Le jeu de la vie

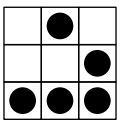
Sujet proposé par Jean-Christophe Filliâtre
mailto:Jean-Christophe.Filliatre@lri.fr

Difficulté : difficile ($\star\star\star$)

Version 1 (6 février 2012)

URL de suivi : `http://www.enseignement.polytechnique.fr/profs/informatique/Jean-Christophe.Filliatre/11-12/INF431/gol/`

Le but de ce projet est de réaliser un simulateur pour le jeu de la vie.



1 Le jeu de la vie

Le jeu de la vie a été inventé en 1970 par le mathématicien John Conway. Il s'agit d'un automate cellulaire à deux états sur une grille infinie. Les éléments de la grille sont appelées des *cellules* et chaque cellule possède deux états : *allumée* ou *éteinte*. Initialement, un nombre fini de cellules sont allumées. Toutes les cellules changent simultanément d'état, selon un temps discret. Le nouvel état d'une cellule est déterminé par l'unique règle suivante :

> Si deux de ses huit voisines sont allumées, elle ne change pas d'état. Si trois sont allumées, elle s'allume. Sinon, elle s'éteint.

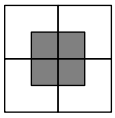
Bien que d'une grande simplicité, cette règle définit un automate cellulaire d'une incroyable complexité, qui ne cesse d'émerveiller et de poser des questions difficiles (par exemple « quelle est la plus petite configuration sans prédécesseur ? »).

L'objectif de ce projet est de simuler le jeu de la vie à grande échelle, c'est-à-dire sur de grandes grilles et de grandes échelles de temps, en utilisant l'algorithme *Hashlife* dû à Bill Gosper [1].

2 L'algorithme *Hashlife*

Cet algorithme introduit la notion de *macro-cellule*. Une macro-cellule est une portion de la grille de taille $2^n \times 2^n$; on parle alors de macro-cellule de *dimension* n. L'algorithme *Hashlife* repose alors sur plusieurs idées clés :

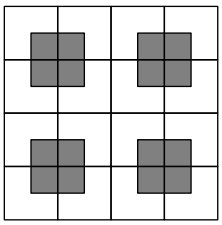
- Si on considère la région centrale de taille $2^{n-1} \times 2^{n-1}$ d'une macro-cellule de dimension n, alors son état dans 2^{n-2} pas de temps peut être calculé indépendamment de tout contexte. En effet, aucune information ne peut parvenir de l'extérieur de la macro-cellule dans ce laps de temps.



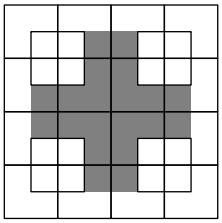
- Une macro-cellule de dimension n peut être représentée par *quatre* macro-cellules de dimension $n-1$, à savoir ses quatre quadrants nord-est, nord-ouest, sud-ouest et sud-est. De plus, sa région centrale dans 2^{n-2} pas de temps est également une macro-cellule de dimension $n-1$; on l'appelle son *résultat*. Ces cinq macro-cellules sont représentées ci-dessus.
- Une même macro-cellule peut être utilisée dans la représentation de plusieurs autres macro-cellules. En particulier, cela permet de réutiliser également son résultat. On peut même adopter la stratégie suivante de *partage maximal* : ne jamais construire deux fois la même macro-cellule et ne jamais calculer deux fois le résultat d'une même macro-cellule.

Il reste à expliquer comment on calcule le résultat d'une macro-cellule de dimension n. Il y a deux cas de figure.

- Si $n = 2$, alors le résultat est composé de quatre cellules seulement et on le calcule directement en appliquant la règle du jeu de la vie.
- Si $n > 2$, alors la macro-cellule est constituée de quatre macro-cellules de dimension $n-1$ dont on peut calculer le résultat récursivement. Cela correspond à la situation suivante, où ces quatre résultats, de dimension $n-2$, sont représentés en gris :

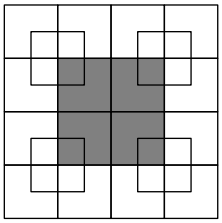


On peut alors calculer les cinq macro-cellules de dimension $n-2$ qui composent la croix encadrées par ces quatre résultats :



Pour cela, il suffit de construire les cinq macro-cellules de dimensions $n-1$ correspondantes et de calculer récursivement leur résultat. Enfin, on peut calculer les

quatre macro-cellules de dimension $n-2$ qui composent le résultat que l'on cherche à calculer, toujours récursivement :



Au total, on aura dû calculer 13 résultats récursivement pour parvenir à nos fins.

3 Travail demandé

Le but de ce projet est de réaliser un programme permettant :

1. de lire une configuration initiale dans un fichier au format RLE et de l'afficher ;
2. de calculer et d'afficher l'état de la grille après t pas de temps.

Cela peut être aussi simple qu'un programme qui propose une ligne de commande de la forme

```
life fichier.rle 10000
```

et ouvre deux fenêtres, l'une montrant la configuration initiale contenue dans le fichier (ici `fichier.rle`) et l'autre montrant la configuration après t pas de temps (ici 10 000).

Si le temps le permet, on pourra également également proposer un programme plus élaboré avec une interface graphique, permettant

1. une animation où la vitesse de déplacement dans le temps peut être modifiée dynamiquement, ainsi que la portion de la grille affichée ;
2. une édition de la grille, pour permettre une exploration interactive du jeu de la vie.

4 Indications

4.1 Représentation et construction des macro-cellules

Il y a deux types possibles de macro-cellules :

- les macro-cellules de dimension 1, qui se réduisent à l'état de quatre cellules c'est-à-dire à quatre booléens ;
- les macro-cellules de dimension $n > 1$, qui se composent de cinq macro-cellules de dimension $n-1$. L'invariant est le suivant : les quatre macro-cellules représentant les quatre quadrants sont quatre macro-cellules valides (*i.e.* non `null`) de même dimension $n-1$. En revanche, la cinquième macro-cellule, représentant le résultat, pourra prendre la valeur `null` tant qu'elle n'a pas été calculée (son calcul se faisant alors à la demande). On pourra également stocker dans la macro-cellule sa dimension n, pour éviter d'avoir à la recalculer.

Le partage maximal des macro-cellules est alors réalisé de la manière suivante. Une table de hachage (`HashMap`) contient toutes les macro-cellules déjà construites (chaque macro-cellule m y est associée à elle-même). Quand on cherche à construire une nouvelle macro-cellule de dimension n à partir de quatre macro-cellules m_0, m_1, m_2 et m_3 de dimension $n-1$, on procède ainsi :

1. on construit l'objet Java correspondant M avec le constructeur de la classe ;
2. on cherche dans la table de hachage s'il existe déjà un objet M' structurellement identique à M (en appliquant la méthode `find` à M) :
   (a) si oui, on renvoie M' (et M, construite inutilement, sera ramassée par le GC) ;
   (b) si non, on ajoute l'association $M \mapsto M$ dans la table et on renvoie M.

Il est très important que la fonction de hachage (`hashCode`) et la fonction d'égalité (`equals`) utilisées par cette table soient de coût $O(1)$. Elles doivent donc prendre en compte les quatre macro-cellules immédiatement inférieures, mais ne pas aller plus loin dans le calcul.

Note : cette technique s'appelle le *hash consing*.

4.2 Opérations sur les macro-cellules

Outre l'opération de construction décrite dans le paragraphe précédent, il pourra être intéressant de coder les opérations suivantes sur les macro-cellules (vous êtes cependant libres de procéder autrement) :

- calcul du résultat : si la macro-cellule M contient déjà son résultat, on le renvoie ; sinon, on le calcule comme expliqué dans la section 2, on le stocke dans M et on le renvoie.
- généralisation du résultat d'une macro-cellule M de dimension n à un laps de temps 2^s avec $s \leq n-2$. On pourra stocker ces résultats-là dans une table indexée par M et s, afin de les réutiliser. L'intérêt de cette généralisation est qu'il devient facile de se déplacer de t instants dans le temps : il suffit de décomposer t comme une somme de puissances de deux.
- simplification : renvoie une macro-cellule de dimension minimale contenant les mêmes cellules allumées et centrée de la même façon. On procède ainsi : si tout le pourtour de largeur 2^{n-2} d'une macro-cellule de dimension n est éteint, on peut considérer la macro-cellule de dimension $n-1$ qui compose son centre (attention : ce n'est pas la même chose que son résultat), et la simplifier récursivement.
- doublement : à l'inverse de la simplification, on peut doubler la taille d'une macro-cellule sans modifier l'ensemble des cellules allumées, en ajoutant un pourtour de cellules éteintes. Ceci est notamment utile pour garantir qu'aucune cellule ne « s'échappera » du calcul du résultat qu'on s'apprête à effectuer.
- conversion depuis et vers une matrice de booléens, pour faciliter l'affichage et la saisie.

4.3 Format de fichier RLE

Des fichiers de configurations sont fournis. Ils portent l'extension `.rle` et leur format est le suivant.

1. L'entête du fichier peut contenir zéro, une ou plusieurs lignes de commentaires. Une ligne de commentaire commence par le caractère `#` et s'étend jusqu'à la fin de la ligne.
2. Vient ensuite une ligne donnant les dimensions de la portion non vide de la grille, sous la forme suivante :

   `x =` n`, y =` m`, rule = B3/S23`

   où n est le nombre de colonnes et m le nombre de lignes. L'indication `rule = B3/S23` décrit la règle utilisée (lire « *born 3, stay 2/3* ») qui sera toujours ici la même, à savoir celle du jeu de la vie.
3. Suivent des caractères qui déterminent l'état des cellules, de gauche à droite, puis de haut en bas. Ces caractères sont les suivants :

   `o` une cellule allumée ;

   `b` une cellule éteinte ;

   `$` un changement de ligne.

   Chacun de ces caractères est éventuellement précédé d'un entier n, qui signifie alors qu'il doit être répété n fois. Ainsi `12o` représente douze cellules consécutives allumées, `3$` représentent trois changements de ligne, c'est-à-dire l'insertion deux lignes vides, etc. Un caractère retour-chariot est ignoré. Une cellule dont l'état n'est pas déterminé (parce qu'elle se trouve sur une fin de ligne non décrite) est éteinte.
4. Le caractère `!` indique la fin de la description.

Ainsi, le fameux *planeur* peut être décrit par le fichier suivant :

```
# the glider
x = 3, y = 3, rule = B3/S23
3o$2bo$bo!
```

Ce fichier est fourni sur la page de suivi du projet [2], sous le nom `glider.rle`.

Références

[1] R. Wm. Gosper. Exploiting Regularities in Large Cellular Spaces. *Physica D : Nonlinear Phenomena* 10(1–2) :75–80, 1984.

[2] Page de suivi du projet. `http://www.enseignement.polytechnique.fr/profs/informatique/Jean-Christophe.Filliatre/11-12/INF431/gol/`