The Puzzle game in Java Swing

In this chapter, we will create a simple puzzle game in Java Swing toolkit.

Puzzle

The goal of this little game is to form a picture. Buttons containing images are moved by clicking on them. Only buttons adjacent to the empty button can be moved.

```
package com.zetcode;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.awt.image.CropImageFilter;
import java.awt.image.FilteredImageSource;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.AbstractAction;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

class MyButton extends JButton {
    private boolean isLastButton;

    public MyButton() {

        super();

        initUI();
    }

    public MyButton(Image image) {

        super(new ImageIcon(image));

        initUI();
    }

    private void initUI() {

        isLastButton = false;
        BorderFactory.createLineBorder(Color.gray);

        addMouseListener(new MouseAdapter() {

            @Override
            public void mouseEntered(MouseEvent e) {
                setBorder(BorderFactory.createLineBorder(Color.yellow));
            }

            @Override
            public void mouseExited(MouseEvent e) {
                setBorder(BorderFactory.createLineBorder(Color.gray));
            }
        });
    }

    public void setLastButton() {

        isLastButton = true;
    }

    public boolean isLastButton() {

        return isLastButton;
    }
}
```

```
public class PuzzleEx extends JFrame {

    private JPanel panel;
    private BufferedImage source;
    private ArrayList<MyButton> buttons;

    ArrayList<Point> solution = new ArrayList();

    private Image image;
    private MyButton lastButton;
    private int width, height;
    private final int DESIRED_WIDTH = 300;
    private BufferedImage resized;

    public PuzzleEx() {

        initUI();
    }

    private void initUI() {

        solution.add(new Point(0, 0));
        solution.add(new Point(0, 1));
        solution.add(new Point(0, 2));
        solution.add(new Point(1, 0));
        solution.add(new Point(1, 1));
        solution.add(new Point(1, 2));
        solution.add(new Point(2, 0));
        solution.add(new Point(2, 1));
        solution.add(new Point(2, 2));
        solution.add(new Point(3, 0));
        solution.add(new Point(3, 1));
        solution.add(new Point(3, 2));

        buttons = new ArrayList();

        panel = new JPanel();
        panel.setBorder(BorderFactory.createLineBorder(Color.gray));
        panel.setLayout(new GridLayout(4, 3, 0, 0));

        try {
            source = loadImage();
            int h = getNewHeight(source.getWidth(), source.getHeight());
            resized = resizeImage(source, DESIRED_WIDTH, h,
                    BufferedImage.TYPE_INT_ARGB);

        } catch (IOException ex) {
            Logger.getLogger(PuzzleEx.class.getName()).log(
                    Level.SEVERE, null, ex);
        }

        width = resized.getWidth(null);
        height = resized.getHeight(null);

        add(panel, BorderLayout.CENTER);

        for (int i = 0; i < 4; i++) {

            for (int j = 0; j < 3; j++) {

                image = createImage(new
FilteredImageSource(resized.getSource(),
                        new CropImageFilter(j * width / 3, i * height / 4,
                                (width / 3), height / 4)));
                MyButton button = new MyButton(image);
                button.putClientProperty("position", new Point(i, j));

                if (i == 3 && j == 2) {
                    lastButton = new MyButton();
                    lastButton.setBorderPainted(false);
                    lastButton.setContentAreaFilled(false);
                    lastButton.setLastButton();
                    lastButton.putClientProperty("position", new Point(i,
j));
                } else {
                    buttons.add(button);
                }
            }
        }

        Collections.shuffle(buttons);
        buttons.add(lastButton);

        for (int i = 0; i < 12; i++) {

            MyButton btn = buttons.get(i);
            panel.add(btn);
            btn.setBorder(BorderFactory.createLineBorder(Color.gray));
            btn.addActionListener(new ClickAction());

        }

        pack();
        setTitle("Puzzle");
        setResizable(false);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private int getNewHeight(int w, int h) {

        double ratio = DESIRED_WIDTH / (double) w;
        int newHeight = (int) (h * ratio);
        return newHeight;
    }

    private BufferedImage loadImage() throws IOException {

        BufferedImage bimg = ImageIO.read(new File("icesid.jpg"));

        return bimg;
    }

    private BufferedImage resizeImage(BufferedImage originalImage, int
width,
            int height, int type) throws IOException {

        BufferedImage resizedImage = new BufferedImage(width, height, type);
        Graphics2D g = resizedImage.createGraphics();
        g.drawImage(originalImage, 0, 0, width, height, null);
        g.dispose();

        return resizedImage;
    }

    private class ClickAction extends AbstractAction {

        @Override
        public void actionPerformed(ActionEvent e) {

            checkButton(e);
            checkSolution();
        }

        private void checkButton(ActionEvent e) {
            int lidx = 0;
            for (MyButton button : buttons) {
                if (button.isLastButton()) {
                    lidx = buttons.indexOf(button);
                }
            }

            JButton button = (JButton) e.getSource();
            int bidx = buttons.indexOf(button);

            if ((bidx - 1 == lidx) || (bidx + 1 == lidx)
                    || (bidx - 3 == lidx) || (bidx + 3 == lidx)) {
                Collections.swap(buttons, bidx, lidx);
                updateButtons();
            }
        }

        private void updateButtons() {

            panel.removeAll();

            for (JComponent btn : buttons) {

                panel.add(btn);
            }

            panel.validate();
        }
    }

    private void checkSolution() {

        ArrayList<Point> current = new ArrayList();

        for (JComponent btn : buttons) {
            current.add((Point) btn.getClientProperty("position"));
        }

        if (compareList(solution, current)) {
            JOptionPane.showMessageDialog(panel, "Finished",
                    "Congratulation", JOptionPane.INFORMATION_MESSAGE);
        }
    }
    public static boolean compareList(List ls1, List ls2) {
        return ls1.toString().contentEquals(ls2.toString());
    }

    public static void main(String[] args) {

        EventQueue.invokeLater(new Runnable() {

            @Override
            public void run() {
                PuzzleEx puzzle = new PuzzleEx();
                puzzle.setVisible(true);
            }
        });
    }
}
```

We use an image of a Sid character from the Ice Age movie. We scale the image and cut it into 12 pieces. These pieces are used by `JButton` components. The last piece is not used; we have an empty button instead. You can download some reasonably large picture and use it in this game.

```
addMouseListener(new MouseAdapter() {

    @Override
    public void mouseEntered(MouseEvent e) {
        setBorder(BorderFactory.createLineBorder(Color.yellow));
    }

    @Override
    public void mouseExited(MouseEvent e) {
        setBorder(BorderFactory.createLineBorder(Color.gray));

    }
});
```

When we hover a mouse pointer over the button, its border changes to yellow colour.

```
public boolean isLastButton() {

    return isLastButton;
}
```

There is one button that we call the last button. It is a button that does not have an image. Other buttons swap space with this one.

```
private final int DESIRED_WIDTH = 300;
```

The image that we use to form is scaled to have the desired width. With the `getNewHeight()` method we calculate the new height, keeping the image's ratio.

```
solution.add(new Point(0, 0));
solution.add(new Point(0, 1));
solution.add(new Point(0, 2));
solution.add(new Point(1, 0));
...
```

The solution array list stores the correct order of buttons which forms the image. Each button is identified by one `Point`.

```
panel.setLayout(new GridLayout(4, 3, 0, 0));
```

We use a `GridLayout` to store our components. The layout consists of 4 rows and 3 columns.

```
image = createImage(new FilteredImageSource(resized.getSource(),
        new CropImageFilter(j * width / 3, i * height / 4,
                (width / 3), height / 4)));
```

`CropImageFilter` is used to cut a rectangular shape from the already resized image source. It is meant to be used in conjunction with a `FilteredImageSource` object to produce cropped versions of existing images.

```
button.putClientProperty("position", new Point(i, j));
```

Buttons are identified by their `position` client property. It is a point containing the button's correct row and colum position in the picture. These properties are used to find out if we have the correct order of buttons in the window.

```
if (i == 3 && j == 2) {
    lastButton = new MyButton();
    lastButton.setBorderPainted(false);
    lastButton.setContentAreaFilled(false);
    lastButton.setLastButton();
    lastButton.putClientProperty("position", new Point(i, j));
} else {
    buttons.add(button);
}
```

The button with no image is called the last button; it is placed at the end of the grid in the bottom-right corner. It is the button that swaps its position with the adjacent button that is being clicked. We set its `isLastButton` flag with the `setLastButton()` method.

```
Collections.shuffle(buttons);
buttons.add(lastButton);
```

We randomly reorder the elements of the `buttons` list. The last button, i.e. the button with no image, is inserted at the end of the list. It is not supposed to be shuffled, it always goes at the end when we start the Puzzle game.

```
for (int i = 0; i < 12; i++) {

    MyButton btn = buttons.get(i);
    panel.add(btn);
    btn.setBorder(BorderFactory.createLineBorder(Color.gray));
    btn.addActionListener(new ClickAction());
}
```

All the components from the `buttons` list are placed on the panel. We create some gray border around the buttons and add a click action listener.

```
private int getNewHeight(int w, int h) {

    double ratio = DESIRED_WIDTH / (double) w;
    int newHeight = (int) (h * ratio);
    return newHeight;
}
```

The `getNewHeight()` method calculates the height of the image based on the desired width. The image's ratio is kept. We scale the image using these values.

```
private BufferedImage loadImage() throws IOException {

    BufferedImage bimg = ImageIO.read(new File("icesid.jpg"));

    return bimg;
}
```

A JPG image is loaded from the disk. `ImageIO's` `read()` method returns a `BufferedImage`, which is Swing's important class for manipulating images.

```
private BufferedImage resizeImage(BufferedImage originalImage, int width,
        int height, int type) throws IOException {

    BufferedImage resizedImage = new BufferedImage(width, height, type);
    Graphics2D g = resizedImage.createGraphics();
    g.drawImage(originalImage, 0, 0, width, height, null);
    g.dispose();

    return resizedImage;
}
```

The original image is resized by creating a new `BufferedImage` with new dimensions. We paint from the original image into this new buffered image.

```
private void checkButton(ActionEvent e) {

    int lidx = 0;
    for (MyButton button : buttons) {
        if (button.isLastButton()) {
            lidx = buttons.indexOf(button);
        }
    }

    JButton button = (JButton) e.getSource();
    int bidx = buttons.indexOf(button);

    if ((bidx - 1 == lidx) || (bidx + 1 == lidx)
            || (bidx - 3 == lidx) || (bidx + 3 == lidx)) {
        Collections.swap(buttons, bidx, lidx);
        updateButtons();
    }
}
```

Buttons are stored in an array list. This list is then mapped to the grid of the panel. We get the indexes of the last button and the clicked button. They are swapped using the `Collections.swap()` if they are adjacent.

```
private void updateButtons() {

    panel.removeAll();

    for (JComponent btn : buttons) {

        panel.add(btn);
    }

    panel.validate();
}
```

The `updateButtons()` method maps the list to the panel's grid. First, all components are removed with the `removeAll()` method. A for loop is used to go trough the `buttons` list to add the reordered buttons back to the panel's layout manager. Finally, the `validate()` method implements the new layout.

```
private void checkSolution() {

    ArrayList<Point> current = new ArrayList();

    for (JComponent btn : buttons) {
        current.add((Point) btn.getClientProperty("position"));
    }

    if (compareList(solution, current)) {
        JOptionPane.showMessageDialog(panel, "Finished",
                "Congratulation", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Solution checking is done by comparing the list of points of the correctly ordered buttons with the current list containg the order of buttons from the window. A message dialog is shown in case the solution is reached.

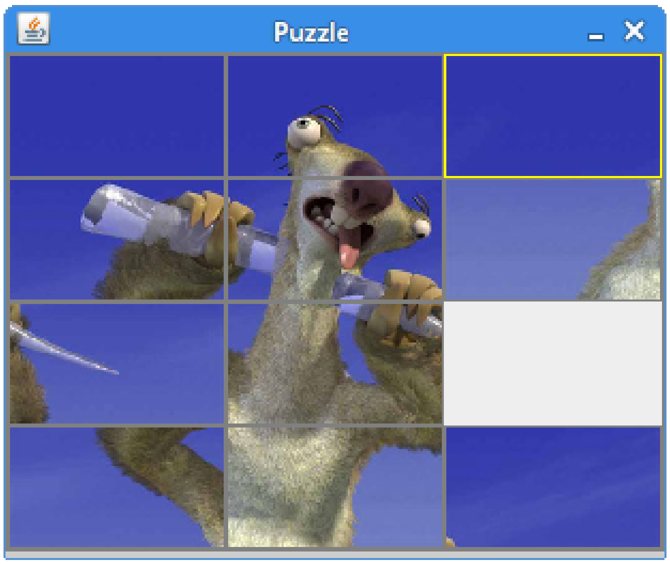

Figure: Puzzle

This was a puzzle game.