RATOUCHNIAK Antoine

Rapport
Projet Labyrinthe

Table des matières

Partie utilisateur....2
Ce que fait le projet....2
Comment le compiler....2
Comment l'utiliser....2
Partie développeur....3
Description informatique du projet....3
Comment sont supprimés les murs ?....3
Comment sont reliées les « classes » ?....3
Les options....4
L'option unique....4
L'option accessibilité....4
L'option victor....4
Le projet en général....5

RATOUCHNIAK Antoine

Partie utilisateur

Ce que fait le projet

Le labyrinthe est un projet qui affiche un labyrinthe avec proposant plusieurs difficultés choisies par l'utilisateur. On peut modifier cette difficulté en modifiant la taille du labyrinthe, la façon dont il est généré (accessibilité de toutes les cellules ou encore chemin unique).

Comment le compiler

Il existe trois manières pour lancer le labyrinthe.
La première méthode est d'utiliser la console et de taper :
« ./Labyrinthe --option1 --option2 --option3 » sur Windows avec les options que nous souhaitons avoir, que nous détaillerons dans le paragraphe suivant.
Il est également possible de recompiler le fichier en tapant :
« gcc -Wall -ansi index.c -o Labyrinthe »
La deuxième méthode est de tout simplement exécuter le fichier Labyrinthe.
La troisième méthode est d'utiliser le makefile en tapant « make » dans le terminal et en lançant le labyrinthe crée.

Comment l'utiliser

En lançant le fichier sans argument, le fichier lancera une interface graphique avec un labyrinthe d'une taille 6 x 8.
Il est cependant possible d'utiliser des arguments. En effet, le passage par la console avec arguments se constitue de la façon suivante :
« ./Labyrinthe --option1 --option2 » (attention aux deux tirets)
avec les options que sont :
--mode=texte permet l'affichage texte du labyrinthe à la place de l'interface graphique
--taille=AxB avec A et B à remplacer par la largeur et hauteur
--graine=X avec X la graine souhaitée
--attente=X avec X l'attente positive entre chaque destruction de mur (-1 pour attendre un clic de la souris ou une entrée clavier, 0 pour supprimer l'attente)
--unique permettant un unique chemin de l'entrée à l'arrivée
--acces rend toutes les cellules accessibles
--victor permet d'afficher le chemin victorieux, c'est-à-dire du le chemin le plus court du départ jusqu'à l'arrivée ; cette option ne marche qu'en affichage graphique
--optim optimise la création du labyrinthe et donc la vitesse de création

RATOUCHNIAK Antoine

Partie développeur

Description informatique du projet

Le labyrinthe utilise le Union-Find pour créer le labyrinthe. L'algorithme principal relie les cases pour les mettre dans la même « classe » jusqu'à ce que l'entrée et l'arrivée soient dans le même arbre.

Comment sont supprimés les murs ?

Les murs sont supprimés de manière aléatoire, mais de manière différente selon l'optimisation. Si l'utilisateur ne décide pas d'appliquer les optimisations alors des murs sont supprimés aléatoirement avec la fonction rand(). En revanche, si l'utilisateur active –optim alors le processus sera différent. Un tableau va d'abord répertorier tous les murs possiblement destructibles dans la fonction mélanger(Mur *tableau, int taille, int tx, int ty) puis les mélanger en parcourant le tableau et en inversant avec une autre position aléatoire du tableau. Cette méthode vue en TP est très efficace quand le labyrinthe atteint une très grande taille.

Comment sont reliées les « classes » ?

Les classes sont un ensemble de cases qui ont tous une racine commune. Comme expliqué en cours, on relie les arbres et donc les classes par fusion par rang. Cela permet d'avoir les arbres les plus petits possibles. Pour une fusion entre un arbre X et Y, on choisit le rang le plus grand, prenons par exemple Y, alors Y devient le père de X. On maintient par cet algorithme des arbres de rang minimum. On utilise un algorithme de trouve et compresse pour toujours limiter la recherche et être le plus rapide possible.

Les fonctions ont toutes un nom explicite telles que :
initialiser : initialise le labyrinthe
affichageTexte : affiche la version texte du labyrinthe
affichageGraphique : affiche la version graphique du labyrinthe
trouver : qui cherche la racine d'un point et compresse aussi
separation : renvoie 1 si deux cases passées en paramètre sont séparées d'un mur
supprimerMur : algorithme qui supprime un mur et relie deux points
relierCases : complète la fonction supprimerMur, lie dans la même classe deux points
calculerRang : estime le plus haut rang d'une classe
coordonneesEgales : renvoie 1 si deux coordonnees sont égales ...

RATOUCHNIAK Antoine

Les options

L'option unique

L'option --unique fonctionne assez simplement. Pour éviter plusieurs chemins menant à la victoire, alors il suffit d'éviter de supprimer les séparations entre deux cases étant dans une même classe, même si elles sont séparées par un mur. On instaure donc une fonction memeClasse(labyrinthe_t laby, int x1, int y1, int x2, int y2), qui vérifie que la racine de la case (x1, y1) et (x2, y2) soit différente. Si c'est le cas alors on peut fusionner les deux arbres et enlever la séparation. Ce paramètre agit dans la fonction supprimerMur.

L'option accessibilité

L'option --acces permet l'accessibilité de toutes les cases du labyrinthe. Après la création du labyrinthe, il reste en théorie beaucoup moins de cases seules ou dans une classe différente de la classe reliant le départ de l'arrivée. On parcourt donc le tableau en partant de (0, 0) et en avançant de longueur puis en hauteur, dès qu'on rencontre une case qui n'est pas dans notre « classe principale », alors on la lie avec la case de gauche, haut, droite ou bas selon sa classe à elle. La classe principale étant la classe reliant le départ de l'arrivée. Une case n'étant pas dans la « classe principale » a forcément un voisin dans la « classe principale ». On lie alors ces deux cases. Ce paramètre agit après la suppression de tous les murs.

L'option victorieux

L'option --victor permet d'afficher le chemin le plus court allant du départ jusqu'à l'arrivée. La fonction cheminPlusCourt(labyrinthe_t laby, coordonnees_t *chemin, int *taille) affectant au pointeur chemin le chemin du départ à l'arrivée et à taille la taille du chemin. L'algorithme utilisé est l'algorithme A*, qui m'a été proposé en TP. L'algorithme fonctionne dans un arbre ou graphe en avançant progressivement en se rapprochant de l'arrivée grâce au point le plus proche de celui-ci. On affecte une valeur à chaque case traversée pour évaluer sa « qualité » et permettre d'évaluer le chemin le plus rapide. La qualité prend en compte la distance du point en question à l'arrivée, la distance du point en question au départ et du dernier point parcouru. On atteint donc l'arrivée et on remonte en prenant la qualité la plus basse, car c'est aussi le chemin le plus court. Je laisse un lien qui m'a bien permis de comprendre comment fonctionnait cet algorithme. Ce paramètre agit quand le départ et l'arrivée sont reliés dans la fonction affichageGraphique.
https://khayyam.developpez.com/articles/algo/astar/

RATOUCHNIAK Antoine

Le projet en général

Le projet labyrinthe m'a été extrêmement bénéfique car il m'a permis de développer davantage ma logique. En effet, les structures Union-Find sont très intéressantes et permettent d'avoir des résultats extrêmement rapidement. J'ai dû me pencher sur plusieurs problématiques dans ce projet, notamment celle du chemin victorieux avec l'algorithme A* qui n'a pas été facile à comprendre au début, mais que j'ai trouvé très intéressant.