Université Galatasaray

Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 55 | génération d'un labyrinthe

Présentation

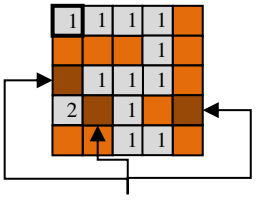
Ce TP est complémentaire de celui sur la recherche de chemin dans un labyrinthe. Il s'agit ici de générer des labyrinthes, de façon partiellement aléatoire. L'archive fournie avec le sujet contient certaines fonctions écrites pour le TP précédent, en particulier `dessine_labyrinthe` et `recherche_profondeur`. De plus, le type énuméré `valeur_case`, utilisé pour représenter la nature des cases du labyrinthe, est déjà créé.

1 Méthode par fusion

Il existe différentes méthodes pour générer aléatoirement un labyrinthe. Nous allons utiliser la méthode par fusion, qui fonctionne en manipulant des *zones* :

- Une zone est un ensemble de cases accessibles.
- Une zone est caractérisée par un numéro unique dans le labyrinthe.
- Une zone est connexe (il est possible d'atteindre n'importe quelle case de la zone à partir de n'importe quelle autre case de la zone).
- Une zone est close (elle contient toutes les cases qu'on peut atteindre à partir de chaque case de la zone).

labyrinthe contenant deux zones



murs utiles

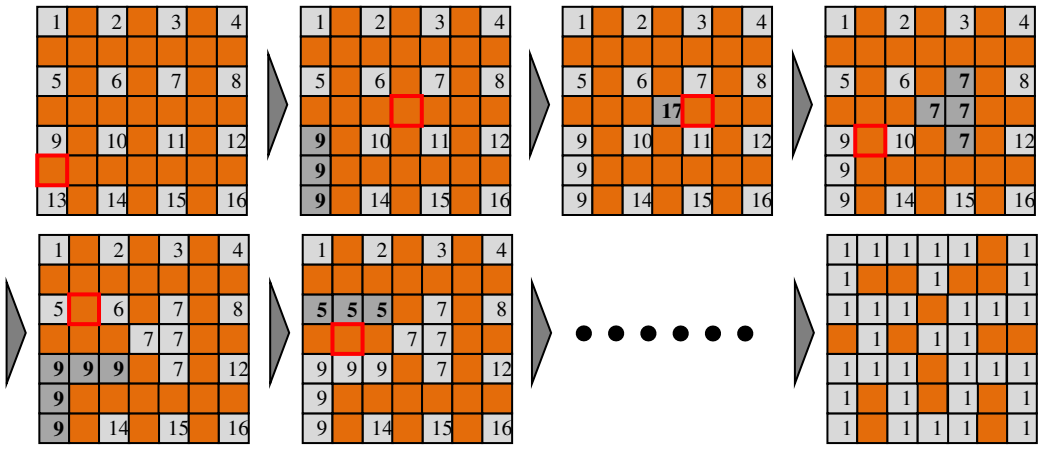
De plus, on dira qu'une case contenant un mur est *utile* si :

- Soit toutes les cases voisines sont aussi des murs.
- Soit la suppression du mur permet de fusionner au moins deux zones différentes.

Algorithme :

- Initialisation :
  - On part d'un labyrinthe dont les murs sont disposés de manière à former une grille.
  - Chaque case libre constitue une zone. Les zones sont numérotées en partant de 1 pour la zone correspondant à la case d'entrée.
- Fusion :
  - On sélectionne aléatoirement une case contenant un mur utile.
  - Le mur est supprimé, et les zones qu'il séparait sont fusionnées.
  - La zone obtenue porte le plus petit numéro des zones fusionnées.
  - Si le mur utile était entouré de murs, une nouvelle zone est créée (avec un nouveau numéro).
- On répète la fusion jusqu'à ce qu'il n'y ait plus qu'une seule zone.

exemple :



Contrairement au TP précédent, nous allons dans un premier temps représenter le labyrinthe sous la forme d'une matrice d'entiers, et non pas de valeurs de type `valeur_case`. On appellera cette structure un pseudo-labyrinthe. On utilisera la valeur 0 pour représenter les murs, et toute autre valeur positive correspondra au numéro d'une zone (donc à une case libre). Cela signifie donc que les zones sont numérotées à partir de 1.

La fonction `dessine_pseudo_labyrinthe` contenue dans la bibliothèque `labyrinthe` est fournie spécifiquement pour afficher ce type de matrice entière : chaque zone est représentée d'une couleur différente.

Exercice 1

Écrivez une fonction `int initialise_grille(int pseudo[DIM_LABY][DIM_LABY])` qui dispose les murs dans le pseudo-labyrinthe de manière à former une grille, et qui numérote les cases libres. La fonction doit renvoyer le nombre de zones créées.

Exercice 2

Écrivez une fonction `int est_utile(int pseudo[DIM_LABY][DIM_LABY], int x, int y)` qui renvoie 1 si le mur de coordonnées (x, y) est utile, et 0 sinon.

Exercice 3

Écrivez une fonction `void remplace_valeur(int pseudo[DIM_LABY][DIM_LABY], int ancienne, int nouvelle)` qui remplace dans le tableau `pseudo` toute occurrence de la valeur `ancienne` par une occurrence de la valeur `nouvelle`.

Exercice 4

Écrivez une fonction `void supprime_mur(int pseudo[DIM_LABY][DIM_LABY], int x, int y, int *n)` qui supprime le mur situé aux coordonnées (x, y). La fonction doit également déterminer les zones à fusionner puis effectuer la fusion (grâce à la fonction `remplace_valeur`), en n'oubliant pas de traiter la case de coordonnées (x, y). Le paramètre `n` sera éventuellement utilisé pour numéroter une nouvelle zone.

Exercice 5

Écrivez une fonction `int teste_zone_unique(int pseudo[DIM_LABY][DIM_LABY])` qui renvoie 1 s'il n'y a qu'une seule zone dans le pseudo-labyrinthe, et 0 sinon.

Exercice 6

La bibliothèque `labyrinthe` contient une fonction `void tire_mur(int pseudo[DIM_LABY][DIM_LABY], int *x, int *y)` qui tire au sort un des murs du pseudo-labyrinthe et qui renvoie ses coordonnées par les paramètres `x` et `y`.

Utilisez cette fonction ainsi que les fonctions précédentes pour écrire une fonction `void genere_pseudo_labyrinthe(int pseudo[DIM_LABY][DIM_LABY])` qui génère aléatoirement un pseudo-labyrinthe en utilisant l'algorithme décrit plus haut. Vous devez utiliser `dessine_pseudo_labyrinthe` pour donner le détail pas-à-pas de cette construction.

Exercice 7

On veut appliquer à notre labyrinthe un algorithme de recherche de chemin écrit lors du TP précédent. Mais pour cela, on a besoin d'une matrice contenant des valeurs de type `valeur_case` et non pas des entiers.

Écrivez une fonction `convertis_labyrinthe(int pseudo[DIM_LABY][DIM_LABY], valeur_case laby[DIM_LABY][DIM_LABY])` qui reçoit un pseudo-labyrinthe `pseudo` déjà créé avec `genere_pseudo_labyrinthe`, ainsi qu'un labyrinthe vide `laby`, et qui initialise `laby` grâce aux valeurs contenues dans `pseudo`.

Exercice 8

Dans la fonction `main`, créez un pseudo-labyrinthe avec `genere_pseudo_labyrinthe`, puis convertissez-le grâce à `convertis_labyrinthe` pour obtenir un labyrinthe, et enfin appliquez `recherche_profondeur` pour en trouver la solution.