Projet génie logiciel

CYNAPSE

Application de génération/résolution de labyrinthe

par Nithiyarajan Sarusan, Jin Arthur, Larabi Omar, Rumeli Yusuf et Lohan Briard

---

Introduction

CYNAPSE est un projet de génération et résolution de labyrinthe, le but étant de faire une application qui permet d'afficher la génération et la résolution du labyrinthe en utilisant des algorithmes tels que DFS et Kruskal pour la génération et A* pour la résolution, mais aussi JavaFX comme outil pour l'interface. Tout cela en ajoutant des fonctionnalités telles que la modification (murs/entrée/sortie), charger/sauvegarder, choix de la taille et la vitesse de génération/résolution du labyrinthe.

Organisation de l'équipe

Dès le début du projet, nous avons mis en place un groupe discord pour permettre de planifier et tenir régulièrement des réunions, ces réunions nous ont aidées à avancer, suivre le projet et répartir les tâches. Afin de faciliter le suivi des versions et le travail des autres personnes de l'équipe, nous avons mis en place un dépôt GitHub commun. Chaque membre mettait son code et les modifications qu'ils ont effectuées. Nous avons scindé l'équipe en deux groupes, l'un qui s'est chargé d'effectuer la partie algorithmique, donc de génération et de résolution, puis l'autre groupe s'est chargé de faire l'interface graphique et l'implémentation des algorithmes pour que tout fonctionne. Cette répartition nous a permis de travailler efficacement en assurant une bonne communication entre les groupes avec des réunions journalières et à notre dépôt partagé.

Problèmes et solutions

1) Problème : La fermeture d'une fenêtre pendant une génération/résolution n'arrête pas l'algorithme, il travaille toujours dans le fond

   Solution : Devoir vérifier à chaque step si la fenêtre est encore ouverte (non implémentée)

2) Problème : Gestion des erreurs (ex : dimensions invalides) et des exceptions

   Solution : Utilisation de try-catch et affichage d'une fenêtre d'erreur différente pour chaque type d'erreur

3) Problème : Taille de la fenêtre selon le labyrinthe, on affichait la même taille de fenêtre pour un labyrinthe de taille 5x5 ou 100x100

   Solution : Calcul de la taille des murs/cellules selon la taille du labyrinthe et utilisation d'une barre de scroll pour visualiser tout le labyrinthe

4) Problème : Manque de clarté dans les couleurs utilisées lors de la résolution du labyrinthe

   Solution : Affichage d'une légende pour faciliter la compréhension pour l'utilisateur

5) Problème : Implémentation du mode pas à pas, on avait uniquement une implémentation complète

   Solution : Modification des algorithmes de résolution et de génération pour avoir une méthode step

6) Problème : L'algorithme qui tourne toujours même après avoir trouvé la solution.

   Solution : Implémenter la variable isFinished et la méthode isFinished().

7) Problème : Les différents algorithmes de génération et résolution n'avaient pas les mêmes variables.

   Solution : Création d'une classe MazeGenerator et MazeSolver qui permet de gérer l'appel d'un algorithme.

8) Problème : Selon la taille du labyrinthe, l'affichage n'est pas fait à chaque étape de l'algorithme.

   Solution : Adapter l'affichage selon le nombre de cases.

Limitations fonctionnelles

Selon le PC, générer un labyrinthe de trop grande taille ne fonctionne pas. Sur le pc de l'école, on est à environ un maximum de 300 000 cellules.

La modification n'est pas dynamique, elle ouvre une nouvelle fenêtre pour que l'utilisateur puisse ajouter/supprimer des murs ainsi que changer l'entrée et la sortie.

Planning

Kick-off du projet : Mercredi 30 avril

Tâches du backlog et diagrammes : Vendredi 2 mai

Premiers algorithmes et maquette : Dimanche 11 mai

Finalisation de l'interface graphique et algorithmes : Vendredi 16 mai

Correction des erreurs et des incohérences : Mercredi 21 Mai

On a également effectué des réunions à un rythme d'un jour sur 2 pour vérifier l'avancement du projet sur l'application Discord

Diagramme de cas d’utilisation

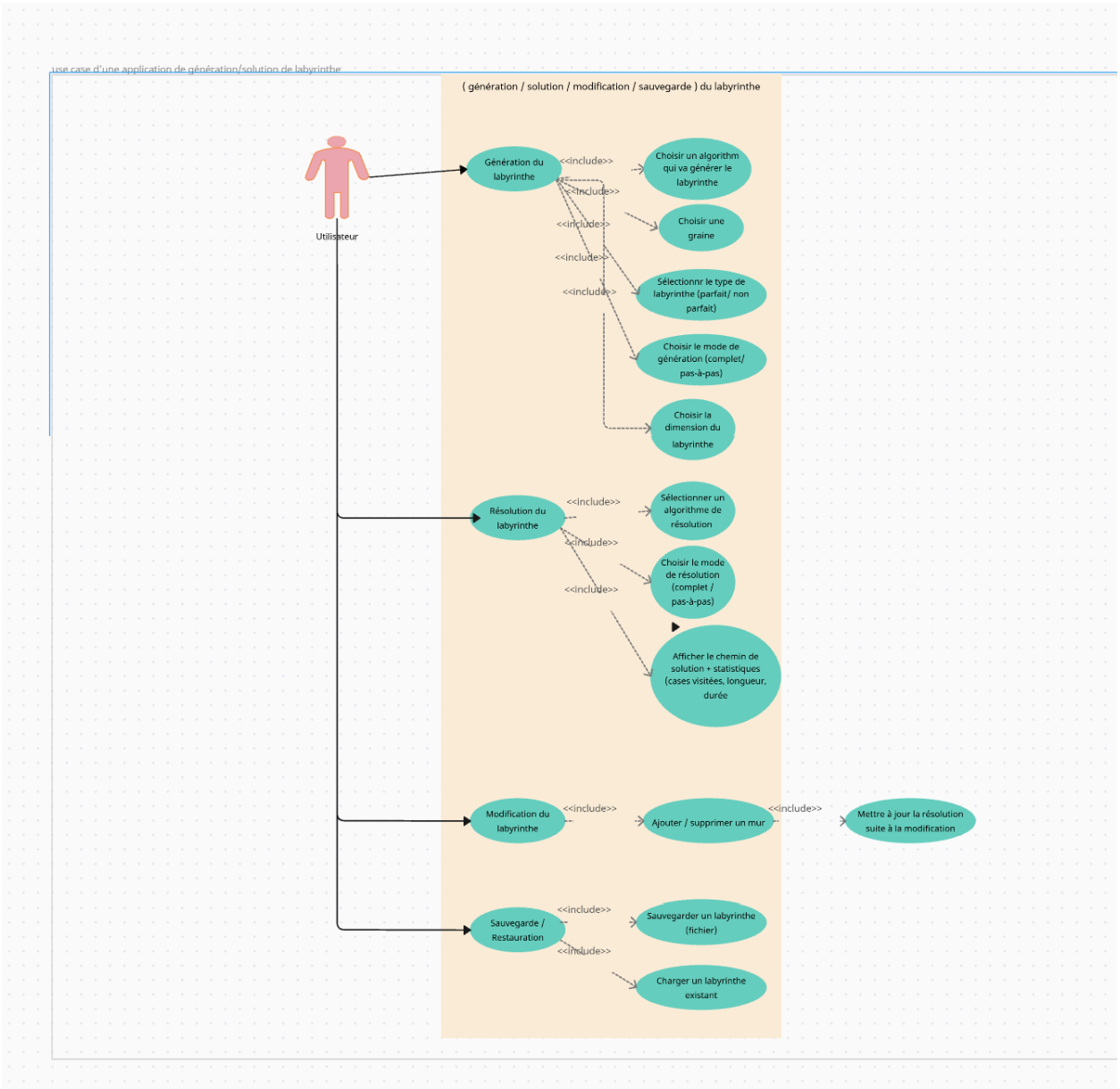

Diagramme de classe