AVIGNON UNIVERSITÉ

Rapport de projet

M1 ALT

Simon Diaz

08/05/2025

Master ILSEN
UE UCE PROGRAMMATION PARALLÈLE

Responsable
ROUVIER Mickael

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Utilisation du programme

Pour utiliser le programme il faudra juste lancer le programme compilé. Si jamais il n'est pas compilé veuillez le compiler avec **g++ -std=c++11 -O3 main.cpp labyrinthe.cpp -o programme.exe**.
Veuillez éxecuter le programme avec ./programme.exe, il vous demandera de quelle manière vous désirez faire le backtracking. De manière séquentielle, avec des threads par direction au départ ou un thread par labyrinthe. Vous taperez 1, 2 ou 3 puis entrer.

2 Justification d'implémentation

Dans le cadre de ce projet, l'objectif principal était de développer un système permettant de résoudre des labyrinthes en récupérant certains objets obligatoires avant d'atteindre la sortie. La résolution repose sur un algorithme de type backtracking.

2.1 Implémentation du backtracking

La fonction backtracking() repose sur un parcours récursif du labyrinthe, où chaque case visitée est temporairement marquée pour éviter les boucles infinies. L'algorithme explore toutes les directions possibles depuis une case donnée (haut, bas, gauche, droite), en revenant en arrière si aucune solution n'est trouvée dans un chemin.

Lorsqu'un objet est trouvé, il est ajouté à une liste de collecte. La sortie du labyrinthe n'est considérée comme atteinte que si tous les objets nécessaires ont été ramassés, ce qui ajoute une contrainte supplémentaire à la recherche.

2.2 Gestion des labyrinthes "Prime"

Une particularité du projet est l'introduction de labyrinthes dits "Prime", accessibles uniquement après avoir marché sur une case spéciale contenant de la TNT. Dans ce cas, la fonction backtracking() est relancée sur le labyrinthe Prime, en héritant de l'état des cases déjà visitées. Cela permet de simuler un changement de labyrinthe tout en conservant la logique de résolution.

2.3 Implémentations multithreadées

Pour améliorer la performance et explorer différentes stratégies de parallélisation, deux variantes utilisant les threads ont été développées :

Un thread par direction possible :
Depuis la position de départ, un thread est lancé pour chaque direction valide (haut, bas, gauche, droite). Chaque thread exécute le backtracking indépendamment. Cela permet de paralléliser l'exploration initiale et de gagner du temps sur des labyrinthes complexes. Cette approche a été abordée en cours lors du premier TP du projet.

Un thread par labyrinthe : Lorsque plusieurs labyrinthes doivent être résolus, un thread est lancé pour chacun. Cela permet d'accélérer le processus lorsque plusieurs labyrinthes sont fournis en entrée. L'usage de cette méthode est pertinent dans un contexte de traitement en lot ou d'environnement multi-labyrinthe.

Choix de conception :
Le code est structuré de manière modulaire avec une séparation claire entre la logique backtracking (résolution, collecte, exploration) et l'affichage. Le fichier labyrinthesManager.cpp centralise la création et le chargement des labyrinthes à partir d'un fichier texte, permettant une extension facile du système. L'utilisation d'un vecteur global pour les objets collectés

(objetRecoltes) est justifiée ici par le besoin de partager l'état entre plusieurs fonctions de résolution, tout en gardant la simplicité du code.