import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.Random;

public class LifeGame extends JApplet implements ActionListener {

private Container c;

private JTextArea displayBoard; // display board

private JButton nextGeneration; // button to generate next generation

int[][]cells=new int[9999][9999]; // The matrix board

int userLine, userRows=0; // The index of matrix that user has been typed

int helperCount=0; // This varable helps to do caculating while running

int yesNo; //used when the matrix has no more changes

int[][]cellsBefore=new int[9999][9999];

@Override

public void init(){ // Applet init() method

// set layout manager

c = getContentPane();

c.setLayout(new BorderLayout());

// setup components

displayBoard = new JTextArea();

nextGeneration = new JButton("Next generation");

nextGeneration.addActionListener(this);

// add components to applet

c.add(displayBoard,BorderLayout.CENTER);

c.add(nextGeneration,BorderLayout.SOUTH);

gameStart();

}

public void displayLife(){

showChanges();

}

// implementation of ActionListener interface

@Override

public void actionPerformed (ActionEvent e){

generateNextGeneration();

displayLife();

}

public void generateNextGeneration(){

SetInBefore();

MakeChanges();

showChanges();

if(checkChange())

{

yesNo=(JOptionPane.showConfirmDialog(null, "There is no more changes, restart?","What's now?"

,JOptionPane.YES\_NO\_OPTION));

if(yesNo==JOptionPane.YES\_OPTION)

{

gameStart();

}

}

}

public void gameStart()

{

String rows, columns;

rows=JOptionPane.showInputDialog("Enter how many lines: ");

columns=JOptionPane.showInputDialog("Enter how many columns: ");

userLine= Integer.parseInt(rows);

userRows=Integer.parseInt(columns);

int randomNum;

String MatrixText="";

for (int i = 0; i < userLine; i++)

{

for (int j = 0; j < userRows; j++)

{

cells[i][j]=randomCellValue();

}

}

showChanges();

}

public int randomCellValue()

{

Random rnd= new Random();

int randomNum=rnd.nextInt(2);

return randomNum;

}

public void showChanges()

{

String MatrixText="";

for (int i = 0; i < userLine; i++)

{

for (int j = 0; j < userRows; j++)

{

if(cells[i][j]==1)

MatrixText += "+ ";

else

{

MatrixText += "- ";

}

}

MatrixText+="\n";

}

displayBoard.setText(MatrixText);

}

public void borning( int userLinex, int userRowy)

{

//cubes spots

//first cube

if (userLinex==0 && userRowy==0 && cells[0][1]==1

&&cells[1][0]==1 &&cells[1][1]==1)

{

cells[userLinex][userRowy]=1;

return;

}

// secound cube

if (userLinex==0 && userRowy==userRows-1 && cells[userLinex][userRowy-1]==1

&&cells[1][userRowy-1]==1 &&cells[1][userRows]==1)

{

cells[userLinex][userRowy]=1;

return;

}

// third cube

if (userLinex==userLine-1 && userRowy==0 && cells[userLinex][userRowy+1]==1

&&cells[userLinex-1][userRowy]==1 &&cells[userLinex-1][userRowy+1]==1)

{

cells[userLinex][userRowy]=1;

return;

}

//forth cube

if (userLinex==userLine-1 && userRowy==userRows-1 && cells[userLinex][userRowy-1]==1

&&cells[userLinex-1][userRowy-1]==1 &&cells[userLinex-1][userRowy]==1)

{

cells[userLinex][userRowy]=1;

return;

}

checkOverflowAndGeneral(userLinex,userRowy);

helperCount=0;

}

public void MakeChanges()

{

for (int i = 0; i < userLine; i++)

{

for (int j = 0; j < userRows; j++)

{

// מס' שכנים = מנה את השכנים של תא זה

// על פי מספר השכנים קבע את מצבו של התא בדור הבא.

if (cells[i][j]==0)

{

//dead cell

borning(i,j);

}

else

{

//alive cell

killCell(i,j);

}

}

}

}

public void killCell (int userLinex, int userRowy)

{

helperCount=0;

//cubes spots

//first cube

if (userLinex==0 && userRowy==0)

{

if(cells[0][1]==1)

{

helperCount++;

}

if(cells[1][0]==1)

{

helperCount++;

}

if(cells[1][1]==1)

{

helperCount++;

}

if(helperCount<=1)

{

cells[userLinex][userRowy]=0;

}

return;

}

//secound cube

if (userLinex==0 && userRowy==userRows-1)

{

if(cells[0][userRowy-1]==1)

{

helperCount++;

}

if(cells[1][userRowy]==1)

{

helperCount++;

}

if(cells[1][userRowy-1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

return;

}

//third cube

if (userLinex==userLine-1 && userRowy==0)

{

if(cells[userLinex][userRowy+1]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy+1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

return;

}

//forth cube

if (userLinex==userLine-1 && userRowy==userRows-1)

{

if(cells[userLinex][userRowy-1]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy-1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

return;

}

checkOverflowAndGeneral(userLinex,userRowy);

}

public void checkNeedToDo(int userLinex, int userRowy)

{

if(helperCount<=1 || helperCount>=4)

{

cells[userLinex][userRowy]=0;

}

if(helperCount==3)

{

cells[userLinex][userRowy]=1;

}

helperCount=0;

}

public void checkOverflowAndGeneral(int userLinex, int userRowy)

{

//over flow

//left overflow

try

{

if(userRowy==0)

{

if(cells[userLinex][1]==1)

{

helperCount++;

}

if(cells[userLinex-1][1]==1)

{

helperCount++;

}

if(cells[userLinex-1][0]==1)

{

helperCount++;

}

if(cells[userLinex+1][0]==1)

{

helperCount++;

}

if(cells[userLinex+1][1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

}

}

catch(ArrayIndexOutOfBoundsException e)

{}

//right overflow

try

{

if(userRowy==userRows-1)

{

if(cells[userLinex-1][userRowy]==1)

{

helperCount++;

}

if(cells[userLinex][userRowy-1]==1)

{

helperCount++;

}

if(cells[userLinex+1][userRowy-1]==1)

{

helperCount++;

}

if(cells[userLinex+1][userRowy]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy-1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

}

}

catch(ArrayIndexOutOfBoundsException e)

{}

//up overflow

try

{

if(userLinex==0)

{

if(cells[0][userRowy+1]==1)

{

helperCount++;

}

if(cells[0][userRowy-1]==1)

{

helperCount++;

}

if(cells[1][userRowy]==1)

{

helperCount++;

}

if(cells[1][userRowy-1]==1)

{

helperCount++;

}

if(cells[1][userRowy+1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

}

}

catch(ArrayIndexOutOfBoundsException e)

{}

//down overflow

try

{

if(userLinex==userLine-1)

{

if(cells[userLinex][userRowy+1]==1)

{

helperCount++;

}

if(cells[userLinex][userRowy-1]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy-1]==1)

{

helperCount++;

}

if(cells[userLinex-1][userRowy+1]==1)

{

helperCount++;

}

checkNeedToDo(userLinex,userRowy);

}

}

catch(ArrayIndexOutOfBoundsException e)

{}

}

public void SetInBefore()

{

for (int i = 0; i < userLine; i++)

{

for (int j = 0; j < userRows; j++)

{

cellsBefore[i][j]=cells[i][j];

}

}

}

public boolean checkChange()

{

for (int i = 0; i < userLine; i++)

{

for (int j = 0; j < userRows; j++)

{

if(cellsBefore[i][j]!=cells[i][j])

{

return false;

}

}

}

return true;

}

}

// מס' שכנים = מנה את השכנים של תא זה

// על פי מספר השכנים קבע את מצבו של התא בדור הבא.