COMPLEMENT DE PROGRAMMATION ORIENTÉÉ OBJET

JEU DE PUZZLE À GLISSIÈRES (TAQUIN)

Equipe de développement :

DOUMBOUYA Sékou

KITSOUKOU Manne Emile

OROU-GUIDOU Amirath Fara

ROUSSEAU Pierre-Alexandre

Parcours : Licence 2 Informatique

Groupe : 2B

Sous la supervison :

CHARRIER Christophe

Avril 2022

Table des matières

1 Présentation du Projet

Le **Taquin** est un jeu de puzzle, constitué d'un carré (le nombre de ligne est égal au nombre de colonne) ou d'un rectangle(le nombre de ligne est différent du nombre de colonne) dans lequel se trouve des cases. Ces cases contenant des lettres de l'alphabet, des nombres ou encore des morceaux d'images, peuvent glisser les unes sur les autres. Parmi les cases, l'une d'entre elles est vide. Le but du jeu du taquin consiste à reconstituer le puzzle en formant une image (lorsque les cases forment une image) ou de ranger les nombres par ordre croissant (lorsque les cases contiennent des nombres) en glissant l'un des éléments contigus à la case vide vers celle-ci. Avec l'interface graphique, l'utilisateur a la possibilité de choisir sa propre image pour le taquin qu'il devra reconstituer.

Le but de ce devoir a été de réaliser un taquin sous forme d'une application de jeu, dotée d'une interface graphique. Cet application devrait être réaliser intégralement MVC, avec un modèle totalement indé- pendant de l'interface graphique.

2 Organisation du projet

2.1 Gestion du projet

Afin de faciliter la communication et le bon déroulement de la conception de notre application, divers moyens ont été mis en œuvre.

2.1.1 Gestionnaire de version

Tout d'abord, nous pouvons citer l'utilisation d'un gestionnaire de version afin de permettre la cen- tralisation du code et rendre le travail en équipe bien plus efficace. Le choix de celui-ci étant imposé (Subversion) mais pour notre équipe nous avons faire le dépôt sur Gitlab.

2.2 Compilation

Pour la compilation, nous avons décidé d'utiliser un script de compilation [1]**ANT**(logiciel créé par la fondation Apache qui vise à automatiser les opérations répétitives du développement de logiciel telles que la compilation, la génération de documents (Javadoc) ou l'archivage au format jar) en Java. Il s'agit de build.xml. Celui-ci possède différentes sous-commandes très utiles permettant d'effectuer les tâches courantes sur le projet suivit de ant. En voici la liste :

- **ant init** initialise le projet
- **ant run** lance le projet
- **ant javadoc** génère le javadoc du projet
- **ant nettoyage** nettoie les dossiers générés
- Et plein d'autre

3 Architecture du projet

3.1 Mise en place du pattern MVC

L'ensemble de l'application a été réalisé en faisant usage au pattern MVC. Ainsi nous avons pu répartir l'application en differents packages essentiels :

- modele
- vue
- controleur

Le tout est completé par un package contenant la classe principale qui permettra le lancement de l'application

1. C'est un Apache en Java

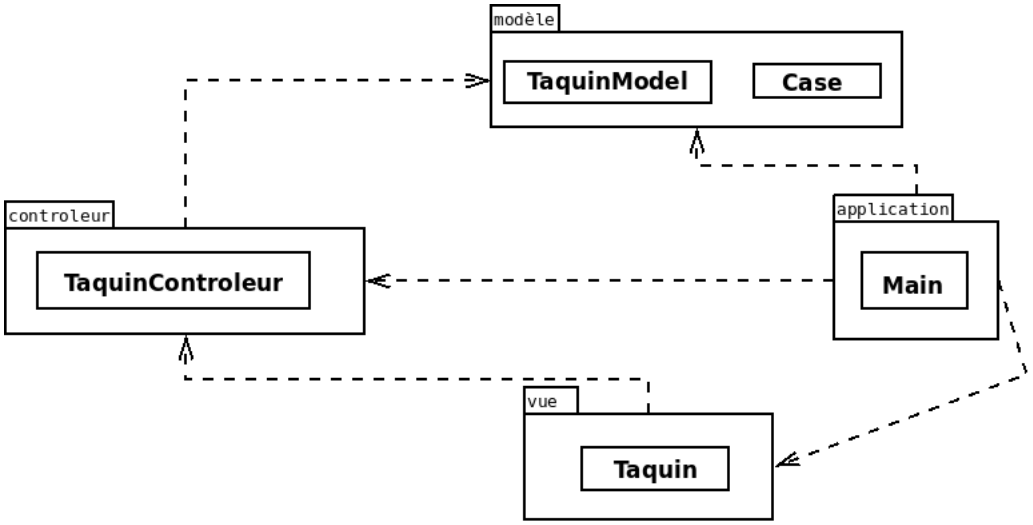

FIGURE 1 – Diagramme des packages du pattern MVC

Modèle

Le choix a été fait de modeliser le jeu de Taquin en l'assimilant à une liste de cases contentant des numeros allant de 0 à (taille-1) du plateau. Ainsi un etat de taquin est identifié par :

- La liste de ses cases
- la dimension du plateau de jeu
- la position actuel de la case vide

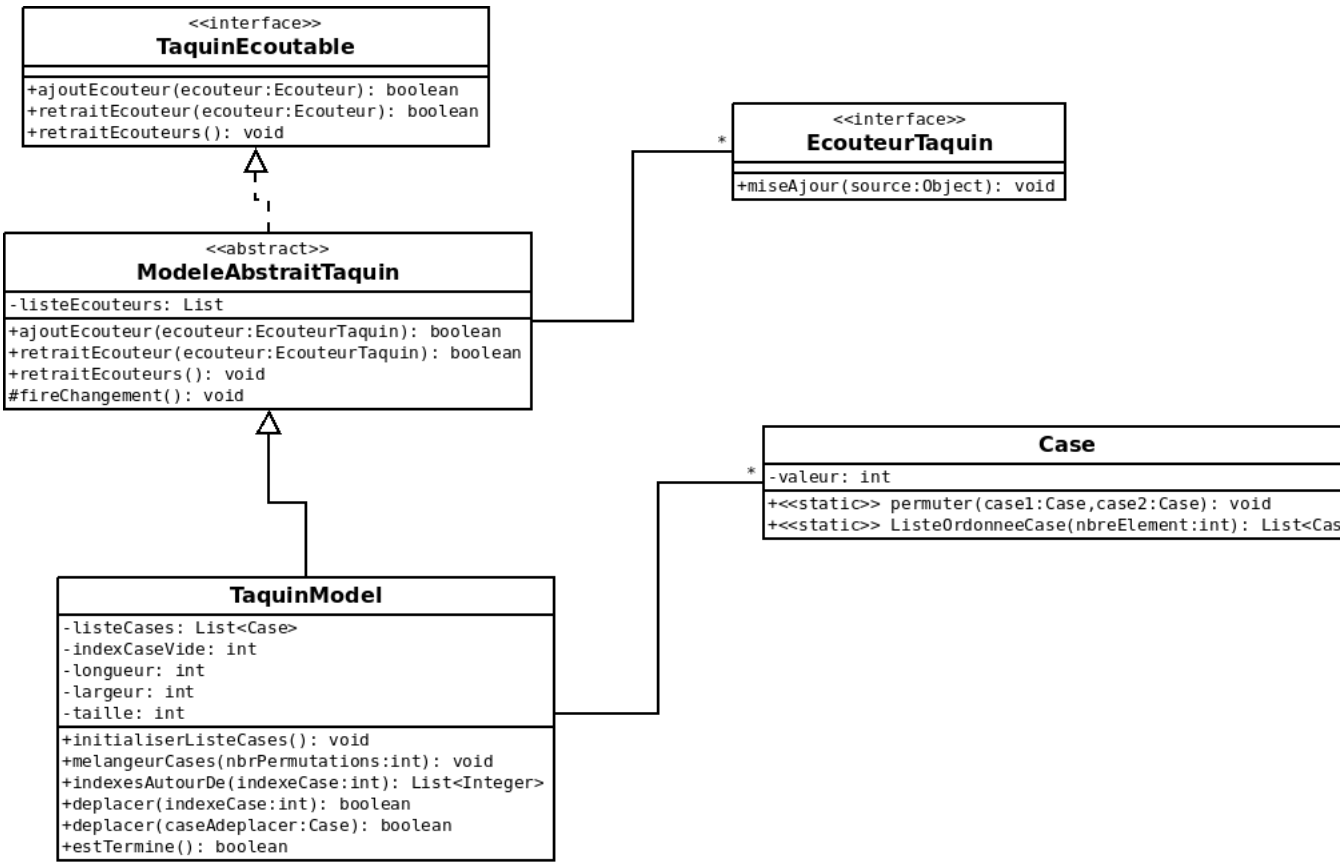

FIGURE 2 – Diagramme du package modele

Les principales méthodes utilisés pour le fonctionnement de l'application est :

- **deplacer** qui permet de deplacer une case de taquin si cela est possible
- **melanger** qui melange le taquin et donne toujours un configuration solvable

Controleur

Ce package contient principalement le controleur. Le but principale de notre controleur est de contraindre le passage par ce dernier pour pouvoir modifier un element du modele. Ce qui fait que nous effectuons principalement 2 types de modifications :

- Le deplacement d'une case du jeu
- Et remelanger completement le plateau de jeu

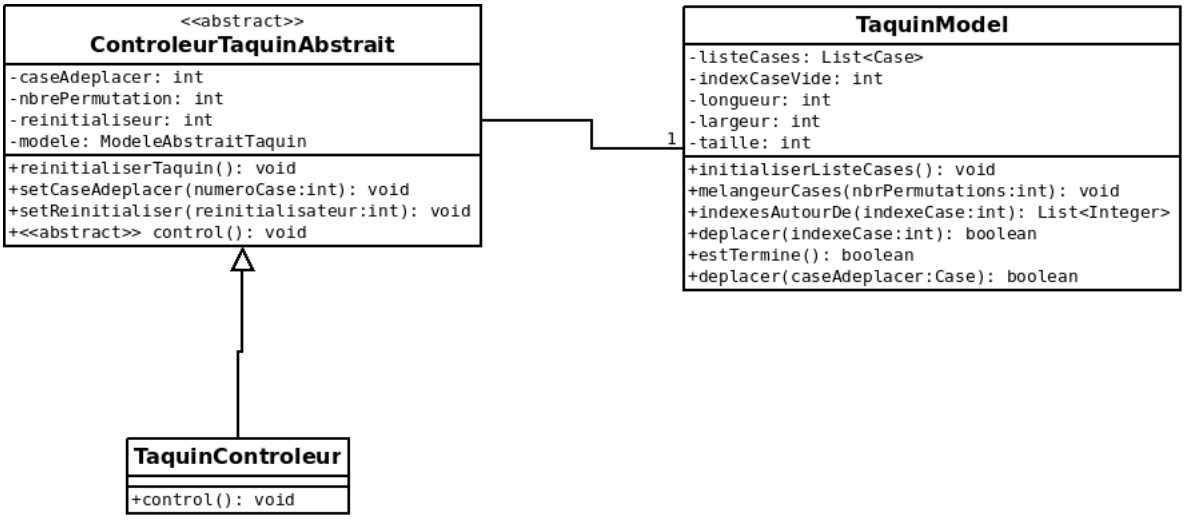

FIGURE 3 – Diagramme du package Controleur

Vue

La partie vue, c'est la partie visible du Taquin. Ce n'est qu'une partie qui vient se greffer à notre modèle. Il permet l'implementation de la partie graphique qui n'est pas indispensable pour le déroulement d'une partie. Principalement nous effectuons uniquement une mise à jour de l'affichage quand il y'a modifications du modele

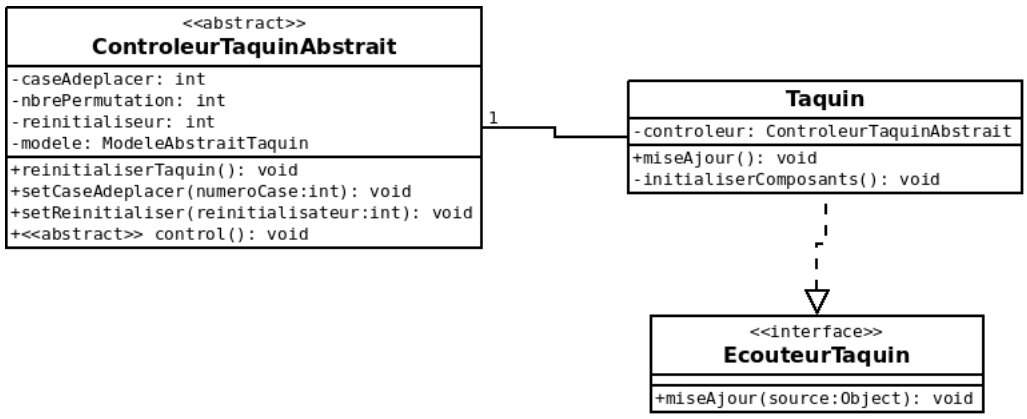

FIGURE 4 – Diagramme du package vue

4 Aspects Techniques

4.1 Retour sur la modelisation

L'état d'un taquin est principalement identifiable par la position des differentes case dans le plateau. Le choix a été fait d'utiliser une liste qui pour une case donnée sa position dans le plateau serait tout simplement son indice dans la liste.

De plus nous voulions que redonner un ordre à ses cases. Généralement, pour des entiers naturels, le chiffre **0** represente le plus petit element de notre ensemble. Cependant le choix a été fait de considerer ce chiffre comme le plus grand des entiers. Ainsi toute case en terme de comparaison sera toujours plus petit que la **case 0**. Néamoins l'ordre globalement des entiers reste le même. Ce choix a été fait pour que lors de la vérification le jeu terminé aura tout simplement une configuration dont la liste des cases est trié de $\{1, 2, .., taille - 1, 0\}$

4.2 Algorithme de déplacement

Pour effectuer un deplacement nous suivons une logique assez spécifique. Cette logique est repris par cet Algorithme :

Algorithme 1 : Algorithme de deplacement

```
1 «««< HEAD =======
  Entrées : Une liste de case listeCases dans un ordre quelconque
  Sortie : Une liste de case ayant une seule difference au maximun
           avec la liste d'entree
2 si indice ∈ indicesAutour(positionVide) alors
3 |  listeCases ← swap(listeCases, positionVide, indice)
4 retourner listeCases
```

4.3 Algorithme de mélange

Pour aboutir toujour à une configuration solvable le melange doit suivre une certaine logique

Algorithme 2 : Algorithme de deplacement

```
  Entrées : Une liste de case listeCases dans un ordre quelconque
  Sortie : La liste listeCases melangé
1 tant que permutation ≠ 0 faire
2 |  indice ← un indice aléatoire dans indicesAutour(positionVide)
  |   listeCases ← swap(listeCases, positionVide, indice)
  |   permutation ← permutation − 1
3 fin
4 retourner listeCases
```

5 Manuel d'utilisation

Pour jouer à l'application avec l'interface graphique vous pouvez :

- Cliquer sur l'image a déplacé avec la case vide
- Deplacer directement la case vide à l'aide des touches directionnelles

Vous pouvez charger n'importe quelle image mais veuillez préférer des images ayant de bonnes résolutions(Voir Menu)

Durant la partie, quand une case est bien positionné, elle est autouré de vert. Quand la case peut être remplacer par la case vide, elle est autouré de violet. Ainsi le jeu est terminé quand toute les cases sont autouré de vert.

Pour melanger le plateau, veuillez utiliser la combinaison **ctrl + m**, le plateau sera melanger environs 2O0 fois. Par défaut au lancer du jeu vous voyez la configuration finale du plateau. Ainsi n'oubliez pas mélanger. Vous pouvez aussi mélanger en fixant vous même le nombre de permutation(Voir menu). »»»> c9594af9a3646ecd517ca6ed487a390652dad6f7

6 Conclusion

6.1 Avis général

Le Taquin a été une approche pratique intéressante concernant le Pattern MVC. L'intérêt de celui-ci apparaît très clairement lors du changement de JPanel où l'on conserve le même modèle (TaquinModèl) bien que la manière d'afficher l'information diffère, soit en affichant une image, soit en affichant des chiffres. Le pattern permet également une nette séparation entre les classes modèles qui se retrouvent réutilisables dans un autre contexte, dû au fait qu'elles n'embarquent pas de code spécifique au contrôle des évènements ou à l'affichage de la vue.

6.2 Éléments à améliorer

- Sauvegarde des scores avec le nom d'utilisateur du joueur
- Ajout d'un mode versus où 2 joueurs seraient en compétition pour finir le plus rapidement pos- sible une même grille
- Le temps mit pour arriver à l'état final