The 8 Puzzle report

DAVIDE ORSUCCI

The result of the 8 puzzle project is shown in figures 1, 2 and 3 which depict how the 8 puzzle works.

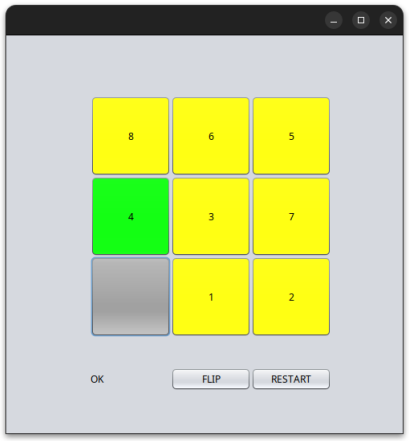

Figure 1: The tile in the correct position becomes green.

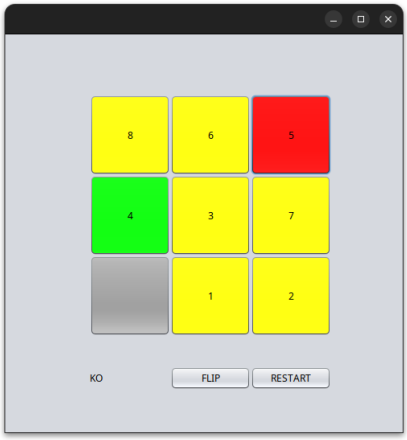

Figure 2: An illegal move turns the tile red one second.

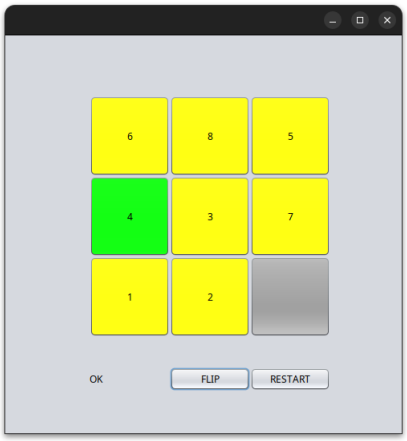

Figure 3: When the hole is in the ninth position, flip can be performed.

The project is made up by three files that can be found inside */src/main/java/com/mycompany/eightboard* folder:

EightTile.java : EightTile is a JavaBean which extends the *JButton* class. The EightTile is made up of two properties: *label* and *position*. At the beginning of each game the tiles are reinitialized using the *restart* method which initializes the tiles using an array containing the permutation of the labels. When the *restart* button is pressed, each EightTile manages the event by executing the following handler:

```
135 @Override
136 public void actionPerformed(ActionEvent ae) {
137     // retrieve the Array from the the restartButton's properties
138     JButton button = (JButton) ae.getSource();
139     if(button.getActionCommand().equals("restart")){
140         int[] retrievedArray = (int[]) button.getClientProperty("labels");
141         this.restart(retrievedArray);
142     }
143 }
```

EightController.java : EightController is a java Bean that extends *JLabel*. It implements the logic of the game by saving only the position of the hole in the private property *holePosition*. As requested, it implements the *vetoableChange* that checks if the player's move is legal or not by using the *isAdjent* method as depicted below.

```
67 @Override
68 public void vetoableChange(PropertyChangeEvent evt) throws PropertyVetoException {
69     String propertyName = evt.getPropertyName();
70     if ("label".equals(propertyName)) {
71         EightTile tile = (EightTile) evt.getSource();
72         int tilePosition = tile.getPosition();
73         // Check if the tile is the hole or not adjacent to the hole
74         if (tilePosition == holePosition || !isAdjacent(tilePosition, holePosition)) {
75             setText("KO");
76             throw new PropertyVetoException("Illegal move", evt);
77         } else {
78             setText("OK");
79             this.holePosition = tilePosition;
80         }
81     }
82 }
```

EightController does the following actions:

- it vetoes the change if the tile is the hole or it is not adjacent to the hole by displaying “KO”. Otherwise it shows ”OK” and allows the player’s move. This is done using the methods *vetoableChange* and *isAdjent* which uses the position (not the label) of the tiles to check if the user’s move is allowed. If the veto is positive, the action is performed by the tiles and the EightController updates the *holePosition* property.
- it is registered as an action listener of the restart button, so when the game is restarted, the *holePosition* is reinitialized and the EightController displays ”START” as a new game.

EightBoard.java extends the *JFrame* class, it’s the entry point of the program and creates the GUI using and manages the tiles and the buttons as requested. The EightBoard presents a constructor which initializes the various elements of the board and also defines the actions to be performed when each EighTile is pressed and when the *restart* and *flip* buttons are pressed.

The Flip Button

The flip button is created inside the *initComponents()* method of EightBoard.java.

```
128 flip = new javax.swing.JButton();
```

After the creation the following code is executed:

```
201 flip.setText("FLIP");
202 flip.addActionListener(new java.awt.event.ActionListener() {
203     public void actionPerformed(java.awt.event.ActionEvent evt) {
204         flipActionPerformed(evt);
205     }
206 });
```

line 201 : sets the text to be shown on the button;

line 202 : adds the action listener to the button. When the button is pressed, the *flipActionPerformed* method is called which contains the logic of the flip button.

After creating the button and adding the action listener to by calling *initComponents()*, the *EightBoard()* executes the following fragment of code:

```
54 // setup the flip button
55 flip.setActionCommand("flip");
56 flip.putClientProperty("eightTile1", eightTile1);
57 flip.putClientProperty("eightTile2", eightTile2);
```

line 54 : sets the action command of the action associated with the ”flip” component to the flip button.

line 55 and 56 : add the two client properties eightTile1 and eightTile2 to the flip component, each one with its own key so the 2 properties can be retrieved by the action listener.

Finally, the *EightBoard* implements the code of *flipActionPerformed* which contains the logic of the flip button.

```
303 private void flipActionPerformed(java.awt.event.ActionEvent evt) {
304     if(eightController1.getHolePosition() == 9){
305         EightTile t1 = (EightTile) flip.getClientProperty("eightTile1");
306         EightTile t2 = (EightTile) flip.getClientProperty("eightTile2");
307         String tempText = t1.getText();
308         t1.setText(t2.getText());
309         t2.setText(tempText);
310     }
311 }
```

line 303 : gets the source of the action performed

line 304 : checks if the flip is legal by calling the method *getHolePosition* of the controller, so that if the hole is in position 9 the flip can be performed, otherwise no.

line 305-309 if *holePosition* is 9, the swap between the two EighTiles’s can be performed, otherwise no.