Le jeu de taquin
Utilisation de swing

D. Olivier

23 mai 2008

Résumé

Cet exercice a pour objectif de vous apprendre à élaborer une interface utilisateur utilisant l'API `swing`. L'exercice s'appuie sur un jeu de puzzle à reconstituer. Cet exercice est emprunté à Frédéric Larue.

1 Pour commencer

Le taquin est un jeu solitaire en forme de damier. La version initiale est composée de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Le problème consiste à remettre dans l'ordre les 15 carreaux à partir d'une configuration initiale quelconque sachant que l'on peut échanger verticalement ou horizontalement une case numérotée avec la case blanche. Certaines configurations initiales sont sans solution.

Nous allons nous inspirer de ce problème pour utiliser l'API `swing`. Ainsi plutôt que de considérer 15 cases numérotées et une case blanche, nous allons utiliser une image que nous allons découper et mélanger. Voici par exemple un aperçu de ce que nous devrions obtenir à la fin de ce TP (cf. figure 1). Nous devrons pouvoir charger une image, la découper, la mélanger et ensuite pouvoir déplacer les morceaux mélangés qui la composent.

2 Modèle - Vue - Contrôleur (MVC)

Nous allons suivre le modèle MVC qui se décompose en trois parties :

1. Le *modèle* c'est le cœur du programme, il contient la représentation du problème.
2. La *vue*, c'est l'IHM (Interface Homme-Machine), l'utilisateur peut interagir et communiquer à loisir avec l'application. Plusieurs vues peuvent correspondre à un même modèle.
3. Le *contrôleur*, il correspond aux différents éléments permettant de connecter l'IHM au modèle. Le contrôleur gère les évènements générés par l'action de l'utilisateur sur l'IHM.

2.1 Le modèle

On va donc définir la classe `ModeleTaquin` suivante :

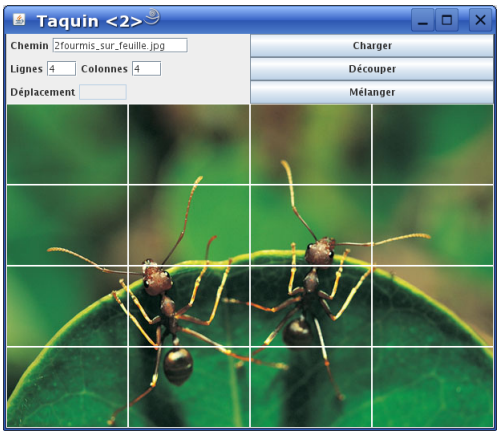

FIG. 1 – Le jeu de taquin et son interface sous `swing`

```
1  class ModeleTaquin
2  {
3   /**
4    * Attributs.
5    */
6      private Case[][] cases;
7      private Case caseVide;
8
9   /**
10   * Le constructeur, on fixe le nombre de lignes et de colonnes.
11   */
12     public ModeleTaquin(int nl, int nc)
13
14  /**
15   * On renvoie la case vide.
16   */
17     public Case caseVide()
18
19  /**
20   * Déplace la case c si elle est voisine à la case vide.
21   */
22     protected void glisserJeton(Case c)
23
24  /**
25   * Déplace la case appropriée dans la direction donnée par sens.
26   * Renvoie false si le deplacement n'est pas valide, true sinon.
27   */
28     public boolean deplacer(Direction sens)
29
30  /**
31   * Affiche l'état courant du jeu.
32   */
33     public String toString()
34
35  /**
```

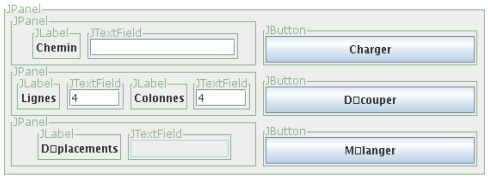

FIG. 2 – Le panneau du haut et ses différents éléments

```
36     * Mélange les cases du jeu de taquin en fonction des voisins,
37     * en effectuant n déplacements aléatoires.
38     * (utilise glisserJeton pour les déplacements)
39     */
40       public void melanger(int n)
41
42    /**
43     * Détermine si le taquin est dans le bon ordre ou pas.
44     */
45       public boolean estRange()
46   }
```

2.2 La vue

Pour commencer, nous allons créer notre conteneur principal une `JFrame`.

```
1  public class VueTaquin extends JFrame {
2    public static void main(String[] args) {
3      SwingUtilities.invokeLater(new Runnable() {
4        public void run() {
5          VueTaquin ihm = new VueTaquin();
6          ihm.setTitle("Taquin");
7          ihm.pack();
8          ihm.setLocationRelativeTo(null);
9          ihm.setResizable(false);
10         ihm.setVisible(true);
11         }
12     });
13   }
14 }
```

Maintenant que la fenêtre est créée, il faut ajouter et agencer différents composants qui constituent l'interface.

2.2.1 Le panneau du haut

La structure est définie sur la figure 2. On a donc un panneau principal : un `JPanel` contenant trois autres panneaux et trois boutons (`JButton`). La création des composants se fera dans le constructeur de la classe `VueTaquin`. Nous aurons besoin de réutiliser certains d'entre eux lors de la mise en place de la partie controleur. Nous allons donc les déclarer en tant qu'attributs de notre classe. Les composants qui nous intéressent sont ceux avec lesquels l'utilisateur peut interagir. Il s'agit donc des quatres `JTextField` ainsi que des trois `JButton`.

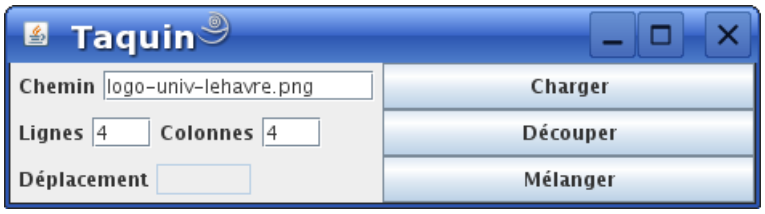

FIG. 3 – Panneau haut pour le dialogue avec l'utilisateur de l'application

FIG. 4 – L'interface du haut et l'image

Pour les 3 `JPanel` contenant respectivement le chemin, le nombre de lignes et de colonnes et enfin la direction de déplacement le gestionnaire de placement par défaut (`FlowLayout`) est suffisant, le placement devra être justifié à gauche. Vous devez obtenir un résultat ressemblant à la figure 3.

2.2.2 Le panneau contenant l'image découpée

Nous allons dans un premier temps charger une image et ensuite l'afficher au niveau de l'interface. Vous devez créer une `ImageIcon` qui implémente l'interface `Icon` et qui se prête bien à la création d'une IHM car on a la possibilité de rajouter de tels objets dans des boutons. Vous devez donc rajouter votre `ImageIcon` à un `JButton` et mettre le tout dans le conteneur principal. Modifier votre constructeur en conséquence.

```
1 // ....
2 Icon icone = new ImageIcon(saisieChemin.getText());
3                     // Nom du fichier associé au bouton chemin
4 JButton boutonImage = new JButton(icone);
5 add(boutonImage, BorderLayout.CENTER);
```

Vous obtiendrez la figure 4.

De la même manière que pour les composants précédents, en fait, nous allons placer notre image dans un conteneur `JPanel`. Cependant, le contenu de ce panneau n'est pas destiné à rester statique au cours de l'exécution : il faudra par la suite pouvoir découper cette image en cellules et pouvoir déplacer les cases du taquin pendant la phase de jeu. Afin d'anticiper sur ces futures modifications, nous allons modifier la déclaration de

l'image et du panneau qui la contient à la liste des attributs pour nous permettre de les manipuler :

```
// attributs.
Icon imageTaquin;
JPanel     panneauImageTaquin;
// ...
```

De la même manière que vous l'avez fait précédemment, ajoutez quelques lignes à la fin du constructeur pour créer le panneau `panneauImageTaquin` et ajoutez le au centre en utilisant le gestionnaire de placement.

Il faut maintenant découper l'image en cellules fonction du nombre de lignes et de colonnes. Pour cela je vous propose d'utiliser la classe `DecoupeImage` suivante :

```
class DecoupeurImage
{

 /**
  * Chargement d'une image à partir du fichier chemin,
  */
  public static ImageIcon lireImage(String chemin) {
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image src1 = tk.getImage(chemin);
    return new ImageIcon(src1);
  }

 /**
  * Chargement d'une image à partir du fichier chemin,
  * retaillée aux dimensions (largeur,hauteur).
  */
  public static ImageIcon lireImage(String chemin, int largeur, int hauteur) {
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image src1 = tk.getImage(chemin);
    Image src2 = src1.getScaledInstance(width, height, Image.SCALE_DEFAULT);
    return new ImageIcon(src2);
  }

 /**
  * Découpage d'une sous-image de dimensions (largeur,hauteur) dans l'image src.
  * (x,y) spécifie le coin supérieur gauche de la sous-image.
  */
  public static ImageIcon decouper(ImageIcon src, int x, int y,
                                   int largeur, int hauteur) {
    ImageProducer ip = src.getImage().getSource();
    CropImageFilter cif = new CropImageFilter(x, y, largeur , hauteur);
    FilteredImageSource fis = new FilteredImageSource(ip, cif);
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image img = tk.createImage(fis);
    return new ImageIcon(img);
  }

 /**
  * Découpage de l'image src selon une grille régulière
  * de nl lignes et nc colonnes.
  * retourne la sous-image correspondant à la case (i,j) de cette grille.
```

```
42     */
43     public static ImageIcon getIcon(ImageIcon src, int i, int j, int nl, int nc) {
44       int largeur = src.getIconWidth() / nc;
45       int hauteur = src.getIconHeight() / nl;
46       return decouper(src, j * largeur, i * hauteur, largeur , hauteur);
47     }
48
49    /**
50     * Découpage de l'image src selon une grille régulière
51     * de nl lignes et nc colonnes.
52     * retourne cette grille sous la forme d'un tableau de sous-images.
53     */
54     public static ImageIcon[][] diviser(ImageIcon src, int nl, int nc) {
55       ImageIcon[][] res = new ImageIcon[nl][nc];
56       for (int i = 0 ; i < nl; ++i)
57         for (int j = 0; j < nc; ++j)
58           res[i][j] = getIcon(src, i, j, nl, nc);
59       return res;
60     }
61   }
```

Une fois votre image lue et découpée il vous suffit de créer un bouton par morceau d'image et ajouter cela à votre `JPanel panneauImage` en utilisant un gestionnaire approprié. Nous effectuons cela pour tester l'ensemble, nous modifierons cette architecture par la uiste. Il nous reste une dernière chose à faire. En effet, que se passe-t-il si on charge consécutivement deux images ? La deuxième image ne remplace pas la première mais vient s'ajouter. Pour cela il suffit de tout supprimer à l'aide de `removeAll()`, mais attention cette méthode supprime également le gestionnaire de placement, il faut donc à nouveau le spécifier.

2.3 Un premier controleur

Pour commencer nous allons gérer les trois boutons du panneau haut (charger, découper, mélanger). Pour que ce controleur (listener) puisse manipuler la vue, nous allons lui en déclarer une copie en attribut, qui sera initialisée par le constructeur, le squelette de classe va s'écrire :

```
1  public class ControleurTaquin implements ActionListener {
2   VueTaquin vue;
3
4   public ControleurTaquin(VueTaquin v) {
5      vue = v;
6   }
7
8   public void actionPerformed(ActionEvent e)
9   {
10  }
11 }
```

Le listener est utilisé pour gérer les trois boutons à la fois. Il faut donc pouvoir les différencier l'un de l'autre et associer les méthodes correspondantes. Vous pouvez éventuellement dans un permier temps, uniquement utiliser une trace.

3 L'application

Commençons par définir une classe `TaquinGraphique` qui hérite de la classe `Taquin`. Celle-ci devra nous permettre de manipuler, en plus du tableau de `Case`, un tableau de `JBouton`. Chacun de ces boutons correspondra à une case du jeu et contiendra une sous-partie (icône) de l'image.

```
public class TaquinGraphique extends ModeleTaquin
{
 JButton[][] cellules;
 int nbLignes;
 int nbColonnes;

 public TaquinGraphique(ImageIcon img, int nbl, int nbc)
 {
 }
}
```

Le constructeur prend trois paramètres :

- `img` : l'image à utiliser pour le jeu. C'est celle qui est actuellement chargée par l'IHM via la méthode chargerImage().
- `nbl` : le nombre de lignes pour le découpage.
- `nbc` : le nombre de colonnes pour le découpage.

Ecrivez le corps de ce constructeur afin qu'il exécute ces tâches.

- Appellez le constructeur de la super classe en lui passant les paramètres appropriés. Cela permettra d'initialiser correctement le jeu.
- Créez le tableau de boutons.
- Créez une instances `JButton` pour chaque élément de ce tableau. Utilisez la méthode `setBorder()` pour spécifier une bordure de très petite taille à vos boutons. Récupérez la sous-image associée à l'aide de la classe `DecoupeurImage`.
- Enfin, grisez le bouton d'indice (0,0) en le désactivant. Il représentera la case vide du jeu.

La classe `ModeleTaquin` repose essentiellement sur la méthode `glisserJeton(Case c)` qui déplace la case `c` dans l'emplacement vide si celle-ci lui est adjacente. La méthode `melanger(int n)` par exemple, utilise `glisserJeton` pour mélanger les cases du jeu. Nous devons donc redéfinir cette méthode pour qu'elle opère également sur notre tableau de boutons. On aura besoin pour cela de récupérer le bouton associé à une case. La classe `Case` propose deux méthodes, `getLigne()` et `getColonne()`, pour récupérer les coordonnées de la case dans le tableau. On ajoute donc à `TaquinGraphique` :

```
private JButton getButton(Case c)
 {
     return cellules[ c.getLigne() ][ c.getColonne() ];
 }
    public JButton getButton(Case c)
    {
        return buttons[ c.getLigne() ][ c.getColonne() ];
    }
```

Vous pouvez maintenant redéfinir la méthode `glisserJeton(Case c)`. Elle doit :

- Récupérer les boutons associés à la case `c` et à la case vide (que l'on récupère avec la méthode `caseVide()` déclarée dans `ModeleTaquin`).

– Inverser les icones de ces deux boutons.
– Réactiver le bouton associé à l'ancienne case vide.
– Désactiver le bouton associé à la nouvelle case vide.
– Appeler la méthode `glisserJeton()` de la super classe pour mettre à jour l'état du jeu en répercutant le le déplacement sur les cases également.

Il vous faut associer un écouteur `ActionListener` à vos boutons.

Pour que le listener puisse commander le déplacement d'une case du jeu lorsqu'un évènement est produit, nous allons déclarer une méthode effectuant ce déplacement, ne connaissant que l'instance du bouton source :

```
1   public boolean deplacerBouton(JButton clickedButton)
2  {
3  }
```

Cette méthode doit :
– Récupérer les cases voisines à la case vide. La classe `ModeleTaquin` propose pour cela la méthode `voisines(Case c)`.
– Vérifier si l'instance du bouton associé à l'une des quatre cases voisines est égale à la source `clickedButton`.
– Si c'est le cas, appeler `glisserJeton()` avec la bonne case.
– Renvoyer `true` si un déplacement à bien eu lieu, `false` dans le cas contraire.

Pour finir, l'IHM devra être capable de récupérer le tableau de boutons pour l'afficher. Plutôt que de renvoyer le tableau lui même, nous allons implémenter dans `TaquinGraphique` une méthode pour remplir un `JPanel` passé en paramètre. Ajoutez donc :

```
1      public void remplirPanneau(JPanel panneauImageTaquin)
2      {
3      }
```

Cette méthode doit d'abord effacer le contenu précédent du panneau puis lui ajouter un nouveau GridLayout aux bonnes dimensions.

```
1      panneauImageTaquin.removeAll();
2      panneauImageTaquin.setLayout(new GridLayout(getNbLignes(), getNbColonnes()));
```

Les boutons seront ensuite ajoutés un à un.

La classe `TaquinGraphique` est désormais complète. Elle nous fournit une extension suffisante de `ModeleTaquin` pour pouvoir être exploitée par notre IHM. Il faut maintenant écrire l'écouteur de placée sur notre vue. Cela donne :

```
1  public class ControleurTaquin implements ActionListener {
2
3      private VueTaquin vue;
4      //.....
5  }
```

Notre vue doit disposer d'un modèle (selon le concept de l'architecture MVC), c'est à dire d'une instance du jeu de taquin. Ajoutons lui donc un attribut :

```
1      public class TaquinIHM extends JFrame
2      {
3          // attributs.
4          ...
5          private TaquinGraphique taquin;
6
```

```
7               ...
8       }
```

Il vous faut écrire les deux méthodes `decouperImage()` et `melangerImage()` si vous ne l'avez pas fait dans `VueTaquin`.

Ecrivez pour finir une classe principale qui lance je jeu.