Rapport Complément Programmation Orientée

Amélie Phok, Antonin Montagne, Lou-Anne Gautherie, Naya Makdessi

8 Avril 2022

Table des matières

1 Introduction

1.1 Description du projet

Nous avions pour but de coder en Java, un jeu de puzzle à glissières qui s'intitule "Taquin". Ce genre de jeu est une grille de n lignes sur m colonnes, avec n*m-1 éléments (c'est-à-dire qu'une case est vide). Le joueur a pour but faire glisser l'un des éléments contigus à la case vide vers cette dernière afin de remettre les cases dans le bon ordre.

1.2 Objectifs du projet

Nous voulions développer une interface graphique où l'on pourrait cliquer sur des boutons pour bouger la pièce vide. Nous avions d'abord pour but de dévélopper la même chose mais sur le **Terminal** ; c'est-à-dire de rentrer des commandes telles que "*haut*" ou encore "*gauche*" pour bouger la pièce vide.

2 Organisation du projet

Nous avons décidé de se répartir les tâches en suivant le concept MVC, c'est-à-dire Modèle, Vue, Contrôleur.

Pour commencer, Amélie a crée la classe qui définit un puzzle, nommée ModeleTaquin. Antonin, Lou-Anne et Naya ont crée le reste des classes MVC qui sont :

- AbstractModeleEcouteur dans controle
- EcouteurModele dans controle
- ModeleEcoutable dans controle
- ModeleGUI dans vue
- VueModel dans vue
- Image dans vue

Notre modèle est donc la classe ModeleTaquin dans modele codé par Amélie. Ce modèle hérite ainsi de la classe AbstractModeleEcouteur qui permet d'*écouter* le modèle lorqu'une pièce est bougée.

Cette classe contient plusieurs méthodes telles que `shuffle()` qui permet de mélanger la grille un nombre aléatoire de fois ou encore `swap(int i, int j)` qui permet d'échanger une case de coordonnées (i,j) avec la case vide.

Enfin, la méthode `isOver()` vérifie si le puzzle est terminé.

Pour la partie affichage de l'interface graphique, c'est Antonin qui a codé la classe VueModel, Image et ModeleGUI.
La classe ModeleGUI hérite de la classe JFrame est permet d'appeler les constructeur des classes Image et VueModel. Elle prend un argument une instance de ModeleTaquin.
La classe Image prend en argument le chemin de l'image que l'on veut utiliser et les dimensions de l'image final. Cette classe contient deux méthodes `redimension` qui permet de redimensionner l'image d'origine en 500*500 et `decoupe` qui permet de découper l'image en parties.
La classe VueModel prend en paramètre une instance de ModeleTaquin, le nombre de lignes et de colonnes, le nom de l'image et la JFrame. Pour afficher le jeu, il appel la méthode `affichageGrille`. Dans cette méthode, il a crée un tableau de `JButton` et une map qui contient pour chaque nombre de la grille ses coordonnées dans le modèle. Puis pour chaque nombre de la grille, il a crée un `JButton` au quel il a donné en argument le chemin de la sous-image qui correspond au numéro de la case. Il a rendu tout les boutons sauf celui de la case vide cliquable pour pouvoir échanger la case avec la case vide. Quand le focus est sur la case vide, il a crée un évènement permettant de déplacer la case avec les flèches. A chaque déplacement, il appel la méthode `swap` du modèle. La méthode `modeleMisAJour` supprime tout les éléments de l'affichage est appel `affichageGrille`. Enfin la fonction `deleteImg` permet de supprimer toutes les sous-images crée une fois la partie gagné.

Pour la partie Terminal, c'est Lou-Anne qui a commencé à coder la classe Terminal, utilisant la classe `Scanner` de Java.
Elle a implémenté ce qu'Amélie avait fait dans ModeleTaquin pour importer un taquin aléatoire définit sur 3 par 3.
Elle a créé un `Scanner` qui récupère la commande rentrée par l'utilisateur, c'est-à-dire "*haut*", "*bas*", *droite*, ou "*gauche*".
Si la commande rentrée n'existe pas, le terminal redemande de saisir une commande correcte.
Elle a ensuite fait des conditions de sorte d'associer la commande rentrée,

avec l'échange correspondant (avec la méthode `swap()` de la classe ModeleTaquin). Sauf que les coordonnées codées par Lou-Anne n'étaient pas les bonnes, Naya a donc prit le relais et a mis les coordonnées correspondantes aux échanges. Elle a aussi fait en sorte que si il y a un mur là où l'utilisateur voulait mettre la pièce vide, le terminal redemande au joueur de rentrer une commande correcte.

3 Architecture du projet

Ce premier diagramme montre comment nous avons organisé l'architecture de notre projet en différents paquets.
Comme on peut le voir, le paquet controle contient les éléments de controle abstrait mais aussi les classes Terminal et Fenetre qui permettent de lancer le jeu dans le terminal et dans une interface graphique respectivement.

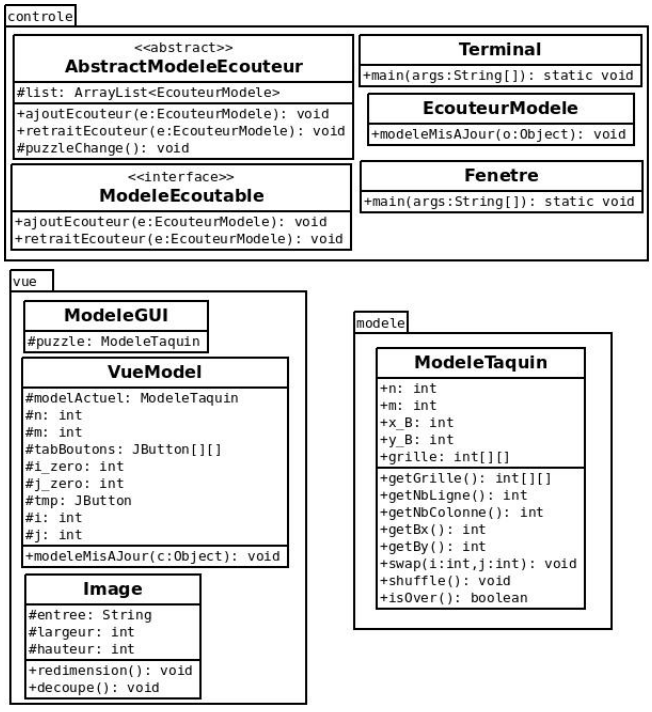


FIGURE 1 – Diagrammes des paquets utilisés pour le projet

Ce deuxième diagramme permet d'illustrer le concept MVC et comment les trois parties dépendent les unes des autres.

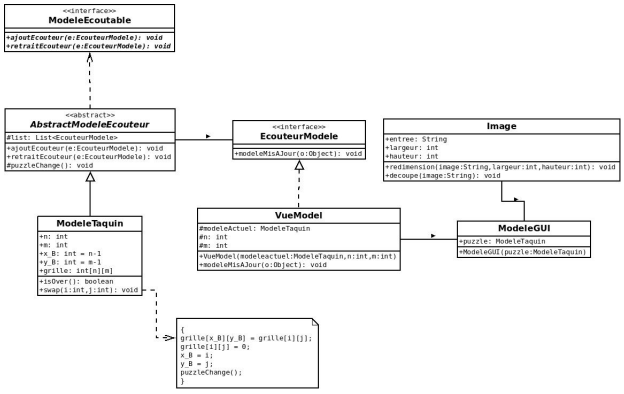


FIGURE 2 – Diagramme des classes avec leurs dépendances

4 Documentations

La documentation a été générée automatiquement avec les commentaires Javadoc formatés avec /** et */.
Dans le dossier doc, c'est la page index.html qui permet d'accéder à l'ensemble de la documentation ainsi générée.

5 Expérimentations

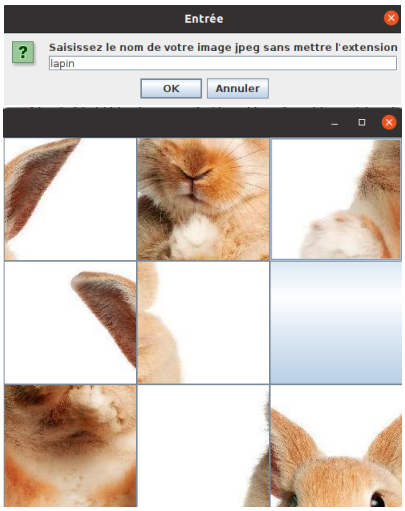
Le jeu peut être lancé à partir de deux fichiers .jar qui sont a exécutés directement dans le dossier dist.

`TaquinTerminal.jar` lance le jeu dans le terminal et `TaquinInterface.jar` lance le jeu dans une fenêtre graphique.

Lorsque vous lancez le jeu, avec `runTerminal`, le Terminal affiche la grille mélangée et vous demande de saisir une commande :

```
    [java] 0 1 6
    [java] 4 3 8
    [java] 7 2 5
    [java]
    [java]
    [java] Vous pouvez choisir les commandes suivantes: monter la pièce vide (h
aut), la descendre (bas), la mettre à gauche (gauche), la mettre à droite (droit
e)
    [java] Entrez une commande:
```

Lorsque vous lancez le jeu, avec `runInterface`, une fenêtre s'ouvre, vous demandant de saisir le nom de l'image que vous voulez jouer (se trouvant dans le répertoire image). L'image apparait découpée, il faut cliquer sur une case pour placer la case vide à l'endroit du clique. On peut aussi cliquer sur la case vide et la bouger avec les flèches.



6 Conclusion

Nous avons ainsi complété la majorité des objectifs de notre projet. Notre puzzle est en effet construit selon le concept MVC avec une partie Modèle, Vue et Contrôleur.
Il est possible de jouer avec des commandes textes dans le terminal ou de jouer dans une interface graphique avec une image à reconstituer.