TP7 : Game of Life

2024-2025

Les objectifs de ce TP sont :

— Conception et implémentation d'une application avec interface graphique.

— Application du motif MVC.

— Création d'une application multi-vue grâce au patron d'architecture MVC.

Le jeu de la vie (*game of life*) est un automate cellulaire imaginé par *John Horton Conway* en 1970. Un automate cellulaire est un modèle où chaque état conduit mécaniquement à l'état suivant à partir de règles préétablies. Le jeu de la vie se déroule sur une grille à deux dimensions dont les cases, appelées cellules, peuvent prendre deux états distincts : **vivante** ou **morte**. Plus d'information sur `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`.

À chaque itération (ou cycle de vie), l'état d'une cellule est entièrement déterminée par l'état de ses huit cellules voisines, selon les règles suivantes :

➤ une cellule morte possédant exactement trois cellules voisines vivantes devient vivante (elle naît) ;

➤ une cellule vivante possédant deux ou trois cellules voisines vivantes le reste, sinon elle meurt.

On veut écrire une application simulant le jeu de la vie. L'architecture (i.e. l'organisation) de l'application suivra le modèle MVC (Modèle-Vue-Contrôleur) :

➤ le modèle contient les données et la logique de l'application ;

➤ la vue est la partie visible de l'application ;

➤ le contrôleur traite les actions de l'utilisateur, modifie les données du modèle et de la vue.

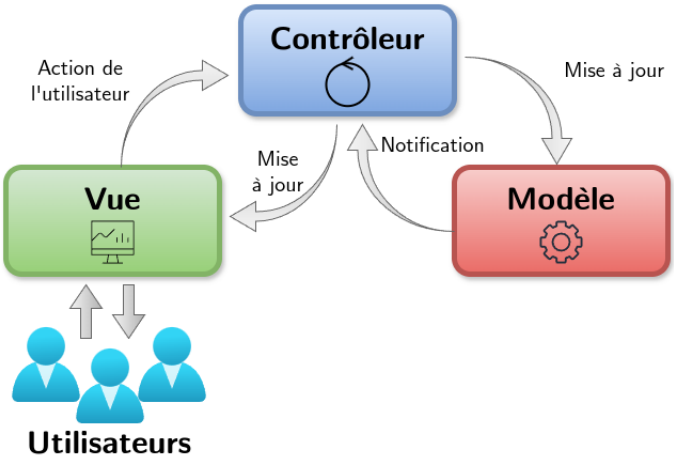

FIGURE 1 – Modèle MVC à suivre pour ce projet.

1 Modélisation

- ➤ Modèle : ce composant gère les données de l'automate et définit la logique métier de fonction, i.e. le passage à l'itération suivante.
- ➤ Vue : permet l'affichage de l'état de l'automate. Le motif MVC permet de prendre en considération plusieurs types de vue. On considérera deux types de vue : une vue en console et une vue GUI (avec Swing). **Attention**, comme l'indique le schéma de la figure 1, les vues n'ont pas d'accès direct au modèle.
- ➤ Contrôleur : traitera les interactions avec l'utilisateur (lancement de la simulation, arrêt de la simulation, etc.) et se chargera de la mise à jour du modèle et des vues (liaison vue-modèle).

Consigne : proposer une modélisation pour cette application respectant le pattern MVC ainsi que les principes SOLID de conception. Pour la modélisation du modèle, vous pourrez envisager une classe `GameOfLifeModel` (cf cas d'étude vue en cours). Vous aurez la possibilité de revenir pour mettre à jour la modélisation au fur et à et mesure de votre implémentation.

2 Implémentation

2.1 Modèle

Implémentez le modèle de votre application. On initialisera la grille de l'automate avec un ratio de 20% de cellules vivantes. **Rappel** : vous n'oublierez pas d'écrire les tests unitaires pour vérifier le bon fonctionnement du modèle avant de passer à la suite.

2.2 Vue (console)

Implémentez la vue qui permettra de visualiser la simulation dans un terminal (console). La figure ci-après présente un exemple d'affichage à une itération.

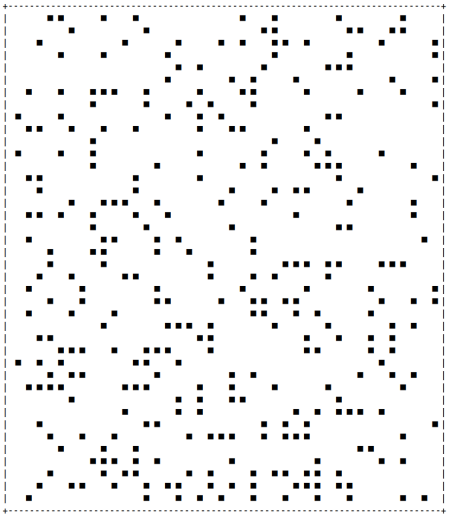

FIGURE 2 – Exemple de vue en console de l'automate.

Écrivez une classe de lancement pour tester la simulation en console.

2.3 Contrôleur

Implémentez le contrôleur qui doit gérer les accès à l'état de l'automate (modèle), et le lancement de la simulation. Le contrôleur doit aussi permettre d'interrompre la simulation. Pensez à utiliser un *thread* pour

lancer la simulation : cela permettra à l'application de traiter en même temps la simulation et les interactions avec l'utilisateur (taper `<Entrée>` au clavier pour le démarrage et l'arrêt de la simulation). **NB** : pour lancer un *thread*, créez une classe héritant de la classe `Thread`, et dont la méthode `run()` définit l'action réalisée dans ce thread, et lancez le en appelant sa méthode start(). Exemple :

```
public void startSimulation(){
  Thread thread = new Thread() {
    @Override
    public void run() {
      // define here the actions to be executed
    }
  }
  thread.start();
}
```

2.4 Vue (GUI)

Implémentez la vue graphique (Swing) qui permettra l'affichage de la grille. Cette interface doit comporter des éléments d'interactions permettant de démarrer ou arrêter la simulation.

3 Classe de Lancement

Proposez une classe de lancement de l'application. L'application au lancement doit permettre une visualisation à la fois en console et GUI.

4 Travail à Rendre

Vous devrez fournir un fichier `jar` exécutable contenant également tous les fichiers sources de votre programme (bien documentés), ainsi qu'un fichier `png` représentant votre diagramme de classes final.