

Complément POO Rapport du Devoir

Beauchamp Aymeric 21301016
Chagneux Dimitri 21606807
Mori Baptiste 21602052
Leblond Valentin 21609038

L2-Info-groupe-4A

Table des matières

Introduction	2
1 La conception du package model	2
1.1 Organisation des classes	2
1.2 Fonctionnement du modèle	3
2 Conception du package GUI	4
2.1 Organisation des classes	4
2.2 Fonctionnement GUI	5
Conclusion	6

Introduction

L'objectif de ce projet de POO est de réaliser un taquin : un casse-tête consistant à déplacer des cases d'un plateau afin de les replacer dans l'ordre et ainsi reconstituer une image ou un motif donné.

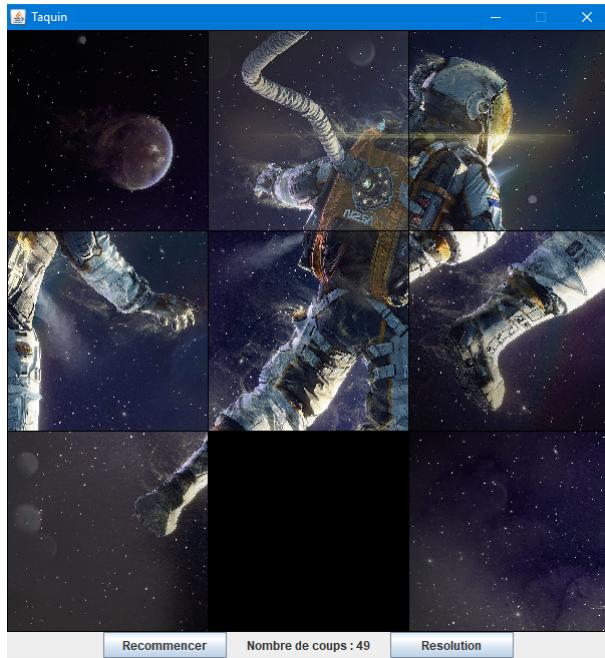


FIGURE 1 – Jeu du Taquin

Nous avons séparé le projet en deux packages, le premier comportant le modèle du jeu utilisable en version console (le package **model**), et le deuxième, l'implémentation de l'interface graphique (le package **GUI**). Nous utilisons également un troisième dossier **ressources** où sont stockées les images utilisées par l'application.

1 La conception du package model

Le modèle de base du taquin est constitué d'une grille d'objets représentant les différentes cases. Les cases pleines ont des identifiants pour permettre de mettre en place la condition de victoire : on gagne si chaque identifiant est placé aux bonnes coordonnées.

1.1 Organisation des classes

Tout d'abord, nous avons représenté les cases par deux classes, **FullTile** pour les cases pleines et **EmptyTile** pour la zone vide qu'on déplacera. Ces classes possèdent des attributs en commun qui sont les coordonnées X et Y dans la grille ; c'est pourquoi nous avons conçu une classe abstraite **Tile** qui possède ces coordonnées et dont héritent FullTile et EmptyTile. La classe FullTile diffère de EmptyTile en ce qu'elle possède en plus un identifiant entier que l'on rend unique pour chaque instance utilisée.

Ensuite, nous avons une classe **Board** qui décrit l'état du jeu et le fait évoluer.

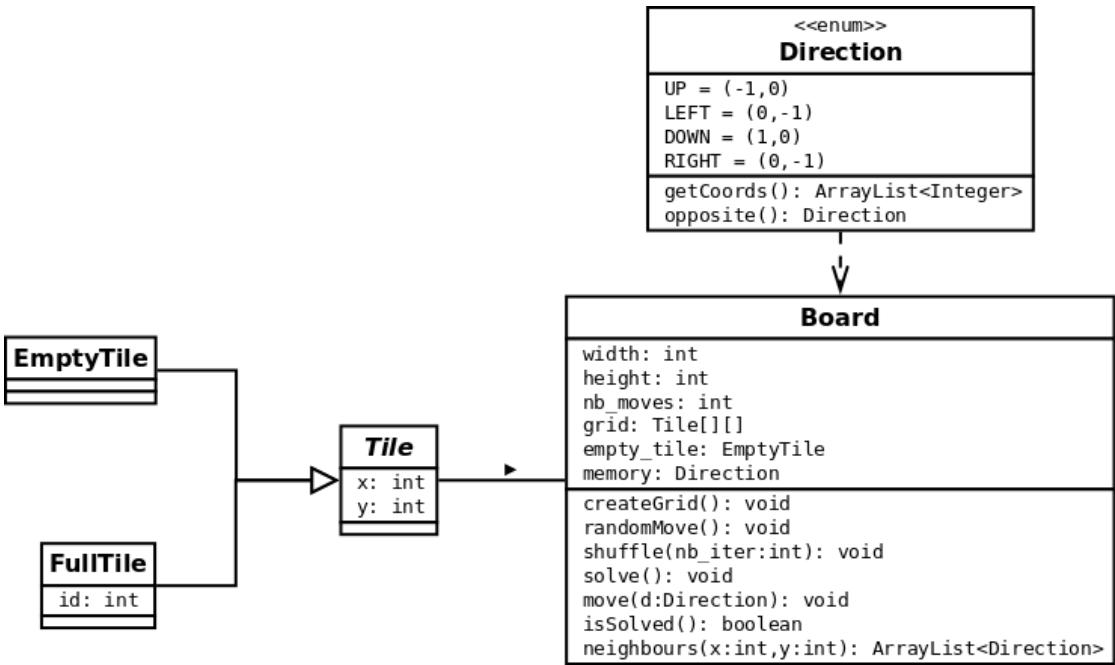


FIGURE 2 – Diagramme de classe du package model

1.2 Fonctionnement du modèle

Comme nous l'avons expliqué, l'initialisation et la mise à jour du modèle se font par le biais de la classe **Board**.

En premier lieu, nous avons une fonction **createGrid** qui permet d'initialiser une grille avec des **FullTile** et une **EmptyTile** et une fonction **toString** qui permet de l'afficher.

La grille ainsi créée est dans un état résolu où les identifiants des cases sont rangés dans l'ordre ci-dessous :

```
$ java model/Main
0 1 2
3 4 5
6 7
```

FIGURE 3 – Affiche d'une grille 3x3 initialisée

Puis, nous avons crée les déplacements de la case vide, nous avons d'abord commencer par créer une enum **Direction** avec quatre instances **UP**, **DOWN**, **LEFT**, **RIGHT** qui représentent une direction de déplacement. Nous utilisons cette enum dans la fonction **move** qui déplace la case vide dans la direction donnée en argument.

Pour mélanger le jeu, on choisit un mouvement aléatoire selon les mouvements possibles de la case vide puis on déplace la case vide jusqu'en bas à droite, à sa position initiale. On garde en mémoire le dernier coup effectué afin d'éviter des coups inutiles qui réduisent l'efficacité du mélange ; si l'on va vers le haut lors d'un coup, on interdit d'aller vers le bas au coup suivant. Le mélange est effectué par la fonction **shuffle** qui prend en argument le nombre de coups à réaliser avant de considérer le mélange comme terminé. Dans la version finale, nous faisons 10 000 mélanges. Cela assure une bonne dispersion des cases pour un temps de calcul négligeable. Comme la grille initiale est résolue, l'état obtenu par le mélange n'est pas bloquant puisqu'il existe une séquence de coups menant de cet état à la solution.

La récupération des coups jouables de la case vide est un voisinage de taille 4 de rayon 1 qui correspond aux quatre directions haut, bas, gauche, droite. La fonction **neighbours** donne selon une coordonnée dans la grille du jeu donne toutes les directions possibles.

Le modèle dispose aussi d'une fonction **solve** qui résout le puzzle. Nous avons utilisé une approche extrêmement simple consistant à jouer un coup aléatoire jusqu'à obtenir la configuration initiale de la grille.

Cette simplicité a un coût : le solveur peut réaliser jusqu'à 500 000 coups pour résoudre un simple puzzle 3x3, et devient très lent dès que l'on veut augmenter la taille. Nous sommes conscients qu'il est possible de réaliser un solveur plus efficace, par exemple via l'utilisation de l'algorithme A*, mais nous avons préféré ne pas nous focaliser sur cet aspect du projet.

La classe principale permet de jouer au taquin en interface console. Le joueur peut déplacer la case vide avec Z,Q,S,D.

2 Conception du package GUI

L'interface graphique est composée de deux fenêtres, celle permettant de choisir l'image du jeu et la deuxième qui est le jeu en lui-même.

2.1 Organisation des classes

L'interface graphique est composée de vue **View**, du model **Board**, **Tile** et du contrôleur présent dans **Interface** (clics de souris et touches du clavier). La vue implémente **ModelListener** qui permet de l'actualiser lorsque un mouvement du model se fait. Le model (ici le **Board**) étend de **AbstractModeleEcouteable** qui permet d'ajouter ou supprimer des écouteurs mais aussi d'une méthode qui permet d'actualiser tous les écouteurs qu'on fera appeler lors de l'exécution de la fonction **move**. Les contrôleurs lancer le mouvement du model qui va lui actualiser la vue.

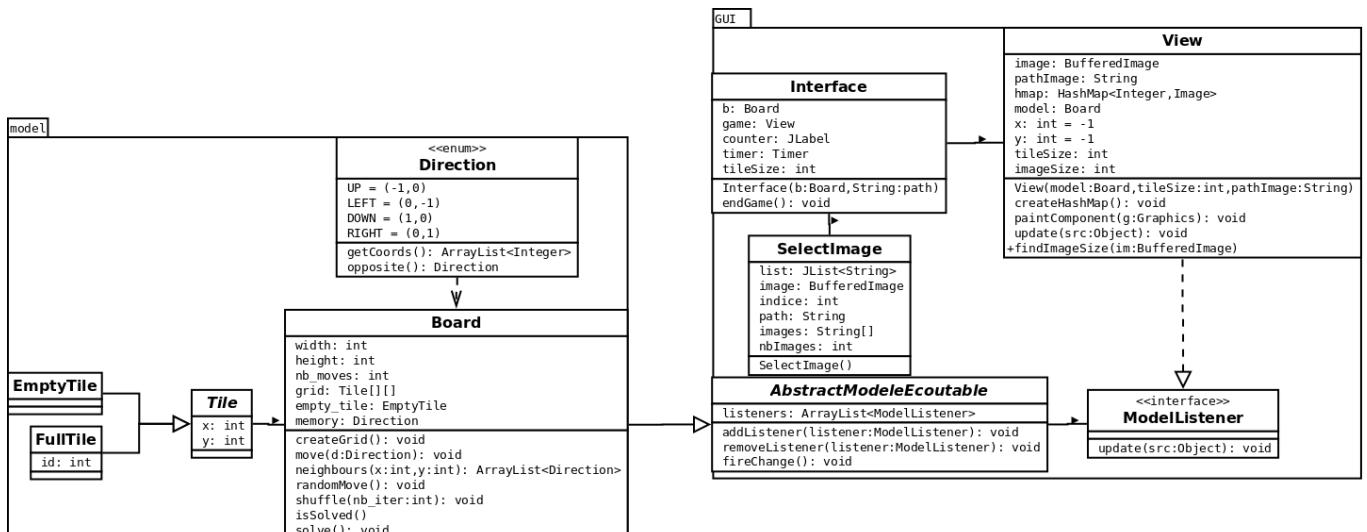


FIGURE 4 – Diagramme de classe MVC

2.2 Fonctionnement GUI

D'abord, pour représenter l'état du jeu, la classe **View** permet à partir d'un chemin, d'une image et d'un Board découper l'image et lier pour chaque identifiant une partie de l'image qui lui correspond. La case vide ne possède pas d'identifiant et pourtant elle a une image qui est associé à son identifiant si elle en possédait un, c'est comme cela que quand le puzzle est résolut, on dessine toute l'image sans la case vide et on enlève aussi le quadrillage pour avoir l'image propre.



FIGURE 5 – Visuel de la résolution du Taquin

Pour choisir l'image qui est en fond, nous avons fait une autre interface qui va chercher tous les éléments qui sont dans le dossier **ressources** (qui sont que des images) et qui seront listé dans une liste (en utilisant **JList**, **Vector** et **JScrollPane** pour gérer une plus grande quantité d'image). On ajoute dans le vecteur que les noms des images sans l'extension et on créer la **JList** avec le **Vector** en argument. Ensuite on créer le **JScrollPane** et on ajoute la **JList**. Pour ajouter le tout, on ajoute le **JScrollPane** de la même manière qu'un **JPanel** avec un **add** dans la frame. Quand on sélectionne une image dans la liste, elle s'affiche sur la droite de la liste. Enfin, une fois l'image voulu sélectionné, en appuyant sur le bouton **Jouer** on lance l'interface du jeu avec l'image choisi en fond. Toutes les images sont possible d'ajouter pour le format et la taille mais si l'image est petite, elle sera étendu et le visuel sera pixelisé.



FIGURE 6 – Images dans le dossier **ressources**

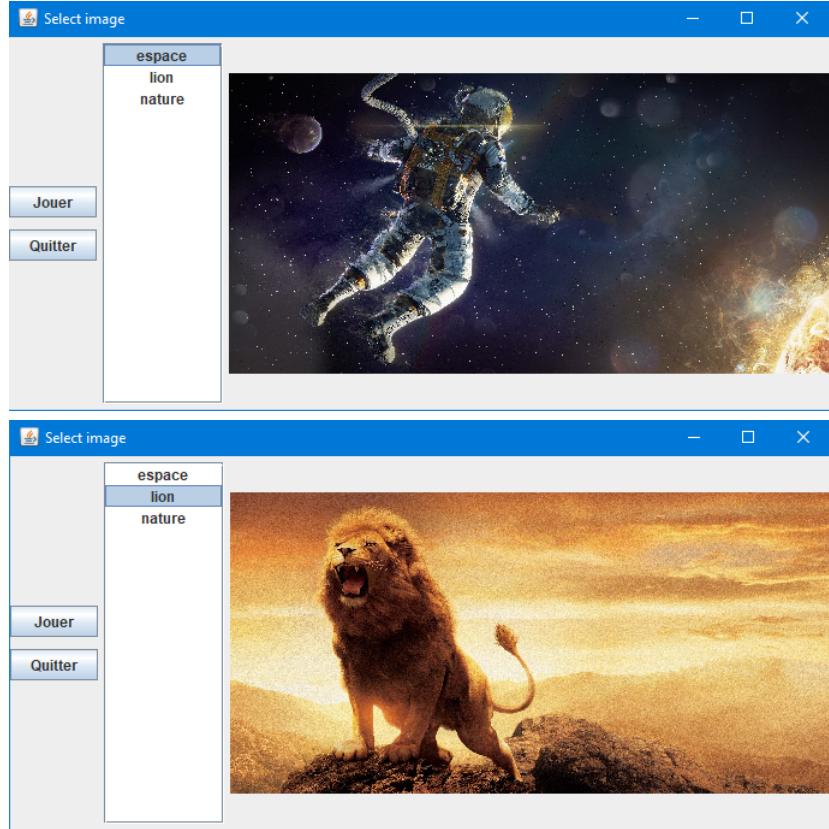


FIGURE 7 – Interface de sélection d'image

Les déplacements sont fait soit avec le clic de la souris ou soit avec les flèches directionnelles du clavier.

Pour le déplacement de la souris, on récupère la position du clic et on le divise par la taille que les cases ont et enfin on prend la partie entière ce qui nous donne les coordonnées en cases du clic de souris. Il nous reste maintenant à tester si le clic de la souris correspond à une énum, on va soustraire les coordonnées de la case cliquée avec la case vide ce qui nous donne le vecteur de déplacement et si ce vecteur correspond à une des énums des positions jouable de la case vide on effectue le mouvement sinon on fait rien.

Pour le déplacement au clavier, c'est plus simple, on teste qu'elle touche est pressée et si c'est la touche du haut on lance la fonction **move** du Board (qui test si la case de destination est dans la grille), de même pour les autres directions.

Un bouton **recommencer** et **resolution** sont présent en bas de l'interface de jeu qui permettent respectivement de recommencer le jeu et de le résoudre. Pour recommencer le jeu, on a juste à mélanger le jeu.

Conclusion

L'objectif du projet a été atteint, nous avons réalisé le jeu du Taquin jouable en console et une version jouable en interface graphique.

Nous aurions pu améliorer ce projet par exemple en proposant au joueur les dimensions de la grille de jeu que nous avons mis par défaut en 3x3 mais nous avons fait en sorte que la grille et l'interface soient adaptatif aux dimension de la grille.

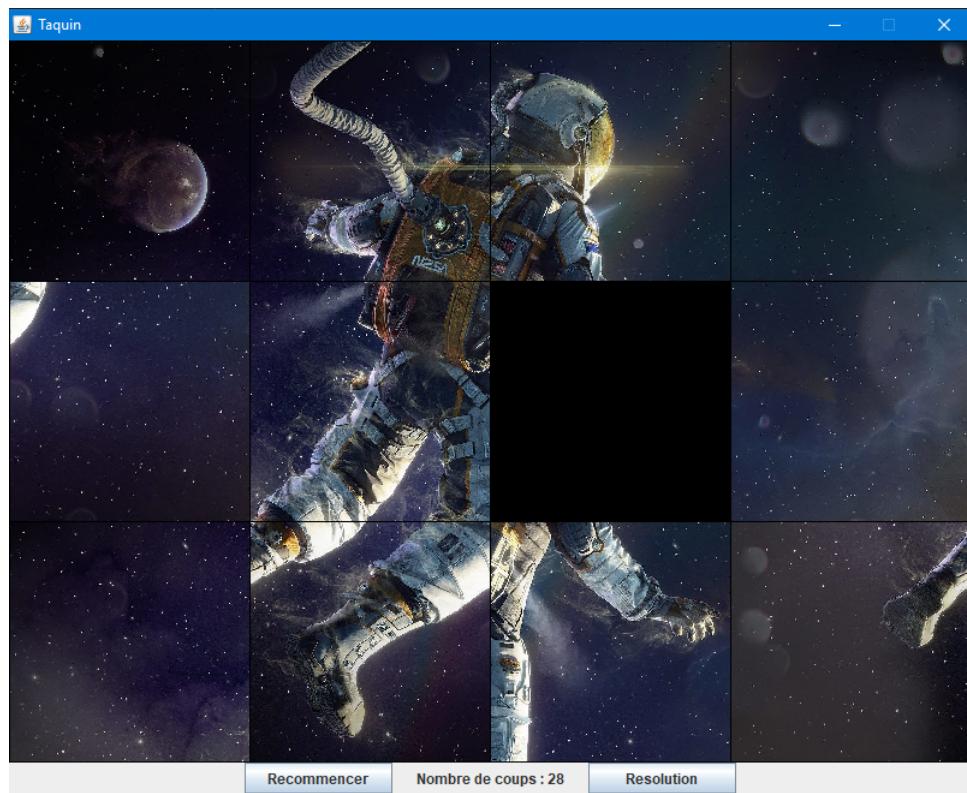


FIGURE 8 – Interface graphique avec une dimension 4x3