```
package ola.jsame;

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

public class ButtonPanel extends JPanel
{
    private OOGAButton myButtons[][];
    private SameGameModel myModel;
    private ActionListener myMoveMaker;
    /**
     * Create a button panel consisting of rows x cols buttons
     */

    ButtonPanel(int rows, int cols, SameGameModel model)
    {
        // construct superclass and make button array
        super(new GridLayout(rows,cols));
        myButtons = new OOGAButton[rows][cols];

        // make listener to do the move chosen by user
        myMoveMaker = new ActionListener(){
                public void actionPerformed(ActionEvent e)
                {
                    myModel.makeMove(new SameMove(e.getActionCommand()));
                }
            };

        myModel = model;
        makeButtons(myMoveMaker);
    }

    /**
     * Makes the buttons for this GUI/view. The number
     * of buttons is determined my the size of the array
     * myButtons. This helper function takes some of
     * the busy work out of the ButtonPanel constructor.
     */
    private void makeButtons(ActionListener moveMaker)
    {
        int count = 0;

        for(int rows = 0; rows < myButtons.length; rows++){
            for(int cols = 0; cols < myButtons[0].length; cols++) {
                String label = ""+count;
                count++;
                // create button with label reflecting count
                OOGAIcon icon = new OOGAIcon();
                myButtons[rows][cols] = new OOGAButton(icon);
                myButtons[rows][cols].setActionCommand(label);
                myButtons[rows][cols].addActionListener(moveMaker);
                add(myButtons[rows][cols]);
            }
        }
    }

    public void highlight(GridPoint[] list)
    {
        for(int j=0; j < list.length; j++){
            GridPoint p = list[j];
            myButtons[p.row][p.col].toggleColor();
        }
        repaint();
    }

    /**
     * Set all the buttons by re-displaying them all
     * in the right order. First we remove all the components
```



```
     * in this panel (that's the buttons). Then we add the
     * buttons to this panel in the right order based on what
     * the ordering of the buttons in the model is.
     * Finally, we revalidate so the buttons are shown (GUI
     * will redraw as a result of the revalidate).
     */
    public void setGrid(int[][] list)
    {
        for(int j=0; j < list.length; j++){
            for(int k=0; k < list[0].length; k++) {
                if (list[j][k] != SameGameModel.INVALID){
                    myButtons[j][k].setColor(list[j][k]);
                    myButtons[j][k].setEnabled(true);
                }
                else {
                    myButtons[j][k].setColor(-1);
                    myButtons[j][k].setEnabled(false);
                }
            }
        }
        revalidate();
    }
}
```

```
public class Debug
{
    public static void print(String s)
    {
        System.out.print(s);
    }
    public static void println(String s)
    {
        System.out.println(s);
    }
    public static void println()
    {
        System.out.println();
    }
}
```

```
package ola.jsame;

public class GridPoint implements Comparable
{
    public int row, col;

    public GridPoint()
    {
        this.row = this.col = 0;
    }

    public GridPoint(int row, int col)
    {
        this.row = row;
        this.col = col;
    }

    public boolean equals(Object o)
    {
        GridPoint p = (GridPoint) o;
        return this.row == p.row && this.col == p.col;
    }

    public int compareTo(Object o)
    {
        GridPoint p = (GridPoint) o;
        int rowDiff = this.row - p.row;
        int colDiff = this.col - p.col;

        if  (rowDiff == 0) {
            return colDiff;
        }
        return rowDiff;
    }
}
```

```
package ola.jsame;

public interface ISameGameView
{
    public void showGrid(int[][] list);
    public void highlight(GridPoint[] list);
}
```

```
package ola.jsame;

import javax.swing.*;
import java.awt.event.*;
import java.awt.Color;
import java.awt.Graphics;

/**
 * Show simple method for animating a button using swing.Timer
 * The button "zooms" by drawing increasing/decreasing rectangles
 * Note: button also has Icon, this is drawn independently
 * of zooming. Most likely all drawing should be consolidated in
 * one place.
 */

public class OOGAButton extends JButton implements ActionListener
{
    private static final int DELAY = 3;       // timing delay
    private static Color BLANK = Color.WHITE; // button not visible?
    private Color myColor;                    // color of button
    private Color mySavedColor = null;        // if null, not zooming

    private Timer myTimer = new Timer(DELAY, this);
    private int myCount;

    // these values are used for zooming in and out

    private static double[] widthPercent = {
        1.0, 0.75, 0.5, 0.25, 0.5, 0.75, 1.0
    };

    private int zoomx,zoomy,zoomw,zoomh;

    public void actionPerformed(ActionEvent e)
    {
        int w = getWidth();
        int h = getHeight();
        myCount++;
        double factor = widthPercent[myCount % widthPercent.length];
        zoomx = (int) (w*(1.0-factor)/2.0);
        zoomy = (int) (h*(1.0-factor)/2.0);
        zoomw = (int) (w*factor);
        zoomh = (int) (h*factor);
        repaint();
    }

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        if (mySavedColor != null) {
            Color c = g.getColor();
            g.setColor(Color.BLACK);
            g.fill3DRect(zoomx,zoomy,zoomw,zoomh,true);
            g.setColor(c);
        }
    }

    public OOGAButton()
    {
        super();
    }

    public OOGAButton(String s)
    {
        super(s);
    }
    public OOGAButton(Icon i)
    {
        super(i);
    }
```



```
    }

    public OOGAButton(String s, Icon i)
    {
        super(s,i);
    }

    public void setColor(int index)
    {
        if (index == -1){
            myColor = BLANK;
        }
        else {
            myColor = ourColors[index];
        }
    }

    public Color getColor()
    {
        return myColor;
    }

    public void setActionCommand(String s)
    {
        super.setActionCommand(s);
    }

    public void toggleColor()
    {
        if (mySavedColor == null){
            mySavedColor = myColor;
            myCount = (int) Math.random()*4;
            myTimer.start();
        }
        else {
            myColor = mySavedColor;
            mySavedColor = null;
            myTimer.stop();
        }
    }

    private static Color[] ourColors= {
        Color.YELLOW, Color.GREEN, Color.RED,
        Color.BLUE,   Color.CYAN,  Color.MAGENTA,
        Color.PINK,   Color.ORANGE, Color.WHITE
    };

}
```

```
package ola.jsame;

public class OOGAConsts
{
    public final static String BLANK = " ";
    public final static int IMAGE_SIZE = 240;
    public final static int OFFSET = 1;
}
```

```
package ola.jsame;

import javax.swing.Icon;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Component;

/**
 *
 * @author Owen Astrachan
 */

public class OOGAIcon implements Icon
{
    /**
     */
    public OOGAIcon()
    {
    }

    /**
     * Draws this plain icon at the proper size.
     * @param c is used to determine how big to draw this Icon
     * @param g is the graphics context in which drawing takes place
     */
    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        int w = c.getWidth();
        int h = c.getHeight();
        Color color = ((OOGAButton) c).getColor();
        doPaint(g,Color.blue,color,w,h);
    }

    private void doPaint(Graphics g, Color background,
                         Color foreground, int width, int height)
    {
        g.setColor(background);
        g.fill3DRect(0,0,width,height,true);
        g.setColor(foreground);
        g.fill3DRect(OOGAConsts.OFFSET,OOGAConsts.OFFSET,
                         width,height,true);
        g.setColor(Color.black);
    }

    /**
     * Needed for interface, not used in this project.
     */
    public int getIconHeight()
    {
        return 5;
    }

    /**
     * Needed for interface, not used in this project.
     */
    public int getIconWidth()
    {
        return 5;
    }
}
```

```
package ola.jsame;

import java.awt.Dimension;
import javax.swing.*;

/**
 * Same game GUI
 * <P>
 * @author Owen Astrachan
 */
public class SameGameApplet extends JApplet
{
    private Dimension getDimension()
    {
        int rows = 6;
        int cols = 15;
        try{
            rows = Integer.parseInt(getParameter("rows"));
            cols = Integer.parseInt(getParameter("cols"));
        }
        catch (Exception e){
            // in case format bad or params missing
        }
        return new Dimension(rows,cols);
    }
    /**
     */
    public void init()
    {
        Dimension d = getDimension();
        SameGameModel model = new SameGameModel(d.height,d.width);
        JPanel panel = new SameGameCore(model);
        setContentPane(panel);
        setVisible(true);
    }
}
```

```
package ola.jsame;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.event.*;
import javax.swing.*;

/**
 * Same game Core for Applet and Application
 * <P>
 *
 * @author Owen Astrachan
 */
public class SameGameCore extends JPanel implements ISameGameView
{
    private SameGameModel myModel;
    private ButtonPanel myButtonPanel;

    /**
     * Create a view with a model
     * @param model is the model for this view
     */
    public SameGameCore(SameGameModel model)
    {
        setLayout(new BorderLayout());

        myModel = model;
        myButtonPanel = new ButtonPanel(myModel.getRows(),
                                        myModel.getCols(),
                                        myModel);

        add(myButtonPanel, BorderLayout.CENTER);
        myModel.addView(this); // draws board
    }

    /**
     * Required by interface, show the grid that's the model
     * @param list represents the model
     */
    public void showGrid(int[][] list)
    {
        myButtonPanel.setGrid(list);
    }

    public void highlight(GridPoint[] list)
    {
        myButtonPanel.highlight(list);
    }
}
```

```
package ola.jsame;

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

/**
 * Same game GUI
 * <P>
 *
 * @author Owen Astrachan
 */
public class SameGameGui extends JFrame
{
    private static final int OUR_WIDTH  = 50;

    /**
     * @param model is the model for this view
     */
    public SameGameGui(SameGameModel model)
    {
        setTitle("OOGA SameGame");
        JPanel panel = new SameGameCore(model);
        makeMenu();

        setContentPane(panel);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pack();
        setSize(OUR_WIDTH*model.getCols(), OUR_WIDTH*model.getRows());
        setVisible(true);
    }

    private void makeMenu()
    {
        JMenu menu = new JMenu("File");

        menu.add(new AbstractAction("Quit")
            {
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }
            });
        JMenuBar menubar = new JMenuBar();
        menubar.add(menu);
        setJMenuBar(menubar);
    }

    public static void main(String args[])
    {
        SameGameModel model = new SameGameModel(6,15);
        SameGameGui gui = new SameGameGui(model);
    }

}
```

```
package ola.jsame;

import java.util.*;

public class SameGameModel
{
    public static int INVALID = -1;
    private static int DEFAULT_RANGE = 6;
    private static int OFFSET = 2000;

    private int myRows;
    private int myCols;
    private int[][] myGrid;
    private ISameGameView myView;
    private int myRange;
    private ArrayList myList;
    private int       myLastMove;

    public SameGameModel(int rows, int cols)
    {
        this(rows,cols,DEFAULT_RANGE);
    }

    public SameGameModel(int rows, int cols, int range)
    {
        myRows = rows;
        myCols = cols;
        myRange = range;
        myGrid = new int[rows][cols];
        myList = new ArrayList();
        init();
    }

    private void init()
    {
        java.util.Random rando = new java.util.Random();
        for(int j=0; j < myRows; j++) {
            for(int k=0; k < myCols; k++) {
                int val = rando.nextInt(myRange);
                myGrid[j][k] = val;
            }
        }
    }

    private void flood(int row, int col, int match, ArrayList list)
    {
        if (0 <= row && row < getRows() && 0 <= col && col < getCols()){
            if (myGrid[row][col] == match){
                list.add(new GridPoint(row,col));
                myGrid[row][col] = myGrid[row][col] - OFFSET;
                flood(row,col+1,match,list);
                flood(row,col-1,match,list);
                flood(row+1,col,match,list);
                flood(row-1,col,match,list);
            }
        }
    }


    public int getRows()
    {
        return myRows;
    }

    public int getCols()
    {
        return myCols;
    }
```



```
    private void shift(int col)
    {
        if (myGrid[myRows-1][col] == INVALID){
//          Debug.println("col removal "+col);
            for(int r=0; r < myRows; r++){
                for(int c=col; c < myCols-1; c++){
                    myGrid[r][c] = myGrid[r][c+1];
                }
                myGrid[r][myCols-1] = INVALID;
            }
        }
    }
    private void clearAll()
    {
        GridPoint[] array = (GridPoint[]) myList.toArray(new GridPoint[0]);
        myView.highlight(array);

        Collections.sort(myList);
        for(int k=0; k < myList.size(); k++){
            GridPoint p = (GridPoint) myList.get(k);

            for(int r = p.row; r > 0; r--){
                myGrid[r][p.col] = myGrid[r-1][p.col];
            }
            myGrid[0][p.col] = INVALID;
        }
        for(int col=myCols-2; col >= 0; col--){
            shift(col);
        }
        myView.showGrid(myGrid);
        myList.clear();
    }

    private void clear(GridPoint[] array)
    {
        for(int k=0; k < array.length; k++){
            myGrid[array[k].row][array[k].col] += OFFSET;
        }
    }
    private void clearOne()
    {
        GridPoint p = (GridPoint) myList.get(0);
        myGrid[p.row][p.col] += OFFSET;
        myList.clear();
    }

    public void reset()
    {
        GridPoint[] array = (GridPoint[]) myList.toArray(new GridPoint[0]);
        myView.highlight(array);
        clear(array);
        myLastMove = -1;
    }

    private boolean inBlock(SameMove move)
    {
        int val = move.getValue();
        int row = val/myCols;
        int col = val%myCols;
        GridPoint gp = new GridPoint(row,col);
        return myList.contains(gp);
    }

    public void makeMove(SameMove move)
    {
        int val = move.getValue();
        int row = val/myCols;
        int col = val%myCols;
        if (myGrid[row][col] == INVALID){
```

```
            return;
        }

        if (inBlock(move)){
            clearAll();
        }
        else {
            reset();
            myList.clear();
            val = move.getValue();
            myLastMove = val;
            row = val/myCols;
            col = val%myCols;
//            Debug.println("move at "+row+" "+col);
            flood(row,col,myGrid[row][col],myList);
            if (myList.size() > 1){
                GridPoint[] array = (GridPoint[]) myList.toArray(new GridPoint[0
]);
                myView.highlight(array);
            }
            else {
                clearOne();
            }
        }
    }

    public void addView(ISameGameView view)
    {
        myView = view;
        myView.showGrid(myGrid);
    }
}
```

```
package ola.jsame;

public class SameMove
{
    private int mySquare;
    public SameMove(String s)
    {
        try{
            mySquare = Integer.parseInt(s);
        }
        catch (Exception e){
            mySquare = SameGameModel.INVALID;
        }
    }

    public SameMove(int square)
    {
        mySquare = square;
    }

    public int getValue()
    {
        return mySquare;
    }

}
```