Programmation fonctionnelle

Énoncé du Travail

Année 2011-2012

1 Génération et résolution de labyrinthes

Soit un ensemble de cellules (nœuds) disposées sur une grille rectangulaire, on représentera un labyrinthe par le graphe de connexion entre ses différentes cellules voisines. (Un arc représentant le fait qu'il est possible de passer d'un nœud à un autre.)

On s'intéressera uniquement aux labyrinthes dits parfaits. Un labyrinthe est dit parfait si pour chaque cellule du labyrinthe il existe un chemin unique la reliant à n'importe quelle autre cellule. (càd : le labyrinthe ne contient ni cycle ni cellule isolée.)

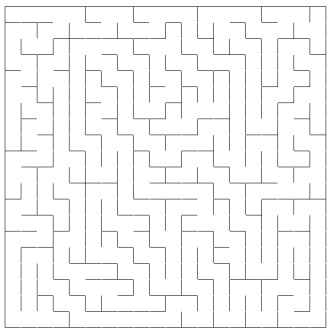

Fig. 1 – Labyrinthe parfait 20 x 20

Plusieurs techniques existent pour générer de tels labyrinthes :
→ `http://en.wikipedia.org/wiki/Maze_generation_algorithm`.

La méthode la plus courante consiste en une simple recherche en profondeur d'abord.

On commence par un labyrinthe où toutes les cellules sont isolées (non connectées à leurs voisines), et on choisit une cellule au hasard.

On marque la cellule courante comme visitée et on récupère la liste de ces voisines. On choisit au hasard une des cellules voisines non encore visitée et on crée un arc entre cette cellule et la cellule courante. On recommence en prenant la cellule voisine comme cellule courante.

L'algorithme se termine lorsque toutes les cellules ont été visitées.

On demande au minimum d'implémenter cette méthode (depth-first). Dans un deuxième temps, on demande de la comparer (efficacité, qualité des labyrinthes générés, ...) à d'autres méthodes (breath-first à la place du depth-first, division récursive, ...).

Une fois le labyrinthe généré, on demandera en plus de chercher une solution permettant de relier la cellule supérieure gauche à la cellule inférieure droite. La solution devant être générée sans connaissance a priori résultant de la construction du labyrinthe.

2 Travail

Il est demandé d'écrire un programme `Scheme` pouvant prendre en entrée deux entiers n et m correspondant respectivement au nombre de lignes et au nombre de colonnes du labyrinthe à générer. Le programme devra être capable de produire en sortie deux représentations graphiques au format `SVG` d'un labyrinthe de taille $n * m$. La première représentation contenant le labyrinthe seul, la seconde contenant le labyrinthe ainsi que sa solution.

Remarques :

- Ce travail est à réaliser par groupe de deux étudiants et doit être envoyé avant le 18 mai 2012 à 23h59 à l'adresse `lens@montefiore.ulg.ac.be`.
- Ce travail doit être envoyé sous la forme d'une archive (`nomA_nomB.tar.gz`) contenant un rapport au format `PDF` et le programme `Scheme` réalisé.
- Le programme fourni doit pouvoir tourner sur *DrRacket*.
- Vos solutions devront être réalisées sans utiliser d'instructions altérantes et/ou vecteurs.

3 Annexes

3.1 Scheme

3.1.1 Écriture dans un fichier

Les mécanismes permettant d'écrire dans un fichier à partir d'un programme Scheme sont décrits sur la page

`http://docs.plt-scheme.org/reference/file-ports.html`

(On s'intéressera en particulier à la fonction `with-output-to-file`.)

3.1.2 Random

Les fonctions permettant de générer un nombre aléatoire dans un programme Scheme sont décrits sur la page

```
http://docs.plt-scheme.org/reference/numbers.html
```

(On s'intéressera en particulier à la fonction `random`.)

3.1.3 Fichiers sources multiples

Pour plus de lisibilité, il est conseillé de répartir les fonctions composant le programme dans plusieurs fichiers sources.

Pour ce faire, on utilisera la fonction `(load "file.scm")` pour charger un fichier dans un autre.

3.2 SVG

Une spécification du format SVG peut être consultée à l'adresse :

```
http://www.w3.org/Graphics/SVG/
```