Programmation structurée - Projet

Génération aléatoire de labyrinthes

Kevin Doolaeghe
God-Belange Aradukunda

Sommaire :

1. Présentation du sujet

Le but du projet proposé est de réaliser un programme qui permet de générer et de résoudre des labyrinthes aléatoires représentés sous forme de matrices à deux dimensions.

Le programme doit proposer les fonctionnalités suivantes :

- Possibilité de générer des labyrinthes de taille arbitraire (N x M où N et M ne sont pas fixés)
- Afficher dans une console le labyrinthe créé (l'affichage ne sera pertinent que pour des labyrinthes de taille relativement petite i.e. inférieur à 25 x 80)
- Possibilité de stocker un labyrinthe créé dans un fichier (via une redirection)
- Possibilité de charger un fichier contenant un labyrinthe (via une redirection)
- Résoudre le labyrinthe chargé (calculer le chemin à suivre)
- Afficher ce chemin sur le labyrinthe
- Tout se fera en console avec une aide pour guider l'utilisateur
- Quelques labyrinthes "exemple" générés par votre programme seront fournis
- Un fichier README.md sera présent pour expliquer comment compiler et utiliser le programme

2. Etude du projet

Après étude du cahier des charges, il nous faut coder en langage C (le programme doit fonctionner sur l'environnement Linux Debian), un programme permettant de générer et résoudre des labyrinthes représentés sous forme de matrices à deux dimensions.

Pour l'affichage, il a été décidé de créer un menu interactif et facile d'utilisation. En effet, les flèches du clavier pourraient permettre la navigation dans un menu et la touche entrée, le déclenchement d'un événement.

Pour la génération et la résolution de labyrinthe, différents algorithmes trouvés sur Internet et disponibles en annexe seront retranscrits en C.

Le code entier du programme se trouverait dans un unique fichier. Chaque opération formerait une fonction à appeler. Cela permettrait d'améliorer la lisibilité et la modification/mise à jour du code. Voici la liste des fonctions utiles dans notre programme :

- `void initRand()` : Initialise la génération de nombre aléatoire selon le temps courant
- `void setDimensions()` : Les variables m et n sont initialisées aléatoirement
- `void createMaze(unsigned int m, unsigned int n)` : Le labyrinthe est alloué dynamiquement en mémoire à partir de m et n
- `void destroyMaze(unsigned int m)` : Le labyrinthe est détruit de la mémoire
- `void createSolution(unsigned int m, unsigned int n)` : La solution du labyrinthe est allouée dynamiquement en mémoire à partir de m et n
- `void destroySolution(unsigned int m)` : La solution du labyrinthe est détruite de la mémoire
- `void createPath(bool **maze, unsigned int m, unsigned int n, unsigned int x, unsigned int y)` : Un chemin est créé dans le labyrinthe
- `void generateMaze()` : Initialise le labyrinthe et lance la génération de chemin
- `bool fillSolution(bool **maze, bool **solution, unsigned int m, unsigned int n, unsigned int x, unsigned int y)` : Cherche la solution du labyrinthe et rempli la variable solution
- `void solveMaze()` : Initialise la solution et lance la recherche de solution

- `void clearTerminal()` : Efface le terminal
- `void resizeTerminal(unsigned int lines, unsigned int columns)` : Redimensionne le terminal
- `void moveCursor(unsigned int x, unsigned int y)` : Déplace le curseur à l'écran
- `void setColor(unsigned int charColor, unsigned int backgroundColor)` : Change la couleur de caractère et du fond
- `int getch()` : Saisie une touche au clavier sans afficher la saisie et sans appuyer sur la touche entrée
- `void displayBorders(unsigned int longueur, unsigned int x, unsigned int y)` : Affiche un encadré aux coordonnées indiquées
- `void displayTitle()` : Affiche le titre
- `void displayMenuOption(unsigned int x, unsigned int y, char* text)` : Affiche une option du menu
- `void displayMenu()` : Affiche le menu
- `void displayMenuSelectionCursor(unsigned int selectedOption)` : Affiche le curseur courant du menu
- `void displayMaze()` : Affiche le labyrinthe
- `void displaySolution()` : Affiche la solution du labyrinthe
- `void saveMaze()` : Sauvegarde le labyrinthe dans un fichier
- `void loadMaze()` : Charge un labyrinthe à partir d'un fichier
- `void handleEvents(unsigned int selectedOption)` : Gère les évènements selon la sélection
- `void loop()` : Boucle du programme
- `void run()` : Lance le programme
- `int main()` : Point d'entrée du programme

De plus, le code a été transféré sur Github pour que les membres du groupe puissent travailler en même temps sur des parties du programme et que chacun puisse accéder au travail des autres.

3. Organisation

Après concertation, le travail est réparti comme présenté ci-dessous.

Kevin	God-Belange
• Interface utilisateur • Sauvegarde et chargement d'un labyrinthe • Structure du code	• Génération du labyrinthe • Résolution du labyrinthe

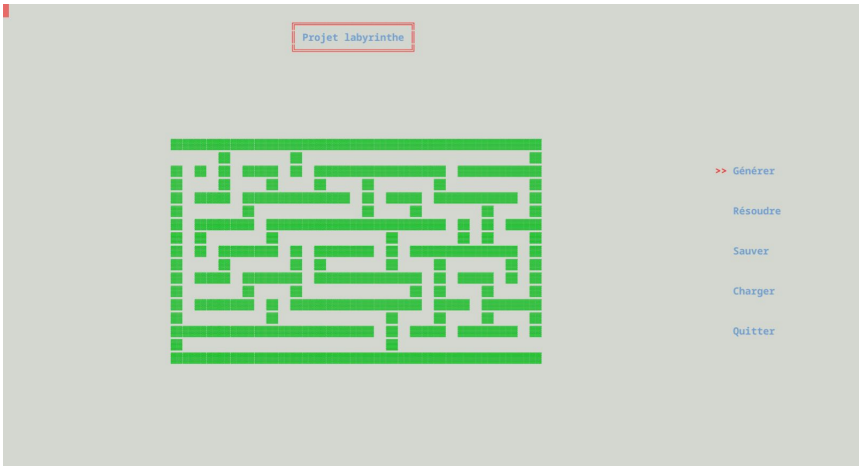
4. Mise en oeuvre

a. Interface Homme-Machine

Nous avons voulu que l'interface puisse afficher un titre, un labyrinthe et un menu de navigation.

Au lancement du programme, une boucle d'affichage tourne tant que l'on ne décide pas de quitter le programme. Dans cette boucle on affiche ce qui est à afficher (labyrinthe et solution si créés, titre et menu), puis on entre dans une boucle qui sert à la saisie et enfin, suite à cette boucle, on effectue les opérations selon la sélection du menu. La boucle de saisie va afficher le curseur selon la sélection et se termine lorsqu'on appuie sur la touche entrée.

L'interface propose cinq options. Il permet en effet de générer un nouveau labyrinthe (à partir des paramètres m et n générés aléatoirement), résoudre et sauvegarder le labyrinthe qui est en mémoire, charger un labyrinthe dans un fichier en mémoire et quitter le programme. La navigation s'effectue à l'aide des touches directionnelles du clavier et de la touche entrée.

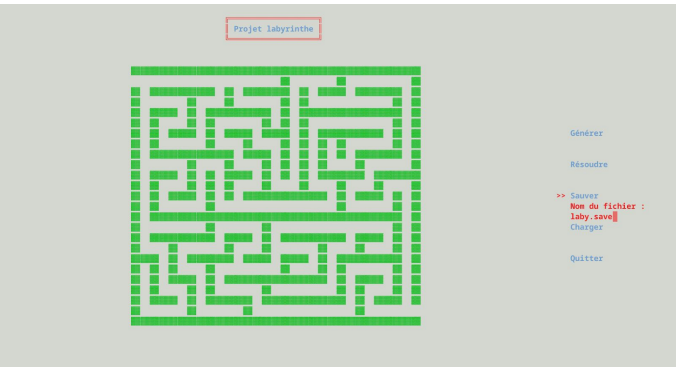


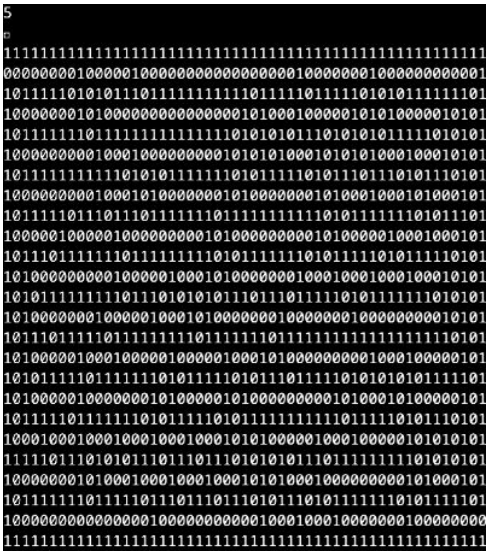
b. Sauvegarde et chargement d'un labyrinthe

Le programme propose la sauvegarde et le chargement d'un labyrinthe au moyen d'un fichier.

La sauvegarde crée un fichier (dont le nom est saisi par l'utilisateur) dans le répertoire où se situe l'utilisateur au lancement du programme et y copie la largeur m, la hauteur n du labyrinthe puis le labyrinthe.

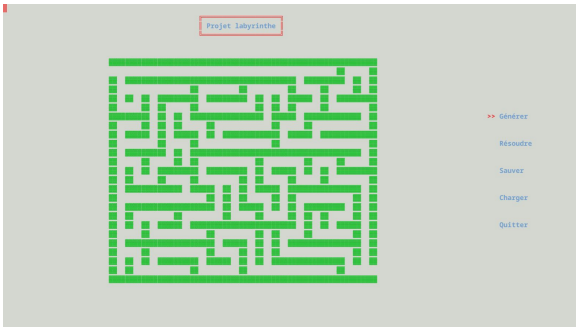
Le chargement récupère les informations du fichier (le nom est saisi par l'utilisateur) afin de reformer le labyrinthe en mémoire. On récupère m et n pour ensuite alloué dynamiquement la variable correspondant au labyrinthe puis on rempli le labyrinthe à l'aide des valeurs lues.

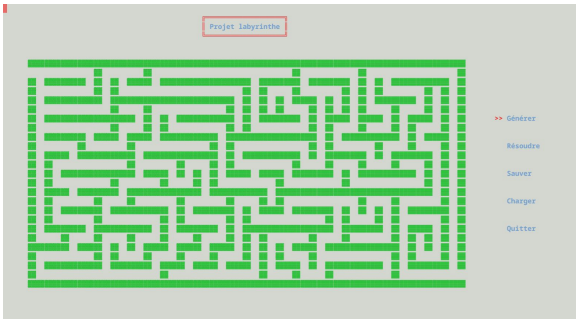




c. Génération d'un labyrinthe

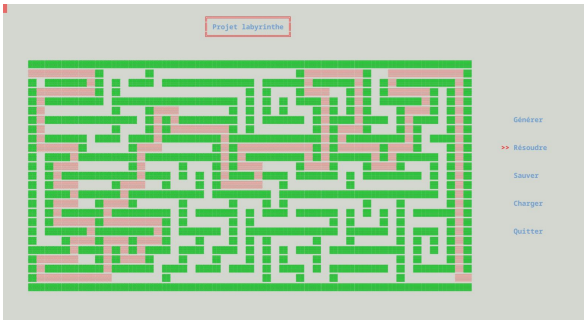
Pour la génération du labyrinthe, on a créé un algorithme après étude du fonctionnement d'un labyrinthe "parfait". L'algorithme que nous avons utilisé se base sur une suite de directions déterminées aléatoirement. Selon la direction, on vérifie que l'on ne soit pas hors du labyrinthe et qu'il y ait un mur. Dans ce cas, on "brise" le mur pour tracer le chemin. Sinon, on s'arrête. Cet algorithme est appliqué pour chaque cellule aux coordonnées impaires du labyrinthe. On obtient ainsi par exemple, les résultats ci-dessous :

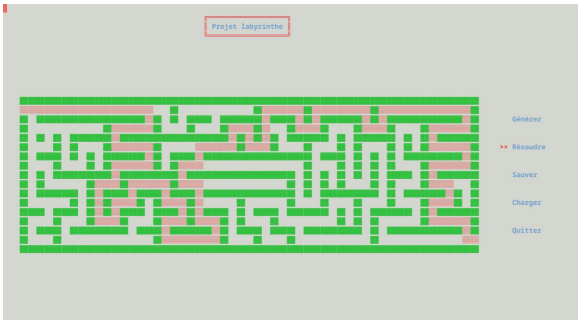




d. Résolution d'un labyrinthe

Il existe différentes méthodes pour résoudre un labyrinthe dont celle de la "main gauche". Nous avons décidé d'utiliser un algorithme récursif qui va emprunter tous les chemins possibles jusqu'à être bloqué. Le premier chemin menant à la sortie est celui qui sera la solution. Il ne s'agit donc pas nécessairement du chemin le plus rapide.





5. Conclusion

L'état final du projet respecte tous les points du cahier des charges :

- Possibilité de générer des labyrinthes de taille arbitraire (N x M où N et M ne sont pas fixés)
- Afficher dans une console le labyrinthe créé (l'affichage ne sera pertinent que pour des labyrinthes de taille relativement petite i.e. inférieur à 25 x 80)
- Possibilité de stocker un labyrinthe créé dans un fichier (via une redirection)
- Possibilité de charger un fichier contenant un labyrinthe (via une redirection)
- Résoudre le labyrinthe chargé (calculer le chemin à suivre)
- Afficher ce chemin sur le labyrinthe
- Tout se fera en console avec une aide pour guider l'utilisateur
- Quelques labyrinthes "exemple" générés par votre programme seront fournis
- Un fichier README.md sera présent pour expliquer comment compiler et utiliser le programme

Cependant, certaines des fonctionnalité ci-dessus peuvent être améliorées davantage. La génération du labyrinthe peut en effet être améliorée en complexifiant encore l'algorithme. De même, le code peut nécessairement être optimisé.

Nous avons essayé de supprimer un maximum de variables globales mais nous avons eu des problèmes avec le passage des doubles pointeurs représentant le labyrinthe et la solution en variables locales. En effet, lorsqu'il a fallu les ajouter en paramètre aux différentes méthodes qui les utilisent, le programme s'arrêtait à cause de débordements mémoires.

6. Annexes

Les sources ci-dessous ont été utilisées pour la génération et la résolution d'un labyrinthe :

- https://ilay.org/yann/articles/maze/
- http://www.encyclopedie-incomplete.com/?Modelisation-et-Creation-d-un
- https://github.com/joewing/maze/blob/master/maze.c