Projet APO
Automates cellulaires

Mathieu Lefort

4 décembre 2023

1 Modalités pratiques

Ce projet est à faire par groupe de 3 ou 4 et est à rendre pour le 4 février 2024.

2 Évaluation

Vous serez évalués sur les points suivants :
— Méthodologie (1 point)
— Conception (8 points)
— Code (8 points)
— Documentation (3 points)
— Extensions optionnelles (? points suivant vos choix)

2.1 Méthodologie

Vous expliquerez votre méthodologie de travail (articulation entre conception et codage, déroulé temporel du projet, ...) et la répartition des tâches entre les différents membres du groupe.

2.2 Conception

Vous utiliserez tous les différents types de diagrammes vus en TDs à bon escient (i.e. permettant à quelqu'un d'extérieur au projet de le comprendre et de pouvoir le coder). Pour rappel, pour le diagramme de cas d'utilisations, chaque cas d'utilisation doit être décrit par un texte.
Votre conception devra être modulaire et donc permettre une intégration sans trop d'efforts des différentes extensions possibles. Il ne vous est pas demandé de modéliser les extensions non faites, mais d'intégrer leur éventuelle insertion dans votre réflexion lors de votre conception.
De plus, les différents choix (majeurs) de votre conception devront être justifiés.

2.3 Code

Le code devra être clair (indentation, noms de variables pertinentes, ...), utiliser la généricité et être autant que possible efficace. Il devra également et surtout être en accord avec la conception. Penser à bien vérifier et traiter (avec des exceptions) les cas d'erreurs dans vos méthodes.
Pensez à utiliser au maximum les packages pour structurer vos fichiers/classes.

2.4 Documentation

Vous générerez la javadoc de votre code (documentation des classes, méthodes, paramètres, ...). Vous penserez également à commenter votre code là où vous le jugez utile.

2.5 Extensions

Plusieurs extensions optionnelles vous sont proposées. À vous de voir celles qui peuvent vous intéresser. Il est tout à fait possible également de proposer (et d'implémenter) vos propres extensions. Si vous décidez de faire des extensions, votre note maximale potentielle peut dépasser 20. L'idée des extensions n'est pas vraiment de "rattraper" des points mais plutôt une opportunité de pousser l'exploration de Java plus loin si vous le souhaitez.

3 Sujet

L'objectif du projet est d'implémenter et d'étudier certains automates cellulaires [1].

3.1 Définition formelle

Un automate cellulaire est un 4-uplet (d, Q, V, δ) où :
— d est la dimension de l'automate, son réseau est alors $\mathbb{Z}^d$, l'espace discret de dimension d (i.e. une grille infinie de cellules)
— Q, un ensemble fini, est son alphabet (i.e. les états possibles pour chaque cellule)
— $V \subseteq \mathbb{Z}^d$ est son voisinage (un sous-ensemble fini du réseau)
— $\delta : Q^a \to Q$ est sa règle locale de transition et $a = |V|$ définit l'arité de l'automate.

On appelle alors configuration l'attribution d'un état à chaque cellule du réseau (formellement un élément de $Q^{\mathbb{Z}^d}$). On définit également la fonction globale d'évolution de l'automate $F_\delta : Q^{\mathbb{Z}^d} \to Q^{\mathbb{Z}^d}$, qui agit sur les configurations et qui est définie par $F_\delta(c) = z \mapsto \delta\big(c(z + v_1), \ldots, c(z + v_a)\big)$ pour toute configuration $c \in Q^{\mathbb{Z}^d}$ avec $V = \{v_1, \ldots, v_a\}$.

Un automate cellulaire est donc un modèle à cheval entre les mathématiques et l'informatique (théorique). Il possède également des liens avec la biologie (une des propriétés historiquement recherchée étant celle de l'auto-réplication) et la physique (la méthode des éléments finis [2] permettant de simuler le comportement dynamique de systèmes physiques, par exemple la propagation de la chaleur, consiste aussi à mettre à jour les valeur d'une grille d'éléments à partir de celles de ces voisins, mais en respectant des équations et des contraintes physiques).

En ce qui concerne ce projet, l'implémentation pratique d'un automate cellulaire nécessite de définir chacun des éléments du 4-uplet (plusieurs (classes) d'exemples sont détaillés par la suite). De plus, le réseau $\mathbb{Z}^d$ ne peut être implémenté puisque de taille infinie. Une taille finie doit donc être donnée à la grille, qui pourra éventuellement être torique (i.e. le voisin de droite/gauche/haut/bas d'une cellule tout à droite/gauche/haut/bas de la grille est une cellule tout à gauche/droite/bas/haut de la grille) afin d'obtenir une grille cyclique infinie. Enfin il faudra définir pendant combien d'étapes la fonction globale d'évolution est appliquée. Classiquement un nombre maximal est fixé à l'avance et/ou un indicateur d'arrêt (par exemple la stabilité de la configuration) est défini.

3.2 1D

Dans ce cas $d = 1$ et l'automate est donc une ligne (infinie en théorie) de cellules. Une des classes d'automate 1D la plus étudiée est celle où $Q = \{0, 1\}$ et V correspond à la cellule et à ses voisines de gauche et de droite. Il y a donc 2^3 configurations de voisinages possibles. Une règle locale peut alors se représenter de la manière suivante :

Configuration du voisinage	111	110	101	100	011	010	001	000
Nouvelle valeur	1	0	1	0	1	0	1	0

La règle précédente est celle où chaque cellule prend à la prochaine étape la valeur de sa voisine de droite. On va également donner un "nom" (un numéro) à cette règle en lisant la ligne Nouvelle valeur du tableau comme un entier (codé sur un octet). La règle précédente est donc la règle 170 (128+32+8+2). On peut ainsi définir 256 règles différentes [3]. Certaines ont des comportements remarquables, par exemple la règle 30 génère des motifs qui se retrouvent sur des coquillages ou ont été utilisés en architecture [4].

3.3 Règle de majorité (optionnel si groupe de 3)

Certains automates sont dit totalistiques, c'est-à-dire que leur règle de transition (plus simple) ne considère pas le motif précis du voisinage, mais le nombre (total) de voisins dans les différents états possibles. C'est le cas pour la règle de majorité où la dimension de l'automate peut être quelconque et où on reste avec $Q = \{0, 1\}$ (même si certains modèles existent aussi avec plus d'états). Le principe est que le prochain état d'une cellule sera celui de la majorité des cellules de son voisinage (qui doit donc avoir une cardinalité impaire). Ce type d'automate permet par exemple d'obtenir des motifs de type tâches/rayures quand il est utilisé avec $d = 2$ [5].

3.4 Jeu de la vie (optionnel si groupe de 3)

Le jeu de la vie est un des automates les plus célèbres [6]. Dans ce cas $d = 2$, $Q = \{0, 1\}$ et V comprend les 8 cases autour de la cellule + la cellule. La fonction locale de transition est la suivante (le terme voisins ci-après ne comprend pas la cellule) :

1. https://fr.wikipedia.org/wiki/Automate_cellulaire
2. https://fr.wikipedia.org/wiki/Méthode_des_éléments_finis
3. https://www.wolframscience.com/nks/p55–more-cellular-automata/
4. https://fr.wikipedia.org/wiki/Règle_30
5. Vous pouvez trouver un petit exemple ici : https://interstices.info/a-la-decouverte-des-automates-cellulaires/
6. https://fr.wikipedia.org/wiki/Jeu_de_la_vie

— le prochain état est 0 si il y a strictement moins de 2 voisins à l'état 1 ("sous population") ou si il y a strictement plus de 3 voisins à l'état 1 ("sur population") ou si il y a exactement 2 voisins à l'état 1 et que la cellule est dans l'état 0 ("conservation")
— le prochain état est 1 si il y a exactement 3 voisins à l'état 1 ("naissance") ou si il y a exactement 2 voisins à l'état 1 et que la cellule est dans l'état 1 ("conservation")

Malgré sa simplicité cet automate possède des propriétés théoriques intéressantes comme sa complétude, i.e. sa capacité à simuler n'importe quel autre automate (avec $d = 2$) [7]. Cela permet donc de simuler une machine de Turing [8] et donc globalement de faire n'importe quel calcul.

3.5 Feu de forêt

On va vouloir modéliser (très simplement) la propagation d'un feu de forêt [9]. Dans ce cas $d = 2$, $Q = \{vide, forêt, en\ feu, brûlé\}$ et le voisinage va être les cellules les plus proches. La fonction locale de transition est la suivante :

— le prochain état est vide si l'état de la cellule est vide
— le prochain état est forêt si l'état de la cellule est forêt et que l'état d'aucun des voisins est en feu
— le prochain état est en feu si l'état de la cellule est forêt et que l'état de l'un des voisins est en feu
— le prochain état est brûlé si l'état de la cellule est en feu

Il faudra a minima positionner une ou plusieurs cellules à l'état en feu initialement pour que cela ait un intérêt.

On peut également rajouter un aspect probabiliste à la fonction de transition :

— le prochain état est vide si l'état de la cellule est vide
— le prochain état est forêt si l'état de la cellule est forêt et que l'état d'aucun des voisins est en feu
— le prochain état est en feu avec une probabilité $k * p$ si l'état de la cellule est forêt et que l'état de k de ses voisins est en feu, sinon le prochain état est forêt avec une probabilité $1 - k * p$
— le prochain état est brûlé si l'état de la cellule est en feu

Là aussi il faudra a minima positionner une ou plusieurs cellules à l'état en feu initialement et/ou dire qu'une cellule forêt peut passer en feu avec une probabilité q même si aucune cellule voisine n'est en feu (i.e. avec une probabilité $q + k * p$ avec k le nombre de voisins en feu).

On peut également rajouter du vent en modifiant la probabilité de la propagation du feu [10]. Ainsi une cellule de forêt qui avait une probabilité p de passer en feu si sa voisine de droite était en feu, aura maintenant une probabilité de $p + f_d$ avec f_d la "force" du vent venant de la droite. La même formule est à appliquer pour toutes les directions possibles du vent avec évidemment $f_d = -f_g$ (et de même pour toutes les autres combinaisons de directions opposées).

4 Tronc commun

Vous devez proposer une conception permettant d'implémenter n'importe quel type d'automate cellulaire. En particulier votre programme permettra :

— de choisir entre les différents automates implémentés, mais également de définir un nouvel automate, de choisir la configuration de départ et d'afficher la configuration finale
— la simulation des automates 1D en donnant directement le numéro de la règle et en affichant l'évolution temporelle de l'automate (affichage 2D avec une dimension pour l'automate et une dimension pour le temps)
— (optionnel si groupe de 3) la simulation des automates de règles par majorité, en particulier ceux avec des voisinages carrés, rectangles ou hexagonales centrés sur la cellule à mettre à jour, permettant l'étude de l'influence de la forme et de la taille du voisinage sur le résultat obtenu
— (optionnel si groupe de 3) le jeu de la vie, avec des configurations initiales aléatoires, définissables "à la main" ou prédéfinies [11]
— la simulation des feux de forêt en permettant d'étudier le temps de propagation et le pourcentage de forêt brûlée suivant différents paramètres (densité initiale de la forêt, force et direction du vent, grille carrée avec 4 ou 8 voisins ou grille hexagonale avec 6 voisins)

L'ensemble de ces fonctionnalités sera accessible via un menu textuel.

7. Plus d'informations ici : https://www.techno-science.net/glossaire-definition/Automate-cellulaire-page-2.html
8. Vous pouvez voir à quoi cela ressemble ici http://rendell-attic.org/gol/tm.htm
9. Un exemple de modélisation plus réaliste (forcément plus compliquée) à partir de la page 153 : http://docnum.univ-lorraine.fr/public/INPL_T_1998_MARGERIT_J.pdf
10. Vous pouvez voir un exemple ici https://ccl.northwestern.edu/netlogo/models/FireSimpleExtension2
11. Vous pouvez trouver des structures intéressantes à regarder ici : https://fr.wikipedia.org/wiki/Jeu_de_la_vie

5 Extensions possibles

5.1 Environnement de travail (0.5 point)

Vous utiliserez un git (dont l'adresse sera à fournir dans le rapport) pour faire le suivi de versions tout au long de votre projet.

5.2 Installation (0.5 point)

Vous fournirez un fichier build.xml (regardez `https://ant.apache.org/`) permettant de compiler votre code (à partir d'un dossier src), de générer la javadoc (dans un dossier doc) et de supprimer l'installation (fichiers compilés et documentation). Si vous faites également une extension demandant l'utilisation d'un framework hors JRE, le fichier devra également s'occuper de l'installation du framework.

5.3 Tests unitaires (2 points)

Vous implémenterez des tests unitaires des méthodes (principales) de votre programme (regardez `https://junit.org/junit5/`).

5.4 Fichier de configuration (1 point)

Faites en sorte que tous les paramètres d'un automate cellulaire soient stockés dans un fichier de configuration (humainement lisible et éditable).

5.5 Sauvegarde (1.5 point)

Sauvegardez dans un fichier (humainement lisible et éditable) la configuration d'un automate cellulaire, ce qui permettra de l'utiliser comme configuration initiale pour une prochaine simulation. Pour les feux de forêt, sauvegarder dans un fichier (type Excel) les résultats des simulations afin de pouvoir faire des graphiques d'analyse a posteriori.

5.6 Interface graphique (4 points)

En plus d'une interface textuelle, vous proposerez une interface graphique pour la visualisation des résultats. Vous respecterez au mieux le *design pattern* MVC (Modèle Vue Contrôleur), à savoir a minima avoir les classes d'affichages séparées de celles du modèle. Cet affichage graphique peut en particulier être efficace pour observer le jeu de la vie ou la propagation du feu de forêt.

5.7 Analyse des automates (2 points)

Générez de manière automatique les résultats et l'affichage de certains indicateurs d'un automate en fonction de certains de ses paramètres (par exemple générer automatique la courbe de pourcentage de forêt brûlée en fonction de la densité et/ou en fonction de la topologie, et/ou de la force du vent, ...). Regardez par exemple la librairie d'affichage *JFreeChart*.