Advanced Programming

Eight Puzzle

Marco Antonio Corallo

531466

Contents

Overview

The *8 puzzle* game is a reduced version of the more famous *15 puzzle* game: starting from a random configuration of the tiles, the puzzle consists of reaching the final configuration (the sorted one) with a sequence of moves. Each move consists of sliding one tile on the hole, thus exchanging the positions of that tile and the hole.
The assignment requires to implement the 8 puzzle using *Java Beans* and *event-based* communication mechanisms.
The system is made of a graphical dashboard, *EightBoard*, containing the board, an *EightController* label, and two buttons: *RESTART* and *FLIP*. The board contains 9 tiles *EightTile*.
The Figure 1.1 shows the expected result; my final product is instead shown in Figure 1.2.
This report presents the requirements specification and the main design choices made for each component.
This report also provides an *interaction diagram* to better explain the event-based interaction between the components.

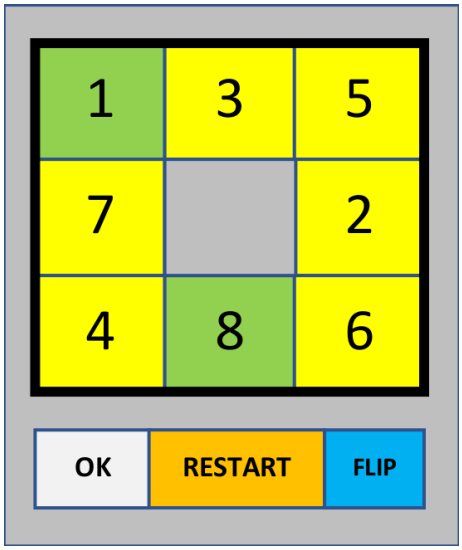


Figure 1.1: Expected result

Figure 1.2: Final result

EightTile

Requirements for the EightTile Bean

The EightTile is a subclass of *JButton*, with two private properties: *position* and *label*, which hold integer values in the range [1, 9]. Position is a constant and it identifies a specific position on the board. The background color of a tile must be **grey** if the label is 9, **green** if position = label (and they are different from 9), and **yellow** otherwise.
The text of the tile must be equal to the value of label, with the exception of the current hole, whose text is empty.
Background color and text of each tile must change if label is changed, in order to preserve these constraints. Note that the in this way the final configuration is the only one where all tiles except the hole are green.
When the player clicks on a tile, the Label property is changed to 9, *if this change is not vetoed*, and the tile must pass its previous label value to the current hole. If instead the change is vetoed, the tile *flashes* changing the color to red for half a second. By the logic of the game, the change of the label should be vetoed if

- the tile is the current hole;
- the tile is not adjacent to the current hole.

For passing the label value of the clicked tile to the old hole are accepted also public methods, but event-based communication is more appreciated.
Tiles must also provide support for a *restart* action, with a method that takes as argument a permutation of [0,9] and sets the label to an initial value. This action will be triggered by the *restart* event of the board, thus tiles have to register as listeners for this event.

Design choices

EightTile doesn't use an auxiliary *VetoableChangeSupport* list, but uses the inherited one. Indeed, EightTile extends *JButton*, that inherits a *VetoableChangeSupport* from *JComponent*.
This choice is mainly due to the method `UpdateUI` of JButton's constructor: the *upcasting* causes a `NPE` invoking the method `addPropertyChange`.
Alternatives could be the *overloading* of these methods with dummy parameters, or the renaming of these methods.

The *getter* method of `label` returns a string value, even if the field is an integer; this is due to a compilation requirement for the overriding. Again, an alternative could be the overloading through a dummy parameter, or the definition of the getter with another name.

The value of the *"hole label"* is stored in a final variable for future extensions (for example the more famous 15 puzzle).

EightController

Requirements for the EightController Bean

The EightController is a bean that extends *JLabel*.
It has to check that only *legal moves* are performed. To this aim, it must be registered as *VetoableChangeListener* to all the tiles.
At the beginning the controller displays *START*. Every time a tile is clicked, it must veto the change if

- the tile is the hole;
- it is not adjacent to the hole,

displaying *KO*. Otherwise, it must display *OK*.
The controller must also provide support for a *restart* action that restores its internal information about the configuration when the game is restarted.
The controller cannot read the labels of the tiles using methods. It must keep internal informations about the configuration of the board to implement correctly the above veto policy. Several choices are possible: the simpler the better.

Design choices

The EightController keeps an inner map `<label, position>` that represents the *initial layout*, updated when a *restart* event is fired. The map is also used for passing the *clicked-tile*'s label to the hole tile. The EightController behaves as an event adaptor: it fires an event to the listener board when a tile is correctly moved, specifying the position of the hole and the label to assign to it.
To do this, the *PropertyChangeSupport* list is overriden: when these

two arguments are equal, the `super.firePropertyChange` does nothing. So I override this method and also the methods `addPropertyChangeListener` and `removePropertyChangeListener`, to avoid the *upcasting* error at the superclass constructors, as specified in the design choices for the EightTile bean.

EightBoard

Requirements for the EightBoard bean

The main class of the application is the EightBoard dashboard, that must be defined as extending *JFrame*. It displays a grid of *3x3 tiles*, an *EightController* label, a *RESTART* button, and a *FLIP* button. When the *RESTART* button is clicked, a random configuration is passed to all beans registered as listener to the *restart* event. At startup the board initializes the grid with the nine EightTile beans, passing to them their position as initial value of `position`. All tiles and the controller are registered as listeners to the *restart* event, and such an event is fired to complete the initialization of the board with a random configuration.
Also, the controller is registered as *VetoableChangeListener* to all the tiles.

Design choices

Like EightTile, also the board reuses an inherited listener list. In this case it inherits *ChangePropertyListener* from *Component* for the bound properties.

The method for the generation of the new configuration returns an *ArrayList* that represents the tiles in a positional way: the *i-th* element is the label for the *i-th* tile.
This is mainly due to the fact the tile access to the map through the position, instead the controller through the label. In this way both the controller and the tile can easily access to the map.

The *restart* has been implemented as a *Bound* property: it fires a

PropertyChange to the registered listeners.
The alternatives I thought was:

1. Simply invoke the *restart* method of the tiles and controller;
2. Register the tiles and the controller as *ActionListeners*;

But

1. The specification explicitly required the registration of the event listeners and
2. the *ActionPerformed* by the *ActionListeners* doesn't take arguments (i.e. the configuration);

The board is a *PropertyChangeListener* to the controller: as seen, the controller fires an event when a tile is (correctly) moved, specifying the hole position and the label to assign to it.
This is the way I choose to implement the passing of the label to the old hole.

The *START* text in the controller disappear after the click; this property is used as a *flag* to know if the game is started.

Flip Button

Requirements for the Flip button

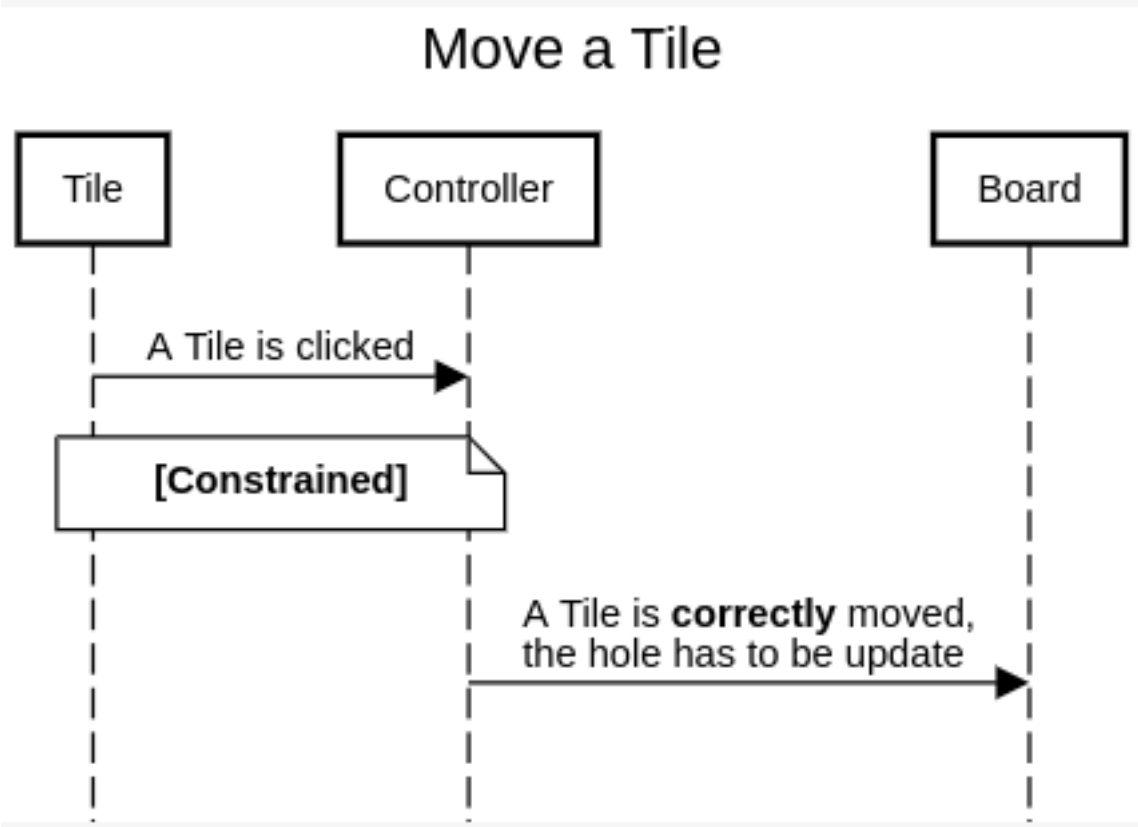
Is proved that in the 8 (or 15) puzzle, from half of the initial configurations the final one cannot be reached with any sequence of moves.
The *Flip* button of the dashboard, when clicked, switches the labels of tiles in position 1 and 2, but **only if** the hole is in position 9, otherwise it has no effect.
Switching the position of two non-hole tiles guarantees that if the previous configuration was not solvable, the new one is.
Any correct solution for the implementation of the flip button is accepted, but solutions which follow the *JavaBean* patterns are more appreciated.

Design choices

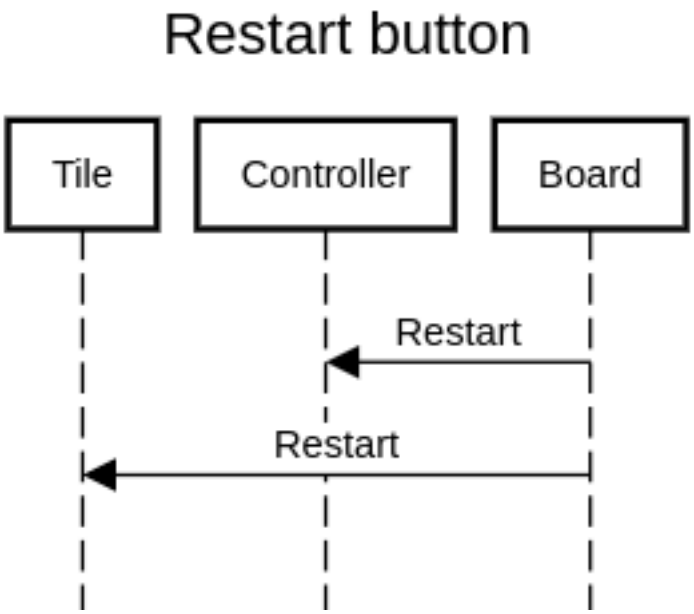
The *flip* event is implemented through a *VetoablePropertyChange*: the controller is triggered and it will decide if the flip is legal; if it is, the board switches the labels;

Interaction Diagram

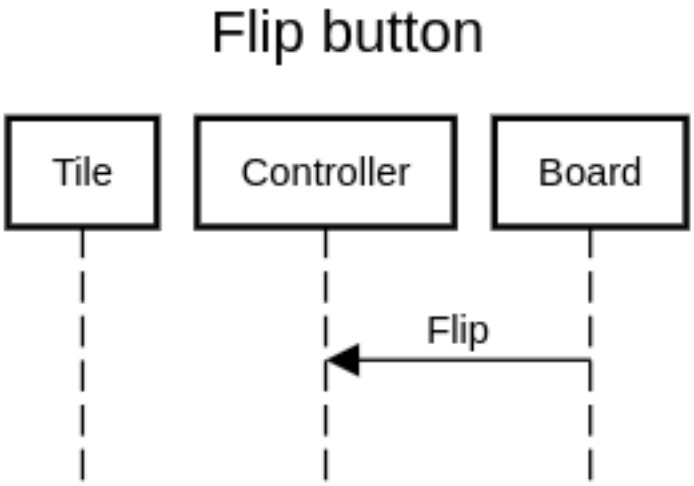
The following sequence diagrams aim to show and explain the event-based communications between the EightPuzzle components.
When a tile is moved, the constrained property `label` has to be changed to 9, so the event is fired to EightController, that checks if the move is legal. If it is, EightController fires the *PropertyChange* to EightBoard, that updates the grid.



When the *restart* button is clicked, the board fires a *PropertyChange* both to the controller and the tiles. The former has to update the internal layout, the latters have to update their label.



When instead the flip button is clicked, the board fires a *VetoableChange* to the controller, that has to check if the 9-tile is in the right-bottom corner.



Conclusions

All the requirements specifications have been met in the development of this *8 Puzzle*, also the *"optional"* ones (i.e. the event-based communication strategies versus the public methods).
Each component has been developed as a single bean, reusable for future works. The development of the whole project followed the *Component-based* programming principles in order to favor the code reuse.
The puzzle has been developed with a test-driven development model, that allowed the detection and the minimization of bugs. At the time this report is written everything seems to work as expected, although is well known that *"program testing can be used to show the presence of bugs, but never to show their absence"*.

Oh, it's also possible to win!