Projet APO

Automates cellulaires

Étudiants :

Noa PETEL
Albin MARTIN
Hugo COURTOIS
Victor LEFEVRE

Tuteur :

Mathieu LEFORT

Semestre 5

Sommaire

I. Introduction

Le but de ce projet est d'implémenter différents automates cellulaires. Un automate cellulaire est un modèle informatique et mathématique qui permet de simuler le comportement dynamique de systèmes physiques. Pour construire un automate plusieurs données en entrées sont nécessaires tel que la dimension de l'automate (1D, 2D , 3D ...), les différents états possibles pour les cellules de l'automate, le nombre et la position des cellules voisines et finalement les règles de transition d'état.

Dans ce projet nous allons créer des automates : un automate général que l'on peut définir à notre guise. Et des automates plus spécifiques tels que le jeu de la vie, un automate 1D, un automate utilisant la règle de la majorité et un automate simulant un feu de forêt.

Puis finalement, nous allons réaliser une interface graphique afin de faire une solution plus friendly-users.

II. Diagrammes

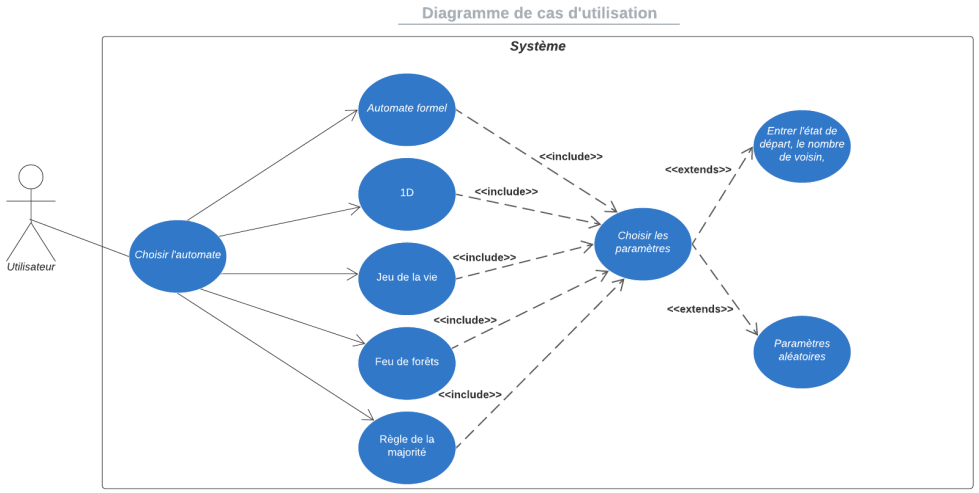


Illustration 1 : Diagramme de cas d'utilisation.

L'utilisateur peut choisir l'automate qu'il souhaite : Automate formel, l'automate 1D, le jeu de la vie, le feu de forêt ou bien la règle de majorité. L'utilisateur devra obligatoirement définir des paramètres ce qui sera fait soit automatiquement ou bien manuellement.

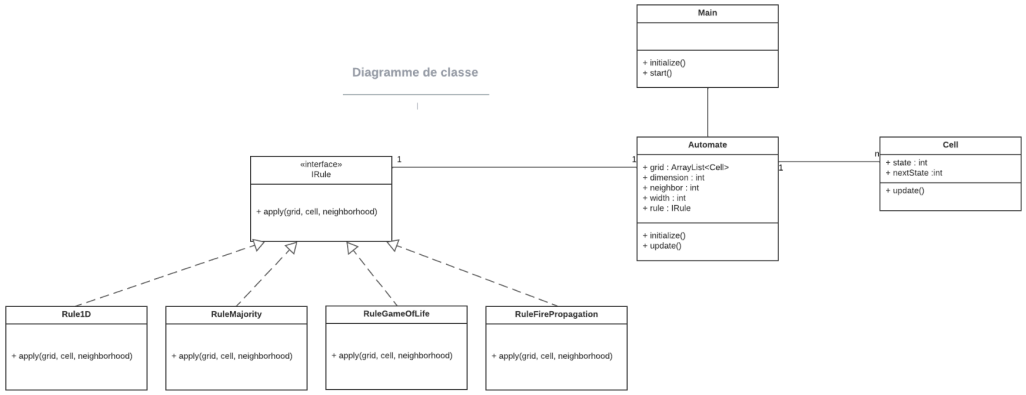


Illustration 2 : Diagramme de classes hypothétique.

Initialement nous avions prévu de faire un projet dont le diagramme de classes ressemble à ça. Malheureusement, nous nous sommes rendu compte que cela aller être plus complexe que prévu alors nous n'avons pas fait une classe mère qui gère tous les automates mais une chacune des classes fille indépendamment.

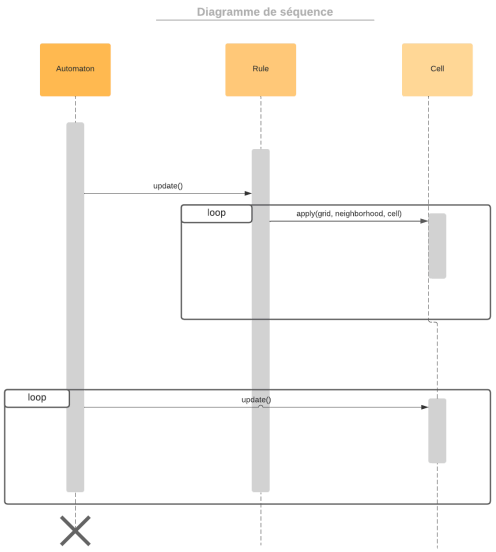


Illustration 3 : Diagramme de séquence hypothétique.

Nous avons réalisé un diagramme de séquence pour détailler le déroulement d'un tick pour l'automate. Dans l'idée, l'automate appellerait la règle actuelle pour qu'elle s'applique à chacune des cellules. Ensuite, l'automate se charge de faire changer l'état des cellules.

III. Définition formelle

L'automate formel est un automate que l'utilisateur peut paramétrer comme il le souhaite. Il a la possibilité de choisir :

- La dimension d de l'automate : ici pour des raisons d'affichage la dimension est limitée à 2.
- Le nombre d'états Q possibles pour chaque cellule
- Le nombre et les coordonnées V des voisin
- Les règles de transitions locales delta.

En partant de cette définition formelle d'un automate cellulaire, il est possible de générer tous les autres automates de dimension 1 ou 2.

Les problèmes que j'ai rencontré étaient surtout liés au typage de la variable etatCourant : en fonction de si d = 1 ou 2, etatCourant possède un type différent où on a en premier lieu ArrayList<Integer>, puis ArrayList<List<Integer>>, ce qui aurait été encore plus compliqué pour d quelconque, d'autant plus pour une méthode qui serait capable d'appliquer les transitions entre les états pour d quelconque. Malheureusement, nous n'avons pas implémenté l'affichage de cet automate.

IV. Automate 1D

Chaque cellule peut se trouver dans l'un des deux états possibles : "vivante" (représentée par le chiffre 1) ou "morte" (chiffre 0). La grille évolue à travers des générations successives, où l'état de chaque cellule dans la nouvelle génération est déterminé par son état actuel et l'état de ses deux voisines.
La règle d'évolution est représentée par un nombre entre 0 et 255, qui dicte comment les cellules évoluent d'une génération à l'autre. Cette règle est décodée en un tableau de 8 bits, où chaque bit représente l'un des 8 états possibles des trois cellules consécutives.
Il suffit d'appliquer la règle pour chacune des cellules et observer leur évolution, en prêtant attention aux effets de bord pour ne pas accéder à des voisins hors de la liste.

On remarque qu'avec la règle 90, on obtient un fractale :

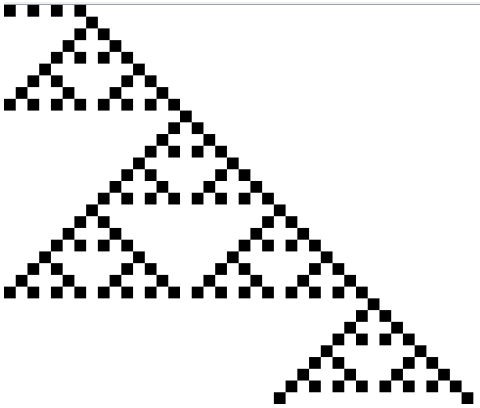


Illustration 4 : Fractale de la règle 90.

V. Règle de majorité

La règle de majorité est une règle s'appliquant sur le même type d'automate que précédemment à la différence que la règle ici compte le nombre de voisins vivants: si la majorité des voisins l'est, la cellule le devient.
La difficulté ici étant de faire un code modulable pour que l'automate soit valide dans toutes les dimensions. Dans notre cas, nous traitons seulement les automates en dimension 1 ou 2.

VI. Jeu de la vie

Le jeu de la vie est un automate en deux dimensions. Les cellules ont deux états possibles : 0 ou 1 (blanc ou noir). Il y a 8 voisins qui sont situés au plus proche de la cellule. Les règles de transition sont les suivantes :

- La cellule prend la valeur 0 si :
  - Il y a moins de deux voisins à l'état 1 (sous population).
  - Il y a plus de trois voisins à l'état 1 (sur population).
  - Il y a deux voisins à l'état 1 et que la cellule est à l'état 0.

- La cellule prend la valeur 1 si :
  - Il y a trois voisins à l'état 1 (naissance).
  - Il y a deux voisins à l'état 1 et que la cellule est à l'état 1.

VII. Feu de forêt

Le feu de forêt est un automate en deux dimensions. Les cellules ont quatre états possibles : vide, forêt, feu et brûlé. Il y a 8 voisins qui sont situés au plus proche de la cellule. Les règles de transition sont les suivantes :

- La cellule prend la valeur vide si son état est vide.
- La cellule prend la valeur forêt si son état est forêt et qu'il n'y a pas de voisin en feu
- La cellule prend la valeur en feu avec une probabilité de ⅛ multiplié par le nombre de voisin en feu.
- La cellule prend la valeur brûlée si son état est en feu ou bien brûlé.

De plus, nous avons ajouté l'effet du vent sur la propagation du feu, c'est-à-dire que si une cellule se trouve proche d'un feu et que le vent est dans sa direction alors la cellule aura une probabilité de ⅛ multiplié par son nombre de voisin + une probabilité qui correspond à la force du vent.

A l'inverse, si la cellule se trouve à l'opposé du vent, la probabilité pour qu'elle prenne feu est de ⅛ multiplié par le nombre de voisin moins la force du vent.

Une difficulté majeure de cet automate a été d'implémenter l'effet du vent. Pour ce faire, nous avons réalisé une méthode CellsInFire() qui récupère les coordonnées des cellules en feu pour chaque nouvelle grille. Puis dans la fonction evolve() on fait un switch suivant la direction du vent, puis on vérifie si la cellule actuelle est à côté d'une cellule en feu, on vérifie également si la cellule est dans la direction du vent ou bien à l'opposé.

VIII. Interface Graphique

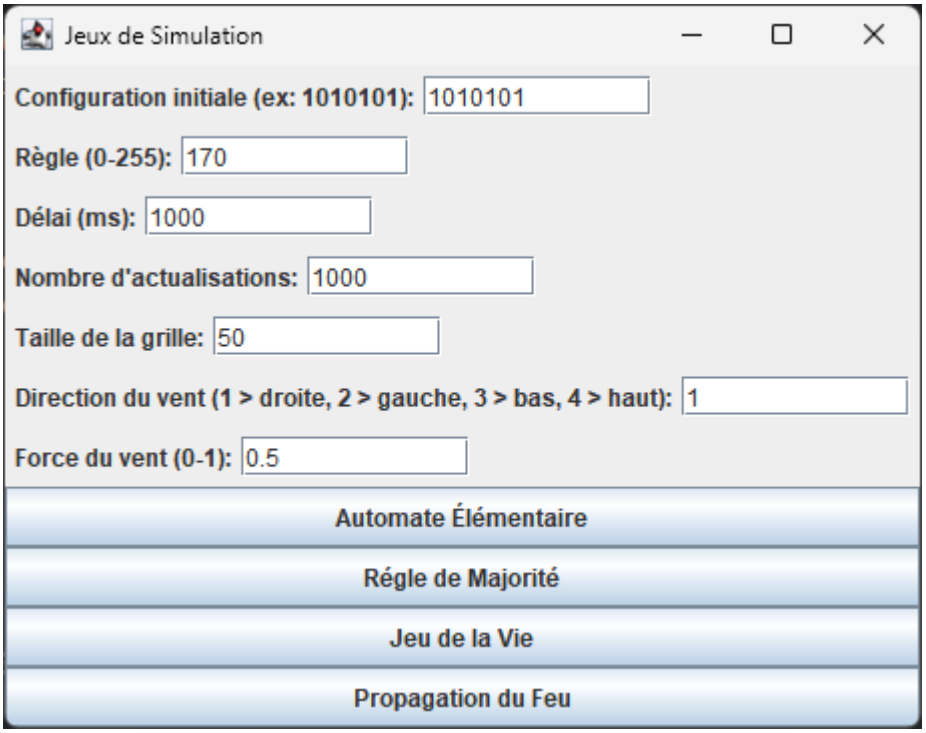


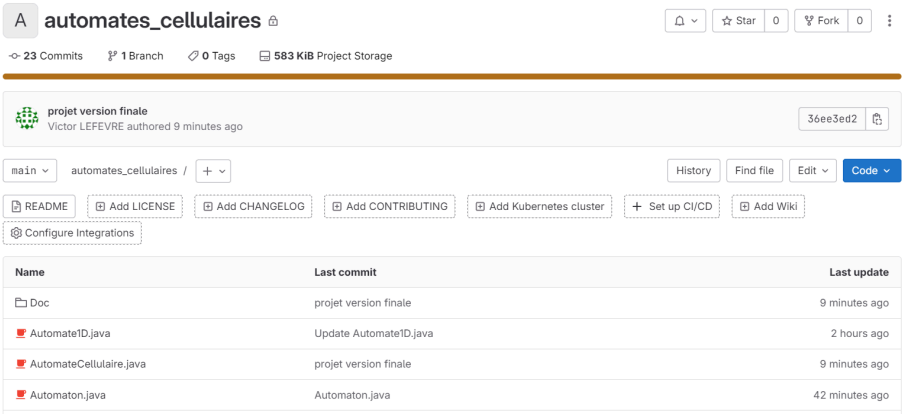
Illustration 5 : Interface graphique.

La classe crée une interface avec plusieurs champs de saisie (JTextField) où l'utilisateur peut entrer des configurations pour les simulations, telles que la taille de la grille, la règle de l'automate, le délai entre les itérations, et d'autres paramètres spécifiques à chaque simulation.
Il y a des boutons en bas de la fenêtre liés aux méthodes (actionListener) afin de démarrer une simulation.
L'affichage graphique est réalisé par javax.swing, d'autres packages sont importés pour gérer la fenêtre et les boutons.

IX. Extensions

Nous avons réalisé deux extensions :

-Réalisation d'une interface graphique.
-Utilisation de GitLab pour le partage de code.



X. Conclusion

En conclusion, ce projet nous a permis de nous familiariser complètement avec la programmation orientée objet. De plus, nous avons appris à créer une interface graphique, chose que l'on n'avait jamais vu auparavent.

Table des illustrations