AMD5
Devoir n° 2 : Labyrinthe

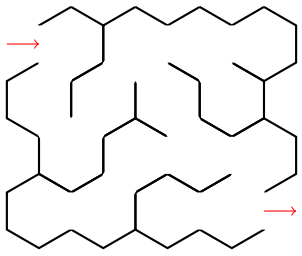
Ce TP est à rendre de façon individuelle sur DidEL jusqu'au 25 octobre.
Regardez attentivement la procédure de rendu en fin de sujet.

I) Fichiers fournis

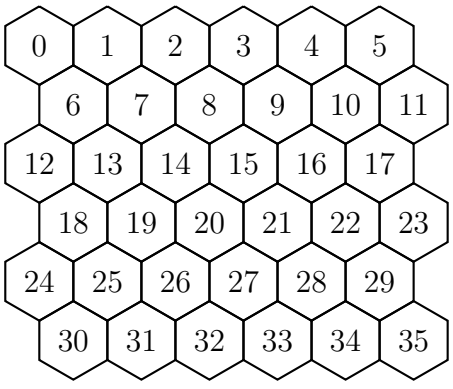
Le sujet est fourni avec quelques fichiers :
- ce présent document PDF détaillant le sujet ;
- 3 fichiers JAVA qui représentent des objets et contiennent quelques méthodes de base à comprendre par vous même :
  . `Cellule.java`,
  . `Neighbor.java`,
  . `Grid.java` ;
- un fichier `Laby.java` qui contient pour le moment juste la méthode `main` avec quelques exemples et devra contenir la méthode `makeLaby` que vous implémenterez ;
- un fichier `Makefile` ;
- un fichier `bonus.xml`, très simple, à compléter avec vos réponses bonus ;
- un script de vérification `check.sh` à utiliser au moment du rendu, cf. partie IV).

II) Présentation

Le but de ce devoir est d'écrire un programme pour engendrer des labyrinthes sur une grille hexagonale (cf. figure 1b), comme celui de la figure 1a. Sur ces figures, les traits noirs représentent les *murs* du labyrinthe.



(a) Un labyrinthe sur une grille hexagonale 4×4.



(b) Une grille hexagonale 6×6 avec numéros des cases.

FIGURE 1 – Exemple d'un labyrinthe et d'une grille hexagonale.

Un *labyrinthe* est une grille hexagonale à laquelle on a retiré des murs de sorte que, entre deux cellules c et c' quelconques, il existe **un unique chemin simple**. On dit alors que

les deux cellules c et c' sont connectées. En particulier, on peut aller de l'entrée (cellule en haut à gauche) à la sortie (en bas à droite).

Pour construire un tel labyrinthe, une implémentation en Java des objets `Cellule`, `Neighbor` et `Grid` est fournie. Celle-ci peut-être complétée, mais le code existant ne doit pas être modifié. La position des voisins d'une case c dans une grille est illustrée à la figure 2.

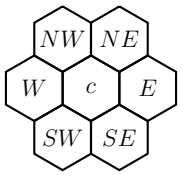


FIGURE 2 – Les voisins d'une cellule c.

L'objectif est de programmer une fonction `makeLaby` qui crée un labyrinthe aléatoire. La fonction que vous programmerez sera basée sur UNION-FIND. La partie qui suit donne des indications sur la façon d'aborder le problème.

III) Algorithme

L'algorithme pourra suivre la structure suivante :

Entrées : les deux dimensions n et m de la grille
Sortie : une grille qui est un labyrinthe.

initialiser la grille de taille n (lignes) et m (colonnes) avec tous les murs construits.
tant que *il existe deux cellules non connectées* **faire**
  pour *chaque cellule c dans un certain ordre* **faire**
    si *il existe un voisin de c non connecté à c* **alors**
      choisir aléatoirement un voisin c' de c non connecté à c
      casser le mur entre c et c'
casser le mur `West` de la cellule en haut à gauche
casser le mur `East` de la cellule en bas à droite

Algorithme 1 : créer un labyrinthe

a) Parcours simple

Commencez par réaliser le programme, en prenant les cellules dans l'ordre de gauche à droite et de haut en bas (c'est à dire l'ordre de leur numéro, cf. figure 1b).

Question bonus 1. *Combien de fois faut-il exécuter la boucle* `tant que` *avant d'arriver, de façon certaine, à un labyrinthe ?*

Question bonus 2. *Trouvez un labyrinthe qui ne peut pas être généré par ce programme.*

b) Parcours aléatoire

La méthode précédente ne permet pas de générer tous les labyrinthes possibles. Pour palier ce problème, on parcourt les cellules (boucle `pour`) dans un ordre aléatoire.

Voici un algorithme qui prend en argument un entier n et renvoie un tableau de taille n contenant, dans un ordre aléatoire, chaque entier de 0 à $n-1$ (une seule fois chacun). Un tel tableau est appelé *permutation des n premiers entiers*.

Entrées : entier n
Sortie : une permutation des n premiers entiers.

Initialiser un tableau `p` de taille n
pour *i de 0 à* $n-1$ **faire**
| `p`$[i] = i$
pour *i de 0 à* $n-1$ **faire**
| choisir aléatoirement un entier j avec $i \leqslant j < n$
| permuter $p[i]$ et $p[j]$

Algorithme 2 : permutation de n entiers

Implémentez la nouvelle méthode pour créer un labyrinthe aléatoire.

Question bonus 3. *Faut-il réaliser la boucle* `tant que` *de l'algorithme 1 plusieurs fois avant d'obtenir un labyrinthe ?*

IV) Rendu

a) Date et heure de rendu

La date limite de dépôt de votre archive sur DidEL est fixée au lundi 26 octobre à 8h.

b) Nom de l'archive

Vous devez rendre une archive TAR, nommé ainsi : `<nom_prenom.tar>`. Le nom et le prénom doivent être écrit en minuscule et sans accent. Les espaces et les apostrophes doivent être remplacées par des tirets. Par exemple, l'étudiant « Jean-Étienne du Château de l'Amérique » devra soumettre l'archive :

```
du-chateau-de-l-amerique_jean-etienne.tar
```

c) Contenu de l'archive

L'archive contiendra obligatoirement les fichiers suivants :
- `Cellule.java`
- `Neighbor.java`
- `Grid.java`
- `Laby.java` contenant la méthode public `makeLaby` demandée
- `Makefile`

et optionnellement les fichiers :

- `bonus.xml`
- d'autres fichiers JAVA, à votre guise (mais modifiez le Makefile en fonction).

d) Test de l'archive

L'archive peut être obtenue par la ligne de commande suivante :

```
tar cf <nom_prenom.tar> <fichier1> <fichier2>...
```

Avant de soumettre l'archive sur DidEL, testez-la à l'aide du script `check.sh` :

```
./check.sh <nom_prenom.tar>
```

Ce script ne teste que la présence ou l'absence des fichiers attendus. Une acceptation ne veut ainsi pas dire que toutes les consignes ont été respectées, en revanche un rejet est de façon certaine éliminatoire. Les fichiers trouvés sont affichés.

e) Modification des fichiers fournis

Les fichiers fournis peuvent être modifiés, à condition de respecter les restrictions suivantes :

- dans les fichiers `Cellule.java`, `Neighbor.java` et `Grid.java`, vous ne pouvez qu'ajouter des méthodes. Vous ne pouvez donc pas modifier l'objet (ni modifier le type de ses attributs ni en ajouter), ni modifier les méthodes et constructeurs déjà présents ;
- un fichier `Makefile` est fourni qui peut être modifié mais la compilation doit pouvoir être effectuée avec la commande `make run` ;
- le fichier `bonus.xml` est à compléter. Vous ne devez supprimer aucun caractère présent dans le fichier, et vous ne pouvez écrire qu'entre les balises `<question>` :

```
...
<!-- QUESTION BONUS 2 -->
<question num="2">
        Écrivez ici votre réponse à la question bonus 2.
</question>
...
```

f) Notation

Les programmes seront corrigés automatiquement. Il faut donc veiller à formater les noms de fichiers et de méthodes comme demandé dans le sujet. Si ces consignes ne sont pas vérifiées, des points en moins, voire une multiplication par 0, pourront être appliqués sur la note du devoir.

Un détecteur de plagia sera utilisé. En cas de copie, des sanctions, pouvant aller jusqu'à l'attribution de la note 0 pour **chacun** des étudiants concernés, seront appliquées.

Les programmes seront aussi considérés manuellement. Il faut donc aussi soigner le code (indentation, commentaires, clarté, noms de variables bien choisis).