RAPPORT DE PROJET: LABYRINTHE

Romain THOMAS
Kevin SOARES

Année 2021/2022

SOMMAIRE

Contents

1 Environnement de travail **3**

2 Description du projet **3**
2.1 Présentation générale du projet 3
2.2 Présentation des algorithmes utilisés 3

3 Bibliothèques et outils utilisés **3**

4 Travail réalisé **4**
4.1 Fonctionnalités prévues 4
4.2 Fonctionnalités réalisées 4
4.3 Répartition du travail 4

5 Difficultés **4**
5.1 Difficultés rencontrées par Romain THOMAS 5
5.2 Difficultés rencontrées par Kevin SOARES 5

6 Bilan **5**
6.1 Bilan personnel Romain THOMAS 5
6.2 Bilan personnel Kevin Soares 5
6.3 Conclusion 5

7 Webographie **6**

8 Annexes **6**
8.1 Cahier des charges 6
8.2 Exemple d'exécution du projet 6
8.3 Manuel utilisateur 6

1 Environnement de travail

Projet réalisé sous Linux avec la machine virtuelle utilisée en cours cette année.

2 Description du projet

2.1 Présentation générale du projet

Ce programme crée un labyrinthe complexe aléatoire, de la taille entrée par l'utilisateur, et affiche le résultat de l'exécution dans un fichier .txt

2.2 Présentation des algorithmes utilisés

Notions utilisées:
-Fonctions
-Allocation dynamique
-Fichiers

Détail des fonctions:

matrice_init: Alloue dynamiquement une matrice de taille 2n plus 1 et met des 0 pour les murs et des chiffres de 1 à n au carré pour les cases

test_cases: Teste chaque case du labyrinthe pour savoir si elles ont la même valeur

wall_break: Casse le mur de coordonnées [x][y] passé en paramètre et met au hasard la valeur de gauche/droite ou haut/bas

random_wall: Renvoie 2 valeurs x et y qui correspondent aux coordonnées d'un mur et utilise ensuite la fonction précédente avec ces coordonnées

murs_restant: Fini ce qu'a commencé la fonction wall_break

affichage_laby: Affiche le labyrinthe sous une forme lisible

fichier_labyrinthe: Affiche le labyrinthe dans un fichier au format .txt

3 Bibliothèques et outils utilisés

stdio.h: Utilisée pour appeler toutes les fonctions "basiques" en C (ex: printf, scanf
stdlib.h: Utilisée pour les fonctions d'allocation dynamique (ex: malloc) et pour l'aleatoire (fonction rand)

time.h: Utilisée pour choisir des nombres aléatoires sans avoir à chaque fois la même suite de nombres

4 Travail réalisé

COMPARAISON ENTRE LE TRAVAIL PRÉVU AU DÉBUT DU PROJET ET LE TRAVAIL RENDU À LA FIN

4.1 Fonctionnalités prévues

L'idée de base était de créer un labyrinthe aléatoire et de le résoudre, et d'afficher ce labyrinthe avec la solution.

4.2 Fonctionnalités réalisées

Le rendu est un labyrinthe complexe (plusieurs chemins possibles au lieu d'un seul), sans sa solution la plus courte.

4.3 Répartition du travail

Le code a été découpé en fonctions qui marchent ensemble. La répartition est la suivante:

Romain THOMAS:
-fonction wall_break
-fonction murs_restants
-fonction affichage_labyrinthe

Kevin SOARES:
-fonction test_cases
-fonction random_wall
-fonction fichier_labyrinthe
-fonction affiche

Réalisé ensemble:
-fonction matrice_init

5 Difficultés

LISTE DES DIFFICULTÉS RENCONTRÉES PAR CHAQUE PERSONNE

5.1 Difficultés rencontrées par Romain THOMAS

Allocation dynamique: C'est la notion qui m'as posé le plus de difficultés dans ce projet. L'allocation dynamique sur des tableaux à deux dimensions est plus compliquée et plus sujette à des erreurs d'inattention que sur des tableaux à une dimension
Dimension de la matrice: Nous avons pris du temps du trouver le format à utiliser pour la matrice

5.2 Difficultés rencontrées par Kevin SOARES

"Randomiser":J'ai eu du mal avec la fonction random_wall qui permet de choisir un mur au hasard
Découpage du programme: Ca a été compliqué de trouver les étapes à réaliser afin de découper le programme en plusieurs fonctions ayant chacune un rôle précis

6 Bilan

6.1 Bilan personnel Romain THOMAS

J'ai trouvé le projet beaucoup plus difficile que ce qui est demandé en TD. Même s'il a fallu parfois se creuser la tête, cela a permis de m'améliorer beaucoup plus rapidement qu'en faisant seulement les TD.
J'aime bien aussi le fait d'avoir un gros projet sur lequel on avance progressivement plutôt que des exercices simples mais parfois un peu abstraits.

6.2 Bilan personnel Kevin Soares

J'ai trouvé le projet très concret par rapport aux TD, ce qui donne une réelle motivation pour s'impliquer dedans. De plus, la difficulté rajoute de la motivation. Et finalement c'est assez amusant à faire.

6.3 Conclusion

Avoir un projet est vraiment formateur. Le fait de ne pas savoir comment structurer les étapes et organiser son code est parfois décourageant, mais cela permet de vraiment s'améliorer. En effet, on doit se renseigner et comprendre par nous mêmes certaines notions qu'il aurait été plus lent d'assimiler en cours.

Cela permet aussi de s'entrainer à travailler en groupe, à comprendre les codes des autres et à savoir expliquer les siens.C'est donc vraiment utile dans des études qui amènent vers les métiers de l'informatique.

7 Webographie

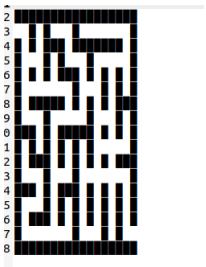
Sites Internet utilisés pour de l'aide:
-KooR.fr
-Stack Overflow.com
-Wikipédia
-Diverses vidéos d'explications sur YouTube

8 Annexes

8.1 Cahier des charges

Le programme est découpé en plusieurs fonctions qui respectivement:
- alloue dynamiquement une matrice de taille 2n plus 1 et met des 0 pour les murs et des chiffres de 1 à n au carré pour les cases
- teste chaque case du labyrinthe pour savoir si elles ont la même valeur
- renvoie 2 valeurs x et y qui correspondent aux coordonnées d'un mur et utilise ensuite la fonction qui suit avec ces coordonnées
- casse le mur [x][y] passé en paramètre et met au hasard la valeur de gauche/droite ou haut/bas
- affiche le labyrinthe avec des #(pour les murs) et des espaces
- créer un fichier (ou le modifie si existant) en y mettant l'affichage du labyrinthe

8.2 Exemple d'exécution du projet



8.3 Manuel utilisateur

Pour faire fonctionner le programme:
-Compiler
-Exécuter
-Entrer la taille souhaitée (attention la taille entrée n devient 2n plus 1 car on compte les murs dans la taille du labyrinthe
-Ouvrir le fichier .txt contenant le résultat de l'exécution