COMPTE RENDU DE PROJET

Jeu de Labyrinthe

Année : 1ère Année
Module : Introduction à la programmation en C

Encadrant : DUCASTEL Mathéo

Étudiant : QACH Yahya

Table des matières

1 Introduction

Ce projet s'agit d'un jeu de labyrinthe avec un générateur de labyrinthes parfaits et un moteur de jeu (déplacements, calcul de score, lecture/ecriture de fichiers, enregistrement de labyrinthes/scores, etc ...).

1.1 Contexte

Ce projet est développé dans le cadre du module d'introduction à la programmation en C en première année cycle ingénieur à l'ensicaen et vise à appliquer les concepts vus en cours et en TD.
La construction de ce projet est faite en suivant des étapes correspondants chacune à une difficulté et un aspect différent des notions vues en cours/TD.

1.2 Structure

Ce rapport se divise en deux parties principales :
— Manuel pour joueur
— Manuel pour le développeur qui reprendra le code, où vous trouverez :

Dans la partie manuel pour joueur vous trouverez tous ce que vous avez besoin pour construire le programme du jeu, lancer le jeu, et comprendre le fonctionnement du jeu.

Dans la partie manuel pour développeur, vous trouverez tous ce que vous avez besoin pour comprendre le fonctionnement du jeu d'un point de vue technique.
Vous y trouverez l'explication de :
— Ce qui a été réalisé
— Ce qui reste à faire
— Les points clés et leurs implémentation et fonctionnement
— Les limitations et les beugs connus

2 Manuel Joueur

2.1 Instructions de compilation et execution

2.1.1 Compilation

Pour compiler le projet, placez-vous à la racine et exécutez la commande suivante dans le terminal :

```
make
```

2.1.2 Exécution du programme

Pour lancer le jeu après compilation, placez vous dans la racine et executez la commande suivante dans votre terminal :

```
./labyrinthe_game
```

2.1.3 Nettoyage

Pour nettoyer le projet (supprimer les fichiers objet et les binaires) :

```
make clean
```

2.1.4 Lancement des tests

Pour compiler et exécuter les tests unitaires placez vous dans la racine et executez la commande suivante dans votre terminal :

```
make test
```

2.1.5 Génération de la documentation

Pour créer la documentation Doxygen :

```
make doc
```

2.1.6 Affichage de la doc

Pour afficher la documentation dans une nouvelle fenêtre :

```
make doc-view
```

2.1.7 Nettoyage de la doc

Pour nettoyer la documentation :

```
make doc-clean
```

2.1.8 Cibles

Pour afficher toutes les cibles possibles avec le fichier makefile disponible executer cette commande dans la racine du projet :

```
make help
```

2.2 Fonctionnement du jeu

Ce projet s'agit d'un jeu de labyrinthe où le joueur (vous, si jamais vous décidez de jouer) se déplace dans un labyrinthe sous forme de grille en cherchant la clé et ensuite atteindre à la sortie avec but de faire le moins de mouvements possibles.

2.2.1 Menu

Au lancement du jeu, l'utilisateur à un choix à effectuer selon l'action qu'il souhaite faire.

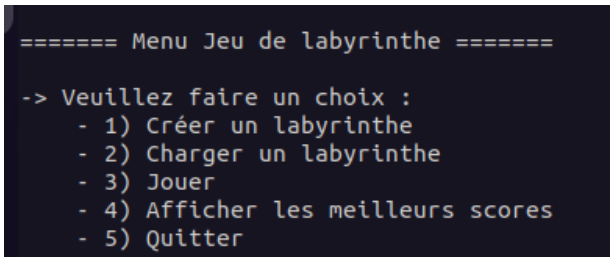

FIGURE 1 – Menu du jeu

Si le joueur a choisi de créer un labyrinthe, il est demandé de choisir :
— Les dimensions du labyrinthe qu'il souhaite créer
— Le nom qu'il souhaite donner au labyrinthe
— La difficulté qu'il souhaite attribuer au labyrinthe (0 : facile, 1 : Difficile)

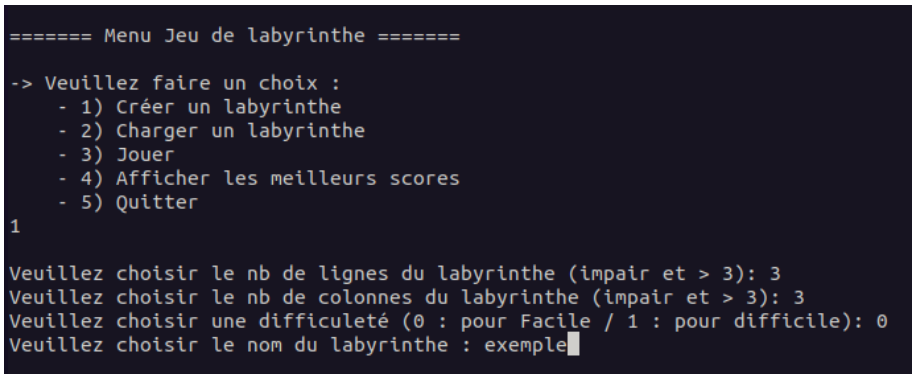

FIGURE 2 – Menu du jeu creer

Sinon si le joueur a choisi de charger un labyrinthe, il est demandé de fournir le nom du labyrinthe qu'il souhaite jouer parmi les labyrinthes enregistrés.

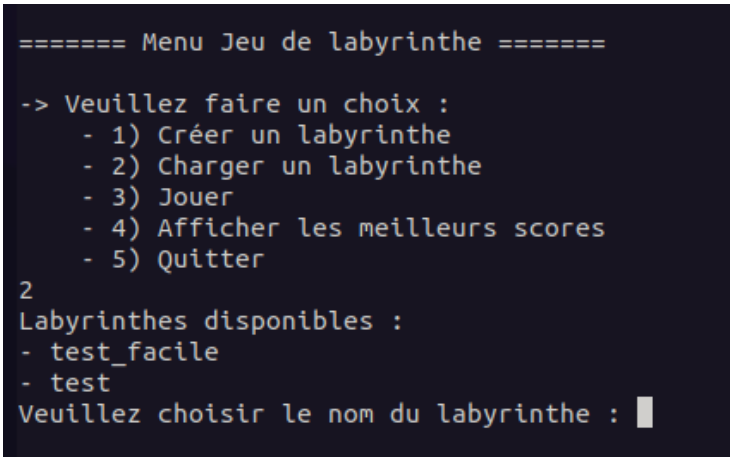

FIGURE 3 – Menu du jeu charger

Sinon si le joueur a choisi d'afficher les meilleurs scores il est demandé defournir le nom du labyrinthe qu'il souhaite jouer parmi les labyrinthes enregistrés.

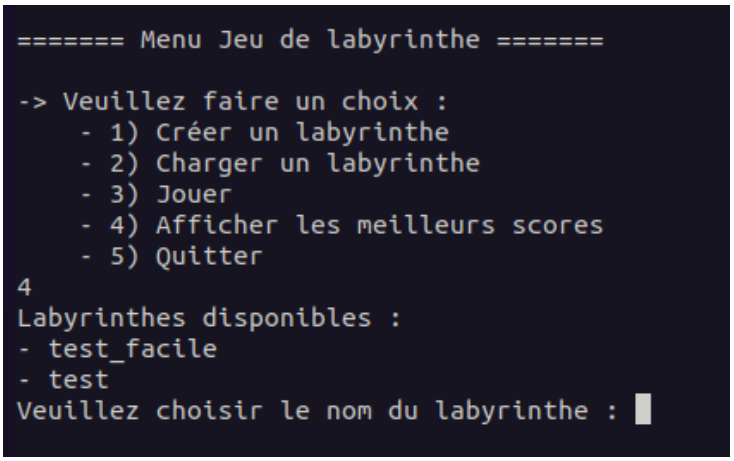

FIGURE 4 – Menu du jeu scores

2.2.2 Jouer

Si le joueur a chois de jouer un labyrinthe avec un labyrinthe chargé en mémoire :

Le joueur se trouve toujours au début du jeu, à la **1ere case en haut à gauche** , et doit atteindre la sortie qui se trouve toujours à **la dernière case en bas à droite.**

Pour pouvoir sortir du labyrinthe, le joueur doit d'abord récupérer la clé qui se trouve quelque part aléatoirement dans le labyrinthe. **Le score** est simplement le nombre de pas que le joueur a effectué pour finir le jeu (récupérer la clé et trouver la sortie), plus le score est petit, meilleur est le joueur.
à chaque tour du jeu, le programme vous indique votre score et si vous avez la clé.

Le labyrinthe possède plusieurs trésors et pièges qui, chacun, accordent des bonus et des malus au joueur, ce qui affecte son score.

- Un trésor enlève un certain nombre de pas de votre score (par défaut bonus=3)
- Un piège ajoute un certain nombre de pas à votre score (par défaut, malus=1)

Les différents éléments du labyrinthe sont représentés par différents caractères :

- Le joueur est représenté par un 'o' bleu : o
- La sortie est représentée par trois tirets en vert : —
- Les murs sont représentés par : [#]
- Les trésors sont représentés par une signe plus : +
- Les pièges sont représentés par le caractère étoile : *
- La clé est représentée par un 'k' doré : k

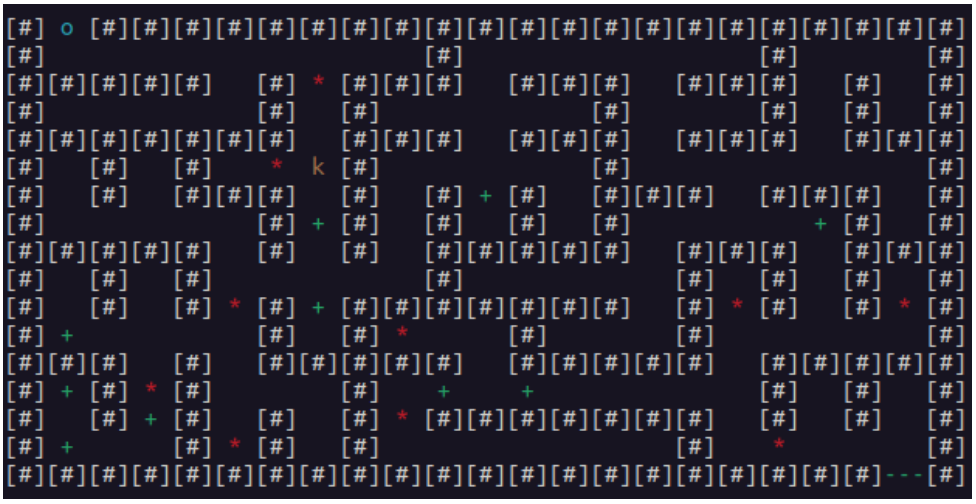

FIGURE 5 – exemple de labyrinthe facile

Si le labyrinthe choisi est un labyrinthe Difficile, il y'a quelques différences avec le labyrinthe facile

Le labyrinthe difficile se distingue par :
- Le labyrinthe dispose maintenant de plusieurs chemins possibles
- Différentes sortes de monstres jalonnent le parcours

Les monstres sont représentés comme suit :
- Les ogres sont représentés avec un signe dollar magenta : -$-
- Les fantômes sont représentés avec un signe mu jaune : -µ-

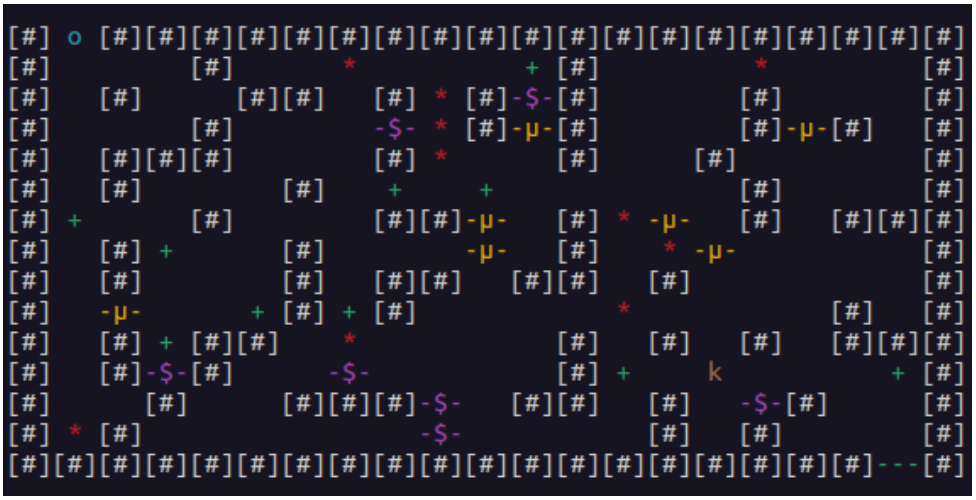

FIGURE 6 – exemple de labyrinthe difficile

à chaque tour de jeu le joueur peut se déplacer en appuyant sur une touche parmi celles ci-dessous suivies par **la touche entrée :**
- La touche **z : haut**
- La touche **s : bas**
- La touche **q : gauche**
- La touche **d : droite**

à la fin du jeu, si le score du joueur est parmi le top 10, le programme lui demande d'ecrir son pseudo pour enregistrer son score dans le fichier score correspondant au labyrinthe.

3 Manuel Développeur

3.1 Objectifs

L'objectif principal de ce projet est la réalisation d'un générateur de labyrinthe parfait et de son utilisation dans un jeu interactif.
Le projet doit impérativement avoir les fonctionnalités suivantes à chaque étape :

- **Étape 1 :** Un générateur de labyrinthe parfait de taille fixe 11×25 en utilisant l'algorithme de fusion aléatoire de chemins, avec affichage en mode console
- **Étape 2 :** Un système de menu interactif permettant la création, sauvegarde (.cfg), chargement et jeu sur des labyrinthes de taille variable
- **Étape 3 :** Système de jeu avancé avec objets (clé, trésors, pièges), système de score basé sur la rapidité et les bonus/malus, et classement des 10 meilleurs scores (.score)
- **Étape 4 :** Niveaux de difficulté avec labyrinthes imparfaits (mode difficile) et système de monstres mobiles utilisant des pointeurs de fonctions

3.2 Fonctionnalitées implémentées

Vous pouvez lire la socumentListe exhaustive des fonctionnalitées implémentée :

3.2.1 Étape 1 : Génération et affichage de base

Génération de labyrinthe parfait

- Création de la grille de jeu avec dimensions configurables
- Implémentation de l'algorithme de fusion aléatoire de chemins
- Génération de labyrinthes parfaits (un seul chemin entre deux points)
- Garantie de connectivité entre toutes les cellules du labyrinthe

Interface d'affichage console

- Affichage complet du labyrinthe en mode console
- Représentation visuelle des murs, couloirs, entrée et sortie, joueur, trésor, pièges, monstres
- Affichage de la position du joueur en temps réel

3.2.2 Étape 2 : Interface utilisateur et persistance

Système de menu interactif

- Menu principal s'affichant au démarrage du programme
- Gestion et nettoyage des entrées utilisateur pour la sélection d'actions
- Retour automatique au menu après chaque action terminée

Création de labyrinthe

- Saisie utilisateur de la taille du labyrinthe (hauteur et largeur)
- Attribution d'un nom au labyrinthe
- Génération automatique du labyrinthe selon les paramètres choisis
- Enregistrement automatique au format .cfg
- Création automatique du fichier .score qui maintient les scores des joueurs sur ce labyrinthe

— Chargement immédiat en mémoire après création

Gestion des fichiers et chargement
— Sauvegarde des labyrinthes au format .cfg lors création
— Chargement de labyrinthes existants depuis fichiers .cfg
— Fonctionnalité bonus : affichage de la liste des labyrinthes disponibles
— Lecture complète et reconstruction du labyrinthe en mémoire
— enregistrement de score après jeu dans le fichier .score
— affichage des 10 meilleurs scores d'un labyrinthe depuis son nom

Système de jeu de base
— Lancement de parties avec un labyrinthe préalablement chargé
— Contrôles de déplacement : z (haut), q (gauche), s (bas), d (droite) + Entrée
— Détection des collisions avec les murs
— Vérification automatique de la condition de fin de jeu
— Détection de l'arrivée du joueur à la sortie du labyrinthe
— interaction avec monstres/pièges/trésors/clé

3.2.3 Étape 3 : Objets, scores et classements

Système d'objets spéciaux
— Définition de constantes pour les objets : clé (CLE), trésor (TRESOR), piège (PIEGE)
— Configuration des valeurs de bonus (X) et malus (Y) paramétrisables
— Placement aléatoire d'une clé unique dans le labyrinthe lors de la création
— Placement aléatoire de plusieurs trésors (quantité configurable) lors de la création
— Placement aléatoire de plusieurs pièges (quantité configurable) lors de la création

Mécanisme de verrouillage de la sortie
— Attribut "a_clé" pour le joueur
— Autorisation de sortie uniquement après récupération de la clé

Système de score avancé
— Variable de score intégrée à la structure de joueur
— Calcul du score basé sur le nombre de déplacements (objectif : minimiser)
— Modification du score lors de la collecte de bonus (réduction de X points)
— Modification du score lors de l'activation de pièges (ajout de Y points)
— Affichage du score en temps réel après chaque déplacement
— Affichage du score final à la fin de chaque partie

Interaction avec les objets
— Collecte automatique de la clé lors du passage du joueur
— Mise à jour de l'état "a_clé" et suppression de la clé du labyrinthe
— Collecte automatique des trésors avec ajout des points bonus
— Suppression des trésors collectés du labyrinthe
— Activation automatique des pièges avec déduction des points
— Suppression des pièges activés du labyrinthe

Système de classement (Top 10)
- Création automatique de fichiers .score pour chaque labyrinthe
- Lecture et écriture des meilleurs scores depuis/vers les fichiers
- Vérification de l'éligibilité du score du joueur pour le top 10
- Saisie du nom du joueur pour les scores qualifiants
- Insertion automatique du score à la position correcte dans le classement
- Maintien d'un classement de 10 meilleurs scores maximum

Interface d'affichage des scores
- Nouvelle option de menu "Afficher le classement des meilleurs scores"
- Affichage formaté du top 10 pour le labyrinthe actuellement chargé
- Possibilité d'afficher le classement pour un labyrinthe spécifique
- Gestion de l'affichage lorsqu'aucun score n'existe

3.2.4 Etape 4 : Monstres et difficulté

Difficulté
- Nouvelle option, choix de difficulté lors de la création du labyrinthe
- Enregistrement du niveau de difficulté lors de l'enregistrement du labyrinthe dans le fichier .cfg
- Suppréssion des murs additionnels si le joueur a choisi Difficile

Monstres
- Placement d'un nombre paramétrable de monstres (Ogres,Fantômes) dans le labyrinthe lors de la création
- Gestion de l'intéraction du joueur avec les monstres

3.2.5 Fonctionnalités transversales

Robustesse et gestion d'erreurs
- Gestion des cas particuliers (pointeurs NULLs, scores inexistants)
- Vérification de l'existence des fichiers avant chargement
- Validation des entrées utilisateur (valeurs invalides, fichiers manquants)
- Gestion de la mémoire avec libération complète des blocs alloués
- Prévention des fuites mémoire dans tous les scénarios d'utilisation

Tests et validation
- Vérification du placement correct de tous les objets
- Tests du mécanisme de verrouillage/déverrouillage de la sortie
- Validation de la gestion du score dans tous les cas d'usage
- Tests complets du système de classement et top 10
- Vérification de la robustesse face aux entrées invalides
- Tests de gestion mémoire et détection des fuites

3.3 Fonctionnalitées pas encore implémentées

- Déplacement des Monstres
- Paramètre de mobilité : mobilité définie suivant une règle régie par une fonction.

— Paramètre étendue de mobilité : territoire de mobilité calculé selon le nombre de pénalité
— Déplacement des monstres à chaque déplacement joueur
— pointeurs vers des fonctions pour les déplacements

3.4 Conception et Implémentation

Vous pouvez visualiser la documentation en executant la commande suivante depuis la raçine du projet :

```
make doc-view
```

3.4.1 Architecture générale

Le projet est organisé selon une architecture modulaire :
- **headers/** : Fichiers d'en-tête (.h) définissant les signatures des fonctions, les différentes structures utilisées dans le programme et les différentes constantes.
- **src/** : Fichiers source (.c) contenant l'implémentation
- **tests/** : Tests unitaires pour valider les fonctionnalités
- **ressources/** : Données du jeu (labyrinthes et scores sauvegardés)
- **minunit/** : Bibliothèque de tests unitaires légère
- **bin/** : Rassemble les différents fichiers objets issus de la compilation

Chaque fonction de chaque fichier source est responsable d'une seule tâche pour assurer la modularité et la lisibilité et la maintenance du code éventuelle.

La grille du labyrinthe est définie comme une grille de valeurs entières, ces valeurs correspondent à des identifiants (objets,cellules,sortie, etc..).
Les valeurs négatives correspondent aux objets (joueur : 0, sortie : -2, mur : -1, etc ...) et Les valeurs positives représentent des cellules vides.

Avec ce choix de design il est beaucoup plus simple de manipuler la grille comme une grille de valeurs entières qu'une grille de caractères que ça soit lors des déplacement où lors de la création et initialisation, l'affichage en caractère est assuré par la fonction **afficherlabyrinthe()** qui attribue à chaque identifiant son caractère.

La génération de labyrinthe est un problème algorithmique classique qui nécessite de créer un graphe connexe sans cycles. Le défi principal consiste à :
- Garantir qu'il existe un chemin unique entre chaque paire de cellules
- Éviter la création de boucles dans le labyrinthe
- Assurer une distribution aléatoire des passages
- Optimiser la performance pour des labyrinthes de grande taille

La fonction qui gère cette résponsabilité est la fonction **init_maze()** avec l'aide de quelques fonctions utilitaires.

3.4.2 Fonctionnement détaillé de la fonction init_maze()

La fonction `init_maze()` implémente l'algorithme de fusion aléatoire de chemins pour générer des labyrinthes parfaits. Voici son fonctionnement étape par étape :

Étape 1 : Initialisation de la grille La grille est initialisée avec la valeur -1 pour toutes les cases, ce qui correspond aux murs. Ensuite, on place une valeur unique sur une cellule sur deux, générant ainsi un ensemble de cellules séparées par des murs.

exemple sur une grille 5x5

```
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Devient :

```
# # # # #
# 1 # 2 #
# # # # #
# 3 # 4 #
# # # # #
```

Chaque chiffre représente une cellule avec son identifiant unique, les symboles # représentent les murs.

Étape 2 : Création de la liste des murs internes L'algorithme génère une liste complète de tous les murs potentiels entre les cellules adjacentes :

— Les murs horizontaux entre chaque cellule et celle de droite
— Les murs verticaux entre chaque cellule et celle du bas

Dans l'exemple ci-dessus, on identifie :

— Mur entre (1,1) et (1,3) - horizontal entre cellules 1 et 2
— Mur entre (1,1) et (3,1) - vertical entre cellules 1 et 3
— Mur entre (1,3) et (3,3) - vertical entre cellules 2 et 4
— Mur entre (3,1) et (3,3) - horizontal entre cellules 3 et 4

Étape 3 : Mélange aléatoire La liste des murs est mélangée aléatoirement pour garantir que chaque génération produit un labyrinthe différent et imprévisible.

Étape 4 : Algorithme de fusion (Union-Find) Pour chaque mur dans la liste mélangée :

1. On examine les deux cellules séparées par ce mur
2. **Si elles ont des identifiants différents** (ne sont pas encore reliées) :
   — On supprime le mur (création d'un passage)
   — On fusionne les composantes : toutes les cellules du second chemin reçoivent l'identifiant du premier
3. **Si elles ont le même identifiant**, on conserve le mur (évite les cycles)

Exemple de fusion : Si on traite le mur entre les cellules 1 et 2 :
— Cellule 1 a id=1, cellule 2 a id=2 → identifiants différents
— On ouvre le mur et on unifie les identifiants
— Résultat après fusion :

```
# # # # #
# 1   1 #
# # # # #
# 3 # 4 #
# # # # #
```

Le processus continue avec les autres murs, en ne créant des ouvertures que si les cellules appartiennent à des composantes connexes différentes.

Étape 5 : Résultat final À la fin de l'algorithme, toutes les cellules sont reliées par un chemin unique. Le labyrinthe obtenu est dit "parfait" car :
— Il existe exactement un chemin entre chaque paire de cellules
— Aucune boucle n'est présente dans la structure
— Toute cellule est accessible depuis n'importe quelle autre
— La suppression d'un mur quelconque créerait une boucle

3.5 Résultats et Tests

Le projet inclut une suite de tests unitaires utilisant la bibliothèque minunit :
— Tests de génération de labyrinthe (validité, connectivité)
— Tests des fonctions de déplacement
— Tests de sauvegarde et chargement des fichiers
— Tests du système de scores
— Tests d'intégration du menu principal
Défis rencontrés :
— Gestion de la mémoire dynamique pour les labyrinthes de grande taille
— Optimisation de l'affichage en mode console
— Implémentation du système de scores avec tri et sauvegarde
— Gestion des cas d'erreur lors du chargement de fichiers
— libération de tout les blocs mémoire alloués lors de l'execution pour éviter les fuites mémoire

3.6 Améliorations futures

Plusieurs améliorations pourraient être apportées au projet :
— Interface graphique avec SDL pour un rendu visuel amélioré
— Algorithmes de résolution automatique (A*, Dijkstra)
— Génération de labyrinthes avec différents algorithmes (Prim, Kruskal)
— Mode multijoueur avec networking
— Système de chronométrage et de statistiques avancées

4 Conclusion

Ce projet de jeu de labyrinthe m'a permis d'explorer les concepts fondamentaux de la programmation en C tout en développant un programme intéressant,fonctionnel et complet.

- Programmation modulaire et bonnes pratiques en C
- Gestion des fichiers et persistence des données
- Gestion de lecture/ecriture des fichiers
- Utilisation d'outils de développement (Make, Doxygen, valgrind, etc...)

L'architecture modulaire adoptée, avec une séparation claire entre les fonctionnalités, facilite la maintenance et les extensions futures. Le système de tests unitaires garantit la robustesse du code, tandis que la documentation améliore sa lisibilité.

Les objectifs initiaux ont été atteints avec succès : le programme génère des labyrinthes parfaits, offre une interface utilisateur complète, et gère efficacement la sauvegarde des données. Les performances sont satisfaisantes même pour des labyrinthes de grande taille.