PICTURE-PUZZLE

Java Project

Nayan Shah (08412EN008)

Shishir Kumar (08412EN002)

IMD Part III

Mathematics & Computing

IT-BHU (Varanasi)

Under the guidance of :

Dr. L.P.Singh,
Professor, Applied Mathematics

Contents :

- ✓ Project Description
- ✓ Technical Details
- ✓ Source Code

Project Description

We have an image, the goal of this little game is to form the original picture from 16 rectangular pieces of that image. Code is written in Java for solving this Picture- Puzzle game.

RULES OF THE GAME

When the game starts a picture is cut into a number (typically 16) of rectangular pieces (all of the same size). The piece on the bottom right is removed. The remaining pieces must then be randomly shuffled around by **New** in menu bar. To solve the puzzle you have to recreate the original picture again by sliding pieces into the empty spot.

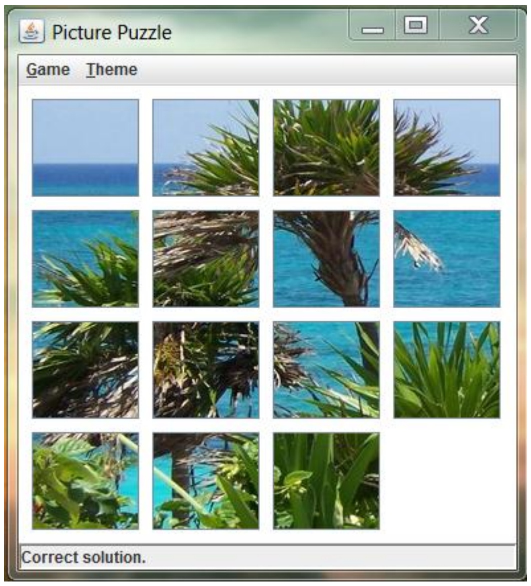
MOUSE CONTROL

If you click a piece surrounding the empty spot it will slide into the empty spot. You can move any suitable piece with one click as expected.

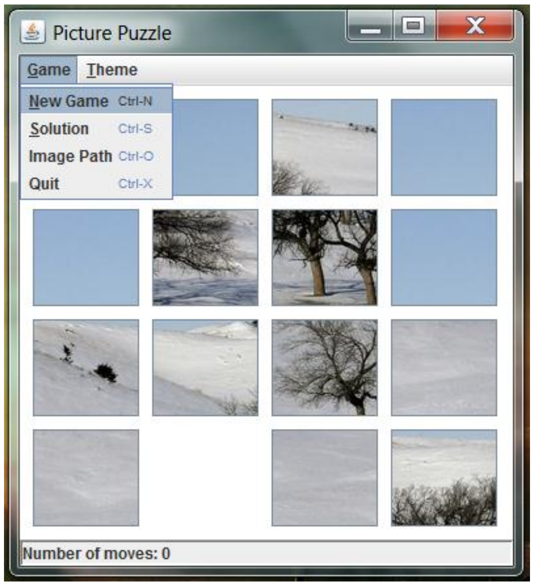
KEYBOARD CONTROL

Ctrl + N	Starts a new game by shuffling the images.
Ctrl + S	Shows the correct solution.
Ctrl + O	To the change the background picture of the puzzle.
Ctrl + X	Exits the game.

SCREENSHOTS



Correct solution.



Number of moves: 0

Technical Details

DISTRIBUTION

An Executable Jar file is provided, **PicturePuzzle.jar**

PLATFORMS

Runs on all platforms with JRE : Linux, Windows, Macintosh

EDITOR

Notepad++ was the primary editor.

COMPILER

JDK 1.6

IMAGE PROCESSING

ImageMagick (software for both Linux and Windows) was used for cropping, slicing and resizing new background images.

VERSION CONTROL

Git was used for version control during development.

PACKAGES

javax.swing.*	(For GUI)
Javax.swing.event.*	(Event Handling)
java.util.Collections	(Randomization)

MAIN CLASSES

PicturePuzzle	(Main class)
ButtonEvents	(Handling mouse actions)

Source Code

```
package com.whiz;
import java.awt.Point;
import java.awt.Rectangle;
import java.util.*;
import javax.swing.*;
import javax.swing.event.MouseInputAdapter;

public class PicturePuzzle extends JFrame {

        private class ButtonEvents extends MouseInputAdapter {

                private Rectangle dragSourceArea = null;
                private Rectangle dragTargetArea = null;
                private Rectangle panelArea = null;
                private Point cursorOffset = null;
                private boolean isClick;

                public void mousePressed(java.awt.event.MouseEvent e) {
                        isClick = true;
                        JButton b = (JButton) e.getSource();
                        // check to see if the target button can be moved
                        if (!solved && check(buttons.indexOf(b))) {
                                // hide the target button
                                b.setVisible(false);

                                // Figure out the bounds of the areas where source
                                // and target buttons are located
                                int menuOffset = getJMenuBar().getHeight();
                                dragSourceArea = b.getBounds();
                                dragTargetArea = ((JButton) buttons.get(hiddenIndex))
                                                .getBounds();
                                dragSourceArea.translate(0, menuOffset);
                                dragTargetArea.translate(0, menuOffset);

                                // setup the bounds of the panel to limit the locations on the drag
                                // layer
                                panelArea = new Rectangle(0, menuOffset,
                                                getJPanel().getWidth(), getJPanel().getHeight());

                                // Setup and show the drag button on the upper layer
                                getDragButton().setText(b.getText());
                                getDragButton().setBounds(dragSourceArea);
                                getDragButton().setVisible(true);

                                // Offset when repositioning the drag button later
                                cursorOffset = new Point(e.getX(), e.getY());
                        }
                }

                public void mouseDragged(java.awt.event.MouseEvent e) {
```

```
		if (dragTargetArea != null) {
			// Since we're dragging, this is no longer considered a click
			isClick = false;

			// Draw the target feedback and position the drag button based
			// on the mouse's new location
			e.translatePoint(dragSourceArea.x, dragSourceArea.y);
			Point p = makeSafePoint(e.getX(), e.getY());
			paintTargetFeedback(p.x, p.y);
			getDragButton().setLocation(p.x - cursorOffset.x,
					p.y - cursorOffset.y);
		}
	}

	public void mouseReleased(java.awt.event.MouseEvent e) {
		if (dragTargetArea != null) {

			// Turn off the drag button and feedback
			getDragButton().setVisible(false);
			paintTargetFeedback(-1, -1);

			e.translatePoint(dragSourceArea.x, dragSourceArea.y);
			Point p = makeSafePoint(e.getX(), e.getY());

			JButton b = (JButton) e.getSource();
			if (isClick || dragTargetArea.contains(p.x, p.y)) {
				// if we're considered a simple click, or the dragging ended
				// within the target area, perform the move.
				slide(buttons.indexOf(b));
			} else {
				// The drag finished outside the target, so just show the
				// hidden button and don't move anything.
				b.setVisible(true);
			}
			dragTargetArea = null;
			dragSourceArea = null;
		}
	}

	/**
	 * Position the target feedback if necessary for
	 * the given cursor position
	 *
	 * @param x cursor position x
	 * @param y cursor position y
	 */
	private void paintTargetFeedback(int x, int y) {
		if (dragSourceArea.contains(x, y)) {
			// If the cursor is in the source area, move the feedback
			// to the source area.
			getHighlightBorder().setBounds(dragSourceArea);
			getHighlightBorder().setVisible(true);
		} else if (dragTargetArea.contains(x, y)) {
				// If the cursor is in the target area, move the feedback
				// to the target area.
				getHighlightBorder().setBounds(dragTargetArea);
				getHighlightBorder().setVisible(true);
			} else {
				// Otherwise just hide the feedback
				getHighlightBorder().setVisible(false);
			}
		}

		/**
		 * Translate the given mouse cursor position into a
		 * point safely within the parent panel area.
		 *
		 * @param cx cursor position x
		 * @param cy cursor position y
		 * @return a safe point
		 */
		private Point makeSafePoint(int cx, int cy) {
			Point op = new Point(cx, cy);
			Rectangle t = new Rectangle(cx - cursorOffset.x, cy
					- cursorOffset.y, cx - cursorOffset.x
					+ dragSourceArea.width, cy - cursorOffset.y
					+ dragSourceArea.height);

			// Check to see if the entire bounds is within
			// panel area
			if (panelArea.contains(t)) {
				return op;
			}

			// if the drag top is above the panel top
			if (t.x < panelArea.x) {
				t.x = panelArea.x;
			}
			// if the drag bottom is below the panel bottom
			if (t.width > panelArea.x + panelArea.width) {
				t.x = panelArea.x + panelArea.width - dragSourceArea.width;
			}
			// if the drag left is before the panel left
			if (t.y < panelArea.y) {
				t.y = panelArea.y;
			}
			// if the drag right is after the panel right
			if (t.height > panelArea.y + panelArea.height) {
				t.y = panelArea.y + panelArea.height - dragSourceArea.height;
			}

			t.x += cursorOffset.x;
			t.y += cursorOffset.y;

			return t.getLocation();
		}
```

```
	}

	private ButtonEvents buttonEvents = new ButtonEvents();

	private JPanel jContentPane = null;
	private JPanel jPanel = null;
	private JLabel statusLabel = null;
	private JButton jButtons[] = new JButton[16];
	private ImageIcon icons[] = new ImageIcon[16];
	private JMenuBar jJMenuBar = null;
	private JMenu gameMenu = null;
	private JMenuItem jMenuItem = null;
	private JMenuItem jMenuItem1 = null;
	private JMenuItem jMenuItem2 = null;
	private JMenuItem jMenuItem3 = null;
	private JMenu jMenu = null;
	private JFileChooser jFileChooser = null;
	private JRadioButtonMenuItem jRadioButtonMenuItem = null;
	private JRadioButtonMenuItem jRadioButtonMenuItem1 = null;
	private JButton dragButton = null;
	private JPanel highlightBorder = null;

	private ArrayList buttons;
	private ArrayList<String> correct;
	private ArrayList<String> current;

	private int hiddenIndex;

	private int moves = 0;
	private boolean solved;
	private boolean solution;

private String path = "/home/nayan/prog/Puzzle/assets/ocean/";

	public static void main(String[] args) {
		new PicturePuzzle().setVisible(true);
	}

	/**
	 * This is the default constructor
	 */
	public PicturePuzzle() {
		super();
		initialize();

	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {
		this.setSize(380, 400);
		this.setContentPane(getJContentPane());

		// Add the drag button and feedback square to the drag layer of the
		// frame's LayeredPane.
		this.getLayeredPane().add(getDragButton(), JLayeredPane.DRAG_LAYER);
		this.getLayeredPane()
				.add(getHighlightBorder(), JLayeredPane.DRAG_LAYER);

		this.setJMenuBar(getJJMenuBar());
		this.setTitle("Picture Puzzle");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		// Set up the lists of buttons, correct orders and current order
		buttons = new ArrayList<Object>(Arrays.asList(getJPanel().getComponents()));
		correct = new ArrayList<String>(16);
		for (int i = 1; i <= 16; i++) {
		   // Add the icons in correct order
		   icons[i-1] = new ImageIcon(path + i + ".jpg");
			correct.add(String.valueOf(i));
		}
		current = new ArrayList<String>(correct);
		solution = false;
		shuffle();
	}

	/**
	 * Shuffle the numbers on the buttons
	 */
	private void shuffle() {
		// Randomize the order
		if(solution)
		   solution = false;
		else
		Collections.shuffle(current);

		// Reset the stats
		moves = 0;
		solved = false;

		// Reset each of the buttons to reflect the randomized
		// order
		for (int i = 0; i < buttons.size(); i++) {
			JButton b = (JButton) buttons.get(i);
			String value = (String) current.get(i);
			int val = Integer.parseInt(value) - 1;
			b.setText(value);
			b.setBackground(java.awt.Color.orange);
			b.setIcon(icons[val]);
			if (value.equals("16")) {
				b.setVisible(false);
				hiddenIndex = i;
			} else {
```

```
                        b.setVisible(true);
                    }
            }

            // Reset the status line
            getStatusLabel().setText("Number of moves: " + moves);
    }

    /**
     * Check to see if the button at the given index can be moved
     * @param index the button to check
     * @return true if the button can move, false otherwise
     */
    private boolean check(int index) {

            if (index < 0 || index > buttons.size() - 1)
                    return false;

            boolean valid = false;
            // check up
            if (index > 3) {
                    valid = (hiddenIndex == index - 4);
            }
            // check down
            if (valid == false && index <= 11) {
                    valid = (hiddenIndex == index + 4);
            }
            // check right
            if (valid == false && (index % 4) != 3) {
                    valid = (hiddenIndex == index + 1);
            }
            // check left
            if (valid == false && (index % 4) != 0) {
                    valid = (hiddenIndex == index - 1);
            }
            return valid;
    }

    /**
     * React to the pushing of the given button
     *
     * @param index  The number of the button pushed
     */
    private void slide(int index) {

            // Don't do anything if the puzzle is solved
            if (solved)
                    return;

            // if not a valid click return
            if (!check(index)) {
                    return;
            }
```

```
		// swap positions
		current.set(index, current.set(hiddenIndex, current.get(index)));

		// swap strings
		JButton b = (JButton) buttons.get(index);
		b.setText((String) current.get(index));
		b.setIcon(icons[Integer.parseInt(current.get(index)) - 1]);
		b.setVisible(false);
		b = (JButton) buttons.get(hiddenIndex);
		b.setText((String) current.get(hiddenIndex));
		b.setIcon(icons[Integer.parseInt(current.get(hiddenIndex)) - 1]);
		b.setVisible(true);

		// update the position of the blanked spot
		hiddenIndex = index;

		// Increment the number of moves and update status
		moves++;
		getStatusLabel().setText("Number of moves: " + moves);

		// if you've won
		if (current.equals(correct)) {
			solved = true;
			getStatusLabel().setText("Solved the game in " + moves + " moves.");
			// Change the buttons colors to green
			Iterator itr = buttons.iterator();
			while (itr.hasNext()) {
				((JButton) itr.next()).setBackground(java.awt.Color.green);
			}
		}
	}

	/**
	 * Update the Look and Feel of the game
	 * @param lnf fully qualified look and feel class name
	 */
	private void changeLnF(String lnf) {
		try {
			UIManager.setLookAndFeel(lnf);
			SwingUtilities.updateComponentTreeUI(this);
		} catch (Exception e) {
		}
	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return JPanel
	 */
	private JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new JPanel();
```

```
				jContentPane.setLayout(new java.awt.BorderLayout());
				jContentPane.add(getJPanel(), java.awt.BorderLayout.CENTER);
				jContentPane.add(getStatusLabel(), java.awt.BorderLayout.SOUTH);
			}
			return jContentPane;
		}

	/**
	 * Creates a JButton.
	 * @param Default text for the button
	 * @return The JButton Object
	 */
	private JButton createButton(String text) {
	   JButton b = new JButton();
			b.setText(text);
			b.setBackground(java.awt.Color.orange);
			b.addMouseListener(buttonEvents);
			b.addMouseMotionListener(buttonEvents);
			return b;
	}

		/**
		 * This method initializes jJMenuBar
		 *
		 * @return JMenuBar
		 */
		private JMenuBar getJJMenuBar() {
			if (jJMenuBar == null) {
				jJMenuBar = new JMenuBar();
				jJMenuBar.add(getGameMenu());
				jJMenuBar.add(getJMenu());
			}
			return jJMenuBar;
		}

		/**
		 * This method initializes gameMenu
		 *
		 * @return JMenu
		 */
		private JMenu getGameMenu() {
			if (gameMenu == null) {
				gameMenu = new JMenu();
				gameMenu.add(getJMenuItem());
				gameMenu.add(getJMenuItem3());
				gameMenu.add(getJMenuItem2());
				gameMenu.add(getJMenuItem1());
				gameMenu.setText("Game");
				gameMenu.setMnemonic(java.awt.event.KeyEvent.VK_G);
			}
			return gameMenu;
		}
```

```
/**
 * This method initializes jMenuItem
 *
 * @return JMenuItem
 */
private JMenuItem getJMenuItem() {
		if (jMenuItem == null) {
				jMenuItem = new JMenuItem();
				jMenuItem.setText("New Game");
				jMenuItem.setMnemonic(java.awt.event.KeyEvent.VK_N);
				jMenuItem.setAccelerator(javax.swing.KeyStroke.getKeyStroke(
								java.awt.event.KeyEvent.VK_N, java.awt.Event.CTRL_MASK,
								false));
				jMenuItem.addActionListener(new java.awt.event.ActionListener() {
						public void actionPerformed(java.awt.event.ActionEvent e) {
								shuffle();
						}
				});
		}
		return jMenuItem;
}

/**
 * This method initializes jMenuItem1
 *
 * @return JMenuItem
 */
private JMenuItem getJMenuItem1() {
		if (jMenuItem1 == null) {
				jMenuItem1 = new JMenuItem();
				jMenuItem1.setText("Quit");
				jMenuItem1.setMnemonic(java.awt.event.KeyEvent.VK_X);
				jMenuItem1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(
								java.awt.event.KeyEvent.VK_X, java.awt.Event.CTRL_MASK,
								false));
				jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
						public void actionPerformed(java.awt.event.ActionEvent e) {
								System.exit(0);
						}
				});
		}
		return jMenuItem1;
}

private JMenuItem getJMenuItem2() {
		if (jMenuItem2 == null) {
				jMenuItem2 = new JMenuItem();
				jMenuItem2.setText("Image Path");
				jMenuItem2.setMnemonic(java.awt.event.KeyEvent.VK_O);
				jMenuItem2.setAccelerator(javax.swing.KeyStroke.getKeyStroke(
								java.awt.event.KeyEvent.VK_O, java.awt.Event.CTRL_MASK,
								false));
				jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
```

```
                                        public void actionPerformed(java.awt.event.ActionEvent e) {
                                                //System.exit(0);
                                                jFileChooser = new JFileChooser();
                                                jFileChooser.setDialogTitle("Select image directory.");
                                                jFileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
                                                jFileChooser.setAcceptAllFileFilterUsed(false);
                                                if (jFileChooser.showOpenDialog(jPanel) ==
JFileChooser.APPROVE_OPTION) {
            path = jFileChooser.getSelectedFile() + "/";
                                            solution = true;
                                current = new ArrayList<String>(correct);
                                for (int i = 1; i <= 16; i++) {
                                  // Add the icons in correct order
                                  icons[i-1] = new ImageIcon(path + i + ".jpg");
                                }
                                shuffle();
                                        getStatusLabel().setText("New game started.");
                                        // Change the buttons colors to green
                                        Iterator itr = buttons.iterator();
                                        while (itr.hasNext()) {
                                                ((JButton) itr.next()).setBackground(java.awt.Color.green);
                                        }
                                      }
                                          }
                                });
                        }
                        return jMenuItem2;
                }

                private JMenuItem getJMenuItem3() {
                        if (jMenuItem3 == null) {
                                jMenuItem3 = new JMenuItem();
                                jMenuItem3.setText("Solution");
                                jMenuItem3.setMnemonic(java.awt.event.KeyEvent.VK_S);
                                jMenuItem3.setAccelerator(javax.swing.KeyStroke.getKeyStroke(
                                                java.awt.event.KeyEvent.VK_S, java.awt.Event.CTRL_MASK,
                                                false));
                                jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
                                        public void actionPerformed(java.awt.event.ActionEvent e) {
                                          solution = true;
                        current = new ArrayList<String>(correct);
                                          shuffle();
                              getStatusLabel().setText("Correct solution.");
                              // Change the buttons colors to green
                              Iterator itr = buttons.iterator();
                              while (itr.hasNext()) {
                                      ((JButton) itr.next()).setBackground(java.awt.Color.green);
                              }
                                        }
                                });
                        }
                        return jMenuItem3;
                }
```

```
    /**
     * This method initializes jPanel
     *
     * @return JPanel
     */
    private JPanel getJPanel() {
            if (jPanel == null) {
                    jPanel = new JPanel();
                    java.awt.GridLayout layGridLayout7 = new java.awt.GridLayout();
                    layGridLayout7.setRows(4);
                    layGridLayout7.setColumns(4);
                    layGridLayout7.setHgap(10);
                    layGridLayout7.setVgap(10);
                    jPanel.setLayout(layGridLayout7);

                    // Another way to do this, but wouldn't
                    // have a visual in the editor.
                    //      for (int i = 1; i <= 16; i++) {
                    //              String label = String.valueOf(i);
                    //              JButton b = new JButton(label);
                    //              b.addMouseListener(buttonEvents);
                    //              b.addMouseMotionListener(buttonEvents);
                    //              jPanel.add(b);
                    //      }

   for(int i = 0; i < jButtons.length; i++) {
                jButtons[i] = createButton(String.valueOf(i+1));
    jPanel.add(jButtons[i], null);
   }

                    jPanel.setBackground(java.awt.Color.white);
                    jPanel.setBorder(javax.swing.BorderFactory.createEmptyBorder(10,
                                    10, 10, 10));
            }
            return jPanel;
    }

    /**
     * This method initializes statusLabel
     *
     * @return JLabel
     */
    private JLabel getStatusLabel() {
            if (statusLabel == null) {
                    statusLabel = new JLabel();
                    statusLabel.setText("Number of moves: 0");
                    statusLabel.setBorder(javax.swing.BorderFactory
                                    .createBevelBorder(javax.swing.border.BevelBorder.LOWERED));
            }
            return statusLabel;
    }
```

```
/**
 * This method initializes jMenu
 *
 * @return JMenu
 */
private JMenu getJMenu() {
		if (jMenu == null) {
				jMenu = new JMenu();
				jMenu.add(getJRadioButtonMenuItem());
				jMenu.add(getJRadioButtonMenuItem1());
				jMenu.setText("Theme");
				jMenu.setMnemonic(java.awt.event.KeyEvent.VK_T);
				ButtonGroup bg = new ButtonGroup();
				bg.add(getJRadioButtonMenuItem());
				bg.add(getJRadioButtonMenuItem1());
		}
		return jMenu;
}

/**
 * This method initializes jRadioButtonMenuItem
 *
 * @return JRadioButtonMenuItem
 */
private JRadioButtonMenuItem getJRadioButtonMenuItem() {
		if (jRadioButtonMenuItem == null) {
				jRadioButtonMenuItem = new JRadioButtonMenuItem();
				jRadioButtonMenuItem.setText("Default");
				jRadioButtonMenuItem.setSelected(true);
				jRadioButtonMenuItem.setAccelerator(javax.swing.KeyStroke
								.getKeyStroke(java.awt.event.KeyEvent.VK_D,
										java.awt.Event.ALT_MASK, false));
				jRadioButtonMenuItem.setMnemonic(java.awt.event.KeyEvent.VK_D);
				jRadioButtonMenuItem
								.addActionListener(new java.awt.event.ActionListener() {
									public void actionPerformed(java.awt.event.ActionEvent e) {
										changeLnF(UIManager

.getCrossPlatformLookAndFeelClassName());
									}
								});
		}
		return jRadioButtonMenuItem;
}

/**
 * This method initializes jRadioButtonMenuItem1
 *
 * @return JRadioButtonMenuItem
 */
private JRadioButtonMenuItem getJRadioButtonMenuItem1() {
		if (jRadioButtonMenuItem1 == null) {
				jRadioButtonMenuItem1 = new JRadioButtonMenuItem();
				jRadioButtonMenuItem1.setText("System");
				jRadioButtonMenuItem1.setMnemonic(java.awt.event.KeyEvent.VK_Y);
				jRadioButtonMenuItem1.setAccelerator(javax.swing.KeyStroke
						.getKeyStroke(java.awt.event.KeyEvent.VK_Y,
								java.awt.Event.ALT_MASK, false));
				jRadioButtonMenuItem1
						.addActionListener(new java.awt.event.ActionListener() {
							public void actionPerformed(java.awt.event.ActionEvent e) {

		changeLnF(UIManager.getSystemLookAndFeelClassName());
							}
						});
			}
			return jRadioButtonMenuItem1;
		}

		/**
		 * This method initializes dragButton
		 *
		 * @return JButton
		 */
		private JButton getDragButton() {
			if (dragButton == null) {
				dragButton = new JButton();
				dragButton.setVisible(false);
				dragButton.setBackground(new java.awt.Color(51, 102, 255));
			}
			return dragButton;
		}

		/**
		 * This method initializes highlightBorder
		 *
		 * @return JPanel
		 */
		private JPanel getHighlightBorder() {
			if (highlightBorder == null) {
				highlightBorder = new JPanel();
				highlightBorder.setOpaque(false);
				highlightBorder.setBorder(javax.swing.BorderFactory
						.createLineBorder(java.awt.Color.black, 2));
				highlightBorder.setBackground(java.awt.Color.white);
				highlightBorder.setVisible(false);
			}
			return highlightBorder;
		}
	}
```