IN103

Victor Priser Mathieu Petitbois Charles Meynard

March 2020

1 Introduction

Nous avons décidé d'effectuer notre projet d'in103 sur les labyrinthes parfaits. C'est à dire les labyrinthes tel qu'il existe unique chemin qui relie deux points du labyrinthe (en particulier il n'y a donc pas de boucles dans le labyrinthes).

Les principaux outils utilisés sont les piles, la fonction aléatoire afin de pouvoir générer un labyrinthe différent à chaque lancement du programme et la bibliothéque sdl pour pouvoir afficher le labyrinthe puis la résolution de ce dernier.

Ce projet contient exactement 3 parties ce qui rendit la division du travail plus facile:

1. Génération d'un labyrinthe aléatoire
2. Trouver le chemin entre l'entrée et la sortie (du point en haut à gauche au point en bas à droite)
3. Affichage du labyrinthe et des résultats

2 Création aléatoire du labyrinthe

2.1 Structures

Avant de commencer l'algorithme de création du labyrinthe on définit des structures qui vont faciliter l'implémentation de l'algorithme.

Une structude de pile et les fonctions associées

Une structure de case du labyrithe cas qui contient 6 informations:

1. Le nombre de murs et leurs positions (Nord, Sud, Est, Ouest), les variables étant mises à 1 pour un mur et 0 s'il n'y en a pas.
2. Un état qui indique si la case a déjà été visitée originellement à 0 puis à 1 une fois la case visitée.
3. La position de la case dans le labyrinthe (x et y).

Une structure de labyrinthe qui contient 3 informations:

1. sa hauteur
2. sa largeur
3. une matrice contenant toutes les cases du labyrinthe

2.2 Principe de l'algorithme

On construit d'abord un labyrinthe dont tous les murs sont fermés et dont aucune cases n'a été visitée.

1. On choisit une case du labyrinthe au hasard on met son état à 1 et on l'ajoute à la pile
2. tant que la pile n'est pas vide:
   (a) on visite la dernière case empilée
      i. Si toutes les cases autour de la case empilée sont traitées on la dépile
      ii. Sinon parmi les cases adjacentes à cette dernière qui n'ont pas été
3. on renvoit le labyrinthe ainsi créé

3 Résolution du labyrinthe

On utilise la méthode du fil d'arianne on peut aussi le voir comme le parcours d'un graphe en profondeur où les intersections correspondent à des noeuds. On va utiliser la structure de pile.

A l'origine nous n'avons que des cases à l'état 1. Pour dire qu'elles sont visitées, on les met à l'état 0. La méthode s'articule de la manière suivante:

1. Initialisation: On empile la première case. Et on dit qu'elle est visitée.
2. On prend une case au hasard adjacente possible (c'est à dire non visitée et accessible sans traverser de mur). Deux cas sont alors possibles.
   (a) Il y a pas de cases disponibles. Dans ce cas, on dépile notre case.
   (b) Il existe une case disponible et on indique alors que cette case est visitée.
3. On recommence avec le sommet de la pile jusqu'à que le sommet de la pile soit la case d'arrivée.
4. La pile finale est donc notre chemin.

On utilise une méthode d'affichage qui permet de bien visualiser l'empilement et le dépilement des cases.

4 Affichage du labyrinthe

On doit passer d'un labyrinthe sous forme de cases entourées de murs à un affichage à générer pixels par pixels. Pour ce faire, nous utilisons SDL. On procède de la manière suivante :

1. On définit un nombre de pixels longueur (800 horizontalement) et en hauteur (700 verticalement) qui définiront les dimensions de l'image affichée.
2. On doit ensuite adapter la taille en pixels d'une case, en fonction du nombre de cases du labyrinthe. On calcule donc $wc = \frac{LongueurEnPixels}{LongueurEnCasesDuLabyrinthe}$ et $hc = \frac{HauteurEnPixels}{hauteurEnCasesDuLabyrinthe}$ (divisions entières).
3. On doit ensuite adapter la taille en pixels d'une case, en fonction du nombre de cases du labyrinthe. On calcule donc $wc = \frac{LongueurEnPixels}{LongueurEnCasesDuLabyrinthe}$ et $hc = \frac{HauteurEnPixels}{hauteurEnCasesDuLabyrinthe}$ (divisions entières).
4. On retrouve ensuite les dimensions en pixels $w = wc * longueur$ en cases du labyrinthe et $h = hc * hauteur$ en cases du labyrinthe. Cela permet d'avoir des dimensions en pixels entières.
5. On crée ensuite la figure en commençant par tracer les bords grâce aux dimensions w et h en pixel.
6. Enfin, on parcourt la matrice du labyrinthe. Chaque case peut être vue comme une matrice de pixels de dimension (wc,hc). Pour éviter tout tracé de mur redondant, on ne s'occupe que des murs Ouest et Nord. S'il y a effectivement ces murs, on trace en noir les rectangles correspondant aux premières ligne et colonne de la matrice en pixels de taille respectives wc et hc.

5 Compilement du code

Pour compiler le code on exécute:

```
gcc labyrinthe.c `sdl-config --cflags --libs`
```

Les cases empilées dans l'algorithme de résolution s'affichent au fur et à mesure sur le labyrinthe. Puis le chemin final s'affiche d'une couleur différente. (en vert)