UNIVERSITÉ DE STRASBOURG

Licence de Sciences
Mention Informatique $2^{ème}$ année
Semestre 4, Session automne
Année 2016/2017
U.E. L2S4 SDA2
Projet
reçu le Lundi 3 Octobre 2016
rendu le Lundi 12 Décembre 2016

Génération et recherche de chemin dans un labyrinthe

David Braun - david.braun@etu.unistra.fr

1 Historique

Le «Labyrinthe» vient du grec laburinthos, qui signifie littéralement «palais des haches». Les labyrinthes nous renvoient 4000 ans en arrière dans l'histoire. Ils prirent, pendant une longue période (3000 ans), la forme d'une courbe sinueuse sans jonctions. Il ne s'agissait pas de jeux mais d'un endroit dans lequel prenaient place les marches rituelles, les processions et les courses. C'est dans la mythologie grecque que l'on trouve le labyrinthe le plus connu, celui qui enferma le Minotaure, fils de Knossos, tué par Thésée. Quelle que soit la véracité de ce mythe, on trouve le dessin d'un labyrinthe crétois à 7 anneaux sur des pièces de monnaie datant du 1er siècle avant J.C.. De nos jours, de nombreux labyrinthes existent, c'est même une passion qui prend de l'ampleur, notamment en Angleterre où l'on trouve les plus beaux labyrinthes de haies.

2 Sujet

2.1 Introduction

Créez un fichier Makefile permettant de compiler vos programmes. Vous indiquerez les dépendances entre fichiers objets (.o) et fichiers d'en-tête (.h) de manière à recompiler ce qui est nécessaire (et rien de plus) si un fichier d'en-tête est modifié. Vous penserez à documenter votre projet dans son intégralité selon le schéma suivant : *pré-conditions*, *post-conditions*, *paramètres*, *sortie*, *descriptif*. Pensez à utiliser des outils comme gdb, valgrind ou gprof pour faciliter votre développement et optimiser votre programme.

2.2 Ensembles

Ecrivez une bibliothèque de manipulation d'ensembles de couples d'entiers (x,y) en utilisant la programmation par mutation. Les entiers x et y seront dans un intervalle fixe de valeurs (entre 0 et une valeur maximale donnée). Cette bibliothèque contiendra au minimum les fonctions suivantes :

- EnsAlloc : création d'un nouvel ensemble
- EnsFree : libération éventuelle de la mémoire utilisée
- EnsEstVide : teste si un ensemble est vide
- EnsAjoute : ajoute un couple à un ensemble
- EnsSuppr : retire un couple à un ensemble
- EnsEstDans : teste si un couple appartient à un ensemble
- EnsTaille : nombre d'éléments dans l'ensemble
- EnsTirage : tire un couple aléatoirement dans un ensemble, et le retire de l'ensemble

Ces différentes fonctions seront appelés fréquemment, choisissez donc soigneusement le type d'implémentation que vous allez utiliser pour optimiser les performances de génération du labyrinthe.

2.3 Matrices

Ecrivez une bibliothèque de manipulation de matrices h * l d'entiers, en utilisant la programmation par mutation. Les entiers h et l (hauteur et largeur de la matrice) seront donnés en paramètre à la fonction de création de la matrice. Cette bibliothèque contiendra au moins les fonctions suivantes :

- MatAlloc : création d'une nouvelle matrice h * l initialisée à 0
- MatFree : libération éventuelle de la mémoire utilisée
- MatVal : renvoie la valeur entière matrice[x,y]
- MatSet : stocke une valeur entière dans matrice[x,y]
- MatSauve : enregistre une matrice dans un fichier
- MatLit : lit une matrice dans un fichier

2.4 Bibliothèque graphique

Utilisez la bibliothèque *graph* fournie pour afficher votre labyrinthe. Vous pouvez éventuellement ajouter vos propres fonctions dans une bibilothèque supplémentaire.

2.5 Génération du labyrinthe

La générateur de labyrinthe fonctionnera de la manière suivante :

- initialisation de la matrice vide (0 représente une case vide, 1 représente un mur)
- création d'un certain nombre de graines initiales, sur les bords et à l'intérieur de la matrice : des murs peuvent être construits autour des graines
- initialisation de l'ensemble des cases constructibles
- boucle principale. Tant qu'il reste des cases constructibles :
  - choisir aléatoirement une case constructible
  - y construire un mur
  - mettre à jour les cases voisines dans l'ensemble des cases constructibles

Vous écrirez une fonction *EstConstructible*, qui teste si une case de la matrice est constructible : elle doit posséder un mur sur un de ses 4-côtés, et, parmi les 8-voisins de cette case, les 5 voisins opposés à ce mur doivent être vides.

Ce programme aura la syntaxe suivante :
`./genlab [-v] [-d] [-l <largeur>] [-h <hauteur>] <fichier>`
`-v` : visualise le labyrinthe pendant sa génération
`-d` : visualise le labyrinthe et les cases constructibles pendant la génération (implique -v)
`-l` <largeur> et -h <hauteur> : largeur et hauteur du labyrinthe, en nombre de cases
`<fichier>` : nom de fichier dans lequel le labyrinthe sera sauvegardé
Valeurs par défaut : largeur = 300, hauteur = 200, taille d'une case affichée = 3*3 pixels, taille de la fenêtre 1000*800

2.6 Recherche de chemin

Ecrivez un programme `cheminlab` qui lit un labyrinthe depuis un fichier, et recherche un chemin depuis le coin supérieur gauche jusqu'au coin inférieur droit du labyrinthe. Ce programme aura la syntaxe suivante :
`./cheminlab [-v] <fichier>`
`-v` : visualise le parcours du labyrinthe pendant la recherche
`<fichier>` : nom du fichier labyrinthe à lire.

Le but est d'établir une méthode efficace pour rechercher un chemin au sein d'un labyrinthe. Pour cela nous allons étudier de manière incrémentale la recherche d'une solution.

- Recherche aléatoire du chemin (choix aléatoire parmi les voisins)
- Recherche avec une heuristique (choix du voisin avec la plus faible distance à vol d'oiseau avec l'arrivée)

- Recherche du plus court chemin (Algorithme de Dijkstra : *https://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra*)
- Recherche du meilleur chemin (Algorithme A* : Dijkstra + heuristique)
  http://khayyam.developpez.com/articles/algo/astar/
  *https://en.wikipedia.org/wiki/A*_search_algorithm*

Extension : il est possible d'améliorer encore l'algorithme A* en améliorant l'heuristique. Etudier un la complexité de vos algorithmes et l'optimalité de vos structures avec le but recherché pour trouver **un chemin dans les plus brefs délais**. On souhaite obtenir une solution avec le temps d'exécution le plus faible en moyenne sur un labyrinthe généré de manière aléatoire.

3 Modalités pratique

Sauf exception, le projet devra être réalisé en binôme. Le langage à utiliser est le C. En plus du code documenté et correctement indenté, vous devrez fournir un rapport clair et concis de quelques pages (jusqu'à 5 pages au maximum) reprenant les spéficités des principales procédures de votre implémentation (choix d'implémentation, complexité ...). Pour chaque structure manipulée, illustrez et justifiez vos choix d'implémentation. Le projet est à rendre pour le 12 décembre 2016 (date limite stricte) et sera pré-évalué en séance de TP le 12 décembre 2016. Le code et le rapport devront être déposés sous la forme d'une archive (zip ou tar.gz) sur le dépôt moodle3. En cas de problème, contactez moi : *david.braun@etu.unistra.fr*.