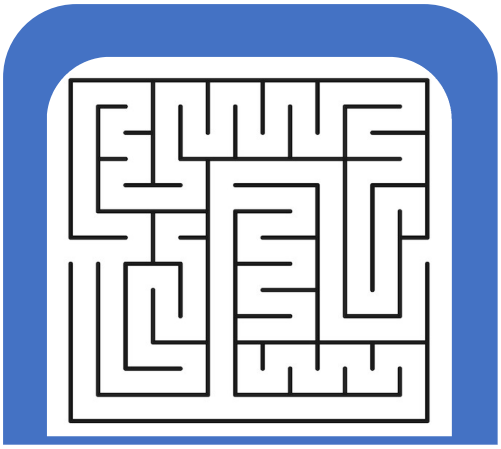

CYNAPSE

RAPPORT DE PROJET

–

GENIE LOGICIEL

24 MAI 2025

BOUHADOU WAALLY, CHIBANTE JULIAN, LAMBERT BAPTISTE, LIPPENS EVAN, VERRIERE CHARLOTTE

Table des matières

Lien vers le dépôt GitHub : https://github.com/fleefie/proj-genie-logiciel/

Introduction :

Dans le cadre de ce projet, nous avons développé une application java interactive dédiée à la génération et à la résolution de labyrinthes parfaits. Les principales fonctionnalités du projet sont :

- Pouvoir générer des labyrinthes, de taille choisie, parfaits ou non ;
- Pouvoir modifier ces labyrinthes en temps réel, en tant qu'utilisateur ;
- Pouvoir résoudre des labyrinthes avec au moins trois algorithmes différents ;
- Pouvoir sauvegarder et charger des labyrinthes pour pouvoir ensuite les résoudre.

Nous avons bien respecté ce cahier des charges. Nous avons cinq façons de générer un labyrinthe, trois en génère des parfaits, et deux des labyrinthes classiques. Nous pouvons modifier les murs en cliquant sur les cases du labyrinthe, cela même pendant la phase de génération. Nous avons bien trois algorithmes permettant de résoudre les labyrinthes, qui peuvent être décomposés pas par pas. Et enfin nous pouvons sauvegarder et charger des labyrinthes sans problème.

Organisation en équipe :

L'organisation du projet a été fortement impactée par les partiels et la semaine de révision qui se trouvait au milieu de la réalisation du projet. Nous avons alors dû organiser notre projet dans un temps réduit, en deux phases distinctes.

Nous nous sommes d'abord concentrés sur la création d'une classe pour le labyrinthe pour pouvoir tout attacher facilement, la création, la résolution, l'affichage et la sauvegarde. Le labyrinthe et le graphe associé, ainsi que la structure générale du projet, ont été la première partie abordée.

Nous avons ensuite fait une réunion pour réfléchir à comment organiser les classes qui ne permettrait de manipuler les labyrinthes. Nous avons alors décidé de faire des interfaces pour les constructeurs, IBuilder, et pour les résolveurs, ISolver, ce qui permet de simplifier le lien entre le front end et la backend de l'application. Avec ces informations, nous avons pu faire un premier diagramme de classe (Annexe 1). Cela nous a aussi permis de nous répartir les rôles ainsi :

- Julian : interface utilisateur, JavaFX ;
- Evan et Baptiste : Résolution des labyrinthes ;
- Charlotte : Création des labyrinthes ;
- Waally : en soutien là où il y aurait besoin.

Cette phase a notamment inclus l'interface utilisateur, partie majeure du projet.

Réalisation du projet :

Algorithmes utilisés :

Pour réaliser ce projet nous avons utilisé différents algorithmes de manipulation de graphe. On peut les séparer en deux catégories, d'abord ceux qui ont été utilisés pour la création des labyrinthes, ensuite ceux qui ont été utilisés pour la résolution des dits labyrinthes.

Pour créer les labyrinthes parfaits, nous partons d'un graphe représentant la grille du labyrinthe. Ses arêtes représentent la possibilité de passer d'une case à l'autre, donc la présence d'un chemin. Le graphe ne contient pas d'arêtes au départ. Les deux premiers algorithmes sont **des parcours en largeur et en longueur** (DFS et BFS) avec une part d'aléatoire, pour ne pas avoir un unique graphe final. Le parcours peut être réalisé grâce au lien entre la grille et le graphe, il n'y a que 4 sommets (maximum) pouvant être le voisin du sommet de départ. Le but est donc de créer un arbre couvrant

de la grille du labyrinthe. Le troisième algorithme est **l'algorithme d'Eller**, il agit de façon similaire à l'algorithme de Kruskal mais nécessite moins de mémoire et est plus rapide. « Il crée le labyrinthe une ligne par une ligne, et lorsqu'une ligne a été générée, il n'en a plus besoin, parce qu'il ne s'en servira plus. Chaque cellule dans une ligne fait partie d'un groupe, et les cellules sont dans le même groupe s'il y a un chemin qui les relie entre elles depuis le début de la génération du labyrinthe. » (Pons, s.d.)

Une fois les labyrinthes créés, il faut les résoudre. Pour faire cela, nous avons trois algorithmes. Nous avons d'abord **l'algorithme A*** et **l'algorithme de Dijkstra**. Ils sont codés ensemble car l'algorithme de Dijkstra correspond à celui d'A* avec une fonction heuristique nulle. Il y a donc 3 fonctions heuristiques possibles : la fonction nulle, la distance de Manhattan et la distance euclidienne. Le troisième algorithme est **l'algorithme de Trémaux**. Il est utile pour trouver la sortie lorsqu'on n'a pas toutes les informations au départ (ex, où est l'arrivée) et si on peut marquer les chemins. Il correspond à un parcours en profondeur d'un labyrinthe. Cependant, l'algorithme de Trémaux, contrairement à l'algorithme A*, ne donne pas forcément le chemin le plus court.

Problèmes rencontrés et solutions,

Nous avons eu peu de problème. Ceux qui nous ont le plus embêté sont ceux-là.

D'abord c'est l'enregistrement propre d'un état de résolution pour pouvoir ensuite le récupérer et le réutiliser. La solution fut d'utiliser l'interface Iterable avec l'interface Serializable. Cependant, le labyrinthe, MazeView, s'affiche alors en s'agrandissant pour une raison que nous n'avons pas réussi à déterminer.

Un autre problème, aussi lié à la sérialisation, est que la classe Color de JavaFX n'est pas sérialisation dans notre cas. Nous avons dons dû faire une énumération pour remplacer le paramètre couleur dans la classe Cell pour pouvoir l'enregistrer.

Limites fonctionnelles

L''algorithme d'Eller n'a pas pu être implémenté en raison de sa difficulté lorsqu''il est utilisé dans un contexte de génération séquentielle étape par étape au lieu d'un seul procédé.

La seule limite fonctionnelle que nous avons pu rencontrer est l'impossibilité de sauvegarder les couleurs des cases de la grille dans la résolution, comme expliqué ci-dessus.

Bibliographie et Annexes

Bibliographie

Cantwell, T. (2020, Octobre 1). *Eller's Maze Algorithm in JavaScript*. Récupéré sur Medium: https://cantwell-tom.medium.com/ellers-maze-algorithm-in-javascript-2e5742c1a4cd

Goulu. (2020, Avril 24). *Résolution de labyrinthe - Algorithme de Trémaux*. Récupéré sur Wikipédia: https://fr.wikipedia.org/wiki/R%C3%A9solution_de_labyrinthe#Algorithme_de_Tr%C3%A9maux

mguru4c05q. (2024, Juin 21). *Difference Between Dijkstra's Algorithm and A* Search Algorithm*. Récupéré sur GeeksForGeeks: https://www.geeksforgeeks.org/difference-between-dijkstras-algorithm-and-a-search-algorithm/

Pons, O. (s.d.). *Algorithme d'Eller*. Récupéré sur Olivier Pons - Blog technique: https://www.olivierpons.fr/labyrinthes/algorithme-deller/

Annexe 1 :

Premier diagramme de classe du projet