**实验二、谢林模型模拟**

输入：n\*n的矩阵，随机布局的两种节点

输出：

1）调节参数后输出相应的结果

2）需要有界面显示

相关概念：

当不同种族的邻居数目达到一定数目时或者同类种族邻居书目不足给定数目时，就有搬家的动机。

实验需要考察不同阈值下网络的变化。

算法分析：







代码如下：

package schelling;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JPanel;  
import javax.swing.JTextField;  
  
public class frame1 extends JFrame {  
  
 JLabel label = new JLabel("请输入矩阵大小：");  
 JTextField text = new JTextField();  
 JPanel panel = new JPanel();//面板1  
  
 JButton button1 = new JButton("创建");//按钮2  
  
 public frame1()  
 {  
 this.setSize(600,400);  
 init();  
 setVisible(true);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 }  
  
 private void init() {  
  
 // *TODO Auto-generated method stub* this.add(panel);  
 panel.setLayout(null);  
 label.setBounds(10, 10, 150, 20);  
 panel.add(label);  
 text.setBounds(120,10,50,20);  
 panel.add(text);  
 button1.setBounds(50, 50, 100, 30);  
 panel.add(button1);  
  
  
  
 button1.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 String s = text.getText();  
 Integer a = Integer.*parseInt*(s);  
 System.*out*.println(a);  
 dispose();  
 new mainFrame2(a);  
  
 }  
 });  
  
 }  
  
 public static void main(String[] args) {  
 // *TODO Auto-generated method stub* new frame1();  
 }  
  
}

package schelling;  
  
import java.awt.Color;  
import java.awt.Font;  
import java.awt.GridLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import javax.swing.\*;  
  
  
  
class mainFrame2 extends JFrame  
  
{ JLabel label = new JLabel();  
 JLabel label1 = new JLabel();  
 JPanel myPanel1 = new JPanel();//面板1  
 JPanel myPanel2 =new JPanel();//面板2  
 JButton button1 = new JButton("返回");//按钮2  
 JButton button2 = new JButton("确定");  
 JButton button3 = new JButton("单步搬家");//按钮3  
 JButton button4 = new JButton("搬家");  
 JButton[][] b;  
 Integer a ;  
 JSplitPane jSplitPane =new JSplitPane();//设定为左右拆分布局  
 public mainFrame2(int a)  
 { this.a =a;  
 this.setSize(1000,800);  
 init(a);  
 setVisible(true);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
  
  
  
 }  
  
 void init(int a)  
 { createArray c = new createArray(a);  
 jSplitPane.setOneTouchExpandable(true);//让分割线显示出箭头  
 jSplitPane.setContinuousLayout(true);//操作箭头，重绘图形  
 //jSplitPane.setPreferredSize(new Dimension (100,200));  
 jSplitPane.setOrientation(JSplitPane.*HORIZONTAL\_SPLIT*);//设置分割线方向  
 myPanel1.setSize(800, 800);  
 myPanel2.setSize(200, 800);  
 jSplitPane.setLeftComponent(myPanel1);//布局中添加组件 ，面板1  
 jSplitPane.setRightComponent(myPanel2);//添加面板2  
 jSplitPane.setDividerSize(1);//设置分割线的宽度  
 //jSplitPane.setDividerLocation(100);//设置分割线位于中央  
 jSplitPane.setDividerLocation(800);//设定分割线的距离左边的位置  
 setContentPane(jSplitPane);  
 //pack();  
 myPanel2.setLayout(null);  
 label.setText("单机一下蓝色");  
 label.setBounds(10, 10, 150, 40);  
 myPanel2.add(label);  
 label1.setText("单机两下红色");  
 label1.setBounds(10,30,150,40);  
 myPanel2.add(label1);  
 button1.setBounds(50, 80, 100, 30);  
 button2.setBounds(50,130,100,30);  
 button3.setBounds(50,180,100,30);  
 button4.setBounds(50,230,100,30);  
 myPanel2.add(button1);  
 myPanel2.add(button2);  
 myPanel2.add(button3);  
 myPanel2.add(button4);  
  
  
  
  
  
 JButton[][] b=new JButton[a][a];  
 myPanel1.setLayout(new GridLayout(a,a,4,4));  
 for(int i=0;i<a;i++){  
 for( int j =0;j<a;j++) {  
  
 b[i][j]=new JButton();  
 myPanel1.add(b[i][j]);  
  
 b[i][j].addActionListener(new ActionListener() {  
  
  
 public void actionPerformed(ActionEvent e) {  
  
  
 JButton button = (JButton) e.getSource();  
 if(button.getBackground() == Color.*BLUE*) {  
 button.setBackground(Color.*RED*);  
 }  
 else {button.setBackground(Color.*BLUE*);  
 }  
  
 }  
 });  
 }  
 }  
  
 /\* createArray c = new createArray(a);  
 for(int i=0;i<a;i++){  
 for( int j =0;j<a;j++) {  
 if(b[i][j].getBackground()==Color.BLUE) {  
 c.array[i][j] =1;  
 }else if(b[i][j].getBackground()==Color.RED) {  
 c.array[i][j] =2;  
 }else {  
 c.array[i][j] =0;  
 }  
 }  
 }  
  
 \*/  
 button1.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 new frame1();  
  
 }  
 });  
  
 button2.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
  
  
 for(int i=0;i<a;i++){  
 for( int j =0;j<a;j++) {  
 if(b[i][j].getBackground()==Color.*BLUE*) {  
 c.*array*[i][j] =1;  
 }else if(b[i][j].getBackground()==Color.*RED*) {  
 c.*array*[i][j] =2;  
 }else {  
 c.*array*[i][j] =0;  
 }  
 }  
 }  
  
 for(int i=0;i<a;i++) {  
 for(int j =0;j<a;j++) {  
 System.*out*.print(c.*array*[i][j]);  
 }  
 System.*out*.println();  
  
 }  
  
 }  
 });  
 button3.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 c.checkNeighbor();  
 for(int i =0;i<a;i++) {  
 for(int j = 0;j<a;j++) {  
 if(c.*array*[i][j]==2)  
 {  
 b[i][j].setBackground(Color.*RED*);  
 }  
 else if(c.*array*[i][j]==1) {  
 b[i][j].setBackground(Color.*BLUE*);  
 }else  
 {  
 b[i][j].setBackground(null);  
 }  
 }  
 }  
  
 }  
 });  
  
 button4.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 for(int i = 0;i<10000;i++) {  
 c.checkNeighbor();}  
 for(int i =0;i<a;i++) {  
 for(int j = 0;j<a;j++) {  
 if(c.*array*[i][j]==2)  
 {  
 b[i][j].setBackground(Color.*RED*);  
 }  
 else if(c.*array*[i][j]==1) {  
 b[i][j].setBackground(Color.*BLUE*);  
 }else  
 {  
 b[i][j].setBackground(null);  
 }  
 }  
 }  
  
 }  
  
 });  
  
  
 }  
  
  
  
}

package schelling;  
  
public class createArray {  
  
  
 private int n;  
 static int [][]*array*;  
  
 public createArray(int n) {  
 this.n =n;  
 *array* = new int[n][n];  
 for(int i=0;i<n;i++) {  
 for(int j =0;j<n;j++) {  
 *array*[i][j] = 0;  
 }  
 }  
 }  
  
  
  
  
 public void checkNeighbor() {  
 for(int i=0;i<n;i++) {  
 for(int j =0;j<n;j++) {  
 if(*array*[i][j]!=0) {  
 int count = 0;  
  
 if(i-1 >=0&&j-1>=0&&*array*[i-1][j-1] ==*array*[i][j] )count ++;  
 if(i-1>=0&&*array*[i-1][j]==*array*[i][j])count++;  
 if(i-1>=0&&j+1<n&&*array*[i-1][j+1]==*array*[i][j])count++;  
 if(j-1>=0&&*array*[i][j-1]==*array*[i][j])count++;  
 if(j+1<n&&*array*[i][j+1]==*array*[i][j])count++;  
 if(i+1<n&&j-1>=0&&*array*[i+1][j-1]==*array*[i][j])count++;  
 if(i+1<n&&*array*[i+1][j]==*array*[i][j])count++;  
 if(i+1<n&&j+1<n&&*array*[i+1][j+1]==*array*[i][j])count++;  
 // System.out.println("与("+i+","+j+")"+"相同的邻居数为"+count);  
  
 if(count <3) {  
 int temp = *array*[i][j];  
  
  
 int x = i;  
 int y = j;  
 while(*array*[x][y]!=0) {  
 x = (int) (Math.*random*()\*n);  
 y = (int) (Math.*random*()\*n);  
 }  
  
 *array*[x][y] = temp;  
 *array*[i][j] =0;  
  
  
  
 }  
  
  
  
 }  
 }  
 }  
  
 }  
  
  
}

测试如下：



