UE Programmation Orientée Objet

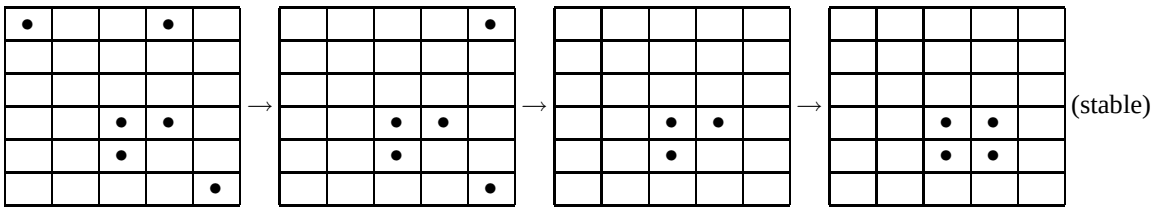
TP Jeu de la Vie

(d'après le sujet de DS de janvier 2002)

Le "jeu de la vie" consiste à faire évoluer un ensemble de cellules réparties sur une grille rectangulaire torique. À chaque cycle l'état de toutes les cellules est évalué **simultanément** et elles évoluent comme suit :

- Une cellule inactive s'active si elle est entourée d'exactement 3 cellules actives.
- Une cellule active ne survit que si elle est entourée de 2 ou 3 cellules actives.

Voici un exemple d'évolution sur 2 cycles (la situation d'arrivée est stable) :



Pour programmer ce problème, nous allons modéliser les cellules par des objets d'une classe `Cell`, la grille de cellules sera représentée par un objet de la classe `Environment` et enfin une classe `GameOfLife` qui gèrera l'évolution de l'environnement et son affichage. Ces classes appartiendront à un paquetage `jeudelavie`.

Pour cette réalisation, nous allons nous appuyer sur le paquetage `grid`. Vous trouverez les interfaces et classes de ce paquetage dans le fichier `jeu_de_la_vie.zip` sur le portail. Il est impératif de l'étudier avant de commencer. Ce paquetage est contenu dans l'archive `grid.jar`.
L'interface `grid.GridDisplayer` permet un affichage d'objet du type `grid.Grid`. Deux réalisations de cette interface sont fournies, l'une permet un affichage en mode texte (`grid.textGridDisplayer`) et l'autre en mode graphique (`grid.GraphicalGridDisplayer`).

Réfléchissez à l'ensemble de la modélisation avant de commencer à coder !

Les cellules Une cellule est définie par son état actif ou inactif. Afin de permettre l'affichage d'une cellule, en mode texte ou graphique, on décide qu'une cellule active sera caractérisée par le caractère `'*'` et la couleur `java.awt.Color.blue` et une cellule inactive par le caractère espace (`' '`) et la couleur blanche (utilisez des attributs de classe tels que `ACTIVE_COLOR`, `INACTIVE_COLOR`, `ACTIVE_CHAR`, `INACTIVE_CHAR`).

L'environnement L'environnement est une grille torique, afin de pouvoir facilement **réutiliser** les classes d'affichage en mode texte et graphique du paquetage `grid`, un objet qui permet de représenter l'environnement sera du type `grid.Grid`.

Un objet de cette classe sera défini par sa hauteur h, sa largeur l et un tableau de $h \times l$ instances de `Cell`. Il faut définir les méthodes qui permettent de modifier ou de récupérer la cellule à une position (cf `grid.Position`) donnée. Afin de gérer l'évolution de l'environnement, il faut pouvoir déterminer le nombre de cellules voisines actives d'une position donnée.

Le jeu de la vie Pour jouer au "jeu de la vie" il faut gérer l'évolution **simultanée** de toutes les cellules de l'environnement, et répéter cette évolution pendant un nombre donné de cycles. On veut un affichage du nouvel état de l'environnement après chaque étape de l'évolution (en fonction du jeu de la vie créé cet affichage pourra être graphique ou texte). Vous pouvez choisir de partir d'un état de l'environnement aléatoire (éventuellement avec un pourcentage de cellules actives à fixer) ou d'une configuration prédéfinie.

Un squelette de la classe `GameOfLife.java` est fournie sur le portail.

Test Vous pouvez avoir un aperçu de ce que doit donner le programme que vous devez écrire en utilisant l'archive `jeudelavie.jar` présente sur le portail.

Vous pouvez tester par :

```
java -jar jeudelavie.jar
```

ou en utilisant des motifs prédéfinis dans des fichiers (contenus dans l'archive et dans le répertoire `motifs`):

```
java -jar jeudelavie.jar /motif?.txt
```

avec `?` prenant une valeur de 1 à 4 (le "/" est nécessaire pour utiliser les fichiers fournies **dans** l'archive – voir plus loin).

La structure des fichiers `motif?.txt` est la suivante :

- on trouve d'abord la largeur et la hauteur de l'environnement
- puis sur les lignes suivantes les couples des coordonnées (abscisse puis ordonnée) des cellules initialement actives dans l'environnement. Les autres cellules (celles non citées) étant inactives.

Comme vous le constaterez il est possible de créer des motifs stables ou cycliques. Créez vos propres motifs...

Lecture dans un fichier Pour pouvoir avec votre programme créer une configuration initiale avec un fichier comme proposé dans l'archive fournie, vous pouvez vous inspirer du contenu du fichier `LectureFichier.java`, du paquetage `exemple`, pour définir la méthode `initFile` qui initialise votre environnement à partir des infos lus dans un fichier.

Utilisation de ressources placées dans un `jar`

Il est possible, c'est même souvent le cas, qu'une application utilise des ressources autres que des classes. Il peut s'agir de fichiers texte, d'images, etc. Dans un certain nombre de cas, ces ressources sont placées dans l'archive `jar` de l'application.
Il est cependant nécessaire d'anticiper dès l'écriture du code que l'on va pouvoir accéder à ces ressources situées dans le `jar`. Le fait que les ressources se situent dans le `jar` nécessite un chargement particulier de celles-ci. On doit passer par le "`ClassLoader`" utilisé[1].
On utilise les méthodes `getResource` qui fournit un objet `URL` et `getResourceAsStream` qui fournit un `InputStream` de la classe `Class`. Celles-ci utilisant les méthodes similaires sur le `ClassLoader` ayant permis le chargement de l'objet `Class` utilisé.
Voici donc ce qu'il faut écrire, pour accéder à une image (`javax.swing.ImageIcon`):

```
ImageIcon image = new ImageIcon(getClass().getResource("image.gif"));
```

et pour un `InputStream`, qui permet d'accéder à un fichier :

```
InputStream is = this.getClass().getResourceAsStream("/myfile.txt");
```

La manière dont sont formées les url construites par `getResource` est importante et peut être trouvée dans la javadoc.
Par exemple dans le cas précédent si ces lignes de codes se situent dans la méthode d'une classe `pack.souspack.Chose` (`Chose` sera donc l'objet "`this.getClass()`"), alors les ressources, images et fichiers, doivent se trouver dans `pack/souspack` pour la première et à la racine du `jar` pour la seconde. On doit donc avoir les fichiers `pack/souspack/image.gif` et `myfile.txt` dans le jar.

Il y a un exemple d'accès à une telle ressource dans le code de `exemple.LectureFichier`.

[1] Ce point est un peu technique et vous pouvez consulter la javadoc des méthodes ci-dessous et/ou rechercher des informations complémentaires sur le web. Les `ClassLoader` sont également mentionnés au S5 dans le cours sur la réflexivité.