TD8 — Génération de terrains

1. Motivation

On va maintenant s'intéresser à des programmes-robots plus «complets», indépendants des terrains sur lesquels ils sont utilisés, pour permettre une plus grande autonomie du robot.

Pour tester les programmes-robots, on peut :

— leur fabriquer à la main des terrains particuliers : s'assurer qu'ils sortent de terrains «faciles», mesurer leur performance sur des terrains «difficiles»...
— vouloir les tester sur un grand nombre de terrains pour évaluer leur comportement moyen.

Pour que cette notion de «comportement moyen» ait un sens, on va tester les automates sur des terrains de mêmes caractéristiques : dimensions et densité/nombre d'obstacles.

Pour fabriquer un grand nombre de terrains (de caractéristiques données), on souhaite les générer aléatoirement.

Au final, on pourra comparer les performances de différents programmes-robots sur différents types de terrains : grands avec peu d'obstacles, petits avec beaucoup d'obstacles...

Remarque : on garantit que les terrains ont tous une issue (le programme est fourni — cf. exercice 4).

2. Génération de terrains

Il s'agit donc de fabriquer un fichier de `N` terrains de dimensions `l` et `h`, avec une certaine densité ou avec un certain nombre d'obstacles.

2.1 Génération de séquences pseudo-aléatoires (Rappel du TP2)

On dispose en C de la fonction `int rand(void)` de la librairie `stdlib.h` qui nous renvoie des nombres *pseudo-aléatoires*. Il s'agit d'une séquence de nombres très grande qui semble donc aléatoire mais qui est entièrement définie par une valeur initiale : la «*graine*». La fonction `void srand(int)` permet d'initialiser cette séquence en fournissant cette «graine». On peut réutiliser la même valeur à chaque exécution (pour reproduire l'expérience dans les mêmes conditions), ou utiliser par exemple la fonction `int time(NULL)` (du paquetage `time.h`, qui renvoie le nombre de secondes écoulées depuis le 1/1/1970), pour obtenir une séquence différente à chaque exécution du programme.

Exercice 1.

— Écrire un programme qui calcule 100 tirages de nombres entre deux naturels a et b donnés.
— Écrire un programme qui calcule 100 tirages d'une pièce pile/face.
— Écrire un programme qui calcule 100 tirages d'un couple de dés à 6 faces.

Exercice 2. Écrire un programme qui produit un fichier contenant un entier `N` suivi de `N` tirages aléatoires d'un dé à 6 faces.

3. Production d'un ensemble de terrains

Notre problème est de créer un fichier contenant un entier `N` suivi de `N` terrains de dimensions `L` et `H` donnés ayant un certain nombre d'obstacles. Cela revient à reprendre l'exercice 2 en remplaçant «dé à 6 faces» par «terrain de dimensions `L` et `H` avec `X` obstacles».

Le terrain étant quelque chose d'un peu plus compliqué qu'un dé, il peut être pratique de le générer entièrement, c'est-à-dire de calculer une variable de type `terrain`, avant de l'écrire dans le fichier.

Exercice 3. Reprendre la solution de l'exercice 2 et l'adapter à notre problème.

4. Validité d'un terrain avec obstacles

Supposons qu'on ait une méthode pour générer un terrain avec un certain pourcentage d'obstacles. Il faut aussi s'assurer que le terrain est valide i.e. possède un chemin formé des cases libres adjacentes menant de la position initiale du robot au bord.

.	.	X	.	.
X	X	X	.	X
X	.	C	.	X
.	X	.	.	X
X	.	X	X	.

Terrain valide

.	.	X	.	.
X	X	X	X	.
X	.	C	.	X
.	X	.	.	X
X	.	X	X	.

Terrain invalide

(X correspond à une case occupée indifféremment par de l'eau ou des rochers)

Exercice 4. Proposer un algorithme permettant de détecter si un terrain donné est valide ou non.