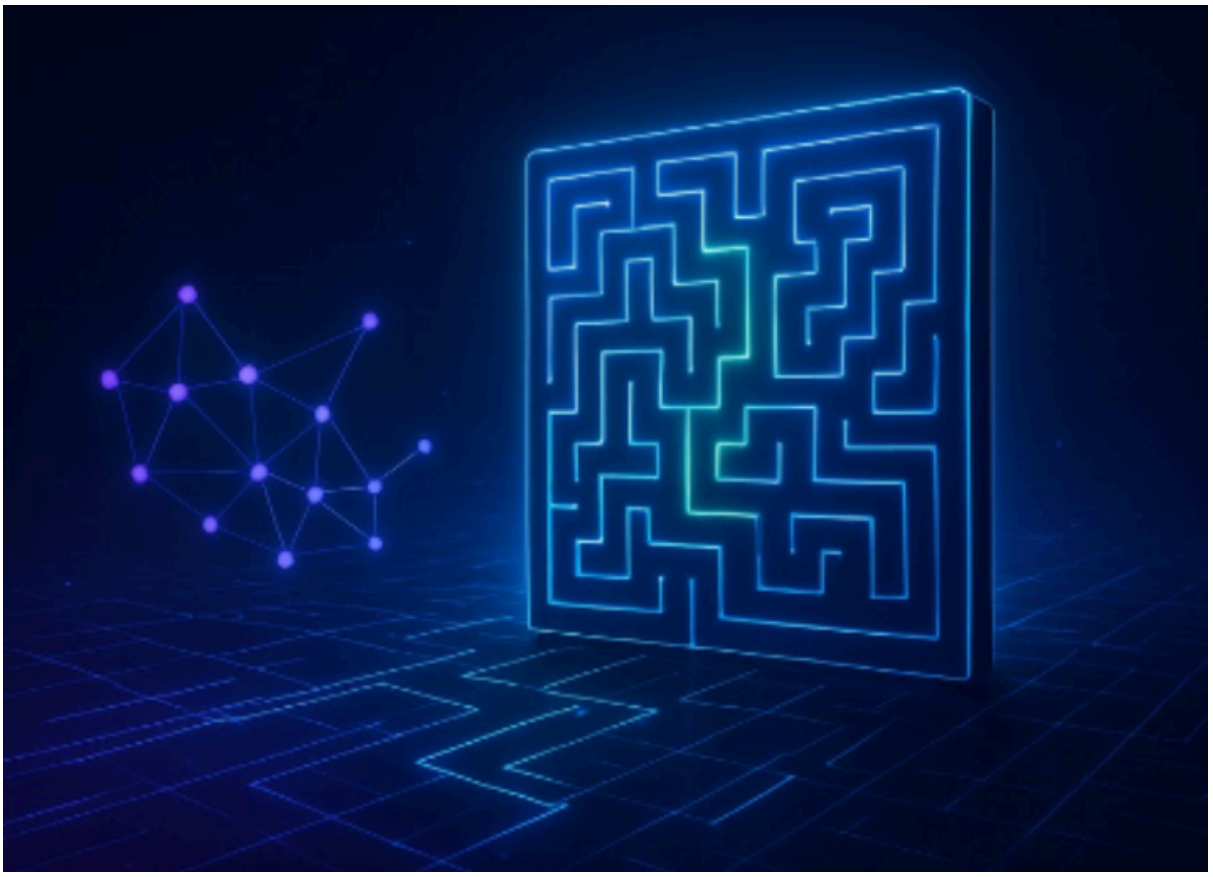
CYnaspe
Rapport de stage



Le but de cette application est de créer une interface graphique permettant de générer puis de résoudre des labyrinthes.

ING1 GI - 2024-2025
Hicham Tougri, Abhishek Suresh, Yuness Smail, Yassir El jarrari, Dylan Mei

23 Mai 2025

SOMMAIRE

I - INTRODUCTION **3**
1 - Présentation générale du projet 3
2 - Les objectifs 3
3 - Les enjeux 3
4 - Choix des algorithmes 3
II - RÉPARTITION DES TÂCHES **5**
1 - Répartition des tâches 5
2 - Diagramme de Gantt 5
3 - Principaux outils utilisés pour le travail collaboratif 6
III - ÉTAPES DE RÉALISATION DU PROJET **7**
IV - PROBLÈMES RENCONTRÉS, SOLUTIONS ET LIMITES FONCTIONNELLES **9**
PROBLÈMES RENCONTRÉS 9
LIMITES FONCTIONNELLES 9
V - DIAGRAMMES **10**
1 - Diagramme de classes 10
2 - Diagramme cas d'utilisations 12
VI- CONCLUSION ET PERSPECTIVES **13**
1 - Conclusion 13
2 - Perspectives d’amélioration 13
3 - Lien GitHub 13

I - INTRODUCTION

1 - Présentation générale du projet

Le projet CYnapse a pour objectif de concevoir une application graphique Java permettant de générer et résoudre des labyrinthes, en mettant en œuvre des algorithmes sur des graphes non orientés. L'application doit proposer différents modes de génération (parfait ou non), des résolutions par plusieurs algorithmes (DFS, BFS, A*, etc.), une visualisation dynamique, et des modifications locales du labyrinthe.

Ce projet vise à sensibiliser les étudiants à l'algorithmique avancée et à la conception d'interfaces en JavaFX, dans un cadre ludique, interactif et technique. Une version console permet de tester toutes les fonctionnalités indépendamment de l'interface graphique.

2 - Les objectifs

- Génération aléatoire ou paramétrée de labyrinthes.
- Résolution visuelle avec différents algorithmes.
- Modification locale dynamique.
- Interface JavaFX complète.
- Sauvegarde/restauration des labyrinthes.

3 - Les enjeux

- Approfondir les connaissances en structures de données et algorithmique.
- Créer une interface ergonomique et réactive.
- Travailler en équipe dans un environnement de projet.
- Rendre les processus de génération/résolution compréhensibles par tous.

4 - Choix des algorithmes

Génération :

- **Kruskal (version simplifiée)** – en option pour les imparfaits.

Nous avons choisi cet algorithme, car il décrit par Internet que c'est la plus satisfaisante à regarder lors de la génération, et la plus efficace comparé aux autres algorithmes.

Résolution :

- **BFS (Breadth-First Search)**

Garantit le plus court chemin mais prend le plus de mémoire car cela garde tous les chemins possible dans une même profondeur. Il explore tous les voisins de la case actuelle avant de passer à l’autre.

- **Recursive Maze Solver (Depth First Search)**

Facile à comprendre et à faire. Permet une visualisation plus "exploratoire" du labyrinthe. Il est efficace en mémoire car il garde le chemin en cours et est proportionnel à la profondeur du labyrinthe.

- **Djikstra**

Contrairement au BFS et DFS, Djikstra marche selon si les cases ont des coûts. Il se rend d’abord aux chemins les moins coûteux, ce qui le rend efficace et optimal.

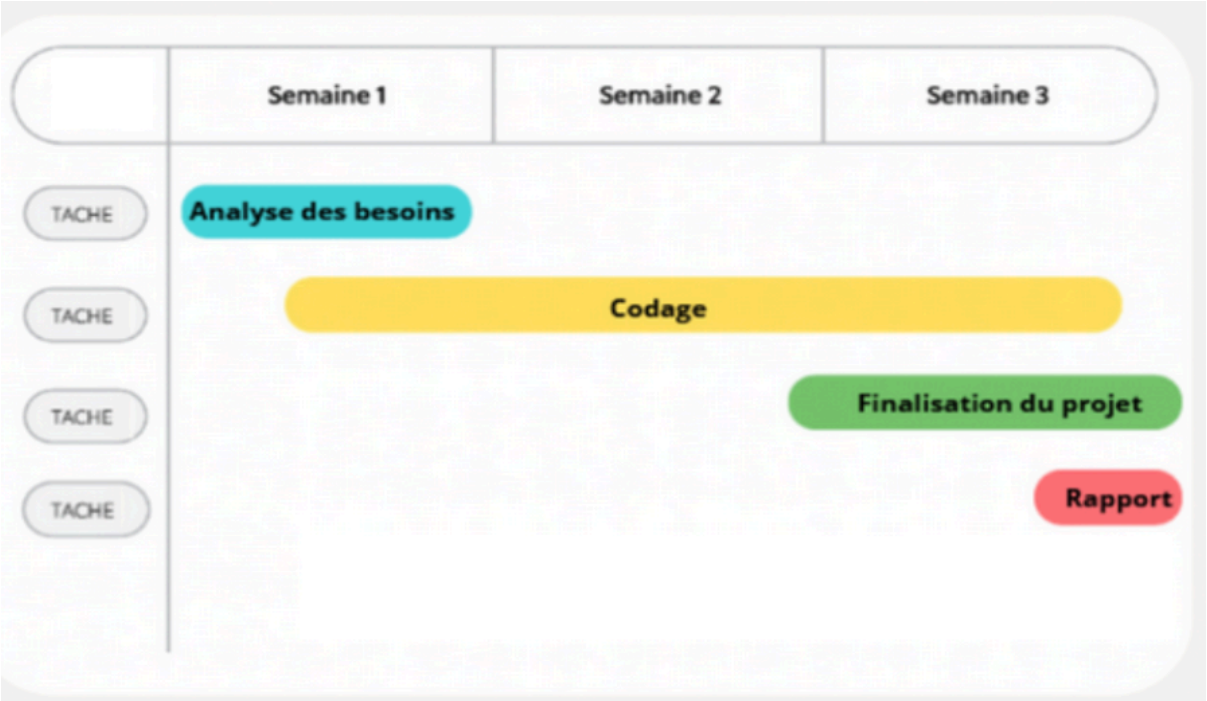
II - RÉPARTITION DES TÂCHES

Nous avons réparti les tâches en fonction des compétences et des points forts de chacun, afin de travailler de manière efficace et collaborative tout en valorisant les savoir-faire individuels

1 - Répartition des tâches

Tâches	Nom des étudiants
Génération des labyrinthes	Dylan
Algorithmes de résolution	Hicham, Abhishek, Yassir
Interface JavaFX	Dylan
Sauvegarde / Chargement	Yuness
Version Console	Dylan
Documentation UML et rapport	Hicham

2 - Diagramme de Gantt



3 - Principaux outils utilisés pour le travail collaboratif

Pour assurer une bonne coordination entre les membres du groupe tout au long du projet, nous avons utilisé plusieurs outils et méthodes de travail collaboratif.

Tout d’abord, nous avons principalement communiqué via Whatsapp, en utilisant les messages écrits et les appels vocaux pour organiser les tâches, partager les avancées et résoudre les problèmes rapidement. Dû à des problèmes de branche, nous avons donc parfois partagé nos différents fichiers via discord.

III - ÉTAPES DE RÉALISATION DU PROJET

- **Etape 1 : Construction des généralités**

Puisque le projet repose sur une structure de données, il fallait d'abord coder une représentation d'un labyrinthe et de ses cases. On a commencé avec une interface graphique simple grâce à FXML et Scene Builder qui permettent de construire facilement des interfaces. Au début, l'interface affichait seulement un labyrinthe généré par le code. Il n'était pas possible de générer en passant par l'interface.

- **Etape 2 : Intégration des algorithmes de résolution**

Pour éviter un blocage du projet en cas de retard dans la mise en place des généralités, les personnes qui s'occupaient des algorithmes pouvaient commencer de leur côté pour comprendre comment ils marchent, même en dehors des labyrinthes. Lorsque les généralités ont été finies, nous avons ensuite rendu compatible les algorithmes avec le labyrinthe. Nous avons commencé par faire le mode complet, car on avait besoin d'une interface plus avancée pour développer le mode pas à pas.

- **Etape 3 : Développement de l'interface graphique (JavaFX et FXML)**

Nous avons fait une fenêtre qui permet de configurer les paramètres du labyrinthe pour sa génération. Ensuite, nous avons mis en place les différents widgets permettant de choisir la vitesse de génération/résolution et l'algorithme de résolution, ainsi que son mode.

Pour faciliter la communication entre l'interface et les données. Nous avons utilisé le modèle MVC qui nous a permis d'être plus à l'aise. Comme mentionné dans l'étape 1, Scene Builder nous a aidé à concevoir rapidement et efficacement des interfaces graphiques. De plus, elle est utile pour la conception du modèle MVC.

- **Etape 4 : Ajout de la visualisation pas à pas**

C'est lors du développement de l'interface que les algorithmes de résolution ont été modifiés afin d'être compatibles avec le mode pas à pas, grâce à la possibilité de visualiser leur exécution. C'est également à ce moment-là que nous avons intégré des widgets permettant d'afficher les statistiques de performance des algorithmes.

- **Etape 5 : Intégration des fonctionnalités de modification dynamique**

Une fois la visualisation fonctionnelle, nous sommes ensuite passés à la modification topologique du labyrinthe. Les modèles utilisés pour représenter le labyrinthe et ses cases nous ont permis de développer cette fonctionnalité avec efficacité. De plus, les événements liés à l'interface permettent une interaction fluide entre l'utilisateur et les modèles de données.

L’une des principales difficultés a été d’identifier les cas dans lesquels le programme renvoyait des erreurs, ainsi que de concevoir une interaction ergonomique permettant à l’utilisateur de modifier ou supprimer un mur de manière intuitive.

- **Etape 6 : Implémentation de la sauvegarde/chargement**

Nous sommes ensuite passés à l’ajout des fonctionnalités de sauvegarde et de chargement des labyrinthes, puisque les éléments nécessaires à leur construction, visualisation et modification étaient désormais en place.

- **Etape 7 : Ajout du mode console**

Enfin, en cas de problème avec JavaFX, nous avons développé une version simplifiée utilisable en console. Pour activer ce mode, l’utilisateur doit passer l’argument « console » lors de l’exécution. Ce mode demande la configuration du labyrinthe, puis l’algorithme de résolution, et affiche ensuite le labyrinthe résolu.

IV - PROBLÈMES RENCONTRÉS, SOLUTIONS ET LIMITES FONCTIONNELLES

PROBLÈMES RENCONTRÉS

- Lors de la résolution, il était possible de générer un nouveau labyrinthe ou d'en charger un existant, ce qui conduisait à des erreurs.

Solution : On interdit à l'utilisateur de créer un nouveau labyrinthe ou d'en charger une.

- Lors de la génération en mode pas à pas, sauvegarder pendant ramène à un labyrinthe qui n'était pas fini.

Solution : On interdit à l'utilisateur de sauvegarder pendant la génération.

- Lorsqu'on appuyait sur une flèche (la touche sur la clavier), cela mettait en focus les boutons radio. Or cela était un problème car on utilisait ces touches pour choisir la direction du mur pour la modification et la suppression.

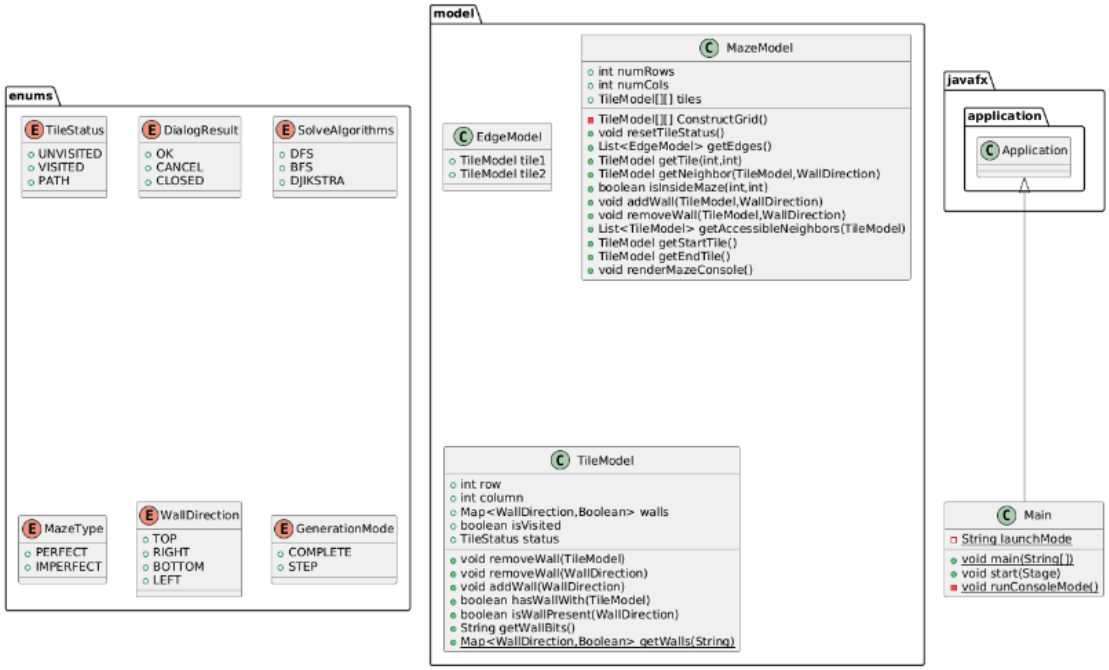
Solution : On a utilisé les touches ZQSD car ce sont des touches qu'on est habitué a utiliser dans les jeux vidéo.

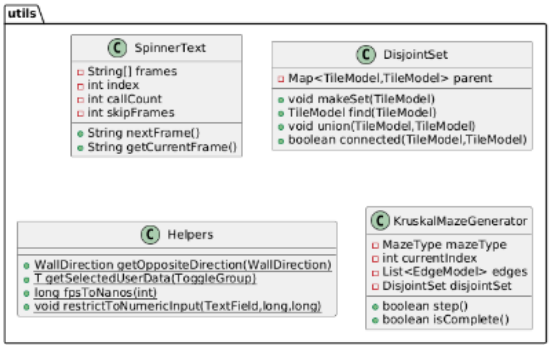
LIMITES FONCTIONNELLES

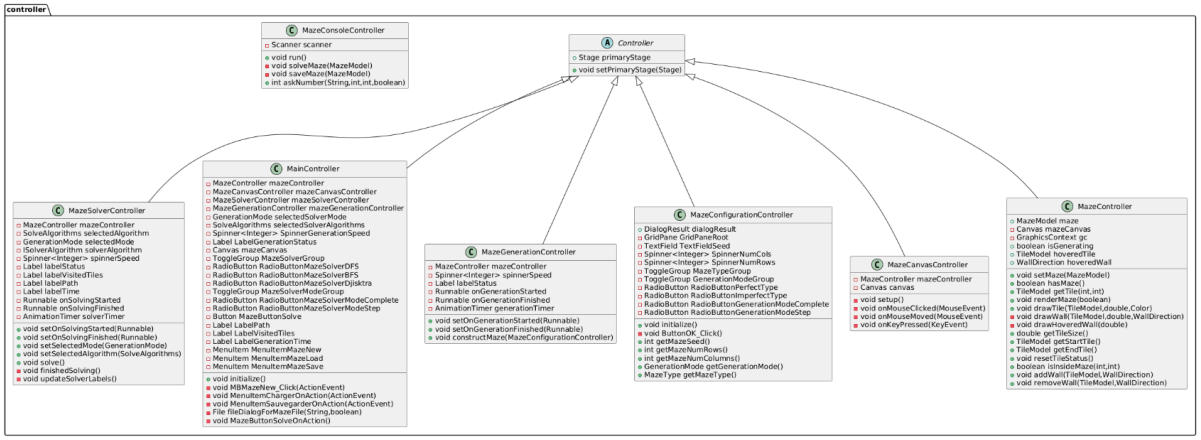
- Pour la résolution, lors du mode pas à pas, le temps de génération dépend de la vitesse de génération. Donc si l'utilisateur joue avec la vitesse, le temps de génération ne sera pas précis.
- La vitesse, le choix du nombre de ligne et de colonne sont limités par un Spinner. Il n'est pas possible d'écrire un nombre directement.
- La case du début et de fin sont placées automatiquement, il n'est pas possible de les modifier.
- L'application ne supporte pas un agrandissement ou un rétrécissement plus dynamique.
- On n'a pas testé sous Linux vu que le développement à été fait majoritairement par Windows. La raison est qu'on utilisait nos ordinateurs personnels aussi pour coder.

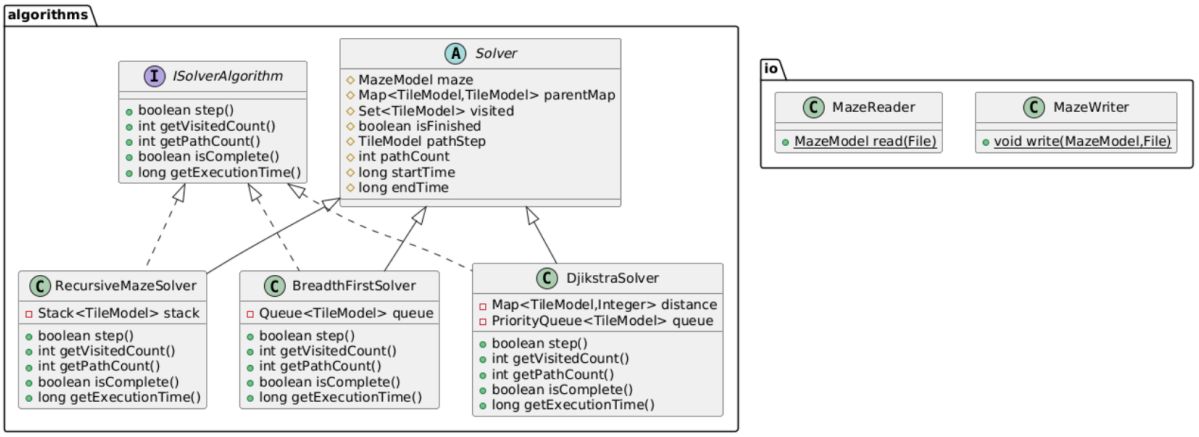
V - DIAGRAMMES

1 - Diagramme de classes

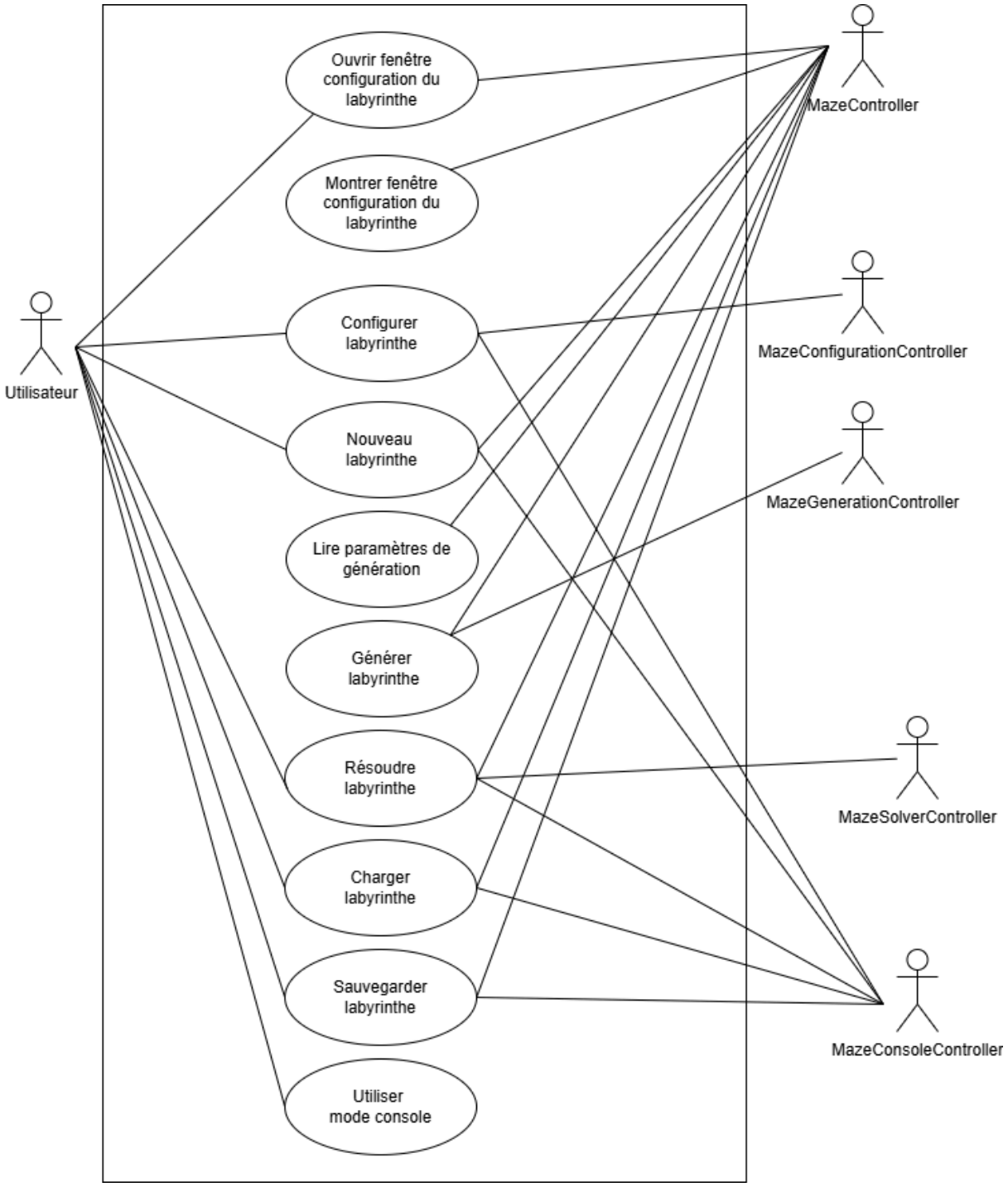







algorithms
ISolverAlgorithm
boolean step()
int getVisitedCount()
int getPathCount()
boolean isComplete()
long getExecutionTime()
Solver
MazeModel maze
Map<TileModel,TileModel> parentMap
Set<TileModel> visited
boolean isFinished
TileModel pathStep
int pathCount
long startTime
long endTime
RecursiveMazeSolver
Stack<TileModel> stack
boolean step()
int getVisitedCount()
int getPathCount()
boolean isComplete()
long getExecutionTime()
BreadthFirstSolver
Queue<TileModel> queue
boolean step()
int getVisitedCount()
int getPathCount()
boolean isComplete()
long getExecutionTime()
DjikstraSolver
Map<TileModel,Integer> distance
PriorityQueue<TileModel> queue
boolean step()
int getVisitedCount()
int getPathCount()
boolean isComplete()
long getExecutionTime()
io
MazeReader
MazeModel read(File)
MazeWriter
void write(MazeModel,File)

2 - Diagramme cas d'utilisations



VI- CONCLUSION ET PERSPECTIVES

1 - Conclusion

Le projet **CYnapse** nous a permis d'explorer les dimensions techniques et visuelles d'un logiciel interactif basé sur des graphes. Il a renforcé nos compétences en algorithmique, en conception modulaire Java, ainsi qu'en gestion de projet agile. Il nous a permis de comprendre plus en détail comment marche un modèle MVC, et de connaître les avantages et inconvénients.

2 - Perspectives d'amélioration

- Améliorer la gestion des grands labyrinthes (c'est-à-dire par rapport à la résolution).
- Intégrer une base de données pour suivre l'historique des labyrinthes générés, ou des algorithmes utilisés.
- Exporter les labyrinthes en image ou fichier JSON.
- Ajouter d'autres algorithmes de résolution.
- Permettre à l'utilisateur de modifier la case de départ et la case de fin.
- Avoir un agrandissement et rétrécissement plus dynamique.

3 - Lien GitHub

https://github.com/Wirbelwind03/CYnaspe