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.

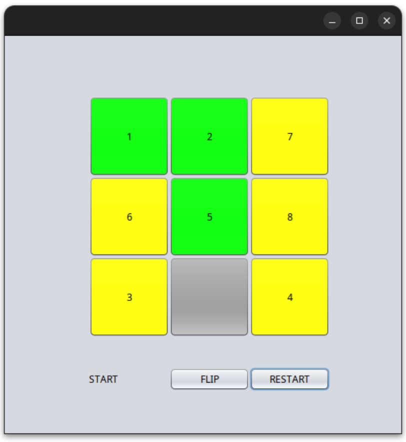

Figure 1: When the game starts the game shows "START".

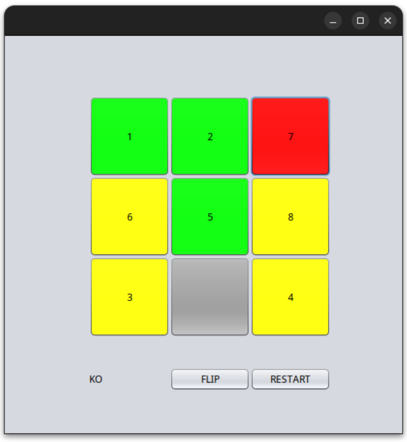

Figure 2: An illegal move turns the tile red and displays "KO".

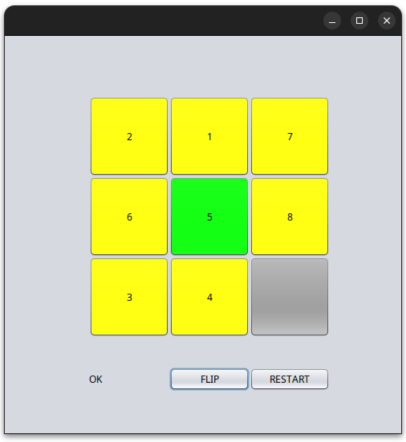

Figure 3: The hole is in the ninth position allows the flip.

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* with their setters and getters. At the beginning of each game labels of the tiles are setted using the *setTileLabel* method which takes the new label to be assigned after shuffling the labels. When the *restart* button is pressed, the action triggers the execution of the EightTile's *actionPerfomed* method which sets the new label according to the position of the tile property. The *MoveTile* method is called by *eightTileListener* defined on the *EightBoard* when the Tile is clicked and assigns the new label to the tile if it is not vetoed by the *EightController* (as described below).

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.

```java
@Override
public void vetoableChange(PropertyChangeEvent evt) throws PropertyVetoException {
    String propertyName = evt.getPropertyName();
    if ("label".equals(propertyName)) {
        EightTile tile = (EightTile) evt.getSource();
        int tilePosition = tile.getPosition();
        if (tilePosition == holePosition || !isAdjacent(tilePosition, holePosition)) {
            setText("KO");
            throw new PropertyVetoException("Illegal move", evt);
        } else {
            setText("OK");
            this.holePosition = tilePosition;
        }
    }
}
```

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 and it's the entry point of the program. The EightBoard is responsible for the initialization and the configuration of the various elements of the GUI and manages the events of clicking the *restart* and each *EightTile* buttons. In particular when a tile is clicked, it triggers the execution actionPerformed code associated to the *eightTileListener* which calls the *moveTile* method on the moved tile, assigns the label of the pressed tile to the *hole* and promotes the pressed tile as new *hole* (if it is not vetoed by the *EightController*, as described before:

```java
@Override
public void actionPerformed(ActionEvent evt) {
    try {
        EightTile clickedTile = (EightTile) evt.getSource();
        int oldLabel = clickedTile.getTileLabel();
        clickedTile.moveTile(9);
        hole.setTileLabel(oldLabel);
        hole = clickedTile;
    } catch (InterruptedException ex) { }
}
```

The Flip Button

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

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

After the initialization of the components is complete, the code sets the "flip" as the action command of the button and adds 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:

```java
// setup the flip button
flip.setActionCommand("flip");
flip.putClientProperty("eightTile1", eightTile1);
flip.putClientProperty("eightTile2", eightTile2);
```

At the end of the constructor the flip button adds the EightController as an action listener of the *flip* button by running the following line of code, so that when the user clicks the *flip* button the EightController will take care of it.

```java
flip.addActionListener(eightController1);
```

Finally, when the flip button is clicked the *actionPerformed* method of the controller is triggered:

```java
82      @Override
83      public void actionPerformed(ActionEvent ae) {
84          JButton button = (JButton) ae.getSource();
85          ...
86          if(button.getActionCommand().equals("flip")){
87              if(holePosition == 9){
88                  EightTile t1 = (EightTile) button.getClientProperty("eightTile1");
89                  EightTile t2 = (EightTile) button.getClientProperty("eightTile2");
90                  int templabel = t1.getTileLabel();
91                  t1.setTileLabel(t2.getTileLabel());
92                  t2.setTileLabel(tempLabel);
93              }
94          }
95      }
```

The code above does the following things:

line 84 : retrieves the button that was clicked and stores it in button;

line 86 : checks if the action is triggered by the flip button.

line 87 : if the empty tile position is in the ninth position, the flip operation can be performed.

lines 88-89 : the controller retrieves the two tiles to be swapped that are stored inside the button's custom properties (as shown before, *putClientProperty()* was used earlier to store them).

lines 90-92 : perform the swap between the labels of the two tiles