RAPPORT DE CRÉATION D'UN JEU DE TAQUIN OU PUZZLE À GLISSIÈRES

mis en page par
Romain ANDRES, Marta BOSHKOVSKA, Sara SALÉ, Xue YANG

Licence 2 Informatique
Groupe 4A

Avril 2022

UE : PROGRAMMATION AVANCÉE

Table des matières

1 Introduction

Actuellement en deuxième année de Licence Informatique, au cours de l'UE Programmation Avancée il nous a été demandé de réaliser un jeu de Taquin ou puzzle à glissière en utilisant le langage de programmation orientée objet Java.
Ainsi tout au long de ce rapport, nous vous présenterons les moyens mis en place pour la réalisation du projet, quelques éléments du processus de réalisation et le fonctionnement de notre jeu.

2 La composition de notre répertoire

Notre projet se subdivise en 3 packages. L'un des points importants de notre cahier des charges est le motif MVC ou Modèle-Vue-Contrôleur. Il nous a été demandé de réaliser impérativement notre travail sous ce motif et de particulièrement de faire à attention à ce que le modèle soit totalement indépendant de l'interface graphique (la vue). Ainsi nos packages sont les suivants :

- **Model**,
- **Control**,
- et **View**.

2.1 Le package Model

Ce package représente le modèle comme l'indique bien son nom. Le modèle représente le socle de notre projet. Il est composé des classes *Board.java* et *Element.java*. C'est là que sont implémentées les fonctionnalités de base. Pour ce jeu de taquin, nous avons décidé de travailler avec un tableau à deux dimensions : une pour les lignes de la grille et une autre pour les colonnes.

La gestion de ce tableau est faite par la classe *Board.java*. Au total, nous avons eu besoin de 12 méthodes pour implémenter les fonctionnalités qui nous semblaient essentielles au fonctionnement du jeu. Ce sont des méthodes qui permettent de mélanger l'ordre des éléments du tableau, d'échanger les cases selon des directions ou encore de déterminer si l'utilisateur a gagné.
Notre tableau est rempli d'objets de la classe *Element.java*. Ces objets ont chacun une valeur précise, des chiffres sans répétition à partir de 1 et le dernier est donné selon la dimension de la grille. Par exemple une grille de dimension 3x3 sera composée de chiffres de 1 à 9 inclus et une grille de 6x6 sera composée de chiffres de 1 à 36 inclus. Puisqu'il s'agit d'un puzzle à glissière nous avons trouvé le moyen de retirer une case afin de permettre aux autres pièces de se mouvoir par la suite. La case retirée est la case au plus grand chiffre du tableau et est remplacée par NULL. Donc un objet ELEMENT a une valeur NULL tandis que les autres conservent leur valeur d'entiers. Le principe de déplacement des cases est alors simple. Si une case est directement voisine de la case où se trouve NULL, les deux cases peuvent être échangées.
Parallèlement dans la classe *Board.java*, nous avons utilisé des ENUM pour manager les directions. En effet, nous en avons 4 : **Up, Down, Left, Right**. Parce que nous travaillons avec un tableau à deux dimensions, nous avons la possibilité de manipuler les coordonnées et chaque direction représente une opération sur les coordonnées.

2.2 Le package Control

Il s'agit de la partie Contrôleur du MVC. Nous nous sommes véritablement basé sur l'élaboration de contrôleur que nous avons eu à voir pendant le TP6. Il regroupe les classes *EcouteurModele.java* et *AbstractModeleExecutable.java*. La première est une interface contenant une seule méthode, MODELEMISAJOUR(). La deuxième est une classe abstraite héritée par *Board.java*. Grâce à cette dernière, les objets BOARD sont écoutés. Elle permet également la manipulation des objets ECOUTEURMODELE ou objets écoutés. De plus, par sa méthode, EXCHANGECHANGEMENT() appelée dans la méthode EXCHANGE() de *Board.java*, on sait immédiatement s'il a eu un échange au niveau des cases de la grille. C'est grâce à cette classe que nous obtenons réellement un échange équilibré et sans dépendance entre la partie modèle et la partie vue.

2.3 Le package View

Ce package concerne exclusivement la partie graphique. Nous y retrouvons les classes *TaquinGUI.java*, *GUI.java* et *ViewBoard.java*. C'est avec *TaquinGUI.java* que nous gérons la page d'accueil. Ici, l'utilisateur doit faire des choix tels que jouer une version du puzzle avec des chiffres ou une version avec des images. Dans une dynamique d'optimisation du code, nous avons décidé que ce soit la même fenêtre pour les deux versions mais selon le choix de l'utilisateur la composante de cette fenêtre change. Cette fenêtre est paramétrée grâce à la classe*GUI.java*. On y retrouve des composantes qui permettent au joueur de choisir son niveau de jeu, de choisir le format de sa grille ou de voir le nombre de mouvements qu'il a joué. Dans cette même dynamique d'optimisation, *ViewBoard.java* qui hérite de la classe *JPanel* nous permet de mettre en oeuvre visuellement la grille de jeu, selon la version choisie par l'utilisateur. Selon, le choix e l'utilisateur, le JPanel affiche la grille avec les chiffres ou la grilles avec les morceaux d'images.

Voici présentés nos packages regroupés dans le répertoire **src**. Nous avons aussi les répertoires **lib** et **build** respectivement destinés à la gestion des images utilisées et de fichiers exécutés. Il existe également un *Makefile* afin de simplifier la compilation et l'exécution du code.

3 Éléments techniques

3.1 Le mélange du tableau et la fin de partie

La méthode SHUFFLE() Pour commencer une partie, il faut que les valeurs du tableau soient préalablement mélangées. Cependant, ce mélange est assez complexe car il faut veiller à ce que la partie reste solvable. En ce sens, au lieu de donner à nos objets ELEMENT des valeurs totalement aléatoires, on a mis en place une méthode de mélange un peu "déterminé" selon le niveau de jeu souhaité par le joueur et son algorithme se présente comme suit :

Par conséquent selon le n entré, on aura un degré de mélange. Ce sont des mouvements plus ou moins aléatoires.

La méthode HASWON() Gagner une partie, c'est revenir au tableau de départ après avoir fait un ou plusieurs mouvements. Au début, on avait des attributs final pour chaque objet ELEMENT qui renvoyaient les coordonnées de départ de l'objet malgré les déplacements. En conséquence, lorsque toutes les cordonnées de tous les objets ELEMENT après déplacements correspondent à leurs coordonnées initiales respectives, le joueur a gagné. Toutefois, cette méthode avait des insuffisances car l'objet ELEMENT avec pour valeur NULL occasionnait des

Algorithme 1 : Mélange des valeurs des cases

Entrées : un entier n

```
1  this.exchange(Board.Direction.Up)
2  this.exchange(Board.Direction.Left)
3  this.exchange(Board.Direction.Left)
4  doublerandomNumber
5  pour i ← 0 a n faire
6  |  randomNumber ← Math.random() * 3
7  |  si randomNumber ==' 0' alors
8  |  |  this.exchange(Board.Direction.Down)
9  |  fin
10 |  si randomNumber ==' 1' alors
11 |  |  this.exchange(Board.Direction.Up)
12 |  fin
13 |  si randomNumber ==' 2' alors
14 |  |  this.exchange(Board.Direction.Right)
15 |  fin
16 |  sinon
17 |  |  this.exchange(Board.Direction.Left)
18 |  fin
19 fin
20 this.exchange(Board.Direction.Right)
```

erreurs.
C'est pourquoi, nous avons préféré créer une copie du tableau initial dans un nouvel objet BOARD avec le mot-clé *final* pour qu'il ne soit pas modifié au fur et à mesure. Lorsque tous les ELEMENT du tableau courant correspondent aux ELEMENT du tableau copié alors l'utilisateur a gagné.

3.2 Réaliser l'interface graphique

Notre cahier de charge nous a exigé de pouvoir lancer le jeu en ligne de commande mais aussi de pouvoir jouer à partir d'une interface graphique. Nous nous sommes donc attelés à réaliser cette interface.

Au départ, nous voulions mettre en place une JPanel et grâce à la disposition `gridLayout`, celle-ci serait disposée sous forme de grille dont chaque case contiendrait un JPanel pour représenter un objet ELEMENT. Par exemple, si la grille devait contenir 25 cases, on se retrouverait avec 26 JPanel. Comme vous le constatez, ce n'est pas du tout la solution optimale. Par conséquent, suivant les conseils du professeur, nous nous sommes plutôt servi de la méthode PAINTCOMPONENT de JPanel que nous avons réécrit. Nous nous sommes servis des dimensions de la JPanel que nous divisions selon le nombre de lignes et de colonnes de la grille. On pouvait avoir des coordonnées très exactes et dessiner correctement la valeur de chaque objet ELEMENT. Cette simple transformation nous a permis d'optimiser notre code mais surtout d'avoir une facilité à pouvoir gérer les différents évènements souris.
Par ailleurs, nous avons pu mettre en place une version imagée du puzzle grâce à un code trouvé sur le blog Kalani's Tech blog. Ce code nous a permis de récupérer une image et de la séparer en morceaux. Nous récupérons ces morceaux dans un tableau. Ensuite, parce que chaque morceau d'images a un indice précis dans le tableau sous forme d'entier, il ne nous reste plus qu'à lier chaque indice à la valeur d'un ELEMENT. Parallèlement, nous avons implémenté

l'interface ActionListener et réécrit sa méthode mousePressed. Notre objectif a été de détecter si lorsque la souris est pressée à un endroit de la composante on peut déterminer si les coordonnées correspondent à un Element voisin de Null. Au terme de tout cela, nous avions une grille de jeu complète et la possibilité de modifier cette grille de jeu à souhait.

4 Fonctionnement du programme et présentation en images du rendu

Grâce au *Makefile* présent dans notre répertoire principal, la compilation et l'exécution se font aisément. Au lancement du programme, il est demandé à l'utilisateur différentes informations, notamment :

- le format souhaité pour sa grille de jeu ;
- la version du jeu, soit en ligne de commandes, soit avec l'interface graphique ;
- et le niveau de jeu (facile, moyen ou difficile).

Une fois ces informations bien fournies, le jeu est lancé. Pour la personne qui joue en ligne de commande, c'est très simple. Il saisit une lettre qui correspond à une direction et la case Null est échangé par rapport à cette direction.
Du côté de l'interface graphique, l'utilisateur doit faire un choix : avoir une grille remplie de chiffres ou remplie de morceaux d'images. Ici, c'est sous l'action de la souris que le changement se produit. Lorsque l'utilisateur clique dans la zone d'un élément remplaçable, il y a automatiquement changement.

Après vous avoir présenté brièvement la constitution de notre code et son fonctionnement, les images suivantes montrent comment tout se présente au lancement du programme.

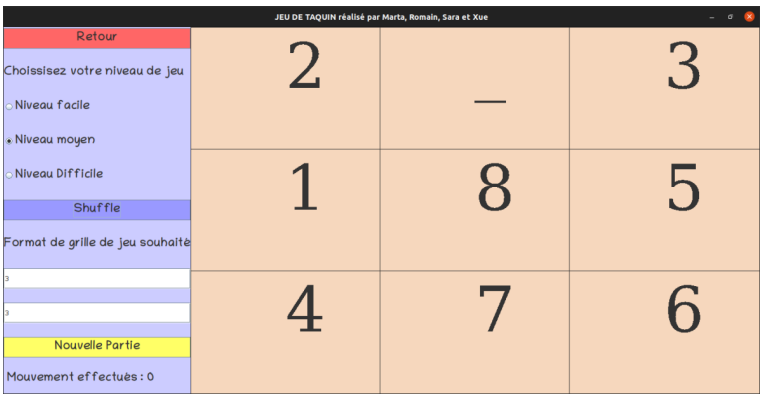
Figure 1 – **Aspect du terminal à l'exécution du code.**

FIGURE 2 – **Aperçu du menu au lancement de la partie version graphique.**

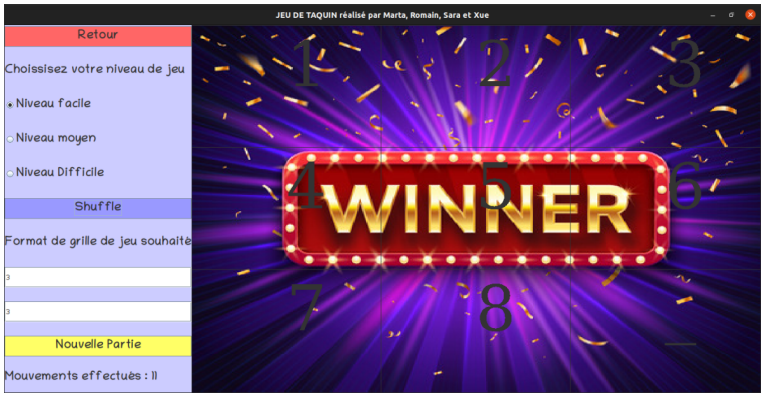
Les cases n'ont pas été mélangées.

Les cases ont été mélangées.

FIGURE 3 – **Extraits d'une partie avec la version image du puzzle**



Les cases ont été mélangées



L'utilisateur a gagné la partie

FIGURE 4 – **Extraits d'une partie avec la version chiffre du puzzle**

5 Conclusion

Pour conclure, ce projet qui devait obligatoirement se faire selon le motif MVC a été l'occasion pour nous travailler avec plus de rigueur et d'organisation. On retient surtout que l'interface graphique peut être développée sans aucunement empiéter sur les autres parties et que le modèle MVC est bien pratique pour avoir un code parfaitement modulaire. Dans l'optique d'améliorer notre jeu, nous pourrions mettre en place un système de sauvegarde des parties, un enregistrement des scores des joueurs et ajouter un chronomètre.