UNIVERSITÉ
CAEN
NORMANDIE

Rapport Projet
Complément POO
PUZZLE

ATTY Abla Cathérine Gloria
Erwan Phillipe MENSAH
TOURE Papa Samba Khary
Daouda TRAORE

L2 Informatique 2021-2022 Université Caen Basse-Normandie.

Table des matières

1 Introduction

Ce rapport a pour objectif de décrire le processus de conception d'une application de jeux dotée d'une interface graphique qui consiste en un puzzle à glissière(voir image ci dessous).

Le jeux consiste à faire glisser les cases adjacentes à la case vide qui en fonction de la situation peut varier entre **deux** à **quatre** possibilités. Le jeux se termine une fois l'image est reconstituée ou une fois que l'on a retrouvé le bon ordre des cases.

Le but du devoir est donc la conception d'une telle application mais intrégralement realisée en utilisant l'architecture ***Model-View-Controller*** (**MVC**) dont nous reparlerons plus en detail plus tard.

Ce rapport contient l'ensemble des éléments ayant permis la conception du modèle, de la vue et du contrôleur, du mode fonctionnement le l'architecture **MVC**, ainsi que comment le travail fut répartit entre les quatre membres du groupe.

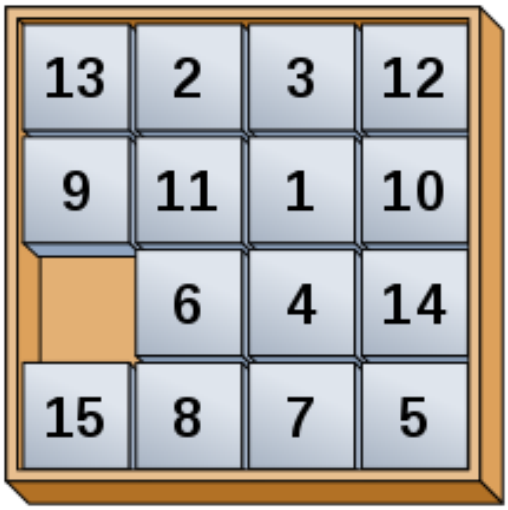

Figure 1: Exemple Taquin

2 Répartition du travail

ATTY :

- View
- Controller
- Rapport

Erwan MENSAH :

- Model
- Controller
- Rapport

Daouda TRAORE :

- View
- Controller
- Rapport

TOURE Papa :

- Model
- View
- Controller
- Rapport

Plutôt qu'une répartition en groupe c'était plutôt un travail collectif dans tous l'ensemble du projet.

3 Architecture du projet

3.1 Description du pattern MVC

L'architecture **MVC** (*Model-View-Controller*) spécifie q'une application consiste en un ***Modèle*** contenant les données et les différentes fonctions de l'application, d'une ***Vue*** assurant la présentation de l'information et d'un ***Contrôleur*** gérant l'information.

La principale motivation dérrière une telle architecture etait de permettre la création d'un interface graphique pour n'importe quel type d'objet. De nos jours le pattern MVC est adopté dans la majorité des applications et des langages de programmation.

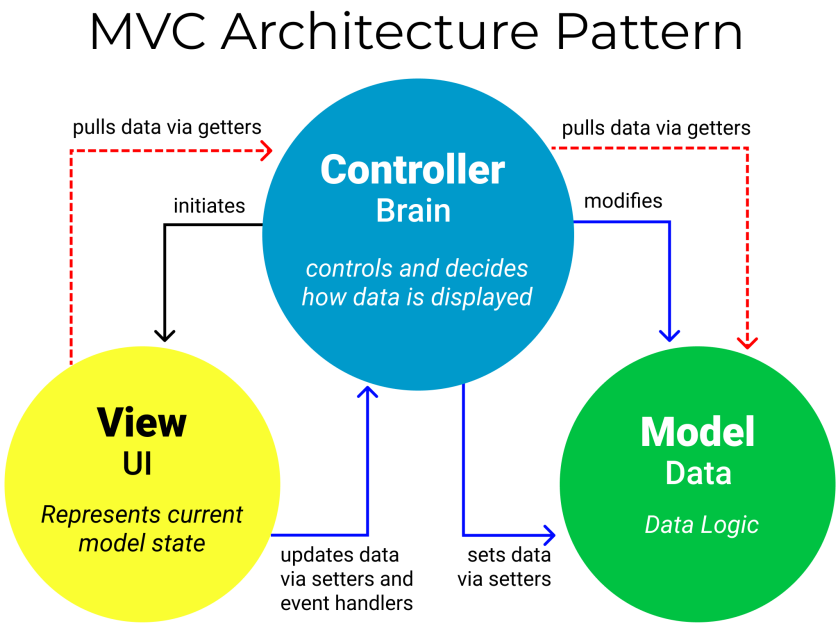

Figure 2: Architecture MVC

* Le ***Modèle*** contient uniquement les données pures de l'application, il ne contient aucune méthode describant comment afficher les données à l'utilisateur et est plus ou moins complétement isolé de la vue.

* La ***Vue*** présente les donnée du modèle à l'utilisateur. La vue sait comment acceder aux données du modèle mais n'en connait ni leur signification ni en quoi l'utilisateur peut en faire.

* Le ***Contrôleur*** existe entre le modèle et la vue. Il agit sur le modèle en fonction des demandes de l'utilisateur.

3.1.1 Avantages

L'architecture dispose de plusieurs avantages :

- La Séparation des tâches, séparer la logique métier, l'interface utilisateur et la dynamique du système.
- Plusieurs développeurs peuvent travailler simultanément sur le Contrôleur, la vue ou modèle.
- Permet la création de multiples vues pour un unique modèle.
- La Reutilisabilite impliquant ainsi un gain en temps.

3.1.2 Désanvantages

Néanmoins des désavantages existent :

- La navigation de l'architecture devient plus complex car elle introduit un niveau supplémentaire d'abstraction.
- Un nombre plus grand de fichiers supplémentaires à manipuler.

4 Mise en place de MVC dans le cadre du projet

4.1 Arborescence du projet

L'application est organisée de la manière suivante: Comme nous pouvons le

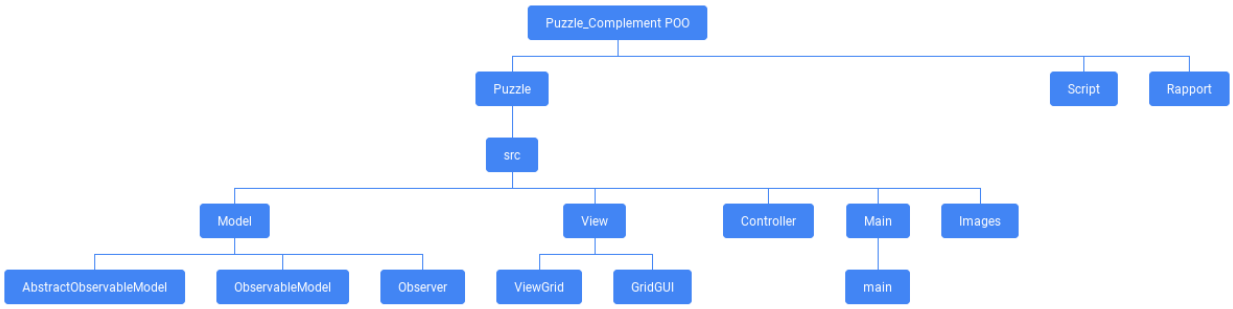

Figure 3: Arborescence Projet

voir le projet est composé principalement de trois répertoires:

- Script : contenant les différents script
- Rapport : contenant le rapport du projet
- src : composé en cinq packages dont ***model, view, controller*** reprenant ainsi l'architecture MVC. Un sous package ***main*** contenant l'exécutable et repertoire image contenant les différentes images.

4.2 Diagramme UML

Dans le projet il nous a été demandé d'utilisé le design pattern MVC. Il a été implémenté de la manière suivante:

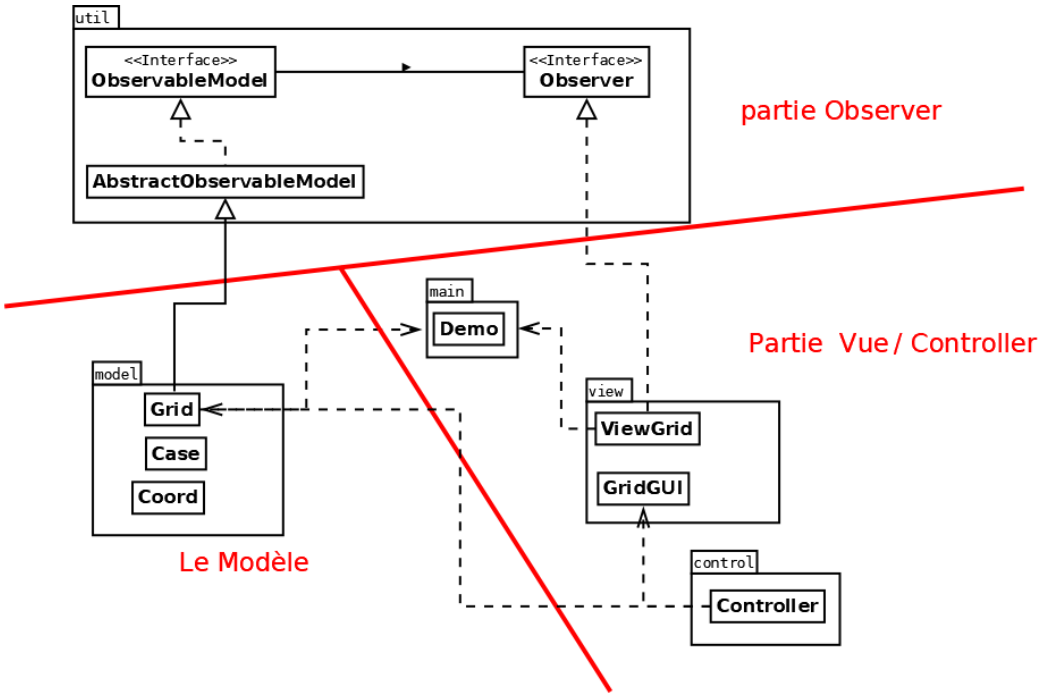

Figure 4: Diagramme de l'application

4.2.1 Model

Le modèle est essentiellement la classe Grid ou se place la totalité de la logique. Tous les traitements restent identiques quelques soit le mode d'affichage souhaité. Les classes Coord et Cases peuvent être considérées comme étant des composantes.

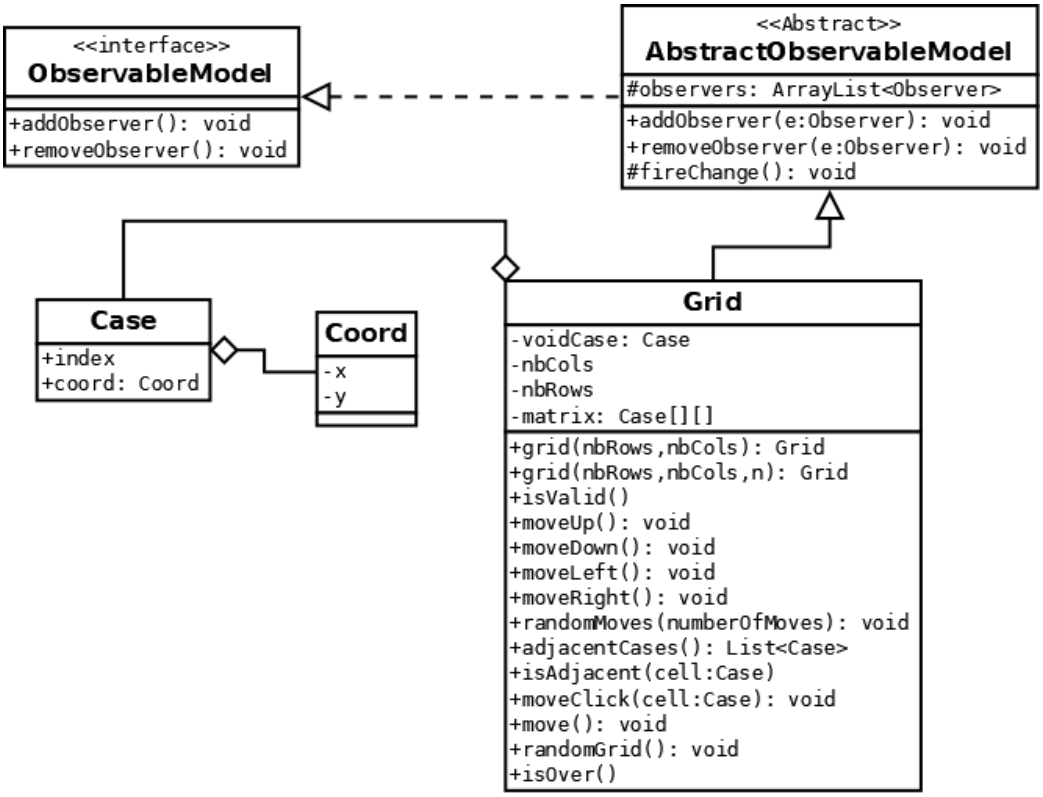

Figure 5: UML du modèle

Comme nous pouvons le voir la classe Grid hérite de AbstractObservable-Model permettant ainsi aux observers d'être notifié à chaque fois que son état change grace à la méthode fireChange().

```java
protected void fireChange() {
    for(Observer e : observers) {
        e.updateModel(this);
    }
```

4.2.2 View + Controller

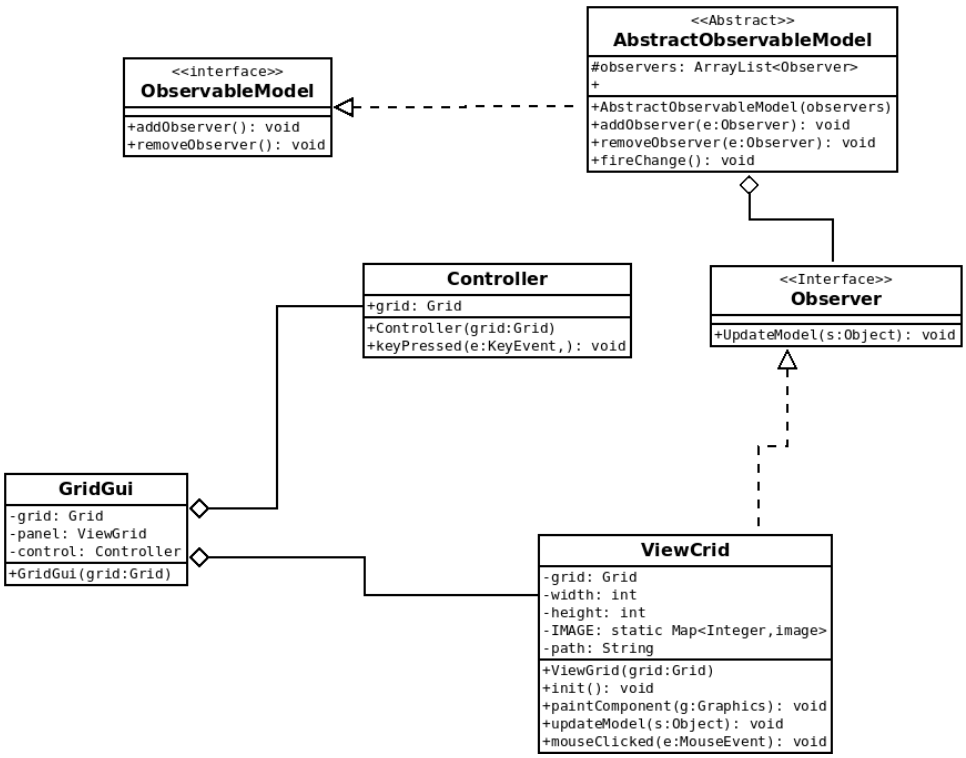

Figure 6: UML View et Controller

La classe **ViewGrid** s'abonne au modèle en implémentant l'interface **Observer** et en prenant bien le modèle **Grid** comme attribut. Ainsi à chaque changement d'état la vue est notifiée et se met à jour.

GridGUI est la frame assurant l'affichage de l'application.

La classe **Controller** permet de faire déplacer une case en utilisant les fléches directionnelles.

5 Eléments techniques

5.1 Grid

Commençons par parler de la classe **Grid**. En effet il nous a été demandé de faire en sorte que le puzzle soit mélangé mais de sorte qu'il soit quand même solvable. En effet un méthode brute-force avec une utilisation de chiffres randomment générés aller à coup sur résulter à la creation d'un puzzle dont il est impossible de résoudre. Pour contourner ainsi ce problème rentre en scène la méthode ***randomMoves***.

Elle prends un certains nombre de moves à faire et génére un chiffre random entre 0 et 4 exclus avec 0 étant la direction *Nord* , 1 *Sud*, 2 *Est* et *Ouest*. La méthode est faite en sorte de garder en mémoire à chaque le direction précédente empêchant ainsi de faire des allers et retours. Exemple : prendre la direction nord puis la direction sud ce qui revient à ne pas bouger(voir algorithme ci-dessous).

Algorithm 1: randomMoves(int numberOfMoves):void

```
   Input: Grid
   Output: Shuffled grid
 1 past ← null
 2 while numberOfMoves > 0 do
 3 |   r ← new Randow()
 4 |   nbrRandom ← r.nextInt(4)
 5 |   move ← null
 6 |   if nbrRandom == 0 and past <> 1 then
 7 |   |   move ← UP
 8 |   |   past ← 0
 9 |   |   numberOfMoves ← numberOfMoves − 1
10 |   if nbrRandom == 1 and past <> 0 then
11 |   |   move ← DOWN
12 |   |   past ← 1
13 |   |   numberOfMoves ← numberOfMoves − 1
14 |   if nbrRandom == 2 and past <> 3 then
15 |   |   move ← LEFT
16 |   |   past ← 2
17 |   |   numberOfMoves ← numberOfMoves − 1
18 |   if nbrRandom == 3 and past <> 2 then
19 |   |   move ← RIGHT
20 |   |   past ← 3
21 |   |   numberOfMoves ← numberOfMoves − 1
22 end
```

5.2 ViewGrid

La méthode ***init()*** assure le découpage de l'image en subimages. Le découpage est fait en sorte que l'image sera découper en **n * m - 1** sous images avec n étant le nombre de lignes et m le nombre de colonnes, puis ensuite chacune d'entre elle est stockée dans une *HasMap*. La hauteur et la largeur de chaque sous imges est calculée en fonction de celle de l'image d'origine et des dimensions de la grille(Voir code ci dessous.)

```
1  public void init() {
2    BufferedImage img = null;
3    try {
4      img = ImageIO.read(new File(path));
5      width = img.getWidth(null) / grid.getNbCols();
6      height = img.getHeight(null) / grid.getNbRows();
7      for (int j = 0; j < grid.getNbRows(); j++) {
8        for (int i = 0; i < grid.getNbCols(); i++) {
9          int x=img.getWidth()*i / grid.getNbCols();
10         int y=img.getHeight()*j / grid.getNbRows();
11         IMAGE.put(grid.getMatrix()[j][i].getIndex(),
12         img.getSubimage(x, y, width, height));
13       }
14     }
15   } catch (IOException e) {
16     System.out.println("No image found");
17   }
18 }
```

6 Conclusion

6.1 Avis Général

Ce projet fut trés instructif et nous aura mieux permis de cerner et de se familiariser avec le patter **MVC**. L'intêret de celui-ci apparait clairement car permettant différentes visualisations du model sans avoir à le changer que cela soit en ligne de commande via des Sysout ou en affichage intétgrale via un interface graphique. Elle permet également un sépération du modèle par rapport au reste rendant les classes composant le modéle réutilisables car n'ayant aucun code ayant rapport avec la vue/controller.

6.2 Eléments à améliorer

Bien que tout ce qui a été demandé dans le sujet a été remplit, pas mal de choses restent quand même à améliorer mais faute de temps et du nombre de projet à finir en ce fin semestre tout ce que l'on souhaité n'a pas pu être accomplit. Par exemple:

- Faire en sorte que la frame se resize en fonction de l'image chargée.
- Faire en sorte que l'image s'adapte à la taille de la frame lorsqu'on l'aggrandit.
- Faire en sorte que la case survolé par la souris si déplacable soit mise en avant.