrapneighbourgap_FROMGAP = fraction_given_to_trap_neighbour_gap_FROMGAP*energytospread;
                        double energygiventotrappart_fromtrapneighbour = fraction_given_to_trap_part_from_trap_neighbour*energytospread;
                        double energygiventotrappartgap_fromtrapneighbour = fraction_given_to_trap_part_gap_from_trap_neighbour*energytospread;
                        double energygiventonontrappart_fromtrapneighbour = fraction_given_to_non_trap_part_from_trap_neighbour*energytospread;
                        double energygiventonontrappartgap_fromtrapneighbour = fraction_given_to_non_trap_part_gap_from_trap_neighbour*energytospread;
                        double energygiventotrapneighbour_fromtrapneighbour = fraction_given_to_trap_neighbour_from_trap_neighbour*energytospread;
                        double energygiventotrapneighbourgap_fromtrapneighbour =  fraction_given_to_trap_neighbour_gap_from_trap_neighbour*energytospread;
```

```
double energygiventotrappart_fromtrapneighbour_FROMGAP = fraction_given_to_trap_part_from_trap_neighbour_FROMGAP*energytospread;
double energygiventotrappartgap_fromtrapneighbour_FROMGAP = fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP*energytospread;
double energygiventonontrappart_fromtrapneighbour_FROMGAP = fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP*energytospread;
double energygiventonontrappartgap_fromtrapneighbour_FROMGAP = fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP*energytospread;
double energygiventotrapneighbour_fromtrapneighbour_FROMGAP =   fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP*energytospread;
double energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP =fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP*energytospread;
if (rectangle[i][j] && trapneighbours[i][j]==0 && i%2==0) {//spreading from normal seed IN LINE!
   double spentenergy = 0;
   if (inRange(i-1,j) && rectangle[i-1][j]) { //EAST
      if (trapneighbours[i-1][j]==0) {
         if ((i-1)%2==0) { seeds2[i-1][j]+= energygiventonontrappart;  spentenergy += energygiventonontrappart; }
         else { seeds2[i-1][j]+=  energygiventonontrappartgap;  spentenergy += energygiventonontrappartgap; }
      }
      else {
         if ((i-1)%2==0) { seeds2[i-1][j]+= energygiventotrapneighbour;  spentenergy += energygiventotrapneighbour; }
         else { seeds2[i-1][j]+= energygiventotrapneighbourgap;  spentenergy += energygiventotrapneighbourgap; }
      }
   }
   if (inRange(i+1,j) && rectangle[i+1][j] ) { //WEST
      if (trapneighbours[i+1][j]==0) {
         if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventonontrappart; spentenergy += energygiventonontrappart; }
         else {seeds2[i+1][j]+= energygiventonontrappartgap; spentenergy += energygiventonontrappartgap; }
      }
      else {
         if ((i+1)%2==0) { seeds2[i+1][j]+= energygiventotrapneighbour; spentenergy += energygiventotrapneighbour; }
         else {
            seeds2[i+1][j]+= energygiventotrapneighbourgap; spentenergy += energygiventotrapneighbourgap;
         }
      }
   }
   if (inRange(i,j-1) && rectangle[i][j-1] ) { //NORTH
      if (trapneighbours[i][j-1]==0) {
         if (i%2==0) {seeds2[i][j-1]+= energygiventonontrappart; spentenergy += energygiventonontrappart; }
         else {seeds2[i][j-1]+= energygiventonontrappartgap;  spentenergy += energygiventonontrappartgap; }
      }
      else {
         if (i%2==0) { seeds2[i][j-1]+= energygiventotrapneighbour;  spentenergy += energygiventotrapneighbour; }
         else { seeds2[i][j-1]+= energygiventotrapneighbourgap;  spentenergy += energygiventotrapneighbourgap; }
      }
   }
   if (inRange(i,j+1) && rectangle[i][j+1] ) { //SOUTH
      if (trapneighbours[i][j+1]==0) {
         if (i%2==0) {seeds2[i][j+1]+= energygiventonontrappart;  spentenergy += energygiventonontrappart; }
         else {seeds2[i][j+1]+= energygiventonontrappartgap;  spentenergy += energygiventonontrappartgap; }
      }
      else {
         if (i%2==0) { seeds2[i][j+1]+= energygiventotrapneighbour; spentenergy += energygiventotrapneighbour; }
         else {seeds2[i][j+1]+= energygiventotrapneighbourgap; spentenergy += energygiventotrapneighbourgap; }
      }
   }
   seeds2[i][j] += energytospread-spentenergy;
}
if (rectangle[i][j] && trapneighbours[i][j]==0 && i%2==1) { //spreading from normal seed IN GAP!
   double spentenergy = 0;
   if (inRange(i-1,j) && rectangle[i-1][j] ) { //EAST
      if (trapneighbours[i-1][j]==0) {
         if ((i-1)%2==0) { seeds2[i-1][j]+= energygiventonontrappart_FROMGAP;  spentenergy += energygiventonontrappart_FROMGAP; }
         else { seeds2[i-1][j]+=  energygiventonontrappartgap_FROMGAP;  spentenergy += energygiventonontrappartgap_FROMGAP; }
      }
      else {
         if ((i-1)%2==0) {
            seeds2[i-1][j]+= energygiventotrapneighbour_FROMGAP;  spentenergy += energygiventotrapneighbour_FROMGAP;
         }
         else {
            seeds2[i-1][j]+= energygiventotrapneighbourgap_FROMGAP;  spentenergy += energygiventotrapneighbourgap_FROMGAP;
         }
      }
```

```
        }
        if (inRange(i+1,j) && rectangle[i+1][j] ) { //WEST
            if (trapneighbours[i+1][j]==0) {
                if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventonontrappart_FROMGAP; spentenergy += energygiventonontrappart_FROMGAP; }
                else {seeds2[i+1][j]+= energygiventonontrappartgap_FROMGAP; spentenergy += energygiventonontrappartgap_FROMGAP; }
            }
            else {
                if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventotrapneighbour_FROMGAP; spentenergy += energygiventotrapneighbour_FROMGAP; }
                else {
                    seeds2[i+1][j]+= energygiventotrapneighbourgap_FROMGAP; spentenergy += energygiventotrapneighbourgap_FROMGAP;
                }
            }
        }
        if (inRange(i,j-1) && rectangle[i][j-1] ) { //NORTH
            if (trapneighbours[i][j-1]==0) {
                if (i%2==0) {seeds2[i][j-1]+= energygiventonontrappart_FROMGAP; spentenergy += energygiventonontrappart_FROMGAP; }
                else {seeds2[i][j-1]+= energygiventonontrappartgap_FROMGAP;  spentenergy += energygiventonontrappartgap_FROMGAP; }
            }
            else {
                if (i%2==0){  seeds2[i][j-1]+= energygiventotrapneighbour_FROMGAP;  spentenergy += energygiventotrapneighbour_FROMGAP; }

                else      {seeds2[i][j-1]+= energygiventotrapneighbourgap_FROMGAP;  spentenergy += energygiventotrapneighbourgap_FROMGAP; }

            }
        }
        if (inRange(i,j+1) && rectangle[i][j+1] ) { //SOUTH
            if (trapneighbours[i][j+1]==0) {
                if (i%2==0) {seeds2[i][j+1]+= energygiventonontrappart_FROMGAP;  spentenergy += energygiventonontrappart_FROMGAP; }
                else {seeds2[i][j+1]+= energygiventonontrappartgap_FROMGAP;  spentenergy += energygiventonontrappartgap_FROMGAP; }
            }
            else {
                if (i%2==0) {    seeds2[i][j+1]+= energygiventotrapneighbour_FROMGAP;
                        spentenergy += energygiventotrapneighbour_FROMGAP;
                }
                else {    seeds2[i][j+1]+= energygiventotrapneighbourgap_FROMGAP;
                    spentenergy += energygiventotrapneighbourgap_FROMGAP;
                }
            }
        }
        seeds2[i][j] += energytospread-spentenergy;
    }
    if (rectangle[i][j] && trapneighbours[i][j]!=0 && i%2==0) {   //spreading from trapneighbour IN LINE!
        double spentenergy = 0;
        if (inRange(i-1,j) && rectangle[i-1][j] ) { //EAST
            if (traps[i-1][j]==1) {
                if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventotrappart_fromtrapneighbour;  spentenergy += energygiventotrappart_fromtrapneighbour; }
                else {seeds2[i-1][j]+= energygiventotrappartgap_fromtrapneighbour;   spentenergy += energygiventotrappartgap_fromtrapneighbour; }
            }
            else if (trapneighbours[i-1][j]==0) {
                if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventonontrappart_fromtrapneighbour;   spentenergy += energygiventonontrappart_fromtrapneighbour; }
                else {seeds2[i-1][j]+= energygiventonontrappartgap_fromtrapneighbour;   spentenergy += energygiventonontrappartgap_fromtrapneighbour; }
            }
            else {
                    if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventotrapneighbour_fromtrapneighbour;    spentenergy += energygiventotrapneighbour_fromtrapneighbour;
            }
                    else { seeds2[i-1][j]+= energygiventotrapneighbourgap_fromtrapneighbour;    spentenergy += energygiventotrapneighbourgap_fromtrapneighbour;
            }

            }
        }
        if (inRange(i+1,j) && rectangle[i+1][j] ) { //WEST
            if (traps[i+1][j]==1) {
                if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventotrappart_fromtrapneighbour;   spentenergy += energygiventotrappart_fromtrapneighbour; }
                else {seeds2[i+1][j]+= energygiventotrappartgap_fromtrapneighbour;   spentenergy += energygiventotrappartgap_fromtrapneighbour; }
            }
            else if (trapneighbours[i+1][j]==0) {
                if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventonontrappart_fromtrapneighbour;    spentenergy += energygiventonontrappart_fromtrapneighbour; }
```

```
         else { seeds2[i+1][j]+= energygiventonontrappartgap_fromtrapneighbour;   spentenergy += energygiventonontrappartgap_fromtrapneighbour; }
      }
      else {
         if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventotrapneighbour_fromtrapneighbour;       spentenergy += energygiventotrapneighbour_fromtrapneighbour; }
         else {seeds2[i+1][j]+= energygiventotrapneighbourgap_fromtrapneighbour;       spentenergy += energygiventotrapneighbourgap_fromtrapneighbour; }
      }
   }
   if (inRange(i,j-1) && rectangle[i][j-1] ) { //NORTH
      if (traps[i][j-1]==1) {
         if (i%2==0) {seeds2[i][j-1]+= energygiventotrappart_fromtrapneighbour;   spentenergy += energygiventotrappart_fromtrapneighbour; }
         else {seeds2[i][j-1]+= energygiventotrappartgap_fromtrapneighbour;   spentenergy += energygiventotrappartgap_fromtrapneighbour; }
      }
      else if (trapneighbours[i][j-1]==0)
      {
         if (i%2==0) {seeds2[i][j-1]+= energygiventonontrappart_fromtrapneighbour;  spentenergy += energygiventonontrappart_fromtrapneighbour; }
         else {seeds2[i][j-1]+= energygiventonontrappartgap_fromtrapneighbour;  spentenergy += energygiventonontrappartgap_fromtrapneighbour; }
      }
      else {
            if (i%2==0) { seeds2[i][j-1]+= energygiventotrapneighbour_fromtrapneighbour; spentenergy += energygiventotrapneighbour_fromtrapneighbour; }
            else { seeds2[i][j-1]+= energygiventotrapneighbourgap_fromtrapneighbour; spentenergy += energygiventotrapneighbourgap_fromtrapneighbour; }
      }
   }
   if (inRange(i,j+1) && rectangle[i][j+1] ) { //SOUTH
      if (traps[i][j+1]==1) {
         if (i%2==0) {seeds2[i][j+1]+= energygiventotrappart_fromtrapneighbour;   spentenergy += energygiventotrappart_fromtrapneighbour; }
         else {seeds2[i][j+1]+= energygiventotrappartgap_fromtrapneighbour;   spentenergy += energygiventotrappartgap_fromtrapneighbour; }
      }
      else if (trapneighbours[i][j+1]==0) {

         if (i%2==0) {seeds2[i][j+1]+= energygiventonontrappart_fromtrapneighbour;    spentenergy += energygiventonontrappart_fromtrapneighbour; }
      else {seeds2[i][j+1]+= energygiventonontrappartgap_fromtrapneighbour;    spentenergy += energygiventonontrappartgap_fromtrapneighbour; }
      }
      else {
            if (i%2==0){ seeds2[i][j+1]+= energygiventotrapneighbour_fromtrapneighbour;   spentenergy += energygiventotrapneighbour_fromtrapneighbour; }
            else  {seeds2[i][j+1]+= energygiventotrapneighbourgap_fromtrapneighbour;   spentenergy += energygiventotrapneighbourgap_fromtrapneighbour; }

      }
   }
   seeds2[i][j] += energytospread-spentenergy;
}
if (rectangle[i][j] && trapneighbours[i][j]!=0 && i%2==1) {   //spreading from trapneighbour IN GAP!
   double spentenergy = 0;
   if (inRange(i-1,j) && rectangle[i-1][j] ) { //EAST
      if (traps[i-1][j]==1) {
         if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventotrappart_fromtrapneighbour_FROMGAP;  spentenergy += energygiventotrappart_fromtrapneighbour_FROMGAP; }
         else {seeds2[i-1][j]+= energygiventotrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappartgap_fromtrapneighbour_FROMGAP; }
      }
      else if (trapneighbours[i-1][j]==0) {
         if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventonontrappart_fromtrapneighbour_FROMGAP;   spentenergy += energygiventonontrappart_fromtrapneighbour_FROMGAP; }
         else {seeds2[i-1][j]+= energygiventonontrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventonontrappartgap_fromtrapneighbour_FROMGAP; }
      }
      else {
            if ((i-1)%2==0) {seeds2[i-1][j]+= energygiventotrapneighbour_fromtrapneighbour_FROMGAP;
               spentenergy += energygiventotrapneighbour_fromtrapneighbour_FROMGAP; }
            else {seeds2[i-1][j]+= energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP;
               spentenergy += energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP; }
      }
   }
   if (inRange(i+1,j) && rectangle[i+1][j] ) { //WEST
      if (traps[i+1][j]==1) {
         if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventotrappart_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappart_fromtrapneighbour_FROMGAP; }
         else {seeds2[i+1][j]+= energygiventotrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappartgap_fromtrapneighbour_FROMGAP; }
      }
      else if (trapneighbours[i+1][j]==0) {
         if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventonontrappart_fromtrapneighbour_FROMGAP; spentenergy += energygiventonontrappart_fromtrapneighbour_FROMGAP; }
         else { seeds2[i+1][j]+= energygiventonontrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventonontrappartgap_fromtrapneighbour_FROMGAP; }
      }
```

```
                        else {
                              if ((i+1)%2==0) {seeds2[i+1][j]+= energygiventotrapneighbour_fromtrapneighbour_FROMGAP;
                                  spentenergy += energygiventotrapneighbour_fromtrapneighbour_FROMGAP; }
                              else{ seeds2[i+1][j]+= energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP;
                                  spentenergy += energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP; }
                        }
                    }
                    if (inRange(i,j-1) && rectangle[i][j-1] ) { //NORTH
                        if (traps[i][j-1]==1) {
                           if (i%2==0) {seeds2[i][j-1]+= energygiventotrappart_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappart_fromtrapneighbour_FROMGAP; }
                           else {seeds2[i][j-1]+= energygiventotrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappartgap_fromtrapneighbour_FROMGAP; }
                        }
                        else if (trapneighbours[i][j-1]==0)
                        {
                           if (i%2==0) {seeds2[i][j-1]+= energygiventonontrappart_fromtrapneighbour_FROMGAP;  spentenergy += energygiventonontrappart_fromtrapneighbour_FROMGAP; }
                           else {seeds2[i][j-1]+= energygiventonontrappartgap_fromtrapneighbour_FROMGAP;  spentenergy += energygiventonontrappartgap_fromtrapneighbour_FROMGAP; }
                        }
                        else {
                              if (i%2==0) {seeds2[i][j-1]+= energygiventotrapneighbour_fromtrapneighbour_FROMGAP;
                              spentenergy += energygiventotrapneighbour_fromtrapneighbour_FROMGAP; }
                              else {seeds2[i][j-1]+= energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP; spentenergy += energygiventotrapneighbour_fromtrapneighbour_FROMGAP; }

                        }
                    }
                    if (inRange(i,j+1) && rectangle[i][j+1] ) { //SOUTH
                        if (traps[i][j+1]==1) {
                           if (i%2==0) {seeds2[i][j+1]+= energygiventotrappart_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappart_fromtrapneighbour_FROMGAP; }
                           else {seeds2[i][j+1]+= energygiventotrappartgap_fromtrapneighbour_FROMGAP;   spentenergy += energygiventotrappartgap_fromtrapneighbour_FROMGAP; }
                        }
                        else if (trapneighbours[i][j+1]==0) {

                           if (i%2==0) {seeds2[i][j+1]+= energygiventonontrappart_fromtrapneighbour_FROMGAP;    spentenergy += energygiventonontrappart_fromtrapneighbour_FROMGAP; }
                        else {seeds2[i][j+1]+= energygiventonontrappartgap_fromtrapneighbour_FROMGAP;    spentenergy += energygiventonontrappartgap_fromtrapneighbour_FROMGAP; }
                        }
                        else {
                              if (i%2==0) {seeds2[i][j+1]+= energygiventotrapneighbour_fromtrapneighbour_FROMGAP;
                                  spentenergy += energygiventotrapneighbour_fromtrapneighbour_FROMGAP; }
                              else{ seeds2[i][j+1]+= energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP;
                                  spentenergy += energygiventotrapneighbourgap_fromtrapneighbour_FROMGAP; }

                        }
                    }
                    seeds2[i][j] += energytospread-spentenergy;
                 }
       }
       for (int i = 0; i < nx; i++) //record details
          for (int j = 0; j < ny; j++)
          {  if (traps[i][j]==0) {
                simonecount += seeds2[i][j]==1?1:0;
                if (restricttobinary) seeds2[i][j] = seeds2[i][j]==1?1:0;
                simseedcount += seeds2[i][j];
             }
             if (traps[i][j]==0&&i%2==0) simseedcountlines += seeds2[i][j];
             if (traps[i][j]==0&&i%2==1) simseedcountgaps += seeds2[i][j];
             if (traps[i][j]==1) simtrapenergy += seeds2[i][j];
             if (traps[i][j]==1&&i%2==0) simtrapenergylines += seeds2[i][j];
             if (traps[i][j]==1&&i%2==1) simtrapenergygaps += seeds2[i][j];
          }
       double[][] temp = seeds;
       seeds = seeds2;
       seeds2 = temp;
   } //end stepsim
void showseedsProcedure()
{  showseeds = !showseeds;
   f.repaint();
   renderframe.repaint();
}
```

```
void showtrapsProcedure()
{   showtraps = !showtraps;
    f.repaint();
    renderframe.repaint();
}
void fillStripeyProcedure()
{   updateSizeVariables();
    updateSundryVariables();
    fillstripey();
    f.repaint();
    renderframe.repaint();
    display.append
    (
        "\n"+
        "RECT WIDTH:\t"+
        "RECT HEIGHT:\t"+
        "NUMBER OF CELLS:\t"+
        "PROB. CELL FILLED:\t"+
        "CELLS FILLED:\t"+
        "PROB. HOLE PUNCHED:\t"+
        "HOLES PUNCHED:\t"+
        "FILLED CELLS LEFT:\n" +
        nx+"\t"+ny+"\t"+nx*ny+"\t"+p+"\t"+particlecount+"\t"+e+"\t"+holecount+"\t"+(particlecount-holecount)+"\n"+
        "PROB. CELL SEEDED:\t"+
        "NUMBER OF SEEDS:\t"+
        "NUMBER OF ONES:\t"+
        "SEEDS IN LINES:\t"+
        "SEEDS IN GAPS:\n"+q+"\t"+seedcount+"\t"+onecount+"\t"+seedcountlines+"\t"+seedcountgaps+"\n"+
        "PROB. CELL TRAPPED:\t"+
        "NUMBER OF TRAPS:\n"+r+"\t"+trapcount+"\n");
}
void fillstripey() {
    calculated = false;
    particlecount = 0;
    holecount = 0;
    seedcount = 0; onecount = 0;
    seedcountlines = 0;
    seedcountgaps = 0;
    simseedcount = 0; simonecount = 0;
    simseedcountlines = 0;
    simseedcountgaps = 0;
    simtrapenergy = 0;
    simtrapenergylines = 0;
    simtrapenergygaps = 0;
    trapcount = 0;
    step = 0;
    for (int i = 0; i < nx; i++)
        for (int j = 0; j < ny; j++) {
                rectangle[i][j] = false;
                seeds[i][j] = 0;
                traps[i][j] = 0;
                trapneighbours[i][j] = 0;
            }
    for (int i = 0; i < nx; i++)
        for (int j = 0; j < ny; j++)
            if (Math.random() < p || ((i % 2) == 0)) { //i is down, j is across
                particlecount++;
                if (Math.random() < e) {
                    holecount++;
                }
                else {
                    rectangle[i][j] = true;
                    if (Math.random() < r)
                    {   traps[i][j] = 1;
                        {   //mark the trap site and the four compass direction adjacents as trap neighbours
                            if (inRange(i,j)) trapneighbours[i][j] = 1;
                            if (inRange(i,j+1)) trapneighbours[i][j+1] = 1;
                            if (inRange(i,j-1)) trapneighbours[i][j-1] = 1;
                            if (inRange(i+1,j)) trapneighbours[i+1][j] = 1;
                            if (inRange(i-1,j)) trapneighbours[i-1][j] = 1;
                        }
                        trapcount++;
                    }
                    else traps[i][j] = 0;
                    if (Math.random() < q && traps[i][j]==0)
                    {   seeds[i][j] = 1;
                        seedcount++; onecount++;
                        if (i%2==0) seedcountlines++;
                        if (i%2==1) seedcountgaps++;
                    }
                    else seeds[i][j] =0;
                }
            }
            else //i.e. nothing here - background
            {
                rectangle[i][j] = false;
                seeds[i][j] = 0;
                traps[i][j] = 0;
            }
}
boolean inRange(int a, int b)
{   if
    (   (0 <= a && a < nx)
        &&
        (0 <= b && b < ny)
    )
    return true; else return false;
}

Font oursmallfont = new Font("Dialog", Font.PLAIN, 10);
Font ourmonospacedsmallfont = new Font("Monospaced", Font.PLAIN, 10);

public JPanel colPan(JComponent j) {
    return colPan(j, new JLabel(""));
}

public JPanel colPan(JComponent j, JComponent k) {
    JPanel cp = new JPanel(new GridLayout(1,2));
    cp.add(j);
    cp.add(k);
    return cp;
}
public Clusters() {
    colorframe = new JFrame("Colours");
    colorframe.setContentPane(new CompositeColorView(this));
    colorframe.pack(); colorframe.setResizable(false);
    colorframe.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    colorframe.setAlwaysOnTop(true);
    colorframe.show();
    renderframe = new JFrame("Rendering");
    renderframe.setResizable(false);
    renderframe.setContentPane(grid);
    grid.setSize(400,400);
    renderframe.pack();
    renderframe.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    renderframe.setAlwaysOnTop(true);
    renderframe.show();
    fillstripeytechnique.addActionListener(this);
    showseedstechnique.addActionListener(this);
    showtrapstechnique.addActionListener(this);
    startsimulationtechnique.addActionListener(this);
    stopsimulationtechnique.addActionListener(this);
    clearthetexttechnique.addActionListener(this);
    savetheimagetechnique.addActionListener(this);
```

```
reseedtechnique.addActionListener(this);
reseedalltechnique.addActionListener(this);
spreadaccordingtoneighbourstechnique.addItemListener(this);
spreadaccordingtoneighbourstechnique.setFont(oursmallfont);
restricttobinarytechnique.addItemListener(this);
restricttobinarytechnique.setFont(oursmallfont);
display.setFont(ourmonospacedsmallfont);
display.setTabSize(25);
JScrollPane s = new JScrollPane(display);
JPanel keypanel = new JPanel(new GridLayout(16,4));
JPanel displaypanel = new JPanel(new GridLayout(1,2));
displaypanel.add(s);
keypanel.add(setLabelFont(new JLabel("Rectangle width (width x height < 1,000,000,000):")));
keypanel.add(nxtechnique);
keypanel.add(setLabelFont(new JLabel("Rectangle height (width x height < 1,000,000,000):")));
keypanel.add(nytechnique);
keypanel.add(setLabelFont(new JLabel("Probability of cell being filled (0-1):")));
keypanel.add(ptechnique);
keypanel.add(setLabelFont(new JLabel("Probability of hole being punched (0-1):")));
keypanel.add(etechnique);
keypanel.add(setLabelFont(new JLabel("Probability of cell being seeded (0-1):")));
keypanel.add(qtechnique);
keypanel.add(setLabelFont(new JLabel("Probability of cell being trapped (0-1):")));
keypanel.add(rtechnique);
keypanel.add(setLabelFont(new JLabel("Simulation length (steps) (1-1,000,000,000):")));
keypanel.add(simulationlengthtechnique);
keypanel.add(setLabelFont(new JLabel("Display every x steps (1-1,000,000,000):")));
keypanel.add(displayeveryxstepstechnique);
keypanel.add(setLabelFont(new JLabel("Redistribute to centre fraction: (0-1)")));
keypanel.add(redistributetocentrefactortechnique);
keypanel.add(setLabelFont(new JLabel("")));keypanel.add(restricttobinarytechnique);
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(colPan(setLabelFont(new JLabel("SOURCE:")),setLabelFont(new JLabel("SOURCE:"))));
keypanel.add(colPan(setLabelFont(new JLabel("SOURCE:")),setLabelFont(new JLabel("SOURCE:"))));
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(colPan(setLabelFont(new JLabel("site in line")),setLabelFont(new JLabel("site in gap"))));
keypanel.add(colPan(setLabelFont(new JLabel("trap neigh. in line")),setLabelFont(new JLabel("trap neigh. in gap"))));
keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
keypanel.add(setLabelFont(new JLabel("trap in line")));
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(
   colPan(
      fraction_given_to_trap_part_from_trap_neighbour_TECHNIQUE,
      fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE
   )
);
fraction_given_to_trap_part_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_trap_part_from_trap_neighbour_");
fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_");
keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
keypanel.add(setLabelFont(new JLabel("trap in gap")));
keypanel.add(setLabelFont(new JLabel("")));
keypanel.add(
   colPan(
      fraction_given_to_trap_part_gap_from_trap_neighbour_TECHNIQUE,
      fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE
   )
);
fraction_given_to_trap_part_gap_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_trap_part_gap_from_trap_neighbour_");
fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_");
keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
keypanel.add(setLabelFont(new JLabel("site in line")));
keypanel.add(colPan(fraction_given_to_non_trap_part_TECHNIQUE,
fraction_given_to_non_trap_part_FROMGAP_TECHNIQUE));
fraction_given_to_non_trap_part_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_");
fraction_given_to_non_trap_part_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_FROMGAP_");
```

```
    keypanel.add(
       colPan(
          fraction_given_to_non_trap_part_from_trap_neighbour_TECHNIQUE,
          fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE
       )
    );
    fraction_given_to_non_trap_part_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_from_trap_neighbour_");
    fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_");
    keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
    keypanel.add(setLabelFont(new JLabel("site in gap")));

    keypanel.add(colPan(fraction_given_to_non_trap_part_gap_TECHNIQUE,
       fraction_given_to_non_trap_part_gap_FROMGAP_TECHNIQUE

    ));
    fraction_given_to_non_trap_part_gap_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_gap_");
    fraction_given_to_non_trap_part_gap_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_gap_FROMGAP_");
    keypanel.add(colPan(fraction_given_to_non_trap_part_gap_from_trap_neighbour_TECHNIQUE,
       fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE
    ));
    fraction_given_to_non_trap_part_gap_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_gap_from_trap_neighbour_");
    fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_");
    keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
    keypanel.add(setLabelFont(new JLabel("trap neigh. in line")));
    keypanel.add(colPan(fraction_given_to_trap_neighbour_TECHNIQUE,
       fraction_given_to_trap_neighbour_FROMGAP_TECHNIQUE
    ));
    fraction_given_to_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_");
    fraction_given_to_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_FROMGAP_");
    keypanel.add(colPan(fraction_given_to_trap_neighbour_from_trap_neighbour_TECHNIQUE,
    fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TECHNIQUE));
    fraction_given_to_trap_neighbour_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_from_trap_neighbour_");
    fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_");
    keypanel.add(setLabelFont(new JLabel("DESTINATION:")));
    keypanel.add(setLabelFont(new JLabel("trap neigh. in gap")));
    keypanel.add(colPan(fraction_given_to_trap_neighbour_gap_TECHNIQUE,
    fraction_given_to_trap_neighbour_gap_FROMGAP_TECHNIQUE
    ));
    fraction_given_to_trap_neighbour_gap_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_gap_");
    fraction_given_to_trap_neighbour_gap_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_gap_FROMGAP_");
    keypanel.add(colPan(fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TECHNIQUE,
    fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TECHNIQUE));
    fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_gap_from_trap_neighbour_");
    fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setToolTipText("fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_");
    keypanel.add(fillstripeytechnique);
    keypanel.add(startsimulationtechnique);
    keypanel.add(stopsimulationtechnique);
    keypanel.add(showseedstechnique);
    keypanel.add(showtrapstechnique);
    keypanel.add(reseedtechnique);
    keypanel.add(reseedalltechnique);
    keypanel.add(clearthetexttechnique);
    keypanel.add(savetheimagetechnique);
    JPanel p = new JPanel(new BorderLayout());
    p.add(displaypanel, BorderLayout.CENTER);
    p.add(keypanel, BorderLayout.PAGE_END);
    f.getContentPane().add(p);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(800,600);
    f.show();
 }
 JFrame colorframe;
 JFrame renderframe;
 public JLabel setLabelFont(JLabel jl) {
    jl.setFont(oursmallfont);
    return jl;
 }
```

```
private void updateSizeVariables()
{   int nxtemp = 0, nytemp = 0;
    try
    {   nxtemp = Integer.parseInt(nxtechnique.getText());
    }
    catch (Exception e) {};
    try
    {   nytemp = Integer.parseInt(nytechnique.getText());
    }
    catch (Exception e) {};
    if
    (   0 < nxtemp && 0 < nytemp
        &&
        nxtemp*nytemp <= 1000000000//a billion
        &&
        (nxtemp != nx || nytemp !=ny)
    )
    {
        nx = nxtemp; ny = nytemp;
        rectangle = new boolean[nx][ny];
        seeds = new double[nx][ny];
        seeds2 = new double[nx][ny];
        traps = new int[nx][ny];
        trapneighbours = new int[nx][ny];
    }
}
void updateTextFields()
{   nxtechnique.setText(""+nx);
    nytechnique.setText(""+ny);
    ptechnique.setText(""+p);
    etechnique.setText(""+e);
    qtechnique.setText(""+q);
    rtechnique.setText(""+r);
    decayfactortechnique.setText(""+decayfactor);
    redistributetocentrefactortechnique.setText(""+redistributetocentrefactor);
    simulationlengthtechnique.setText(""+simulationlength);
    displayeveryxstepstechnique.setText(""+displayeveryxsteps);
    percentagetospreadtechnique.setText(""+percentagetospread);
    spreadaccordingtoneighbourstechnique.setSelected(spreadaccordingtoneighbours);
    restricttobinarytechnique.setSelected(restricttobinary);
    fraction_given_to_non_trap_part_TECHNIQUE.setText(""+fraction_given_to_non_trap_part);
    fraction_given_to_non_trap_part_gap_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_gap);
    fraction_given_to_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour);
    fraction_given_to_trap_neighbour_gap_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_gap);
    fraction_given_to_non_trap_part_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_FROMGAP);
    fraction_given_to_non_trap_part_gap_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_gap_FROMGAP);
    fraction_given_to_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_FROMGAP);
    fraction_given_to_trap_neighbour_gap_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_gap_FROMGAP);
    fraction_given_to_trap_part_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_trap_part_from_trap_neighbour);
    fraction_given_to_trap_part_gap_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_trap_part_gap_from_trap_neighbour);
    fraction_given_to_non_trap_part_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_from_trap_neighbour);
    fraction_given_to_non_trap_part_gap_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_gap_from_trap_neighbour);
    fraction_given_to_trap_neighbour_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_from_trap_neighbour);
    fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_gap_from_trap_neighbour);
    fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_part_from_trap_neighbour_FROMGAP);
    fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP);
    fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP);
    fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP);
    fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP);
    fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.setText(""+fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP);
}
void updateSundryVariables() {
    try
    {   double ptemp = Double.parseDouble(ptechnique.getText());
        if ( 0 <= ptemp && ptemp <= 1 && p!=ptemp) p = ptemp;
    }
    catch (Exception e) {};
```

```
try
{   double etemp = Double.parseDouble(etechnique.getText());
    if ( 0 <= etemp && etemp <= 1 && e!=etemp) e = etemp;
}
catch (Exception e) {};
try
{   double qtemp = Double.parseDouble(qtechnique.getText());
    if ( 0 <= qtemp && qtemp <= 1 && q!=qtemp) q = qtemp;
} catch (Exception e) {}
try
{   double rtemp = Double.parseDouble(rtechnique.getText());
    if ( 0 <= rtemp && rtemp <= 1 && r!=rtemp) r = rtemp;
} catch (Exception e) {}
try
{   double decayfactortemp = Double.parseDouble(decayfactortechnique.getText());
    if ( 0 <= decayfactortemp && decayfactortemp <= 1 && decayfactor!=decayfactortemp) decayfactor = decayfactortemp;
} catch (Exception e) {}
try
{   double redistributetocentrefactortemp = Double.parseDouble(redistributetocentrefactortechnique.getText());
    if ( 0 <= redistributetocentrefactortemp && redistributetocentrefactortemp <= 1 && redistributetocentrefactor!=redistributetocentrefactortemp)
        redistributetocentrefactor = redistributetocentrefactortemp;
} catch (Exception e) {}
try
{   int stemp = Integer.parseInt(simulationlengthtechnique.getText());
    if ( 0 < stemp && stemp <= 1000000000 && simulationlength!=stemp)
    {
        simulationlength = stemp;
    }
}
catch (Exception e) {};
try
{   int dtemp = Integer.parseInt(displayeveryxstepstechnique.getText());
    if ( 0 < dtemp && dtemp <= 1000000000 && displayeveryxsteps!=dtemp)
    {
        displayeveryxsteps = dtemp;
    }
}
catch (Exception e) {};
try
{   double percentagetospreadtemp = Double.parseDouble(percentagetospreadtechnique.getText());
    if ( 0 <= percentagetospreadtemp && percentagetospreadtemp <= 100 && percentagetospread!=percentagetospreadtemp) percentagetospread = percentagetospreadtemp;
} catch (Exception e) {}
try
{
    double fraction_given_to_non_trap_part_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_TECHNIQUE.getText());
    double fraction_given_to_non_trap_part_gap_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_gap_TECHNIQUE.getText());
    double fraction_given_to_trap_neighbour_TEMP = Double.parseDouble(         fraction_given_to_trap_neighbour_TECHNIQUE.getText());
    double fraction_given_to_trap_neighbour_gap_TEMP = Double.parseDouble(         fraction_given_to_trap_neighbour_gap_TECHNIQUE.getText());
    double fraction_given_to_non_trap_part_FROMGAP_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_FROMGAP_TECHNIQUE.getText());
    double fraction_given_to_non_trap_part_gap_FROMGAP_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_gap_FROMGAP_TECHNIQUE.getText());
    double fraction_given_to_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(         fraction_given_to_trap_neighbour_FROMGAP_TECHNIQUE.getText());
    double fraction_given_to_trap_neighbour_gap_FROMGAP_TEMP = Double.parseDouble(         fraction_given_to_trap_neighbour_gap_FROMGAP_TECHNIQUE.getText());
    if (
    ( 0 <= fraction_given_to_non_trap_part_TEMP && fraction_given_to_non_trap_part_TEMP <= 0.25 )&&
    ( 0 <= fraction_given_to_non_trap_part_gap_TEMP && fraction_given_to_non_trap_part_gap_TEMP <= 0.25 )&&
    ( 0 <= fraction_given_to_trap_neighbour_TEMP && fraction_given_to_trap_neighbour_TEMP <= 0.25 ) &&
    ( 0 <= fraction_given_to_trap_neighbour_gap_TEMP && fraction_given_to_trap_neighbour_gap_TEMP <= 0.25 ) &&
    ( 0 <= fraction_given_to_non_trap_part_FROMGAP_TEMP && fraction_given_to_non_trap_part_FROMGAP_TEMP <= 0.25 )&&
    ( 0 <= fraction_given_to_non_trap_part_gap_FROMGAP_TEMP && fraction_given_to_non_trap_part_gap_FROMGAP_TEMP <= 0.25 )&&
    ( 0 <= fraction_given_to_trap_neighbour_FROMGAP_TEMP && fraction_given_to_trap_neighbour_FROMGAP_TEMP <= 0.25 ) &&
    ( 0 <= fraction_given_to_trap_neighbour_gap_FROMGAP_TEMP && fraction_given_to_trap_neighbour_gap_FROMGAP_TEMP <= 0.25 )
    ){
    fraction_given_to_non_trap_part= fraction_given_to_non_trap_part_TEMP ;
    fraction_given_to_non_trap_part_gap= fraction_given_to_non_trap_part_gap_TEMP ;
    fraction_given_to_trap_neighbour= fraction_given_to_trap_neighbour_TEMP ;
    fraction_given_to_trap_neighbour_gap= fraction_given_to_trap_neighbour_gap_TEMP ;
    fraction_given_to_non_trap_part_FROMGAP= fraction_given_to_non_trap_part_FROMGAP_TEMP ;
```

```
          fraction_given_to_non_trap_part_gap_FROMGAP= fraction_given_to_non_trap_part_gap_FROMGAP_TEMP ;
          fraction_given_to_trap_neighbour_FROMGAP= fraction_given_to_trap_neighbour_FROMGAP_TEMP ;
          fraction_given_to_trap_neighbour_gap_FROMGAP= fraction_given_to_trap_neighbour_gap_FROMGAP_TEMP ;
          }
       } catch (Exception e) {}
       try {
          double fraction_given_to_trap_part_from_trap_neighbour_TEMP = Double.parseDouble(          fraction_given_to_trap_part_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_trap_part_gap_from_trap_neighbour_TEMP = Double.parseDouble(           fraction_given_to_trap_part_gap_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_non_trap_part_from_trap_neighbour_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_non_trap_part_gap_from_trap_neighbour_TEMP = Double.parseDouble(          fraction_given_to_non_trap_part_gap_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_trap_neighbour_from_trap_neighbour_TEMP = Double.parseDouble(fraction_given_to_trap_neighbour_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TEMP = Double.parseDouble(fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TECHNIQUE.getText());
          double fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(          fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          double fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(
             fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          double fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(
             fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          double fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(
             fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          double fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TEMP = Double.parseDouble(fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          double fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TEMP =
             Double.parseDouble(fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TECHNIQUE.getText());
          if (
          ( 0 <= fraction_given_to_trap_part_from_trap_neighbour_TEMP && fraction_given_to_trap_part_from_trap_neighbour_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_trap_part_gap_from_trap_neighbour_TEMP && fraction_given_to_trap_part_gap_from_trap_neighbour_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_non_trap_part_from_trap_neighbour_TEMP && fraction_given_to_non_trap_part_from_trap_neighbour_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_non_trap_part_gap_from_trap_neighbour_TEMP && fraction_given_to_non_trap_part_gap_from_trap_neighbour_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_trap_neighbour_from_trap_neighbour_TEMP && fraction_given_to_trap_neighbour_from_trap_neighbour_TEMP <= 0.25 ) &&
          ( 0 <= fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TEMP && fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TEMP <= 0.25 )
          &&
          ( 0 <= fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP <= 0.25 )&&
          ( 0 <= fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TEMP <= 0.25 ) &&
          ( 0 <= fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TEMP && fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TEMP <= 0.25 )
          ){
          fraction_given_to_trap_part_from_trap_neighbour = fraction_given_to_trap_part_from_trap_neighbour_TEMP;
          fraction_given_to_trap_part_gap_from_trap_neighbour= fraction_given_to_trap_part_gap_from_trap_neighbour_TEMP ;
          fraction_given_to_non_trap_part_from_trap_neighbour= fraction_given_to_non_trap_part_from_trap_neighbour_TEMP ;
          fraction_given_to_non_trap_part_gap_from_trap_neighbour= fraction_given_to_non_trap_part_gap_from_trap_neighbour_TEMP ;
          fraction_given_to_trap_neighbour_from_trap_neighbour= fraction_given_to_trap_neighbour_from_trap_neighbour_TEMP ;
          fraction_given_to_trap_neighbour_gap_from_trap_neighbour= fraction_given_to_trap_neighbour_gap_from_trap_neighbour_TEMP ;
          fraction_given_to_trap_part_from_trap_neighbour_FROMGAP = fraction_given_to_trap_part_from_trap_neighbour_FROMGAP_TEMP;
          fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP= fraction_given_to_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP ;
          fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP= fraction_given_to_non_trap_part_from_trap_neighbour_FROMGAP_TEMP ;
          fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP= fraction_given_to_non_trap_part_gap_from_trap_neighbour_FROMGAP_TEMP ;
          fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP= fraction_given_to_trap_neighbour_from_trap_neighbour_FROMGAP_TEMP ;
          fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP= fraction_given_to_trap_neighbour_gap_from_trap_neighbour_FROMGAP_TEMP ;
          }
       } catch (Exception e) {}
   }
   public static void main(String[] args)
   {  Clusters f = new Clusters();
   }
}
```

ColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.colorchooser.AbstractColorChooserPanel;
public abstract class ColorView extends JColorChooser implements ChangeListener {
    Object page;
    public ColorView(Clusters page)
    {  this.page = page;
       setPreviewPanel(new JPanel());
       AbstractColorChooserPanel[] cp = this.getChooserPanels();
       AbstractColorChooserPanel[] cp2 = new AbstractColorChooserPanel[cp.length-1];
       for (int i = 0; i < cp2.length; i++)
          cp2[i] = cp[i+1];
       this.setChooserPanels(cp2);
       getSelectionModel().addChangeListener(this);
       refresh();
    }
    public abstract void set();
    public abstract void get();
    public void stateChanged(ChangeEvent e)
    {
       set();
    }
    private void refresh()
    {
       get();
    }
}
```

CompositeColorView.java

```
import javax.swing.*;
public class CompositeColorView extends JTabbedPane
{
   public CompositeColorView(Clusters page)
   {  super(LEFT);
      this.addTab("Seeds", null, new StartColorView(page), "gradient start color");
      this.addTab("Trap Neighbour Seeds", null, new TrapNeighbourStartColorView(page), "gradient start color");
      this.addTab("Traps", null, new EndColorView(page), "gradient end color");
      this.addTab("Particles", null, new BorderColorView(page), "border color");
      this.addTab("Trap Neighbour Particles", null, new TrapNeighbourBorderColorView(page), "border color");
      this.addTab("Background", null, new BackgroundColorView(page), "background color");
   }
}
```

EndColorView.java

```
public class EndColorView extends ColorView {
   public void set(){
      ((Clusters)page).setEndColor(getColor());
   };
   public void get(){
      setColor(((Clusters)page).getEndColor());
   };
   public EndColorView(Clusters page)
   {  super(page); }
}
```

PageListener.java

```
public interface PageListener {
   public void modelChanged();
}
```

StartColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
public class StartColorView extends ColorView implements ChangeListener {
   public void set(){
      ((Clusters)page).setStartColor(getColor());
   };
   public void get(){
      setColor(((Clusters)page).getStartColor());
   };
   public StartColorView(Clusters page)
   {   super(page); }
}
```

SwingWorker.java – Author: http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html

```
import javax.swing.SwingUtilities;

/**
 * This is the 3rd version of SwingWorker (also known as
 * SwingWorker 3), an abstract class that you subclass to
 * perform GUI-related work in a dedicated thread.  For
 * instructions on using this class, see:
 *
 * http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html
 *
 * Note that the API changed slightly in the 3rd version:
 * You must now invoke start() on the SwingWorker after
 * creating it.
 */
public abstract class SwingWorker {
    private Object value;  // see getValue(), setValue()
    private Thread thread;

    /**
     * Class to maintain reference to current worker thread
     * under separate synchronization control.
     */
    private static class ThreadVar {
        private Thread thread;
        ThreadVar(Thread t) { thread = t; }
        synchronized Thread get() { return thread; }
        synchronized void clear() { thread = null; }
    }

    private ThreadVar threadVar;

    /**
     * Get the value produced by the worker thread, or null if it
     * hasn't been constructed yet.
     */
    protected synchronized Object getValue() {
        return value;
    }

    /**
     * Set the value produced by worker thread
     */
    private synchronized void setValue(Object x) {
        value = x;
```

```java
}

/**
 * Compute the value to be returned by the <code>get</code> method.
 */
public abstract Object construct();

/**
 * Called on the event dispatching thread (not on the worker thread)
 * after the <code>construct</code> method has returned.
 */
public void finished() {
}

/**
 * A new method that interrupts the worker thread.  Call this method
 * to force the worker to stop what it's doing.
 */
public void interrupt() {
    Thread t = threadVar.get();
    if (t != null) {
        t.interrupt();
    }
    threadVar.clear();
}

/**
 * Return the value created by the <code>construct</code> method.
 * Returns null if either the constructing thread or the current
 * thread was interrupted before a value was produced.
 *
 * @return the value created by the <code>construct</code> method
 */
public Object get() {
    while (true) {
        Thread t = threadVar.get();
        if (t == null) {
            return getValue();
        }
        try {
            t.join();
        }
        catch (InterruptedException e) {
            Thread.currentThread().interrupt(); // propagate
            return null;
        }
    }
}


/**
 * Start a thread that will call the <code>construct</code> method
 * and then exit.
 */
public SwingWorker() {
    final Runnable doFinished = new Runnable() {
       public void run() { finished(); }
    };

    Runnable doConstruct = new Runnable() {
        public void run() {
            try {
                setValue(construct());
            }
            finally {
                threadVar.clear();
            }
```

```
                SwingUtilities.invokeLater(doFinished);
            }
        };

        Thread t = new Thread(doConstruct);
        threadVar = new ThreadVar(t);
    }

    /**
     * Start the worker thread.
     */
    public void start() {
        Thread t = threadVar.get();
        if (t != null) {
            t.start();
        }
    }
}
```

TrapNeighbourBorderColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
public class TrapNeighbourBorderColorView extends ColorView implements ChangeListener {
   public void set(){
      ((Clusters)page).setTrapNeighbourBorderColor(getColor());
   };
   public void get(){
      setColor(((Clusters)page).getTrapNeighbourBorderColor());
   };
   public TrapNeighbourBorderColorView(Clusters page)
   {   super(page); }
}
```

TrapNeighbourStartColorView.java

```
import java.awt.Font;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;
public class TrapNeighbourStartColorView extends ColorView implements ChangeListener {
   public void set(){
      ((Clusters)page).setTrapNeighbourStartColor(getColor());
   };
   public void get(){
      setColor(((Clusters)page).getTrapNeighbourStartColor());
   };
   public TrapNeighbourStartColorView(Clusters page)
   {   super(page); }
}
```

Notes

The main class is Clusters. It takes no parameters.