Université Paris Cité

Compléments en Programmation Orientée Objet
Projet de session 2 : Jeu de la Vie et automates cellulaires

Projet à réaliser seul, à rendre sur Moodle avant le 24 juin.

1 Avant-propos

Le projet consiste en l'implémentation du Jeu de la Vie, l'extension à des automates cellulaires variés et la parallélisation des traitements.
À part pour la parallélisation, nous soulignons que les fonctionnalités de ce projet sont tout à fait réalisables avec les connaissances de L2. C'est pourquoi nous insistons ici sur la conception objet (abstractions objet pertinentes et patrons de conception). Il ne suffira pas de rendre un programme qui fonctionne pour avoir une bonne note ! Pour autant, il sera possible d'avoir une note correcte sans avoir fait 100 % de ce qui est demandé.

2 Instructions de rendu

- Vous travaillerez sur un dépôt git privé hébergé sur le serveur `https://gaufre.informatique.univ-paris-diderot.fr` et veillerez à faire des commits réguliers.
- Vous donnerez l'accès à votre dépôt (notamment à son historique, dans le doute choisissez *maintainer*) aux enseignants du module : @adegorre, @bigeon, @boutglay et @zielonka.
- Le projet sera aussi rendu sur Moodle [1] (redondance) dans une archive ZIP.
- Le point d'entrée de votre documentation sera le fichier `README.md` situé à la racine de votre dépôt dans la branche par défaut.
- Pensez à vous inscrire pour la soutenance (salles réservées pour les 26 et 27 juin) via Moodle [2].
- Les soutenances ayant lieu à la Halle aux Farines, faites en sorte de pouvoir exécuter votre projet là-bas (PC portable ou bien PC fixe de la salle de TP).

3 Le Jeu de la vie

Le "Jeu" de la Vie est un jeu de simulation bien connu, inventé par John Horton Conway en 1970, dont on peut trouver la description ici `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`.
En résumé, le "jeu" consiste en une grille de cellules soit mortes soit vivantes, qui sera mise à jour étape par étape. Pour passer à l'étape $n+1$, chaque cellule compte ses voisines vivantes à l'étape n :

- Si la cellule est morte et qu'elle a 3 voisines vivantes, elle devient vivante.
- Si la cellule est vivante et qu'elle a strictement moins de 2, ou strictement plus de 3 voisines vivantes, elle meurt.

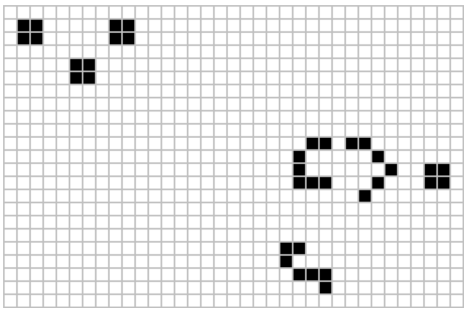

FIGURE 1 – Jeu de la vie – canon à vaisseaux (Wikimedia Commons)

Cette règle très simple permet de simuler des comportements très complexes et difficiles à prévoir (visualiser les nombreuses animations disponibles sur le web).

1. `https://moodle.u-paris.fr/mod/assign/view.php?id=838207`
2. `https://moodle.u-paris.fr/mod/scheduler/view.php?id=838213`

Notez que ce n'est pas un jeu au sens où on l'entend habituellement, dans le sens où il n'y a pas d'interaction avec un joueur et que le déroulé de la simulation ne dépend que de la situation initiale. Pour ce projet, on vous demande un affichage étape par étape de la simulation (1 étape = 1 mise à jour complète de la grille).
Nous ne vous demandons pas de coder un éditeur de configuration. Le plus simple c'est de permettre d'éditer les grilles dans un éditeur de texte et de charger le fichier depuis le jeu. Pour ce faire, suivez le format indiqué ci-dessous (pour une grille de 10 lignes de 5 colonnes) :

```
10
5
0011001110
1001100111
0110011101
0000001010
0011001110
```

Les « 1 » représentent bien sûr la présence d'une cellule vivante, alors que les « 0 » sont des cellules mortes.
Plusieurs exemples de tels fichiers pourront être téléchargés sur Moodle [3]. Vous êtes encouragés à les exécuter et tester les simulations obtenues avec attention.

4 Automates cellulaires

D'après Wikipedia [4] :

> Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est fonction de l'état au temps t d'un nombre fini de cellules appelé son « voisinage ». À chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle « génération » de cellules dépendant entièrement de la génération précédente.

Le Jeu de la Vie est donc un automate cellulaire particulier (à 2 états, sur une grille en dimension 2, avec une règle de mise-à-jour particulière qui dépend seulement de l'état de la cellule et du décompte des voisins vivants, où le voisinage correspond aux 8 autres cellules les plus proches en distance euclidienne).
On peut formaliser un automate cellulaire comme la donnée

- d'un espace d'états pour une cellule,
- d'une « grille » [5] munie d'une notion de voisinage (associant à toute cellule la « liste » [6] de ses voisines)
- ainsi que d'une fonction de mise-à-jour (donnant le prochain état d'une cellule en fonction de son état actuel et de celui du voisinage).

Pour pouvoir exécuter un tel automate, il faut aussi fournir une configuration initiale.
Remarquez que le voisinage peut, en toute généralité, prendre des formes très diverses (ce ne sont pas nécessairement les 8 cellules les plus proches dans une grille 2D en géométrie euclidienne).

3. `https://moodle.u-paris.fr/mod/folder/view.php?id=838225`
4. `https://fr.wikipedia.org/wiki/Automate_cellulaire`
5. À interpréter au sens large : c'est juste une collection de cellules qu'on sait itérer d'une certaine façon. C'est au final surtout la notion de voisinage et à la représentation « graphique » (méthode `toString`) qui fournissent une interprétation géométrique, en tant que grille 2D ou autre.
6. C'est peut-être une map en réalité ! Par exemple une map de { nord, est, sud, ouest} vers des cellules ou leur état.

Votre travail consiste essentiellement à extraire de ce texte les notions (types [7]) pertinentes en termes de programmation (pensez aussi à garder une efficacité algorithmique raisonnable) et d'écrire le programme de simulation basé sur ces types. La configuration initiale sera lue depuis un fichier, mais pour les automates autres que le Jeu de la Vie, le format du fichier est laissé à votre discrétion. Assurez-vous que vous pouvez instancier (et simuler) le Jeu de la Vie en utilisant ce programme, ainsi que plusieurs exemples cités sur la page Wikipédia (prenez des exemples en dimension 1 et en dimension 2, signalez lesquels dans votre `README.md`).

5 Parallélisation

Comme les calculs effectués lors d'une étape de la simulation sont essentiellement le même calcul répété sur de nombreux voisinages différents et ne dépendent que des résultats obtenus à l'étape précédente, ceux-ci sont un candidat idéal à la parallélisation sur plusieurs *threads*.
Ainsi, le projet rendu devra répartir les calculs de simulation sur autant de *threads* qu'il est raisonnable de faire.
Pour paralléliser les traitements, on pourra découper le calcul d'une étape en zones géométriques [8] chacune traitée dans une tâche différente, confiée à un *thread poool*.

6 Quelques indications

- Il est conseillé de commencer par implémenter le Jeu de la Vie seul (sans arrière pensée), puis peut-être une variante ou deux, avant de réfléchir à la conception du simulateur d'automates cellulaires le plus général, quitte à ne rien garder du code initial.
- L'interface utilisateur (que soit dans la console ou en mode graphique) consiste simplement (peut-être après un menu) à afficher les configurations successives, en demandant à l'utilisateur d'appuyer sur une touche pour afficher la configuration d'après. N'affichez surtout pas les éventuels états intermédiaires (mises-à-jour partielles) qui peuvent avoir été obtenus au cours du calcul d'une étape.
- Une suggestion de patron de conception utilisable : un builder pour configurer une instance d'automate cellulaire avant de la simuler.
- Autre suggestion : une abstract factory pour initialiser une grille avec des nouvelles cellules.

(ces deux dernières suggestions peuvent ou non s'appliquer à votre projet, selon vos choix de conception...)

7 Critères d'évaluation

- Le projet est programmé en Java (si version différente de 17, précisez le dans votre `README.md`).
- Il s'agit d'un seul programme incluant toutes les fonctionnalités que vous avez réussi à implémenter (si l'on exclut les programmes de test). Parmi ces fonctionnalités, il devra être possible de simuler plusieurs automates cellulaires différents, dont le Jeu de la Vie et au moins un automate à grille uni-dimensionnelle.
- Le fichier `README.md` indique comment compiler, exécuter et utiliser votre programme et ses tests.
- Le code respecte les conventions de codage usuelles.
- La compilation selon vos instructions doit terminer sans erreur ni avertissement.

7. Pensez à toutes les briques de construction dont vous disposez : interfaces, héritage, composition, généricité, énumérations, enregistrements, lambdas ...

8. Peut-être que cela nécessitera d'ajouter des méthodes à la grille afin de permettre d'itérer sur une partie seulement de celle-ci. Suggestion : une méthode retournant une liste d'itérateurs de cellules.

- Le fichier `README.md` décrit les fonctionnalités effectivement implémentées et explique vos choix techniques (modélisation, patrons de conception, ...).
- Un diagramme de classe rendra la modélisation plus claire !
- Le code est architecturé intelligemment en favorisant la réutilisation de code (autant que possible, pas de code spécifique à un seul automate cellulaire). Notamment, il faut utiliser des patrons de conception vus en cours ainsi que les constructions les plus appropriées fournies par Java.
- L'exécution doit être correcte : elle respecte le cahier des charges et ne quitte pas de façon non contrôlée.
- Les classes et méthodes sont documentées (javadoc).
- Des commentaires expliquent les portions de codes qui ne sont pas évidentes.
- La concurrence doit être utile (réduire le temps de calcul) et sûre (être correctement synchronisée).
- Les tests fournis doivent être aussi exhaustifs que possible.

L'interface graphique n'est pas spécifiquement demandée.