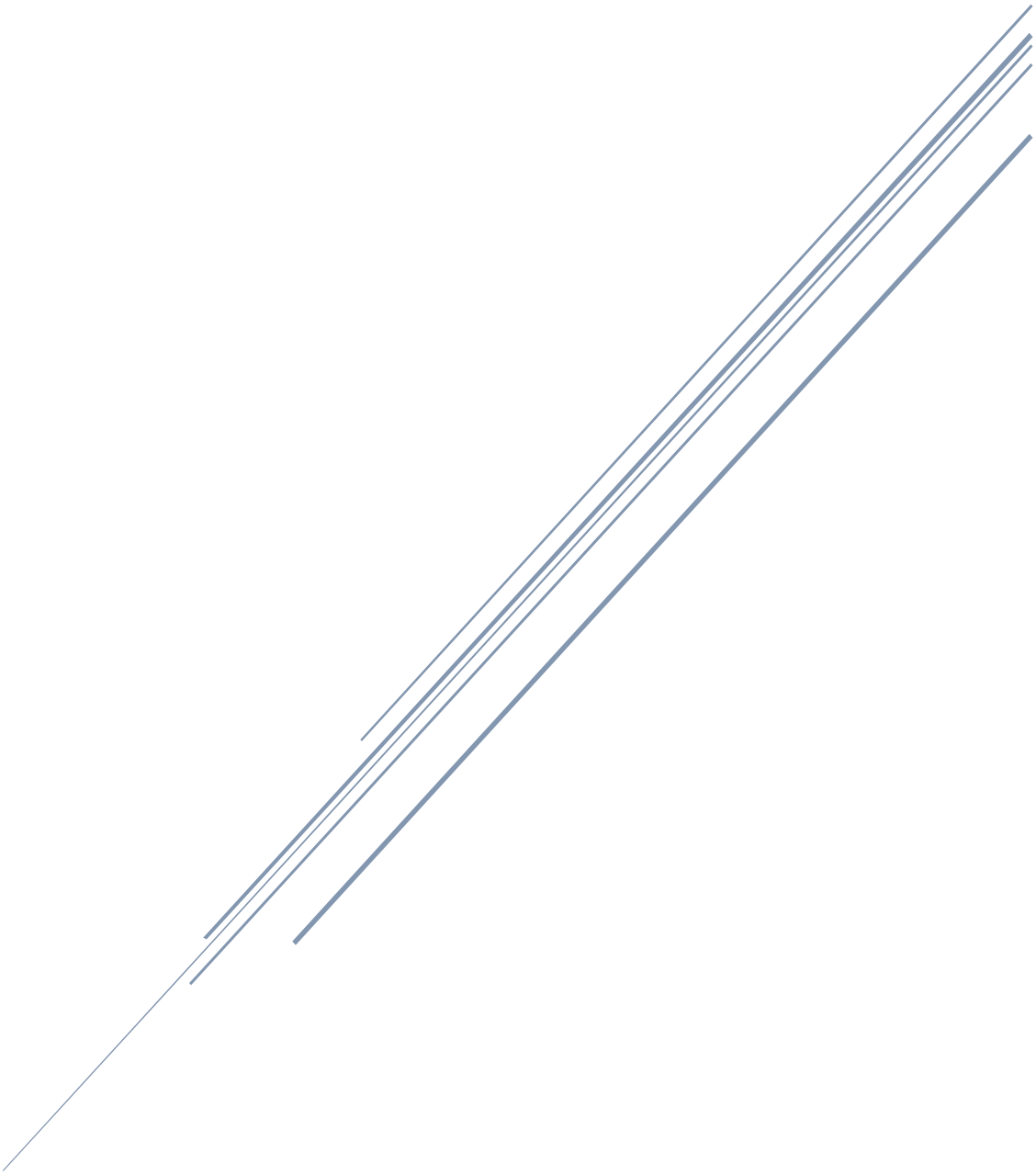
RAPPORT PROJET POO PUZZLE

Gérome FERRAND & Aymeric TOUCHE

Lien GitHub du projet : https://github.com/Skald84/Projet-POO-PuzzleGame



Université Lyon I Clause Bernard

Table des matières

I. Modèle MVC 2
A. Package VueController 3
B. Package Model 3
C. Package Images 3
II. Les fonctionnalités 3
A. Drag and Drop 3
B. Choix du niveau 4
C. Affichage des règles 6
D. Retour au menu 6
E. Aide 6
F. Une zone d'affichage 6
III. Analyse UML 0

I. Méthodes de travail

A. Organisation du travail

Avant toutes choses, il parrait utile de préciser que nous avons procédé en deux temps. D'abord une analyse du sujet et du diagramme d'activité fourni nous a permis de nous accorder sur le fonctionnement du jeu et la manière de procéder. Cette modélisation du problème a représenté 40% du temps total de travail sur ce projet. Puis une phase de développement où le code a été créé a représenté 60% du temps. Ce second temps a été régulièrement entrecoupé par des retours à la première phase afin d'affiner nos idées et d'ajouter les fonctionnalités au fur et à mesure.

B. Git

Nous nous sommes servis de Git (GitKracken ou TortoiseGit) afin de pouvoir travailler séparément sur le code. Notre maitrise de cet outil n'est pas optimale. Ce type de compétences étant fortement sollicité sur le marché du travail nous souhaitions nous entrainer à travailler avec.

II. Modèle MVC

A. Package VueController

Le package « VueController » contient une classe javaFX héritant de la classe Application et implémentant l'interface Observer. Cela nous permet d'exécuter le programme via la fonction héritée start de la classe Application, et de rendre la vue attentive aux changements d'état de nos objets placés dans le package « Model » grâce à l'interface Observer.

Cette classe javaFX va fournir un écran d'accueil aux joueurs qui pourront sélectionner le niveau auquel ils souhaitent jouer (trois niveaux sont disponibles).

B. Package Model

Ce package contient deux types de classes.

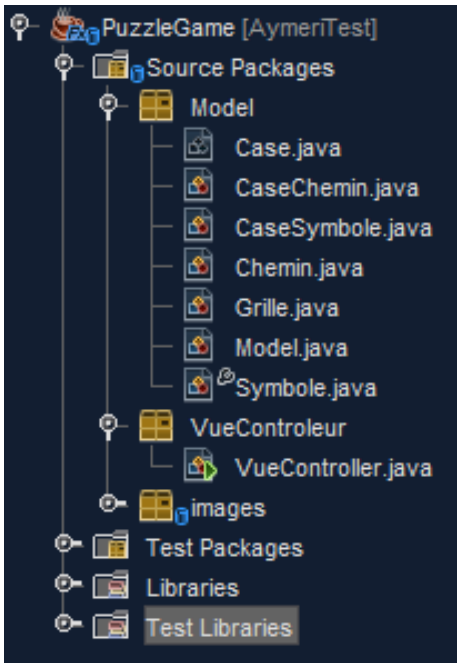
D'abord les classes qui vont instancier les différents éléments du puzzle, la grille, les cases et les chemins. Elles vont comporter des méthodes qui seront appelées par la classe Model.java.

La classe Model.Java est appelée par la vue et le contrôleur selon les actions réalisées par le joueur. C'est elle qui va appeler nos classes citées précédemment : appels des constructeurs pour créer les objets nécessaires, appels des différentes méthodes en fonction des actions du joueur.

Elle permet également de notifier la vue des changements d'état des différents objets afin que celle-ci se mette à jour et propose un affichage cohérent au joueur.

C. Package Images

Contiens les images nécessaires à l'affichage du Puzzle.



III. Les fonctionnalités

A. Drag and Drop (jeu de base)

Il s'agit ici de la fonctionnalité principale, car sans elle le jeu ne pourrait se faire.

La partie contrôleur se charge de récupérer les évènements souris du joueur. Il existe trois fonctions permettant le drag and drop.

- setOnDragDetected : récupère l'évènement clic non relâché
- setOnDragEntered : détecte le mouvement de la souris lorsque le click n'est pas relâché
- setOnDragDone : détecte le relâchement du clic

Chacune de ces fonctions va appeler une méthode de la classe Modele.java

Cette fonctionnalité a été la plus longue à développer, elle a occupé 65 % du temps de développement.

Nous avons utilisé des println pour suivre l'évolution de nos objets dans la console, ce qui nous a permis de mieux cibler les erreurs auxquelles nous faisions face.

B. Choix du niveau

Une page d'accueil a été ajoutée une fois le drag and drop stabilisé. Elle permet de choisir un niveau de jeu et donc des grilles de puzzle différentes.

Ces grilles sont prédéterminées du côté de la vue comme cotée modèle :

```
switch(niveauChoisi) {
    case "1":
        nbLignes = 4;
        nbColonnes = 4;

        // création d'un tableau de conteneur d'image
        ConteneurImageCase = new ImageView[nbLignes][nbColonnes];

        // insertion des images dans le tableau de conteneur d'images + listener Drag & Drop sur chaque conteneur
            for (int row = 0; row < nbLignes; row++) {
                for (int column = 0; column < nbColonnes; column++) {

                    final int fColumn = column;
                    final int fRow = row;
                    ImageView imageView;

                    if ((row == 0 && column == 0) || (row == nbLignes -1 && column == nbColonnes -1)) {

                            // image symbole en début et fin de tableau
                            // création d'un conteneur d'img + ajout image dedans
                            imageView = new ImageView(new Image(VueController.class.getResourceAsStream("/images/S1.png")));
                            ConteneurImageCase[row][column] = imageView;
                            grilleVue.add(ConteneurImageCase[row][column], row, column);
                    } else if ((row == 0 && column == 1) || (row == 2 && column == 3)) {

                            // image symbole en début et fin de tableau
                            // création d'un conteneur d'img + ajout image dedans
                            imageView = new ImageView(new Image(VueController.class.getResourceAsStream("/images/S2.png")));
                            ConteneurImageCase[row][column] = imageView;
                            grilleVue.add(ConteneurImageCase[row][column], row, column);
                    } else {

                        // remplissage des autres cases par la même image vide
                            imageView = new ImageView(new Image(VueController.class.getResourceAsStream("/images/LIBRE.png")));
                            ConteneurImageCase[row][column] = imageView;
                            grilleVue.add(ConteneurImageCase[row][column], row, column);
                    }
```

Côté modèle, le niveau choisi est envoyé en paramètre du constructeur de Grille.java.

Ce constructeur va récupérer un modèle de grille via un attribut tableau statique

```
private final String [][] patron1 = {   {"S1","CC","CC","CC"},
                                        {"S2","CC","CC","CC"},
                                        {"CC","CC","CC","CC"},
                                        {"CC","CC","S2","S1"},
```

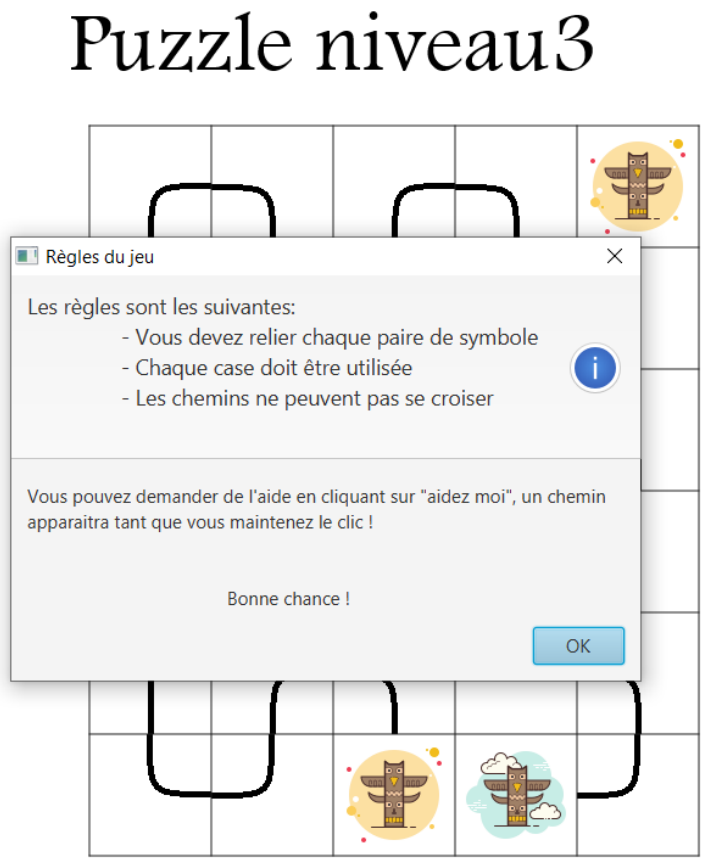
Il va alors lire ce tableau et instancier la grille en lui donnant les bonnes dimensions ainsi qu'en plaçant les cases.

```
public Grille(String niveau){
    nbLignes = 1;
    nbColonnes = 1;
    if(niveau.equals("1")){
        nbColonnes = patron1[1].length;
        nbLignes = patron1.length;

        this.plateauJeu = new Case[nbLignes][nbColonnes];

        for(int i =0 ; i < nbLignes ; i++){
            for(int j =0 ; j < nbColonnes ; j++){
                switch (patron1[i][j]) {
                    case "S1":
                        this.plateauJeu[i][j] = new CaseSymbole(i,j,patron1[i][j]);
                        break;
                    case "S2":
                        this.plateauJeu[i][j] = new CaseSymbole(i,j,patron1[i][j]);
                        break;
                    default:
                        this.plateauJeu[i][j] = new CaseChemin(i,j);
                        break;
                }
            }
        }
    }
```

Il nous aura fallu l'équivalent d'une journée et d'une nuit pour mettre cette fonctionnalité en place, la phase de réflexion et de préparation aura pris quelques heures de plus. Cette focntionnalité représente 19% du temps de développement.

C. Affichage des règles

Le joueur peut demander à consulter les règles du jeu lors d'une partie grâce à un bouton qui les lui affichera en pop up. Cette fonctionnalité représente 1 % du temps de développement (moins d'une heure).



D. Retour au menu

Le joueur peut retourner à tout moment au menu via un bouton (cf. image partie F). Cette fonctionnalité représente 5 % du temps de développement.

E. Aide

Un bouton d'aide a été ajouté, il permet au joueur d'afficher temporairement (le temps du clic) un chemin unique valide (cf. image partie F). Cette fonctionnalité représente 5 % du temps de développement.

F. Une zone d'affichage

Une zone d'affichage a été ajoutée à la fin, elle permet à l'utilisateur de recueillir des informations sur les mouvements effectués, et son avancement.

Cette fonctionnalité représente 5 % du temps de développement (moins d'une heure).

IV. Analyse UML

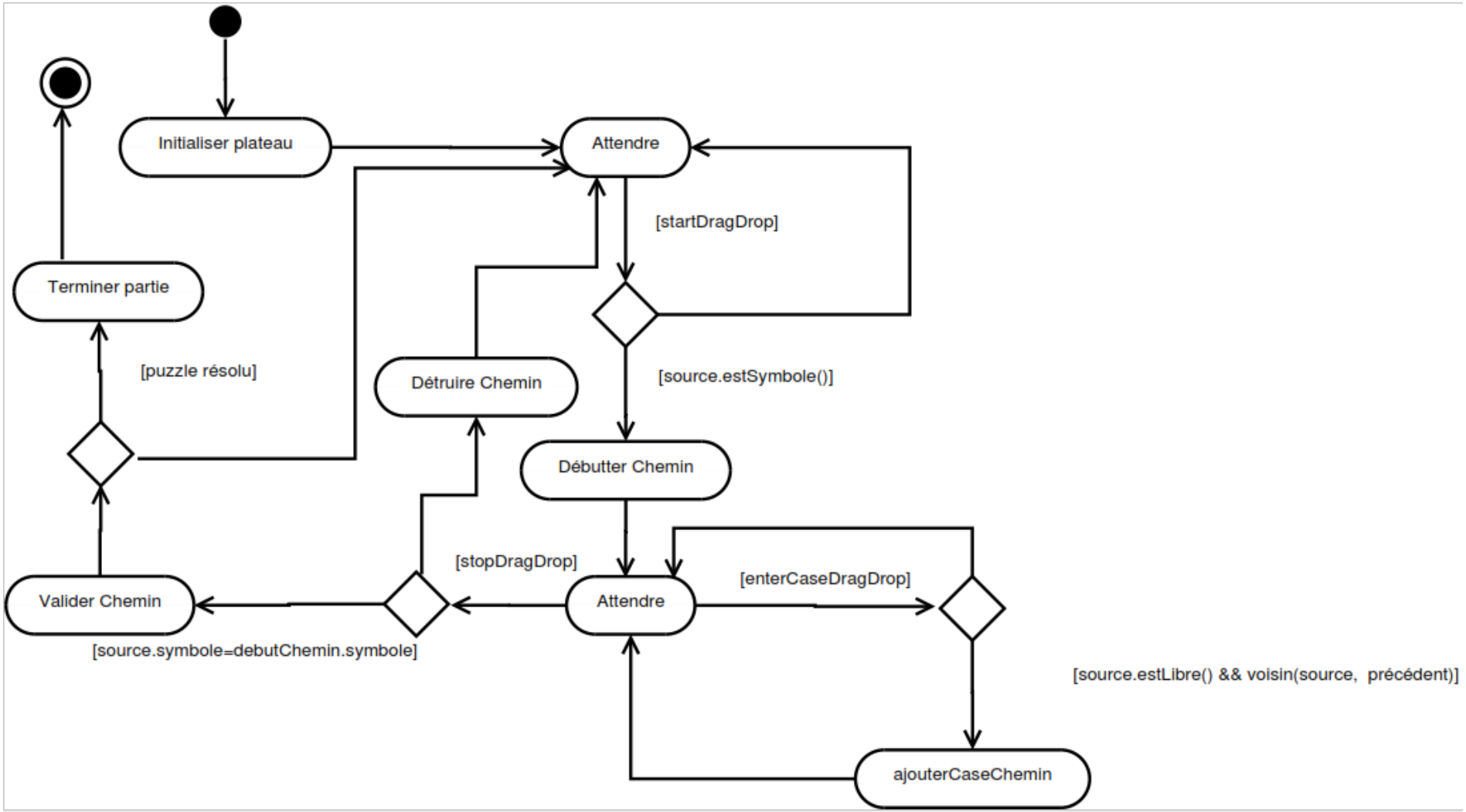


Figure 1 : Diagramme d'activité, proposé par les enseignants

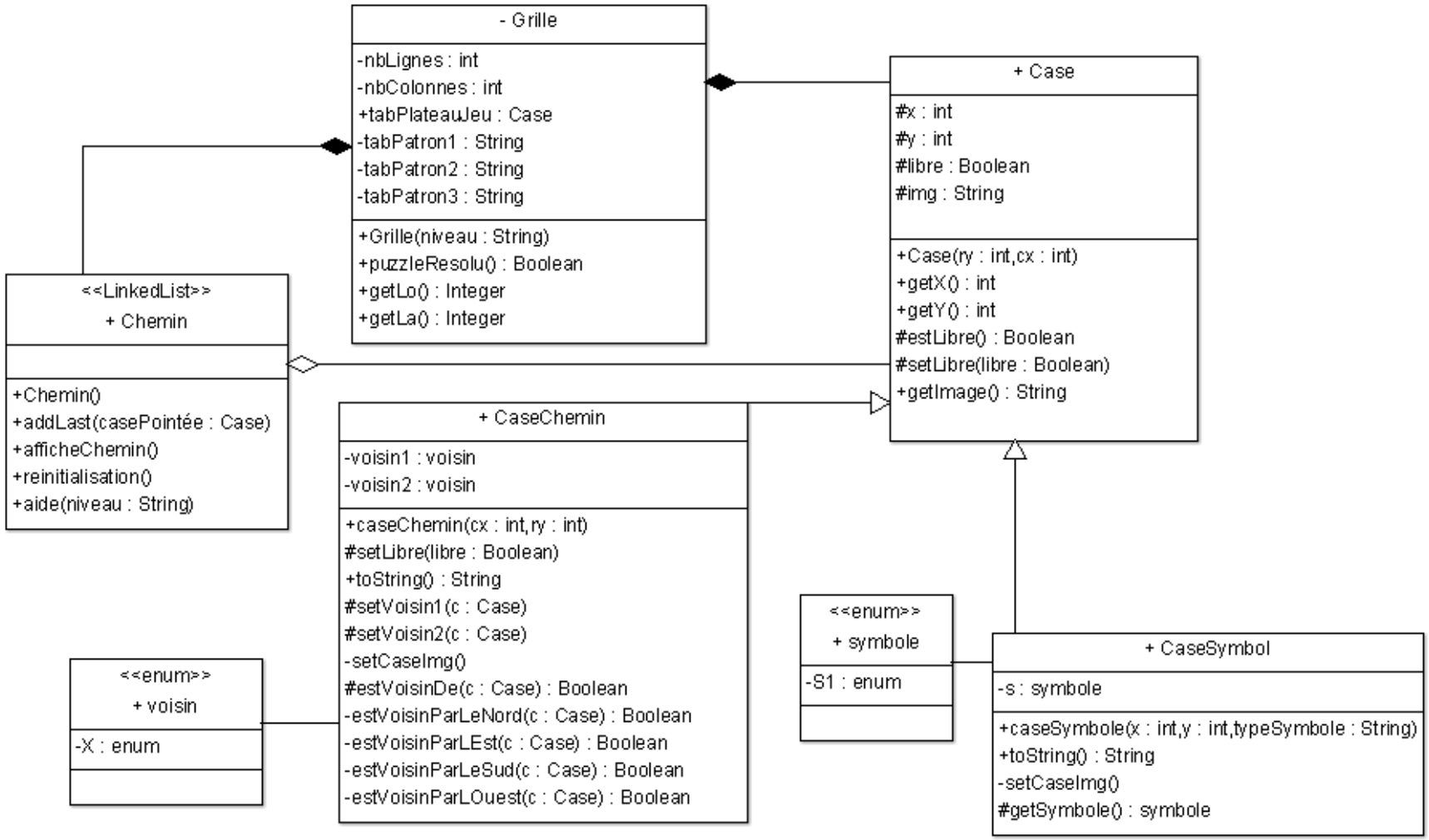


Figure 2 : Diagramme de classes