Xu Frédéric
Lecointre Etienne

2019/2020
L2 Math-info

Algorithme des arbres
Projet
Labyrinthe

Objectif :

L'objectif de ce projet est de créer un labyrinthe en C.
En utilisant l'algorithmique des arbres et surtout l'Union-Find.

Présentation du projet :

partie utilisateur :

Tout d'abord , pour compiler le projet , il suffit d'écrire make dans le terminale.
L'utilisateur peut choisir plusieurs options :
./labyrinthe --texte=texte (pour afficher sous forme de texte utf8 , sinon ca sera par libMLV par défaut)
./labyrinthe --taille=6x8 (Choisir la taille du labyrinthe , longueur sur largeur)
./labyrinthe –taille=nxm --graine=X (X une valeur entre 0 à 4294967295, mettre le même graine génère le même labyrinthe)
./labyrinthe –taille=nxm --attente=X paramètre définissant le temps d'attente entre chaque retrait de mur en *pas bcp* si omis le programme passera immédiatement dans son mode le moins amusant 1 mur en moins par appuie sur la touche entrée si le temps est mis à 0 le programme ometera tout les affichages intermédiaires pour afficher le plus vite possible le résultat
./labyrinthe –taille=nxm –unique (le rendre plus esthétique , mais on a eu un problème là-dessus donc non fonctionnelle)

Il est possible de lancer plusieurs options par exemple :
./labyrinthe –texte=texte –taille=5x5 –graine=50 –attente=0 –unique

partie développeur :

Nous avons choisi de faire le stockage des murs bit par bit 1 mur 0 absence de mur. La raison pour cela est qu'il est beaucoup plus optimisé de faire ainsi. Cela prend beaucoup moins de place que des int ou char.

Nous avons divisé le projet en plusieurs modules :
-affichage pour tout ce qui est affichage ASCII et MLV
- laby pour la création , la validité , les chemins du labyrinthe.
-main pour tous les options .
-mur pour les murs , l'aléatoire et le bit par bit.

Les principales fonctions du module mur sont :

init_wall qui permet une initialisation simple du module

delete_wall permet la suppression d'un mur et renvera sa position relative qui est rendu plus compréhensible avec la fonction suivante

vh_pos

Les principales fonctions du module affichage sont .

aff_wall permet d'afficher le labyrinthe dans un style me rappelant un peu pac-man

aff_wall_base propose une alternative moins esthétique à aff_wall mais n'utilisant que des caractères ascii et ne demandant pas de buffer (à noter que cette version est moins compacte)

Ceux de laby sont :

cree_laby,cree_case,cree_coord pour créer le labyrinthe

laby_valide qui valide le labyrinthe donné

trouve_compresse et fusion_rang qui utilise l'Union-Find pour valider le labyrinthe.

Pour plus de détails , regarder laby.h

Rôles des fonctions dans les modules :

(laby)

cree_laby,cree_case,cree_coord permettent d'initialiser le labyrinthe

laby_valide est utilisé dans le main.

enleve_mur prends des arguments du module mur et utilise fusion_rang pour

rassembler deux cases.

fusion_rang utilise trouve_compresse pour trouver le représentant des deux ensembles et les fusionnes.

trouve_compresse est utilisé par laby_valide et fusion_rang pour trouver les représentant des ensembles de case.

(affichage)

Tout d'abord la structure MLV_du_pauvre contient notamment l'identifiant de la fenêtre de son père, son contexte graphique et sa connection au serveur graphique afin de permettre la gestion de celle-ci on passera donc cette structure pour toute les utilisations de la librairie Xlib

Associée à cette structure, on trouve son initialisation qui initialise toutes les valeurs dont on a besoin alloue la mémoire puis la remplis en attendant la réponse du serveur,cette réponse est nécessaire pour sortir de la fonction afin d'éviter d'utiliser une fenêtre non mapée.

Evidemment une allocation dynamique implique une fonction pour libérer cette mémoire cette fonctionnalitée est assurée par libere_MLV_pauvre qui libérera tout les pointeurs puisqu'ils sont fournis dans la structure.

Enfin on a les fonctions d'affichage

(mur)

Devant la tâche de stocker tout les murs du labyrinthe (potentiellement 2 fois pour plus d'efficacité sur le --unique) et la faible mémoire et puissance de mon ordinateur j'ai décidé de stocker les murs de manière compactes en en stockant 8 par octet de stockage divisée via l'entier BLOCK qui définit la taille d'octet à assigner par block afin de garder un accès rapide sans demander une place mémoire excessive cet entier pourra donc être modifié selon la capacité de la machine sa puissance et l'états de sa mémoire afin d'offrir des performances correctstout en permettant un accès rapide avec le maintiens du nombre de murs par blocks et par charactère .

On a donc l'initialisation de ces structures

La suppression d'un mur se fait par delete_wall qui prend une liste de mur et le numéro du mur à supprimer

la réponse de delete_wall sera simplement traduite par vh_pos pour être compatible au module de l'autre

enfin on a la fonction pour libérer la mémoire

Bilan :

Difficulté rencontré :

On a vraiment eu du mal à afficher correctement les labyrinthes de taille 2x2 sous forme ASCII Extended.
Comprendre le code du partenaire.
Télécharger libMLV , donc on a utilisé Xlib (on a quand même essayé de faire mlv sans pouvoir essayer).
Je ne comprends pas pourquoi unique ne marche pas , pourtant je pense avoir utilisé la bonne méthode.
Quelques soucis avec Xlib , notamment sur leur resize

Répartition des taches :

Frédéric : Labyrinthe , affichage , Rapport.
Etienne : Mur , Main , affichage.