First assignment – Exercise 1

Student: Luca Rizzo

Matricola: 598992

Introduction

The report is structured with a particular emphasis on the Publisher-Subscriber relationships related to each event and how these relationships implement the required game logics.

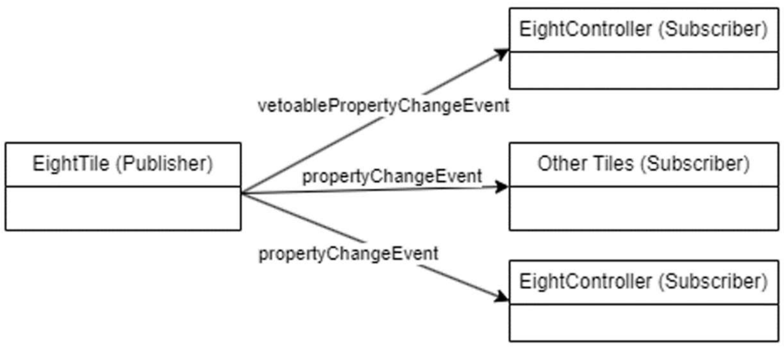
Handling tile value changes

In the **EightTile** class, within its **setLabel()** method, it first notifies all **VetoableChangeListeners** of the vetoable property change.

Subsequently, if no one opposes the change, it confirms the change to all **PropertyChangeListeners**, informing them when the change has actually occurred.

I decided to create two notification event channels for two types of observers:

1. A channel to indicate the "intent" to change the label value — listener can prevent this intent.
2. A channel to indicate the actual change of the label value.


EightController (Subscriber)
vetoablePropertyChangeEvent
EightTile (Publisher)
propertyChangeEvent
Other Tiles (Subscriber)
propertyChangeEvent
EightController (Subscriber)

This way, there is a clear distinction between those who want to register to possibly prevent the change (**EightController**) and those who want to register solely to be informed when the label value has changed (other **EightTile**s).

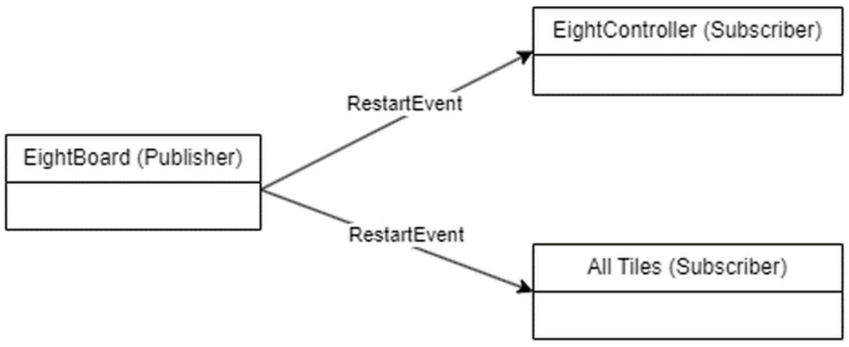
During board initialization, **EightTile**s register each other only for changes in the **label** property so that they are notified, if they are the hole and no one has prevented the change, to change their value to the new "passed" value (the old value of the tile that is changing).

EightController registers for changes in the **label** property of all tiles, both as a **VetoableChangeListener** and as a **PropertyChangeListener**. In this way, with the first callback, it only imposes its veto on the change, allowing other (possible) **VetoableChangeListeners** to prevent the change. With the second callback, it modifies its internal representation because the value has actually changed.

Opposition to the change simply checks whether the tile that is changing is near a hole according to its internal representation. If not, it throws an exception and modifies the text.

Handling restart

Both **EightTile**s and **EightController** register for the **RestartEvent** emitted by **EightBoard**. This event is emitted when the Restart button is clicked, during the initialization of the game and to notify that the Flip must be performed (see later) . The event contains a permutation representing the new state.



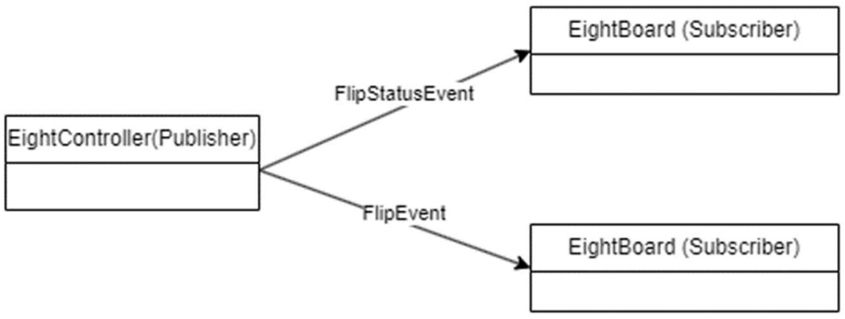
EightController uses it to change its internal representation.

EightTiles use it to modify their **label** value in accordance with the permutation. This is a "special" event not conforming to the game rules, so the value is changed directly without notifying **VetoableChangeListeners** but only **PropertyChangeListeners**. This avoids conflicts with **EightController**'s rule-handling method that prevents such changes.

Handling flip

The flip is managed by registering **EightController** as a listener for the click of the corresponding button in **EightBoard**. Additionally, **EightBoard** registers as a listener for events emitted by **EightController**:

1. **FlipEvent**: contains the new "flipped" permutation. This event is emitted after clicking the Flip button (when enabled).
2. **FlipStatusEvent**: specifies whether the flip is possible or not (only if the hole is in position 9). This event is emitted whenever the **label** property of an **EightTile** changes.



EightBoard, by registering as a listener for these events can, in the first case, notify the new permutation to all restart listeners who will update. In the second case, it enables or disables the flip button. The notification of the flip to all interested listeners has been implemented using the **RestartEvent**, notifying all **RestartListener**s with the "flipped" permutation instead of a random one. This allows for the efficient reuse of the existing **RestartEvent** mechanism