

《管理系统模拟》 课程实验报告

报告书名称：实验 5-2：热虫模拟实验

报告书作者：杨诺彤 郑雅璇

完成日期：2020 年 12 月 27 日

版 本：版本 1

审核签字：金 淳

大连理工大学 经济管理学院 系统工程研究所
金淳
2020 年 12 月

目 录

1	实验原理.....	1
1.1	模型描述.....	1
1.2	参数说明.....	1
2	程序总体说明.....	2
2.1	程序构成.....	2
2.2	程序运行环境.....	2
2.3	程序功能模块.....	2
3	程序设计说明.....	4
3.1	程序描述.....	4
3.1.1	类构成.....	4
3.1.2	类 HeatBugView 基本设计	4
3.1.3	类 HeatBug 基本设计.....	5
3.2	程序功能.....	5
3.3	输入项.....	6
3.4	输出项.....	6
3.5	流程逻辑.....	6
4	其他需要说明的问题.....	7
4.1	本次开发未考虑的因素.....	7
4.2	尚未解决的问题.....	7
4.3	今后的功能追加工作.....	7
5	源程序代码.....	8
5.1	源程序文件.....	8
5.2	源程序规模.....	8
5.3	源程序.....	8
5.3.1	HeatBugModel.java	8
5.3.2	HeatBugView.java.....	11
6	学习收获.....	20

1 实验原理

1.1 模型描述

在被称作“热虫世界”的系统中，初始状态下随机分布着一定数量的热虫（Heatbug）。每一个热虫都能输出少量的热能，并且都有一个适宜自己生存的理想温度，处于理想温度时它会觉得 happy。热虫在释放热量、影响周围环境温度的同时，也在向着更接近于自己理想温度的附近的点不断移动。

一个热虫单凭自身是难以获得足够热量继续生存，因此它们将趋向于聚集在一起来相互取暖。当热虫数量庞大时，整个系统运动相当复杂。热虫系统可看作是一个最优化问题：每个热虫都努力地最小化自己的 Unhappiness。

在本实验中，我们建立了相对比较简单规则：初始时，整个热虫世界温度为 0，随机生成热虫后，每个热虫的理想温度和释放热量均在给定的最小最大范围中间随机生成；热虫向上下左右四个方向的四个点散热，并在四个相邻的点中选取温度与其自身理想温度最接近的点移动；每个格子每次都通过蒸发流失一部分热量；热虫移动到新的点继续向四个方向散热。

1.2 参数说明

numBugs: 热虫数量

minIdealTemp: 最低理想温度

maxIdealTemp: 最高理想温度

idealTemp: 理想温度

minOutputHeat: 最小释放热量

maxOutputHeat: 最大释放热量

outputHeat: 释放热量

x: 热虫位置横坐标

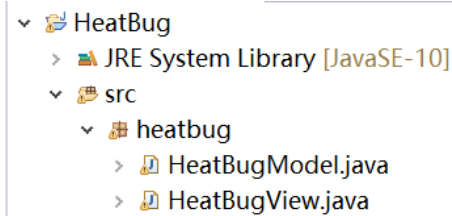
y: 热虫位置纵坐标

unhappiness: 热虫的不高兴程度

evaporation: 系统散热

2 程序总体说明

2.1 程序构成



项目名称HeatBug，heatbug包中含有两个文件：HeatBugModel.java和HeatBugView.java，其中后者包含与程序窗体相关的代码以及程序总入口。

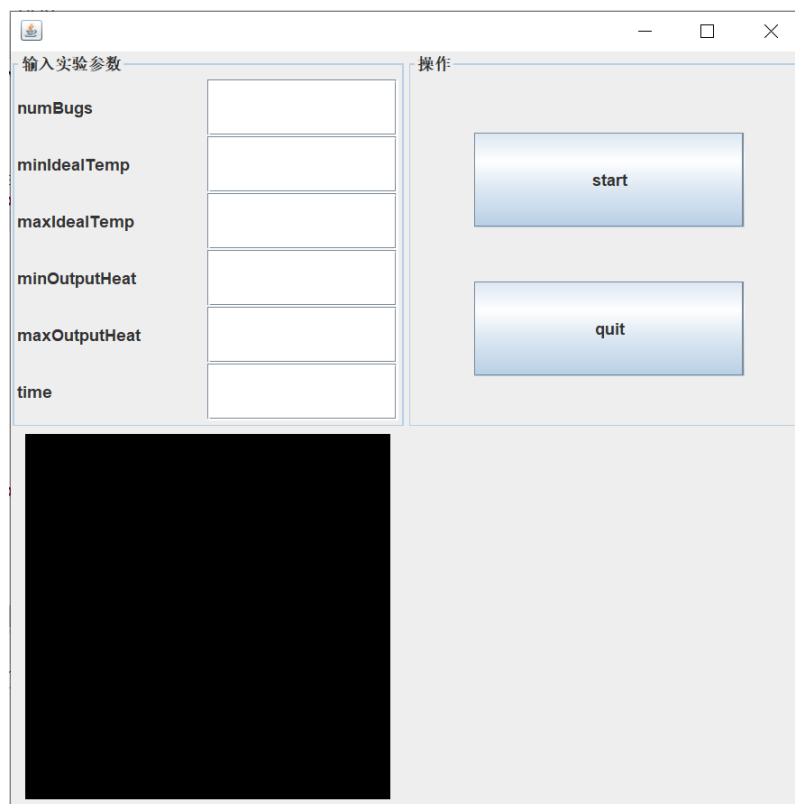
2.2 程序运行环境

参见实验系统总体说明。

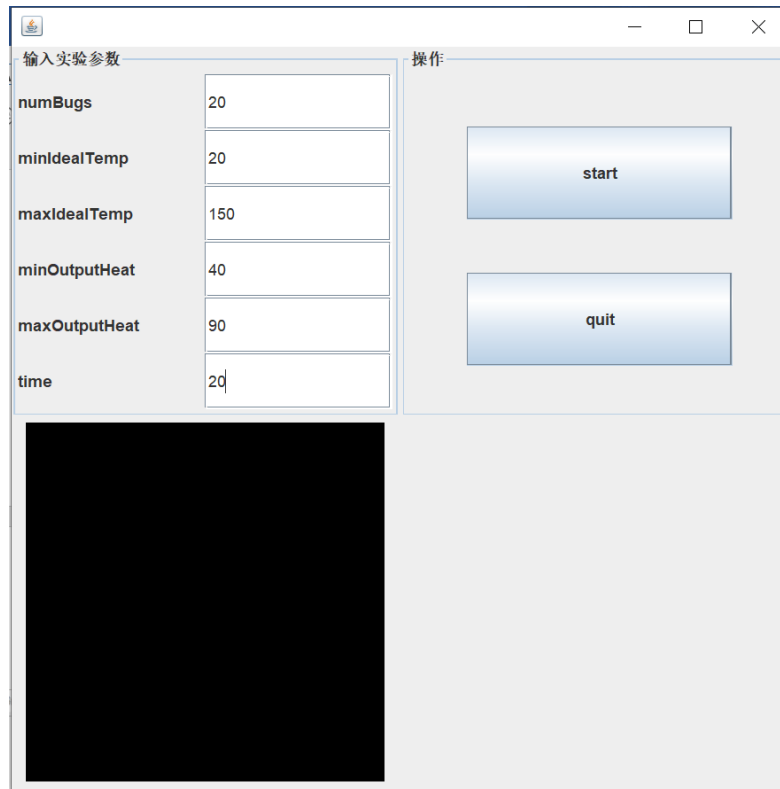
2.3 程序功能模块

程序包含四个功能模块：输入实验参数、实验操作、光栅图、unhappiness 变化图。

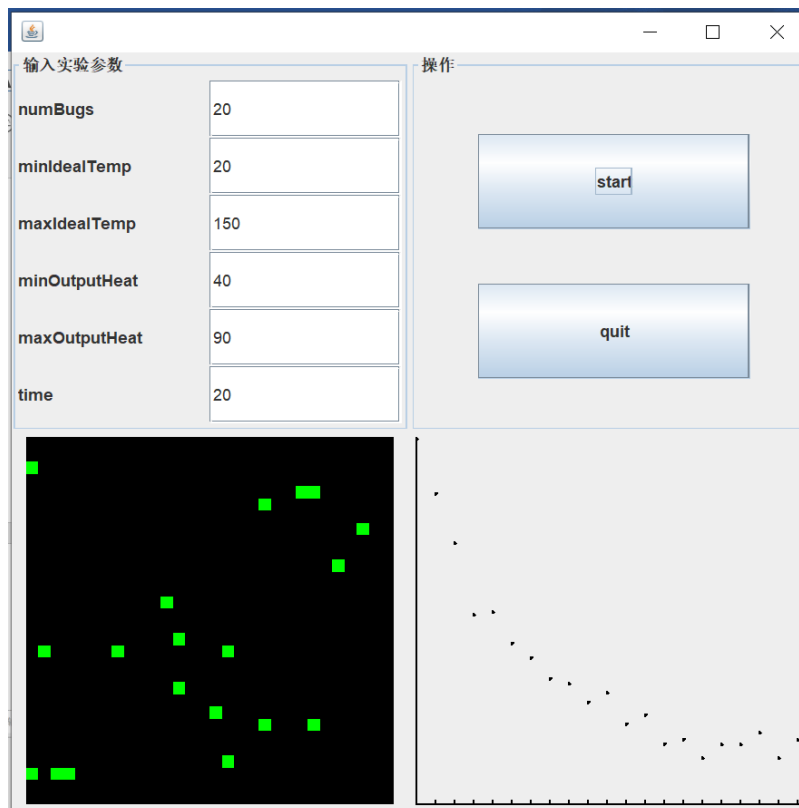
(1) 运行界面



(2) 输入实验参数



(3) 点击“start”，程序开始运行，运行结束后如下



开始运行后，根据用户输入的热虫数量在光栅图内随机生成热虫（绿色方格），随着热虫的运动，绿色方格改变位置，右下角的 unhappiness 图中显示 unhappiness 值整体趋势不断下降。

3 程序设计说明

3.1 程序描述

3.1.1 类构成

本程序包含的所有类如下表所示。

类名	说明
HeatBugView	启动程序，实例化并显示窗口对象
HeatBug	热虫类，封装了热虫运动和散热等函数

以下分别介绍。

3.1.2 类HeatBugView基本设计

(1) 变量设计：

由于变量较多，因此以下只罗列了主要的变量及功能

变量名称	变量类型	说明
frame	Jframe	程序窗体
panel1/2/3/4	Jpanel	面板 1 使用户输入参数；面板 2 使用户控制开始和停止；面板 3 显示热虫光栅图即运动过程；面板 4 实时显示虫的 unhappiness 值
minIdealTemp/maxIdealTemp 最小/最大适宜温度	int	每个虫在这两个变量确定的上下限之间，随机生成一个适宜温度，并保持不变
minOutputHeat/maxOutputHeat 最小/最大散热	int	每个虫在这两个变量确定的上下限之间，随机生成一个散热温度，并保持不变。单位默认，与适宜温度单位相同
time 热虫运动次数	int	用户可以在 panel1 中调整
heatbugs 热虫数组	HeatBug[]	每个元素代表一个虫，完成相应的功能
numBugs 热虫数量	int	用户可以调
temp 格子温度	int[][]	格子的温度会随着附近热虫的运动而受到影响（虫会散热），反过来也会影响它附近的虫的运动
labels 格子	Jlabel[][]	Panel3xianshi 虫的运动过程，它实质上由 30*30 的 label 格子填充。对于一个格子，当有虫在该格子所在位置，则格子显示绿色；若没有虫子则默认显示黑色。
evaporation 蒸发散热	int	每个格子每次通过蒸发流失的

(2) 方法与接口：

方法名称	返回值类型	说明
------	-------	----

public HeatBugView()	Null	构造函数，变量声明和实例化
public init()	Null	初始化函数，初始化光栅图 panel3
public void actionPerformed(ActionEvent e)	void	监听事件触发动作，包括在 panel3 显示虫的运动过程，panel4 显示 unhappiness 变化
public double getOverallUnhappiness()	double	获得所有虫的总体 unhappiness 值
public void drawLineXY(JPanel panel)	void	画出 panel4 中的坐标轴
public void drawXScale(JPanel panel, int time)	void	画出 panel4 中图像横轴刻度
public double getInitialUnhappiness()	double	获取最初的 Unhappiness 值
public void drawPoint(JPanel panel,int[][] temp,int i)	void	在 panel4 的图像中画点
public static void main(String args[])	void	函数入口，类实例化，调用初始化函数

3.1.3 类HeatBug基本设计

(1) 变量设计

变量名称	类型	说明
idealTemp	int	虫的理想温度，每个虫最初会随机生成，后保持不变
outputHeat	int	虫的散热量，每个虫最初会随机生成，后保持不变
unhappiness	int	虫的 unhappiness 值，每个虫最初会随机生成，后保持不变。表示虫的理想温度和其实际所在格子的温度的差值，则差值越小，虫越满意，这个值越小。
x/y	int	虫的坐标，即所在 label 的位置，随每次运动可能有所更新。

(2) 方法与接口：

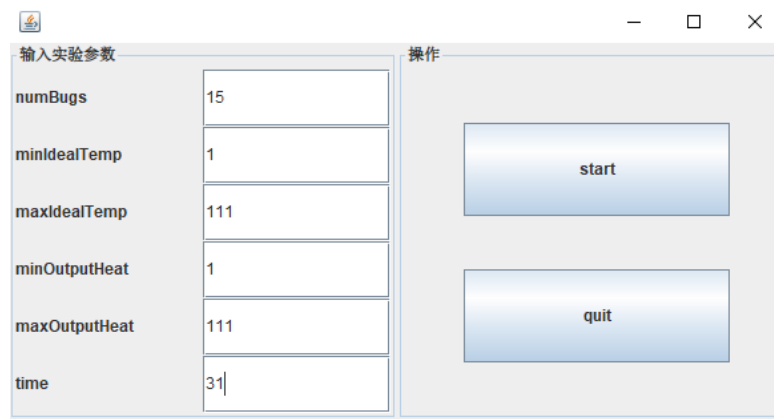
方法名称	返回值类型	说明
public void getIdealTemp(int minIT,int maxIT)	void	获得虫的理想温度
public void getOutputHeat(int minOH,int maxOH)	void	获得虫的散热
public int getUnhappiness(int[][] temp)	int	获得当前虫对象的 unhappiness
public int move(int[][] temp)	int	虫的运动函数。根据附近格子的温度，虫选择趋向接近其适宜温度的地方。返回值 int 代指虫的运动方向
public void spreadHeat(int[][] temp)	void	散热。一般而言虫会向其四周的 label 散热。但如果虫本身处于边界，就向边界内可能的 label 散热。实际程序运行中，每一次都是虫先向四周散热，后根据四周温度情况运动。

3.2 程序功能

参见2.3节的说明。

3.3 输入项

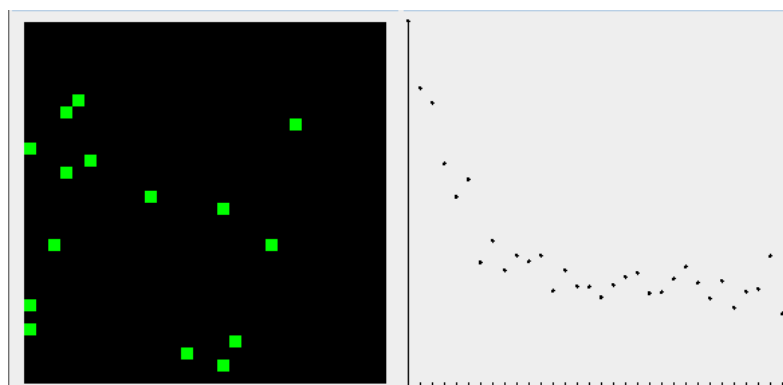
用户输入数据如下：



在 panel1 中，从上向下依次是：热虫数量，最低理想温度，最高理想温度，最小散热，最大散热，运动次数。

3.4 输出项

输出包括两个图：光栅图和unhappiness变化图。



3.5 流程逻辑



4 其他需要说明的问题

4.1 本次开发未考虑的因素

由于时间关系，以下因素在本次暂不考虑：

- (1) 热虫传播热量的速率`diffusion_rate`
- (2) 热虫只向上下左右四个方向散热、移动
- (3) 初始状态下随机生成热虫时、热虫移动时均不考虑可能重叠在同一位置的情况

4.2 尚未解决的问题

- (1) `unhappiness` 曲线图纵轴坐标无法具体显示
- (2) 有时候`unhappiness`在程序运行后半段会不降反升，主要原因是系统散热以及热虫释放热量的规则有待完善。

4.3 今后的功能追加工作

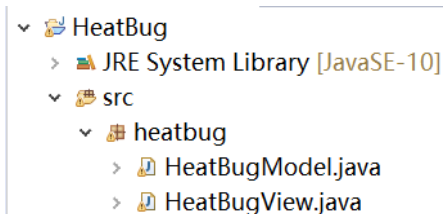
下一步的功能追加建议：

- 1、虫子向周围更多、更远的格子散发热量，且热量用颜色表示出来，较近的格子热量颜色深，教员的格子热量颜色浅
- 2、允许用户输入 `diffusion_rate`、`evaporation_rate` 等参数
- 3、允许用户输入热虫世界的大小（边的长度），动态生成指定大小的光栅图
- 4、避免热虫可能重叠于同一格子的情况
- 5、`Unhappiness` 曲线图纵轴坐标显示具体数值
- 6、各个参数给定范围，用户输入数据超出范围使给出提示
- 7、增加“暂停”按钮
- 8、增加“数值结果”panel，显示运行时间、运行后虫子 `unhappiness` 及其他数值结果

5 源程序代码

5.1 源程序文件

源程序文件有 2 个，列表如下。



5.2 源程序规模

文件	规模（行数）
HeatBugModel.java	132
HeatBugView.java	281

5.3 源程序

5.3.1 HeatBugModel.java

```
package heatbug;

import java.util.*;
import java.math.*;

//热虫类
class HeatBug{
    Random rdm=new Random();
    public int x; //x坐标
    public int y; //y坐标
    public int idealTemp; //理想温度
    public int outputHeat; //释放热量
    public int unhappiness; //一个虫子的unhappiness值

    //获得理想温度
    public void getIdealTemp(int minIT,int maxIT) {
        idealTemp=rdm.nextInt(maxIT-minIT)+minIT; //在用户输入的最小、最大理想温度范围中随机生成
    }

    //获得释放热量
    public void getOutputHeat(int minOH,int maxOH) {
```

```

        outputHeat=rdm.nextInt(maxOH-minOH)+minOH; //在用户输入的最小、最大散发热量范围中随机生成
    }

    //获得unhappiness值
    public int getUnhappiness(int[][] temp) {
        return Math.abs(this.idealTemp-temp[this.x][this.y]); //unhappiness=理想温度与所在格子温度差值的绝对值
    }

    //运动函数，返回运动方向
    public int move(int[][] temp) {
        int nearbyTemp[]=new int[4]; //存放虫子四周（上下左右）温度的数组
        int minimum=100; //理想温度与四周格子温度差距的最小值
        int movement=4; //运动方向
        int count=1; //有相同温度的格子数量（四周）
        int[] a= {5,5,5,5}; //若有格子温度相同，则其方向存放在该数组中
        int k=0;
        Random rdm=new Random();
        //如果所在格子温度正好是虫子理想温度，则原地不动
        if(temp[this.x][this.y]==this.idealTemp)
            return 4;
        //如果所在格子温度不是虫子理想温度，则需要移动
        else {
            //将四周格子温度存入数组，如果虫子本身位于边界，则将相应位置的温度设为-200，使虫子不可能往该方向移动
            nearbyTemp[0]=(this.y==0)?(-200):temp[this.x][this.y-1];
            nearbyTemp[1]=(this.x==0)?(-200):temp[this.x-1][this.y];
            nearbyTemp[2]=(this.x==29)?(-200):temp[this.x+1][this.y];
            nearbyTemp[3]=(this.y==29)?(-200):temp[this.x][this.y+1];
            int delta[]=new int[4]; //存取四周温度与理想温度差值的数组
            //循环4个方向
            for(int i=0;i<4;i++) {
                delta[i]=Math.abs(this.idealTemp-nearbyTemp[i]);
                //如果温度差小于最小值
                if(delta[i]<minimum) {
                    k=0;
                    movement=i;
                    minimum=delta[i]; //重新设最小值
                    count=1;
                    a[k]=i;
                    k++;
                }
                //如果温度差等于最小值
            }
        }
    }

```

```

        else if(delta[i]==minimum) {
            count++;
            a[k]=i;
            k++;
        }
    }
    //4个格子中如果有若干个格子温度相同，则在这些方向中随机选取一个
    if(count!=1) {
        int m=rdm.nextInt(count);
        movement=a[m];
    }
}
return movement;
}

```

//释放热量函数

```

public void spreadHeat(int[][] temp) {
    //如果在左上角
    if(this.x==0&&this.y==0) {
        temp[this.x][this.y+1]+=this.outputHeat/4;
        temp[this.x+1][this.y]+=this.outputHeat/4;
    }
    //如果在右上角
    else if(this.x==0&&this.y==29) {
        temp[this.x][this.y-1]+=this.outputHeat/4;
        temp[this.x+1][this.y]+=this.outputHeat/4;
    }
    //如果在左下角
    else if(this.x==29&&this.y==0) {
        temp[this.x-1][this.y]+=this.outputHeat/4;
        temp[this.x][this.y+1]+=this.outputHeat/4;
    }
    //如果在右下角
    else if(this.x==29&&this.y==29) {
        temp[this.x-1][this.y]+=this.outputHeat/4;
        temp[this.x][this.y-1]+=this.outputHeat/4;
    }
    //如果靠左边界
    else if(this.x==0&&this.y!=0&&this.y!=29) {
        temp[this.x][this.y-1]+=this.outputHeat/4;
        temp[this.x+1][this.y]+=this.outputHeat/4;
        temp[this.x][this.y+1]+=this.outputHeat/4;
    }
    //如果靠右边界
}

```

```

        else if(this.x==29&&this.y!=0&&this.y!=29) {
            temp[this.x][this.y-1]+=this.outputHeat/4;
            temp[this.x-1][this.y]+=this.outputHeat/4;
            temp[this.x][this.y+1]+=this.outputHeat/4;
        }
        //如果靠上边界
        else if(this.y==0&&this.x!=0&&this.x!=29) {
            temp[this.x-1][this.y]+=this.outputHeat/4;
            temp[this.x+1][this.y]+=this.outputHeat/4;
            temp[this.x][this.y+1]+=this.outputHeat/4;
        }
        //如果靠下边界
        else if(this.y==29&&this.x!=0&&this.x!=29) {
            temp[this.x-1][this.y]+=this.outputHeat/4;
            temp[this.x+1][this.y]+=this.outputHeat/4;
            temp[this.x][this.y-1]+=this.outputHeat/4;
        }
        //其他情况（在中间，四周都有格子）
        else {
            temp[this.x][this.y-1]+=this.outputHeat/4;
            temp[this.x][this.y+1]+=this.outputHeat/4;
            temp[this.x-1][this.y]+=this.outputHeat/4;
            temp[this.x+1][this.y]+=this.outputHeat/4;
        }
    }
}

```

5.3.2 HeatBugView.java

```

package heatbug;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.ActionEvent;
import java.awt.geom.Ellipse2D;
import java.util.*;

/*
 * 郑雅璇、杨诺彤
 * 2020 年 12 月
 */

//公有类 HeatBugView

```

```

public class HeatBugView extends JFrame implements ActionListener{

    JFrame frame;
    JPanel panel1; //输入实验参数
    JPanel panel2; //操作
    JPanel panel3; //光栅图
    JPanel panel4; //unhappiness 变化图
    int numBugs; //热虫数量
    int minIdealTemp; //最低理想温度
    int maxIdealTemp; //最高理想温度
    int minOutputHeat; //最小释放热量
    int maxOutputHeat; //最大释放热量
    int time; //热虫运动时间（次数）
    JButton startBtn; //“开始”按钮
    JButton quitBtn; //“退出”按钮
    HeatBug[] heatbugs; //热虫数组
    JTextField numBugsTf; //输入“热虫数量”的文本框
    JTextField minIdealTempTf; //输入“最低理想温度”的文本框
    JTextField maxIdealTempTf; //输入“最高理想温度”的文本框
    JTextField minOutputHeatTf; //输入“最小释放热量”的文本框
    JTextField maxOutputHeatTf; //输入“最大释放热量”的文本框
    JTextField timeTf; //输入“热虫运动次数”的文本框

    //构造函数
    public HeatBugView() {
        frame=new JFrame("热虫模拟");
        frame.setVisible(true);
        frame.setSize(600, 600);
        frame.setLayout(new GridLayout(2,2)); //将 frame 划分成 4 块

        //panel1
        JLabel numBugsLbl;
        JLabel minIdealTempLbl;
        JLabel maxIdealTempLbl;
        JLabel minOutputHeatLbl;
        JLabel maxOutputHeatLbl;
        JLabel timeLbl;
        panel1=new JPanel();
        frame.add(panel1);
        panel1.setLayout(new GridLayout(6,2));
        panel1.setBorder(BorderFactory.createTitledBorder("输入实验参数"));
        //控件实例化
        numBugsLbl=new JLabel("numBugs");
        minIdealTempLbl=new JLabel("minIdealTemp");

```

```

maxIdealTempLbl=new JLabel("maxIdealTemp");
minOutputHeatLbl=new JLabel("minOutputHeat");
maxOutputHeatLbl=new JLabel("maxOutputHeat");
timeLbl=new JLabel("time");
numBugsTf=new JTextField();
minIdealTempTf=new JTextField();
maxIdealTempTf=new JTextField();
minOutputHeatTf=new JTextField();
maxOutputHeatTf=new JTextField();
timeTf=new JTextField();
//将控件加入到 panel1 中
panel1.add(numBugsLbl);
panel1.add(numBugsTf);
panel1.add(minIdealTempLbl);
panel1.add(minIdealTempTf);
panel1.add(maxIdealTempLbl);
panel1.add(maxIdealTempTf);
panel1.add(minOutputHeatLbl);
panel1.add(minOutputHeatTf);
panel1.add(maxOutputHeatLbl);
panel1.add(maxOutputHeatTf);
panel1.add(timeLbl);
panel1.add(timeTf);

//panel2
panel2=new JPanel();
frame.add(panel2);
panel2.setLayout(null);
panel2.setBorder(BorderFactory.createTitledBorder("操作"));
startBtn=new JButton("start");
startBtn.setBounds(50, 60, 200, 70); //设置 “开始” 按钮大小
startBtn.addActionListener(this);
quitBtn=new JButton("quit");
quitBtn.setBounds(50, 170, 200, 70); //设置 “退出” 按钮大小
quitBtn.addActionListener(this);
panel2.add(startBtn);
panel2.add(quitBtn);

//panel3
panel3=new JPanel();
frame.add(panel3);
panel3.setLayout(new GridLayout(30,30)); //将 panel3 划分成 30*30 块

//panel4

```

```

        panel4=new JPanel();
        frame.add(panel4);
    }

    int temp[][]=new int[30][30]; //存放每个格子温度的二维数组
    JLabel[][] labels=new JLabel[30][30]; //存放 30*30 个 label 的二维数组，代表 30*30 个格子

    //初始化，将所有 label 设成黑色，加入到 panel3 中
    public void init() {
        for(int i=0;i<30;i++)
            for(int j=0;j<30;j++) {
                labels[i][j]=new JLabel();
                labels[i][j].setOpaque(true);
                labels[i][j].setBackground(Color.black);
                panel3.add(labels[i][j]);
            }
    }

    //监听事件
    public void actionPerformed(ActionEvent e) {
        //放到单独的线程中，避免主线程阻塞，以实现界面刷新
        new Thread(new Runnable() {
            @Override
            public void run() {
                //点击“退出”按钮
                if(e.getSource()==quitBtn) {
                    System.exit(0);
                }

                //点击“开始”按钮
                int movement=0; //虫子运动方向，0 代表上，1 代表左，2 代表右，3 代表下，4 代表
原地不动

                int evaporation=0; //系统散热
                Random rdm=new Random();
                if(e.getSource()==startBtn) {
                    //获取用户输入数据检验
                    try {
                        numBugs=Integer.parseInt(numBugsTf.getText());
                    }
                    catch (Exception e2) {
                        // TODO: handle exception
                        JOptionPane.showMessageDialog(null, "numBugs 必须为整数!", "提示
",JOptionPane.PLAIN_MESSAGE);
                    }
                    return;
                }
            }
        }).start();
    }

```



```

    }
    try {
        minIdealTemp=Integer.parseInt(minIdealTempTf.getText());
    }
    catch (Exception e2) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "minIdealTemp 必须为整数! ", "提示
",JOptionPane.PLAIN_MESSAGE);
        return;
    }
    try {
        maxIdealTemp=Integer.parseInt(maxIdealTempTf.getText());
    }
    catch (Exception e2) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "maxIdealTemp 必须为整数! ", "提示
",JOptionPane.PLAIN_MESSAGE);
        return;
    }
    try {
        minOutputHeat=Integer.parseInt(minOutputHeatTf.getText());
    }
    catch (Exception e2) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "minOutputHeat 必须为整数! ", "提示
",JOptionPane.PLAIN_MESSAGE);
        return;
    }
    try {
        maxOutputHeat=Integer.parseInt(maxOutputHeatTf.getText());
    }
    catch (Exception e2) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "maxOutputHeat 必须为整数! ", "提示
示",JOptionPane.PLAIN_MESSAGE);
        return;
    }
    try {
        time=Integer.parseInt(timeTf.getText());
    }
    catch (Exception e2) {
        // TODO: handle exception
        JOptionPane.showMessageDialog(null, "time 必须为整数! ", "提示
",JOptionPane.PLAIN_MESSAGE);

```

```

        return;
    }
    if(minIdealTemp>=maxIdealTemp||minOutputHeat>=maxOutputHeat) {
        JOptionPane.showMessageDialog(null, "最大值必须大于最小值!", "提示",
JOptionPane.PLAIN_MESSAGE);
        return;
    }
    if(time>100||time<1) {
        JOptionPane.showMessageDialog(null, "time 取值必须在 1~100!", "提示",
JOptionPane.PLAIN_MESSAGE);
        return;
    }
    if(numBugs>300||numBugs<1) {
        JOptionPane.showMessageDialog(null, "numBugs 取值必须在 1~300!", "提示",
JOptionPane.PLAIN_MESSAGE);
        return;
    }

    heatbugs=new HeatBug[numBugs]; //根据用户输入的热虫数实例化一个 HeatBug
类型的数组

    drawLineXY(panel4); //画 unhappiness 曲线图的 XY 轴
    drawXScale(panel4,time); //画 X 轴刻度

    //生成相应数量的热虫并初始化
    for(int t=0;t<numBugs;t++) {
        heatbugs[t]=new HeatBug();
        //随机生成热虫在网格中的位置
        heatbugs[t].x=rdm.nextInt(30);
        heatbugs[t].y=rdm.nextInt(30);
        //热虫的理想温度和释放热量均在最小最大范围内随机生成
        heatbugs[t].getIdealTemp(minIdealTemp, maxIdealTemp);
        heatbugs[t].getOutputHeat(minOutputHeat, maxOutputHeat);
        //将虫子所在的格子显示为绿色
        labels[heatbugs[t].x][heatbugs[t].y].setBackground(Color.green);
        evaporation+=heatbugs[t].outputHeat;
    }
    evaporation/=900; //计算每个格子的散热

    //循环，使热虫运动 time 次
    int i=0;
    for(i=0;i<time;i++) {
        drawPoint(panel4,temp,i); //每次运动前在 unhappiness 曲线图中画点

        //每个热虫的运动

```

```

for(int j=0;j<numBugs;j++) {
    heatbugs[j].spreadHeat(temp); //向四周散热

    movement=heatbugs[j].move(temp); //获得运动方向

    //根据运动方向来改变网格中 label 的颜色来显示热虫的运动
    switch(movement) {
        //向上移动，原来格子变黑，上方格子变绿
        case 0:{
            labels[heatbugs[j].x][heatbugs[j].y].setBackground(Color.black);
            labels[heatbugs[j].x][heatbugs[j].y-1].setBackground(Color.green);
            heatbugs[j].y--;
            break;
        }
        //向左移动，原来格子变黑，左方格子变绿
        case 1:{
            labels[heatbugs[j].x][heatbugs[j].y].setBackground(Color.black);
            labels[heatbugs[j].x-1][heatbugs[j].y].setBackground(Color.green);
            heatbugs[j].x--;
            break;
        }
        //向右移动，原来格子变黑，右方格子变绿
        case 2:{
            labels[heatbugs[j].x][heatbugs[j].y].setBackground(Color.black);
            labels[heatbugs[j].x+1][heatbugs[j].y].setBackground(Color.green);
            heatbugs[j].x++;
            break;
        }
        //向下移动，原来格子变黑，下方格子变绿
        case 3:{
            labels[heatbugs[j].x][heatbugs[j].y].setBackground(Color.black);
            labels[heatbugs[j].x][heatbugs[j].y+1].setBackground(Color.green);
            heatbugs[j].y++;
            break;
        }
        //原地不动
        case 4:{
            break;
        }
    }
}

//系统散热
for(int m=0;m<30;m++)
    for(int n=0;n<30;n++)

```

```

        temp[m][n]-=evaporation;

        //每次运动后让线程暂停片刻，使运动状况方便观察
        try{
            Thread.sleep(200);
        }
        catch(InterruptedException e2) {
            e2.printStackTrace();
        }
    }
    drawPoint(panel4,temp,i); //最后一次运动后在曲线图画最终的 unhappiness 点
}
}
}).start();
}

//获得所有热虫总体 unhappiness 值
public double getOverallUnhappiness() {
    double overallUnhappiness=0;
    for(int i=0;i<numBugs;i++) {
        int itsUnhappiness=heatbugs[i].getUnhappiness(temp); //获取每个虫子自己的 unhappiness 值
        overallUnhappiness+=itsUnhappiness; //加和
    }
    overallUnhappiness/=numBugs; //求均值
    return overallUnhappiness;
}

//画 unhappiness 曲线图的 XY 轴
public void drawLineXY(JPanel panel) {
    Graphics g=panel.getGraphics();
    Graphics2D g2d=(Graphics2D)g;
    g2d.clearRect(0, 0, panel.getWidth(), panel.getHeight()); //当前背景色清空画板
    g2d.setColor(Color.BLACK); //画笔颜色

    g2d.drawLine(5, 5, 5, panel.getHeight()-5); //画纵轴
    g2d.drawLine(5, panel.getHeight()-5, panel.getWidth()-5, panel.getHeight()-5); //画横轴
    g2d.dispose();
}

//画 X 轴刻度
public void drawXScale(JPanel panel, int time) {
    Graphics g=panel.getGraphics();
    Graphics2D g2d=(Graphics2D)g;

```

```

        g2d.setColor(Color.BLACK); //画笔颜色
        int a=panel.getWidth()/time; //根据运动次数计算两个刻度之间的距离
        //循环画刻度
        for (int i=0; i<(time+1); i++)
            g2d.drawLine(5+i*a, panel.getHeight()-5, 5+i*a, panel.getHeight()-5-3);
        g2d.dispose();
    }

    //获取最初 unhappiness 值
    public double getInitialUnhappiness() {
        double initialUnhappiness=0;
        for(int i=0;i<numBugs;i++)
            initialUnhappiness+=heatbugs[i].idealTemp; //最初每个虫子的 unhappiness 值就是其理想温
        度值（因为初始格子温度为 0）
        initialUnhappiness/=numBugs;
        return initialUnhappiness;
    }

    //在 unhappiness 图中画点
    public void drawPoint(JPanel panel,int[][] temp,int i) {
        Graphics g=panel.getGraphics();
        Graphics2D g2d=(Graphics2D)g;
        g2d.setColor(Color.BLACK);
        double unhappiness=getOverallUnhappiness(); //获得总体 unhappiness 均值
        int a=(panel.getWidth()-10)/time;
        double k=(panel.getHeight()-10)/getInitialUnhappiness(); //纵轴长度与最大 unhappiness 值（初始
        时）之间的比例
        unhappiness*=k; //根据比例将 unhappiness 值换成在纵轴上对应的长度
        Ellipse2D ellipse2d=new Ellipse2D.Double(5+i*a-1, panel.getHeight()-5-unhappiness-1, 2, 2); //圆圈
        g2d.draw(ellipse2d);
        g2d.fill(ellipse2d); //涂成实心
        g2d.dispose();
    }

    public static void main(String args[]) {
        HeatBugView myView=new HeatBugView(); //此类实例化
        myView.init(); //初始化
    }
}

```

6 学习收获

通过本次实验，获得了如下收获：

- (1) 了解 Swarm 模型构成及机理
- (2) 熟悉 java 语言，对面向对象编程、类、封装等概念有更深入的理解
- (3) 通过编写窗体，在 java 图形界面编程上更加熟练
- (4) 在编写展现光栅图中热虫运动情况的代码时，遇到一个困难：虽然在每次的循环中加入了让线程暂停片刻的代码，却还是不能显示热虫运动的每一步，只能看到最初随机生成的热虫和程序运行结束后的热虫状态。在查阅大量资料后，发现需要将那一段代码因为涉及控件的各种操作，只有放入一个单独的新线程中，才能避免主线程阻塞，并且让每次循环控件都能刷新，这样就解决了问题。
- (5) 编代码时遇到困难培养了我们的耐心和毅力
- (6) 培养了合作、交流能力