LABYRINTHE

Aya Moumen Mokhtary, Art Hashani

Mai 2023

1 Introduction

Notre projet porte sur la génération et résolution de labyrinthes aléatoires, plus précisément de labyrinthes parfaits. La programmation a été faite en langage C, nous avons fait usage des notions vues en cours comme l'allocation dynamique et les structures composées. Nous avons codé le plus proprement possible, avec plusieurs fonctions, de sorte à ce que ce soit lisible et compréhensible. Nous avons tout de meme essayé d'intégrer la bibliothèque SDL à notre projet. C'est pour cela que l'on a joint trois versions de codes: une version sans SDL, une version avec SDL où on a la génération d'un labyrinthe qui n'est pas parfait et une version avec SDL qui génère un labyrinthe parfait.

2 Environnement de travail

Nous avons choisi de réaliser notre projet sur Visual Studio Code, qui pour nous est le meilleur IDE qui existe. Nous avons testé les IDE que l'on utilisait en cours, comme Geany ou encore Cod::blocks, mais nous n'avons pas accroché plus que cela. Le système d'exploitation dans lequel nous avons travaillé est le système Linux.

Nous avons organisé notre travail en deux parties:

- Nous avions d'abord codé sans la bibliothèque SDL. Ce travail se faisait en TD, où nous avions le professeur à notre disposition pour quelconques questions.
- Ensuite, nous avions repris le code pour y intégrer cette dernière. Nous avons été aidé par chatGPT et lorsqu'on avait des bloquages, on demandait de l'aide à des membres de nos famille, qui travaillent dans le domaine de l'informatique (pour la partie SDL seulement).

3 Description du projet et objectifs

3.1 Présentation générale du projet

Le but du projet est de générer un labyrinthe parfait, c'est-à-dire construire un programme qui va créer différents chemins sans qu'il y ait une partie de notre labyrinthe qui soit inaccessible (sauf pour la verison01 de SDL). Pour ce faire, nous avons créé des cases formant une grille, chaque extrémité (haut, bas, gauche et droite) de cette case est un mur. La case en haut à gauche de notre labyrinthe sera donc l'entrée et celle en bas à droite sera la sortie. Notre programme demandera deux valeurs à l'utilisateur, les dimensions:la hauteur et la largeur du labyrinthe. La génération consiste à supprimer des murs de cette grille aléatoirement, les murs restants seront affichés sous forme de traits. Une fois la génération terminée, il sera demandé à l'utilisateur s'il souhaite avoir la résolution du labyrinthe ou non. Si la réponse est positive, l'algorithme de résolution parcourra toutes les cases de notre labyrinthe; bloqué dans son chemin par des murs, il trouvera un chemin lui permettant de passer de la case entrée jusqu'à la sortie, le chemin à emprunter pour résoudre le labyrinthe sera donc connu et sera formé grâce à des étoiles qui s'afficheront sur les cases à gagner lors de l'avancée du chemin.

3.2 Présentation des algorithmes utilisés

Pour ce qui est de la génération du labyrinthe, voici la démarche: nous commençons avec une grille de cases numérotées avec des valeurs différentes, allant de 0 à (hauteur * largeur - 1). En utilisant la bibliothèque permettant de générer des nombres aléatoires (time.h), nous choisissons des murs aléatoirement à supprimer. Lorsqu'un mur est enlevé entre deux cases voisines, les deux cases prennent la même valeur. Ce processus est répété jusqu'à ce que toutes les cases de la grille aient la même valeur, ce qui indique qu'il existe un chemin reliant l'entrée à la sortie, et que chaque case est accessible. Ainsi, notre labyrinthe est considéré comme parfait.

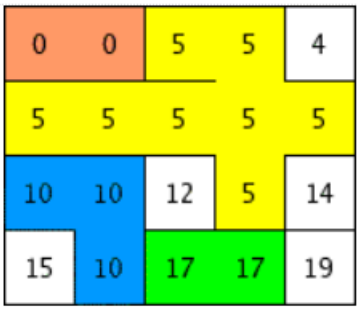

Figure 1: Génération labyrinthe

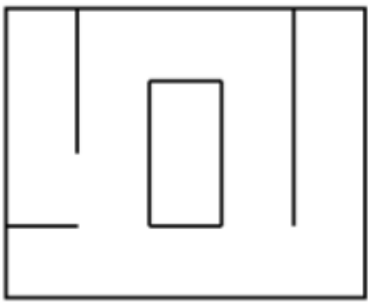

Figure 2: Labyrinthe imparfait

La résolution du labyrinthe est une étape plus complexe. Tout d'abord, nous initialisons toutes les cases à la valeur 0. En partant de la case de sortie, nous parcourons le labyrinthe en sens inverse, c'est-à-dire de la case de sortie jusqu'à la case d'entrée. Nous identifions les cases voisines qui ne sont pas séparées par des murs, et nous leur attribuons la valeur de la case actuelle plus 1. Par exemple, les cases voisines de la case de sortie prendront la valeur 1 (valeur de la case de sortie + 1). Bien entendu, les cases autres que la sortie, qui ont une valeur égale à 0, ne sont pas prises en compte afin d'éviter d'avoir des 1 partout et d'obtenir des valeurs erronées. Ce processus est répété jusqu'à ce que toutes les cases, à l'exception de la case de sortie, aient une valeur différente de 0. Ainsi, toutes les cases possèdent une valeur distincte de 0, sauf la sortie. Pour trouver la solution du labyrinthe, il suffit de partir de la case d'entrée et de suivre les valeurs des cases dans l'ordre décroissant. Ces valeurs indiquent le chemin à emprunter. Une fois la solution trouvée, nous réinitialisons les valeurs du chemin en les remettant à 0, et nous affichons une étoile (*) lorsque la valeur de la case est égale à 0. Ainsi, la résolution du labyrinthe est tracée et affichée avec clarté.

Figure 3: Valeur de résolution

4 Bibliothèques, Outils et technologies

Pour réaliser notre projet, nous avons utilisé plusieurs bibliothèques:

- stdio.h, stdlib.h, string.h: ce sont des bibliothèques standard. La bibliothèque stdio.h fournit des fonctions pour effectuer des opérations d'entrée et de sortie de fichiers, telles que l'ouverture et la fermeture de fichiers, la lecture et l'écriture de données, la gestion des erreurs, etc. Elle fournit également des fonctions pour effectuer des opérations de formatage et d'affichage de données, telles que printf et scanf. La bibliothèque stdlib.h fournit des fonctions pour allouer et libérer de la mémoire dynamiquement, convertir des chaînes de caractères en nombres et effectuer des opérations mathématiques de base, telles que rand et abs. La bibliothèque string.h fournit des fonctions pour effectuer des opérations sur les chaînes de caractères, telles que la copie, la comparaison, la concaténation et la recherche de sous-chaînes.

- stdbool.h : c'est une bibliothèque qui nous a permit d'utiliser des booléens car en C standard, il n'y a pas de type de données booléen natif.

- time.h: on a utilisé cette bibliothèque pour pouvoir genérer des nombres aléatoires.

- SDL2/SDL.h: lors de la deuxième partie du projet, on a eu recours à la bibliothèque SDL pour la création et gestion de la fenêtre, l'affichage du labyrinthe, la gestion des évènements et la libération de la mémoire allouée.

L'intelligence artificielle chatGPT a été un outil nous ayant aidé à la réalisation de ce projet pour la partie SDL.

5 Travail réalisé

Le projet consistait à développer un programme en C qui permet de générer un labyrinthe de manière aléatoire. Pour ce faire, il était nécessaire de prendre en compte plusieurs aspects. Tout d'abord, il fallait permettre à l'utilisateur de saisir les paramètres du labyrinthe, tels que sa taille et ses étapes et/ou ses valeurs. Cette étape nécessitait de mettre en place une interaction avec l'utilisateur, en utilisant un menu. Ensuite, le programme devait générer le labyrinthe. Une fois le labyrinthe généré, le programme devait afficher le labyrinthe à l'utilisateur. Pour cela, il était nécessaire d'utiliser une bibliothèque graphique telle que SDL, qui permet de créer des interfaces graphiques en C. Il fallait donc intégrer la bibliothèque SDL au programme et créer une fenêtre pour afficher le labyrinthe, une bibliothèque que l'on a essayé de manipuler tant bien que mal, mais nous ne sommes parvenus qu'a faire une génération de labyrinthe simple,

sans entrée ni sortie (verison01 et version02 du projet avec SDL). D'ailleurs, nous avons fait deux versions de code avec SDL car la première ne génère pas un labyrinthe parfait tandis que la deuxième si, mais elle contourne un peu les règles... Enfin, le programme devait également afficher la solution du labyrinthe, c'est-à-dire le chemin à emprunter pour sortir du labyrinthe.

Pour ce faire, nous avons distingué plusieurs fonctionnalités que nous avons pu développer:

- Proposer à l'utilisateur plusieurs types de labyrinthe à générer: un labyrinthe simple, un labyrinthe en affichant toutes ses étapes, un labyrinthe avec ses valeurs (valeurs des cases) et un labyrinthe en affichant ses étapes et ses valeurs.
- Demander à l'utilisateur de saisir les dimensions du labyrinthe : la hauteur et la largeur.
- Une fois le labyrinthe généré, il fallait proposer à l'utilisateur une solution de résolution.

La répartition du travail s'est faite de manière égale. Soit le travail était fait en classe ensemble, soit sur discord où on partageait nos écrans et on codait ensemble.

6 Difficultés rencontrées

Les plus grandes difficultés rencontrées ont étées les difficultées liées à la bibliothèque SDL. Dès l'installation de cette bibliothèque, nous avions eu du mal. De plus, la documentation SDL est un peu mal organisée.

D'autres difficultés ont étées rencontrées, la plupart du temps issues de petits bugs que nous réussissions à régler en proposant chacun des solutions que le professeur de TD confirmait ou complétait.

Une fois le programme terminé, nous avons réalisé que nous avions omis d'utiliser la fonction "free" pour libérer la mémoire allouée. Nous sommes conscients de l'importance de cette étape dans la gestion de la mémoire, mais étant donné que notre programme fonctionnait correctement, nous avons décidé de ne pas risquer d'introduire de nouveaux bugs en y apportant des modifications.

Nous avons également été confrontés à un problème de liaison des traits extérieurs, ce qui a entraîné des trous sur les bords du labyrinthe. Pour résoudre ce problème, nous avons fait appel à la bibliothèque SDL et au code ASCII. Le code ASCII nous a permis de remplir correctement les bords horizontaux, mais nous avons rencontré des difficultés pour remplir les bords verticaux. En utilisant la bibliothèque SDL, nous avons réussi à combler tous les bords du labyrinthe.

De plus, nous avons remarqué un décalage des murs vers la droite lorsque la valeur d'une case contenait un nombre à deux chiffres.

7 Bilan

7.1 Bilan personnel de Aya

Cette expérience m'a beaucoup appris en programmation et je me suis rendue compte que le langage C regorge de thématiques que je me tarde d'apprendre en plus approfondi à l'avenir. J'en ressors donc grandis de cette expérience.

7.2 Bilan personnel de Art

Personnellement, il s'agit de mon premier projet informatique. J'ai rencontré plusieurs problèmes de compréhension car certaines technologies que nous devions utilisé ne correspondaient pas à ce que nous étions habitués à voir en cours, en effet les connaissances requises était plus poussées. Il m'a fallu faire un important travail de mon côté, en plus de l'aide du professeur en TD et de Aya pour me permettre de me mettre au niveau pour ce projet.

Cela a été très enrichissant pour moi, car j'ai pu développer un certain début de logique technique grâce aux différentes tâches que j'ai dû accomplir. Maintenant, je dois approfondir cette base durant les années à venir pour être de plus en plus performant et pouvoir atteindre les ambitions que je me suis fixé.

7.3 Conclusion

En conclusion, ce projet a été une expérience très enrichissante sur différents plans:

- D'un point de vue informatique, il nous a permis de mettre en pratique les connaissances acquises en cours et de nous familiariser avec l'application concrète de la programmation.
- D'un point de vue personnel, ce projet a été riche en émotions, car la résolution des différents bugs rencontrés demandait beaucoup de temps et de patience. Parfois, la frustration prenait le dessus et des conflits pouvaient survenir, mais nous avons rapidement appris à travailler ensemble pour les résoudre. Cette expérience nous a permis de développer notre capacité à travailler en équipe dans des situations difficiles, et de mieux comprendre l'importance de la communication et de la collaboration pour mener à bien un projet complexe.

7.4 Perspectives

Ce projet a été d'une grande utilité pour notre projet professionnel, car nous sommes tous les deux en quête d'une licence MIAGE en alternance et devenir respectivement développeur (Art) et consultante en informatique (Aya). En tant que futurs professionnels de l'informatique, nous savons que les développeurs doivent être rigoureux pour mener à bien leurs projets, tandis que les consultants

doivent avoir de la technique et une bonne communication. Ce projet nous a permis de nous familiariser avec les exigences de rigueur, de bonne communication et de patience que peuvent rencontrer les professionnels de l'informatique. En effet, nous avons dû faire preuve de patience et de persévérance pour résoudre les différents problèmes que nous avons rencontrés. Grâce à ce projet, nous avons pu acquérir une expérience précieuse et nous préparer aux exigences du métier. Nous avons également développé nos compétences techniques et organisationnelles en manipulant des structures, en programmant des algorithmes de génération de labyrinthe et en mettant en place une interface utilisateur. En somme, ce projet a été une occasion unique de nous mettre en situation professionnelle et de nous perfectionner dans notre futur métier.

8 Webographie

Pour les algorithmes de génération de labyrinthes et les labyrinthes parfaits:

https://fr.wikipedia.org/wiki/Mod%C3%A9lisation_math%C3%A9matique_d%27un_labyrinthe

http://www.mathsoup.xyz/mathsoup.xyz/content/Informatique/Fiche%20d'activit%C3%A9%204%20-%20g%C3%A9n%C3%A9ration-labyrinthe/g%C3%A9n%C3%A9ration-labyrinthes%20-%20%C3%A9l%C3%A8ves.html

Pour l'algorithme de résolution :

http://www.mathsoup.xyz/mathsoup.xyz/content/Informatique/Fiche%20d'activit%C3%A9%205%20-%20plus-court-chemin/plus-court-chemin-eleves.html

http://www.mathsoup.xyz/mathsoup.xyz/content/Informatique/Fiche%20d'activit%C3%A9%205%20-%20plus-court-chemin/plus-court-chemin-eleves.html

Pour les nombres aléatoires :

https://www.delftstack.com/fr/howto/c/c-generate-random-number/

Pour le code ASCII :

http://www.gecif.net/qcm/information/ascii_decimal_hexa.pdf

Pour la SDL :

https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/

9 Annexe

9.1 Annexe B : Exemple d'exécution du projet

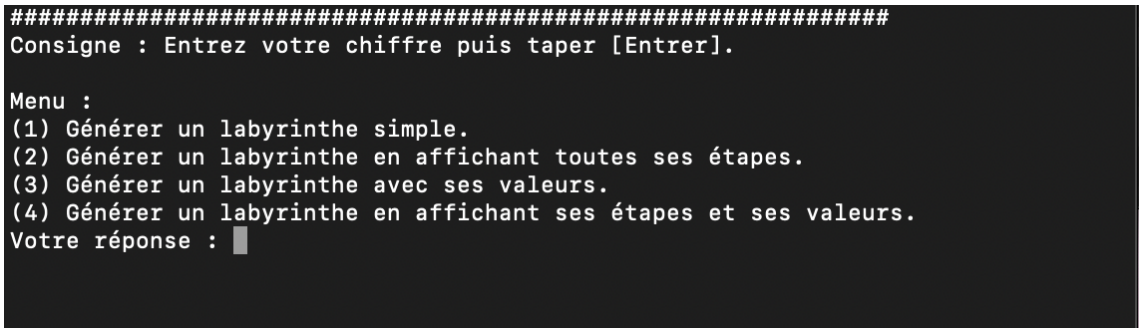

Figure 4: première étape

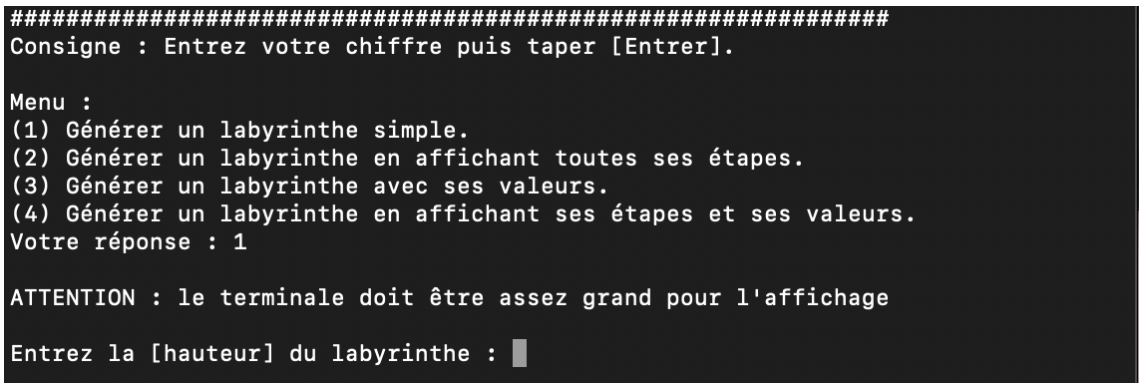

Figure 5: deuxième étape

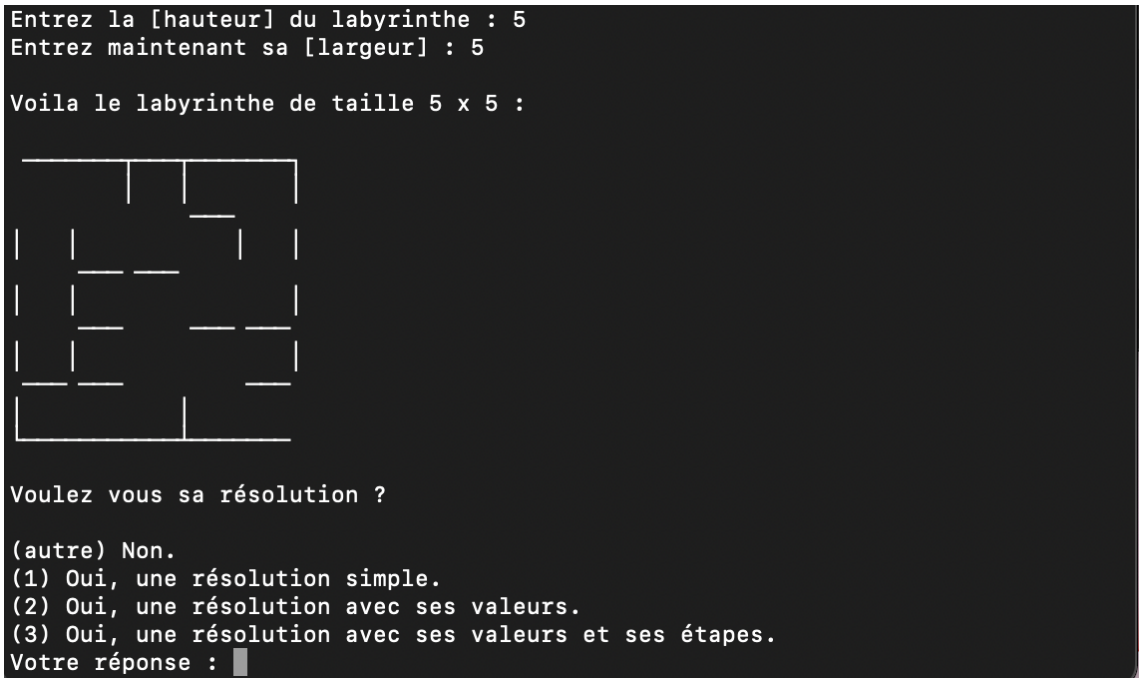

Figure 6: troisième étape

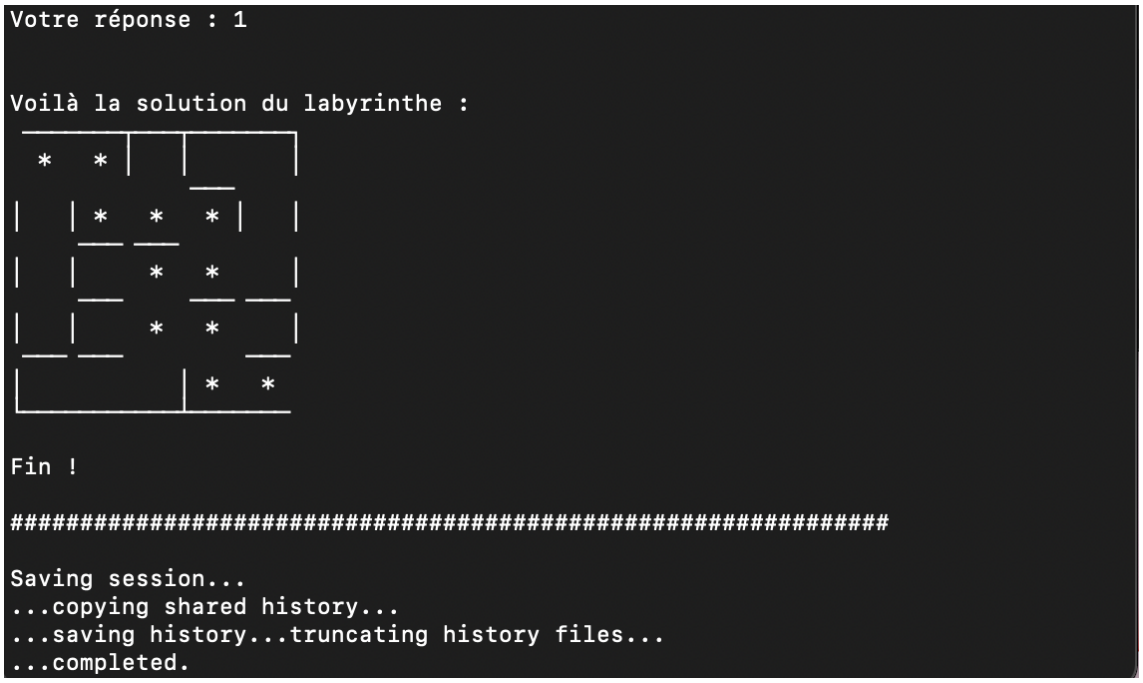

Figure 7: dernière étape

9.2 Annexe C : Manuel utilisateur

Premièrement il vous sera demandé de choisir l'affichage que vous souhaitez parmi ceux proposés ci-dessous :

- 1. Générer un labyrinthe simple.
- 2. Générer un labyrinthe en affichant toutes ses étapes de génération.
- 3. Générer un labyrinthe avec ses valeurs.
- 4. Générer un labyrinthe en affichant ses étapes et ses valeurs.

Si le nombre saisi ne correspond à aucune des options proposées, le programme se relancera automatiquement et vous devrez fournir une nouvelle proposition.

Ensuite, le labyrinthe nécessitera deux entiers qui serviront de dimensions. Veuillez les saisir un par un selon les indications du programme. Si l'un des deux entiers est inférieur ou égal à un, le compilateur signalera potentiellement un bug. Une fois les entiers saisis, le labyrinthe sera généré et affiché.

Enfin, vous serez invité à décider si vous souhaitez afficher la résolution de ce labyrinthe, et à sélectionner l'affichage que vous voudrez parmi les options suivantes :

- 1. Oui, une résolution simple.
- 2. Oui, une résolution avec ses valeurs.
- 3. Oui, une résolution avec ses valeurs et ses étapes.
- (autre) Non.

Si vous entrez un nombre différent des 3 propositions valides, aucune résolution ne sera affichée. Dans ce cas, la solution du labyrinthe sera calculée et affichée selon l'option que vous avez sélectionnée. Cela marque la fin du programme.

ATTENTION : Les valeurs à deux chiffres entraînent des décalages lors de l'affichage des murs.

ATTENTION : Lorsque les dimensions entrées sont trop grandes, il est possible que l'affichage ne soit pas correct en raison d'un manque d'espace ou que la génération prenne beaucoup de temps.