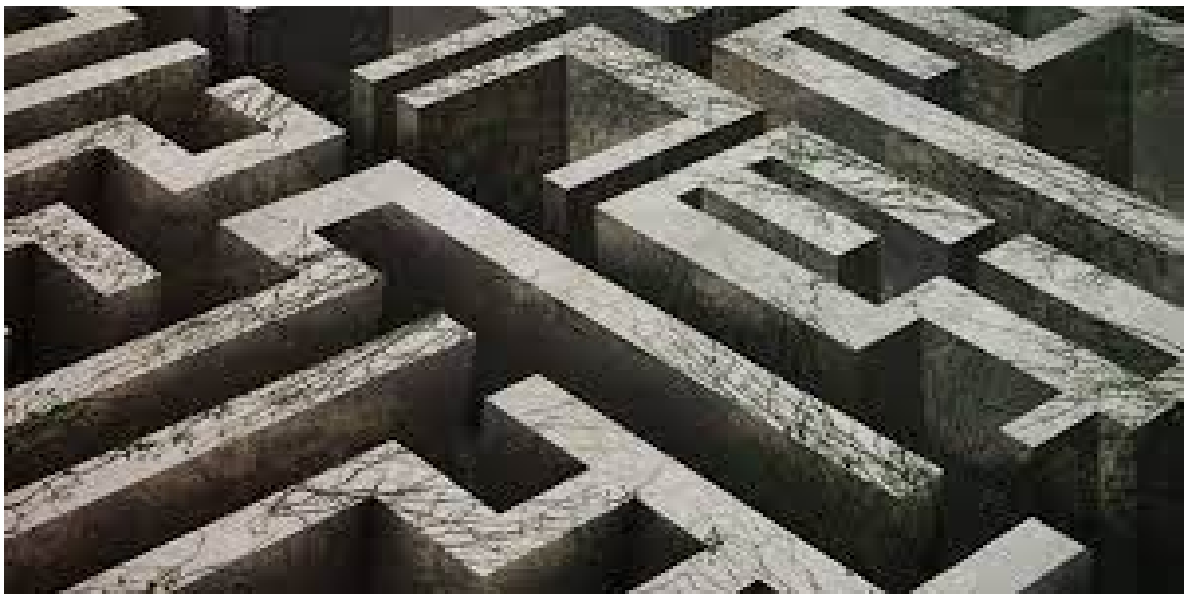
Boudeau Viard Mezou

Projet B3 IngeLog

Maze Generator / Maze Solver



Introduction

Le projet consiste à réaliser un générateur de labyrinthe qui créera un fichier sous format texte ainsi qu'un solver qui lui transformera le fichier texte du générateur afin d'y appliquer un chemin allant de l'entrée à la sortie, cela en passant par une page Web sur laquelle nous enverrons les différents fichier afin d'en avoir un aperçu clair à l'aide de différente technologies tel que le Python pour la génération ou encore du C pour les IA en passant par du React pour l'application Web.

Rôles de chacun

- Arthur : Le rôle d'Arthur consiste au développement du front en react pour le rendu des maps et des ia en C
- Axel : Le rôle d'Axel consiste à s'occuper du générateur de labyrinthe
- Nathan: Le rôle de Nathan consiste à s'occuper de la base de données et a renforcé Arthur sur les algorithmes

Technologies utilisées

- Python3
- React (+hooks)
- C
- Makefile
- sql

Structure algorithmique

Front: Le front est généré grâce à 3 components MazeCell, MazeMap et MazeDashboard.

- **MazeDashbord**: Ce composant est directement appelé dans mon App.tsx. Via un input, on peut récupérer une map avec des Hooks. La fonction fléché se nomme “handleFileChange” car il permet l’insertion de plein maps une par une. Ensuite on y crée un tableau de MazeMap avec le contenu du fichier ou des fichiers.
- **MazeMap**: Celui-ci permet de construire le labyrinth tel qu’on le voit, il se compose d’objet MazeCell représentant 1 caractère stylisé. Donne un effet de “card” au front.
- **MazeCell**: Cet objet attribue une classe CSS à chaque caractère stocké.

L’ia: A* est défini comme un des plus efficaces algorithmes de pathfinding. Le maze est stocké dans tableau et le chemin parcouru dans une liste chainée. A chaque nœud, il calcule de manière heuristique le chemin le plus court de sa position à la fin. La valeur heuristique de chaque case soir “h_cost” est set lors de la mise en place de la map avec les “neighbor”. En lisant la formule on comprend vite que plus nous serons proche de la fin du maze plus le résultat sera faible:

z (neighbor->h_cost = ((map->largeur - 1)- neighbor->x) * 10 + ((map->hauteur - 1) - neighbor->y) * 10;)

Python Maze_gen :

Liste des fonctionnalités validées

- Génération de maze
- Création de fichier txt de la map suite à la génération
- Gestion des arguments pour la taille de la map à la génération
- Algorithme A* fonctionnel
- Front end avec insertion de fichiers pour l'affichage des mazes unsolved et solved
- Affichage clair de la route emprunté par l'algorithme solver dans le front end
- (Affichage du temps réalisé par l'algorithme dans la front end)
- (BDD)

Explications des fonctionnalités principales

- Générateur de labyrinthe : créé un labyrinthe à l'aide d'un script python dans lequel on peut choisir la hauteur ainsi que la largeur
- Solver de labyrinthe : algorithme permettant de résoudre n'importe quel map générée, parfaite ou imparfaite.
- Front end : Permet de mieux comprendre la partie visuelle du projet, c'est-à-dire envoyer les maps et leurs appliquées un style.
- Score : Temps effectué par l'algorithme avec un rendu sur terminal.

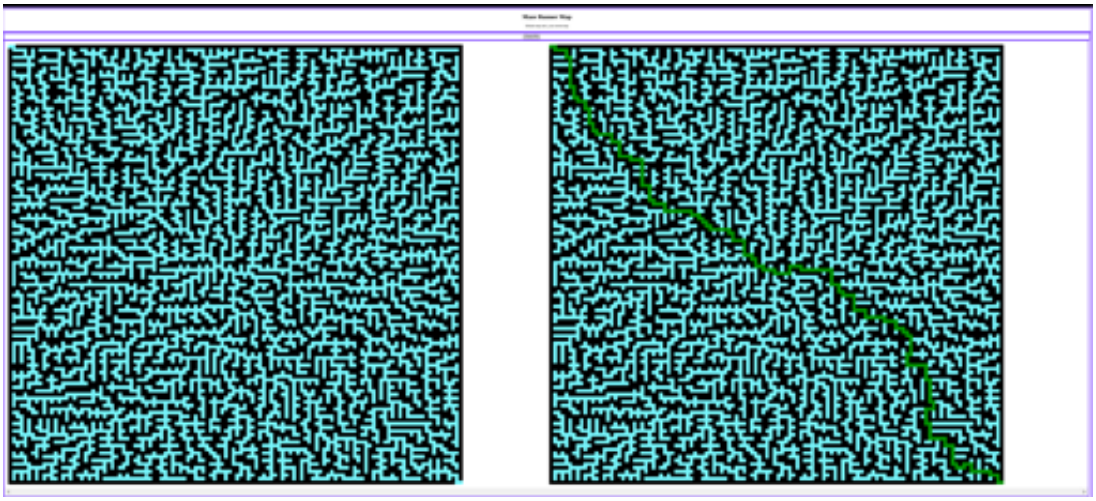
Captures d'écran

Définition: Il existe des labyrinthes dits « parfaits », où un chemin unique passe par toutes les cellules. Et des labyrinthes « imparfaits », où un chemin se recoupant forme des ensembles de cloisons connexes appelées « îlots ».

Front:

25x25

50x50



120x120

IA:


→ Astar git:(main) ✗ ./final_astar/astar ../../Maze_Gen/deposit/90x90.txt
Algorithm A* runtime: 148 ms
→ Astar git:(main) ✗ ./final_astar/astar ../../Maze_Gen/deposit/500x500P.txt
Algorithm A* runtime: 82603 ms
→ Astar git:(main) ✗