Rapport du Projet de Génie Logiciel

COURTADE Orianne
LAM Emilline
LANCE RICHARDOT Mathis
ROBERT Loïc
SOUPPAYA Alban

Groupe 15

UE : Département Informatique
Syllabus : Projet Génie Logiciel : CYnapse
Tuteur : Eva Ansermin

CY Tech — CY Cergy Paris Université
Année universitaire 2024-2025

Table des matières

Introduction

Présentation de l'équipe

Ce projet a été réalisé par une équipe de cinq étudiants passionnés par le développement logiciel et les défis algorithmiques. Les membres de l'équipe sont :

- COURTADE Orianne
- LAM Emilline
- LANCE-RICHARDOT Mathis
- ROBERT Loïc
- SOUPPAYA Alban

L'équipe s'est distinguée par son **efficacité** et une **excellente communication** interne tout au long du projet. Nous avons mis un point d'honneur à organiser des **réunions d'équipe** régulières afin d'assurer un suivi constant de l'avancement, de partager les connaissances et de résoudre collectivement les difficultés rencontrées. L'**implication de tous les participants** a été un facteur clé de succès, chaque membre apportant ses compétences et contribuant activement à l'atteinte des objectifs. Nous avons également veillé **à être à l'écoute des retours de notre tuteur**, cherchant ainsi à améliorer continuellement notre approche et nos livrables.

Pour la collaboration et le développement, nous avons utilisé les outils suivants :

GitHub, **IntelliJ IDEA**, **StarUML**, **Figma**, **Discord**, **Telegram** et **Trello**.

Description du projet

Le projet a pour **objectif principal** le développement d'une application Java interactive dotée d'une interface graphique, dédiée à la **génération et à la résolution de labyrinthes**. L'approche technique repose sur l'application d'**algorithmes de graphes** pour la construction de labyrinthes, qu'ils soient parfaits (chaque point est accessible depuis chaque autre point par un chemin unique) ou imparfaits (pouvant contenir des cycles et des chemins multiples).

L'application offre un ensemble de fonctionnalités clés :

- **Génération de labyrinthes** : Elle permet de créer dynamiquement des labyrinthes en utilisant des algorithmes reconnus, notamment l'**algorithme de Kruskal**, garantissant la création de structures complexes et variées.
- **Résolution automatique** : Une fois un labyrinthe généré, l'utilisateur peut en déclencher la résolution automatique grâce à diverses stratégies performantes, telles que l'**algorithme de Trémaux**, la recherche en largeur (**BFS - Breadth-First Search**), ou encore la méthode "main sur le mur" (**Hand on Wall**).
- **Visualisation** : Le labyrinthe et son chemin solution sont présentés de manière claire et intuitive, avec une option de **visualisation terminale** pour un affichage simplifié et une **interface graphique interactive** pour une expérience utilisateur enrichie.

Le **public visé** par cette application est large, incluant toute personne capable d'interagir avec un ordinateur, des novices curieux aux étudiants en informatique souhaitant visualiser des algorithmes de graphes en action.

Objectifs du cahier des charges

Le développement de ce projet a été guidé par un cahier des charges précis, dont l'objectif était de créer une application de gestion de labyrinthes robuste et fonctionnelle. Les exigences initiales, sur lesquelles notre travail s'est concentré, peuvent être synthétisées comme suit :

- **Génération de Labyrinthes** : L'application devait permettre la génération de labyrinthes de dimensions variables (définies par l'utilisateur), en distinguant les **labyrinthes parfaits** et les **labyrinthes imparfaits**. La génération devait s'appuyer sur une **graine (seed)** choisie par l'utilisateur. L'algorithme de génération utilisé devait être clairement identifié et documenté.

- **Modes de Génération** : Deux modes de génération distincts étaient requis :
  - Un **mode complet** (labyrinthe affiché une fois entièrement généré, avec une indication de progression).
  - Un **mode pas à pas** (affichage en temps réel de la construction du labyrinthe, avec un réglage de la vitesse).

- **Résolution de Labyrinthes** : L'application devait implémenter au moins **trois algorithmes de résolution** différents. La résolution devait également proposer les deux modes (complet et pas à pas). Une visualisation claire du chemin solution et des cases explorées était attendue, accompagnée de statistiques comme le nombre de cases du chemin final, le nombre de cases traitées, et le temps de génération/résolution.

- **Modification Manuelle** : Une fois généré, le labyrinthe devait pouvoir être **modifié manuellement** par l'utilisateur (ajout ou suppression de murs) afin de créer de nouvelles topologies résolubles.

- **Sauvegarde et Restauration** : La fonctionnalité de **sauvegarde** des labyrinthes dans un fichier de format libre était une exigence clé. Parallèlement, la capacité de **restaurer** un labyrinthe précédemment sauvegardé et de poursuivre son utilisation était également impérative.

- **Stabilité et Robustesse** : Le système devait être conçu pour garantir une **stabilité de fonctionnement** et éviter les plantages, assurant ainsi une expérience utilisateur fiable.

Ces exigences ont servi de fondement à toutes les étapes de conception, de développement et de test de notre application.

1 Organisation et méthodologie

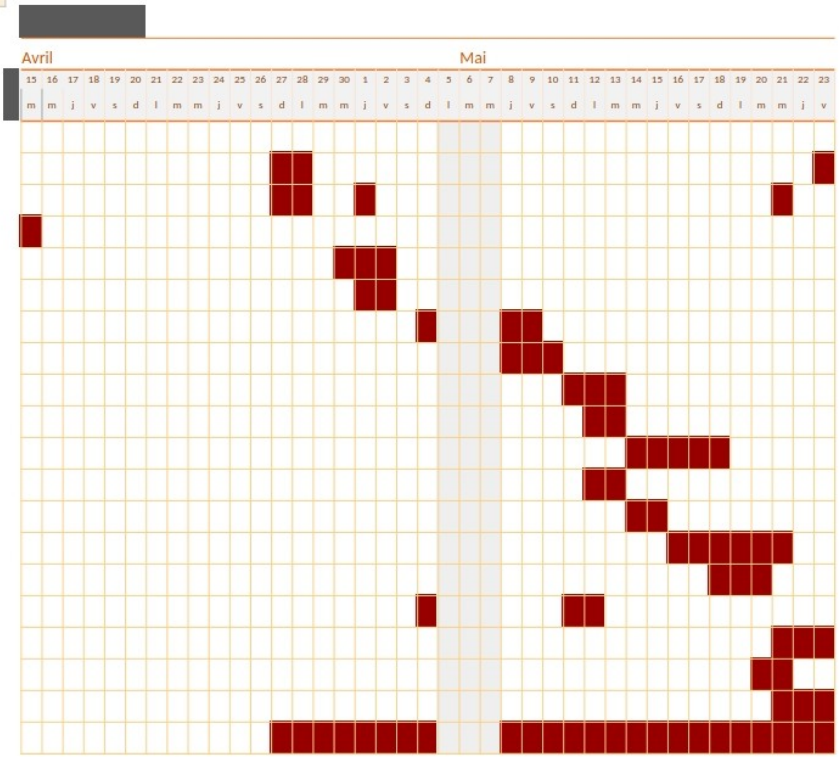
1.1 Répartition des rôles dans l'équipe

CY-Napse

Date de début du projet : 15/4/2025

Marqueur de jalon : 1

Description du jalon	Affecté à	Début
Diagramme de cas d'utilisation	Orianne	27/4/2025
Diagramme de classes	Toute l'équipe	27/4/2025
Création de la sctucture JavaFX	Orianne/Alban	15/4/2025
Génération de labyrinthe	Alban	30/4/2025
Initialisation de l'interface graphique	Alban	1/5/2025
Résolution par Trémaux	Mathis	4/5/2025
Résolution par Hand on wall	Orianne	8/5/2025
Affichage pas à pas	Orianne/Alban/Mathis/Loïc	11/5/2025
Création de la base de donnée	Alban	12/5/2025
Mise en place des sauvegardes	Alban	14/5/2025
Résolution par BFS	Loïc	12/5/2025
Modification des murs internes	Emilline	14/5/2025
Mise à jour de l'interface graphique	Orianne/Alban/Loïc	16/5/2025
Implémentation de l'interface terminale	Orianne/Mathis	18/5/2025
Personnalisation labyrinthe (zoom, dimention, seed, etc)	Toute l'équipe	1/5/2025
Modification des entrer/sorties	Loïc	21/5/2025
Affichage des statistiques	Mathis/Emilline	20/5/2025
JavaDoc	Orianne/Emilline	21/5/2025
Commentaires	Toute l'équipe	27/4/2025



1.2 Outils utilisés

La réalisation de ce projet a nécessité l'emploi d'une panoplie d'outils adaptés à chaque phase du développement, de la conception à la persistance des données, en passant par la visualisation et le prototypage.

1.2.1 Outils de Développement

Pour le développement de l'application elle-même, nous avons principalement eu recours aux environnements suivants :

- **IntelliJ IDEA** : Cet environnement de développement intégré (IDE) a été notre outil principal pour l'écriture, la compilation et le débogage du code Java. Sa robustesse, ses capacités d'auto-complétion et ses outils de refactoring ont grandement contribué à l'efficacité du développement.
- **Visual Studio Code (VS Code)** : Utilisé en complément d'IntelliJ.
- **Scene Builder** : Indispensable pour la conception de l'interface utilisateur graphique en JavaFX.
- **Sora (Génération de GIF)** : Cet outil a été utilisé pour capturer et générer des GIF animés des processus de génération et de résolution des labyrinthes en mode pas à pas, offrant une visualisation dynamique et compacte pour la démonstration.

1.2.2 Outils de Gestion de Base de Données

Pour la persistance des données des labyrinthes (sauvegarde et restauration), nous avons opté pour la combinaison suivante :

- **SQLite** : SQLite a été privilégié pour sa légèreté et sa capacité à être intégré directement dans l'application Java. Il a servi de solution de persistance pour les labyrinthes, offrant une méthode simple et efficace pour la sauvegarde et le chargement des configurations.

1.3 Flux de travail

Notre projet a suivi une approche structurée et itérative, permettant de progresser de la conception théorique à l'implémentation concrète, tout en assurant une intégration continue des retours et des optimisations. Le flux de travail s'est déroulé selon les grandes étapes suivantes :

1. **Phase de Conception Préliminaire :**
   - Nous avons débuté par la formalisation de l'architecture logicielle. Cela a impliqué la création d'un **diagramme de classes** pour modéliser les entités clés du labyrinthe et leurs relations.
   - Parallèlement, nous avons élaboré les **cas d'utilisation** et les **diagrammes de cas d'utilisation** pour identifier et décrire les interactions principales entre les utilisateurs et le système.

2. **Développement des Composants Métier :**
   - Après la conception, nous nous sommes concentrés sur l'implémentation des entités de base.
   - Ceci a été suivi par le développement de la logique de **génération du labyrinthe**, intégrant l'algorithme de Kruskal et gérant les propriétés de labyrinthes parfaits et imparfaits.
   - Simultanément, le travail sur les **algorithmes de résolution** a été lancé, permettant l'implémentation des stratégies BFS, Trémaux et Hand on Wall.
   - En parallèle de la résolution, la **gestion de la base de données** a été mise en place pour la sauvegarde et le chargement des labyrinthes. Ces deux aspects ont été développés en convergence pour assurer une cohérence fonctionnelle.

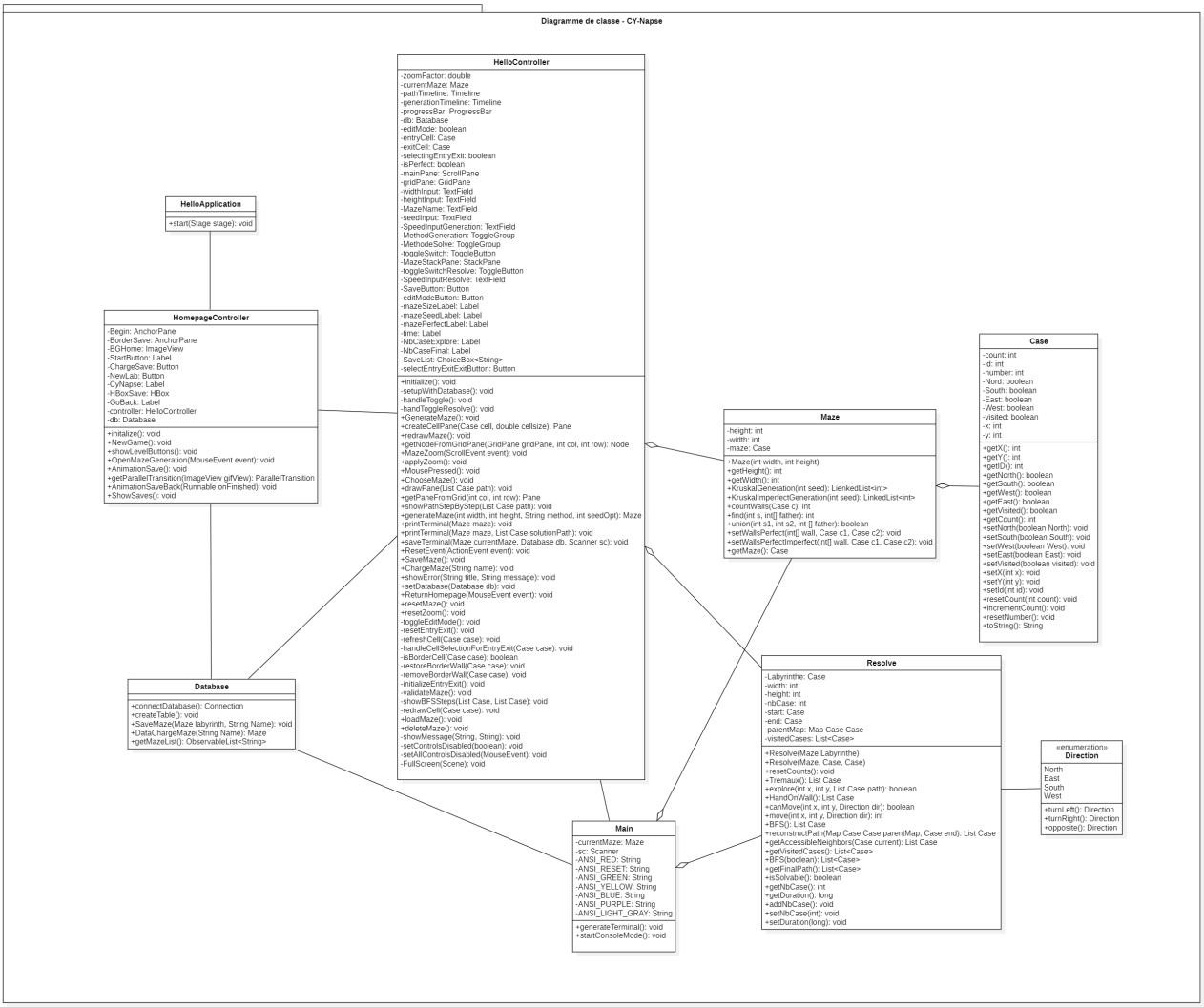
3. **Construction des Interfaces Utilisateur :**
   - Une fois les fonctionnalités métier robustes, nous avons commencé la **construction de l'interface graphique** interactive en JavaFX.
   - Une **interface en mode terminal** a également été développée en complément, offrant une alternative légère pour l'interaction et le débogage initial.

4. **Finalisation et Documentation :**
   - La phase finale a consisté à affiner le code et à garantir sa maintenabilité. Cela a inclus l'intégration de **JavaDoc** pour documenter les classes et méthodes, ainsi que l'ajout de **commentaires** explicatifs dans le code.
   - Un fichier **README** détaillé a été rédigé pour faciliter la prise en main du projet par d'autres développeurs ou utilisateurs.
   - Enfin, la préparation du présent **rapport** a été initiée, et la mise en place des **visualisations pas à pas** a été finalisée pour illustrer dynamiquement le comportement des algorithmes.

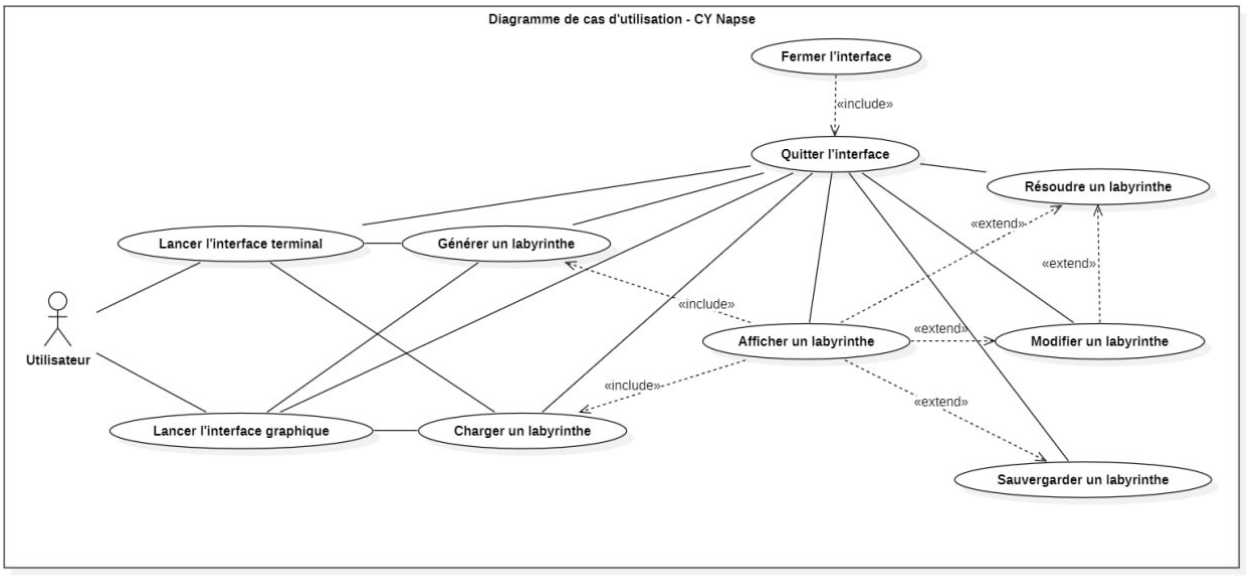
Ce flux de travail nous a permis de gérer la complexité du projet en décomposant les tâches et en assurant une progression méthodique vers l'atteinte de nos objectifs.

2 Conception technique

2.1 Diagramme de Classes



2.2 Diagramme de Cas d'utilisation



3 Problèmes rencontrés et solutions

Au cours du développement de ce projet, plusieurs défis techniques ont émergé, nécessitant une analyse approfondie et l'implémentation de solutions adaptées. Cette section détaille les principales difficultés rencontrées et les stratégies mises en œuvre pour les résoudre.

3.1 Difficultés techniques

1. **Gestion de la Persistance des Labyrinthes (Base de Données) :**
   - **Problème** : Initialement, nous avions envisagé d'utiliser **MySQL via WampServer** pour la sauvegarde et le chargement des labyrinthes. Cependant, cette approche a rapidement révélé des inconvénients majeurs en termes de déploiement et de complexité. L'installation de WampServer et la configuration de MySQL s'avéraient fastidieuses et peu optimales pour une application Java autonome, nécessitant des étapes d'installation supplémentaires pour l'utilisateur final et introduisant des dépendances externes lourdes.

2. **Intégration de Scene Builder et Complexité de l'Interface Graphique (JavaFX) :**
   - **Problème** : L'utilisation de **Scene Builder** a posé des défis inattendus. Nous avons rencontré des difficultés avec la compatibilité des versions de JavaFX (JavaFX 21 utilisé, alors que certaines fonctionnalités ou versions de Scene Builder pouvaient en nécessiter une plus récente comme la 24).
   - De plus, il était difficile de gérer des **interactions complexes directement avec les objets du labyrinthe** via FXML et Scene Builder.

3. **Problèmes d'Affichage du Chemin et du Pas à Pas :**
   - **Problème :** La visualisation dynamique de la résolution des labyrinthes, en particulier le mode "pas à pas", a été source de difficultés. Nous avons rencontré des **problèmes d'affichage du chemin**, où le tracé ne correspondait pas toujours à la logique algorithmique attendue.
   - Un défi spécifique était le "**coloriage inversé**" ou l'inversion des coordonnées (lignes pour colonnes et vice-versa) lors du rendu graphique, ce qui entraînait une représentation incorrecte des chemins.

4. **Gestion du Zoom de l'Interface :**
   - **Problème** : L'implémentation d'une fonctionnalité de **zoom** pour le labyrinthe a présenté des défis d'intégration et de performance. Assurer un zoom fluide et précis qui s'adapte correctement à la taille du labyrinthe, tout en maintenant la réactivité de l'interface et le centrage du labyrinthe agrandi, n'était pas trivial. Des ajustements constants étaient nécessaires pour éviter les décalages visuels ou les performances dégradées.

5. **Rendu Visuel des Murs (Textures) :**
   - **Problème** : Nous avons tenté d'implémenter un rendu visuel plus sophistiqué pour les murs, en essayant d'appliquer des **images de textures (par exemple, "murs en pierre")** directement sur les éléments de l'interface. Cependant, le **rendu final n'était pas visuellement satisfaisant**. Les images ne s'intégraient pas harmonieusement ce qui dégradait l'esthétique générale plutôt que de l'améliorer.

3.2 Solutions implémentées

1. **Transition vers SQLite pour la Persistance :**
   - **Solution** : Pour pallier les inconvénients de MySQL/WampServer, nous avons opté pour **SQLite**. Cette base de données embarquée a permis de simplifier considérablement la gestion de la persistance. L'intégration de SQLite est plus simple et plus directe en Java, éliminant le besoin d'installations de serveurs externes et de configurations complexes pour l'utilisateur. Cela a rendu la solution beaucoup plus portable et facile à compiler et à distribuer.

2. **Gestion Manuelle et Code Java pour l'Interface Graphique :**
   - **Solution** : Face aux limitations de Scene Builder pour les interactions dynamiques et les personnalisations visuelles poussées, nous avons choisi de **privilégier le code Java pur** pour la manipulation fine des éléments graphiques du labyrinthe (dessin des murs, coloriage des cellules, gestion des animations pas à pas).

3. **Correction des Erreurs d'Affichage et de Coordonnées :**
   - **Solution** : Pour résoudre les problèmes d'affichage du chemin et les inversions de coordonnées, une révision minutieuse de la logique de rendu a été effectuée. Cela a impliqué de **standardiser la gestion des coordonnées**.

4. **Structure Spécifique pour le Zoom et Centrage Automatique :**
   - **Solution** : Pour une gestion optimale du zoom, nous avons développé une structure spécifique au sein du contrôleur. Cela inclut des calculs dynamiques de la taille des cellules en fonction du facteur de zoom et de la dimension de la fenêtre. Un mécanisme de centrage automatique du labyrinthe a été mis en place pour garantir que même lors d'un zoom important, le labyrinthe reste visible et bien positionné au centre de la zone d'affichage.

4 Résultats et fonctionnalités

4.1 Fonctionnalités implémentées

Cette section détaille les principales fonctionnalités que nous avons réussies à implémenter dans l'application, en lien direct avec les objectifs du cahier des charges.

Génération de Labyrinthes

L'application offre une capacité robuste de génération de labyrinthes, avec les options suivantes :

- **Algorithme de Kruskal** : Les labyrinthes parfaits sont générés en utilisant l'algorithme de Kruskal, garantissant l'absence de cycles et l'accessibilité de toutes les cellules.
- **Génération Pas à Pas** : Un mode "pas à pas" est disponible pour la génération. Cela permet à l'utilisateur de visualiser en temps réel la progression de l'algorithme de Kruskal, offrant ainsi une compréhension didactique de son fonctionnement et de la construction du labyrinthe.
- **Personnalisation des Dimensions** : L'utilisateur peut définir la taille du labyrinthe (largeur et hauteur) avant sa génération, offrant une grande flexibilité dans la création de différentes complexités de labyrinthe.

Résolution de Labyrinthes

Pour la navigation et la résolution des labyrinthes, plusieurs stratégies ont été implémentées :

- **Multiples Algorithmes de Résolution** : Trois algorithmes distincts sont disponibles pour résoudre automatiquement les labyrinthes :

    - **BFS (Breadth-First Search)** : Une recherche en largeur qui garantit de trouver le chemin le plus court dans les labyrinthes non pondérés.
    - **Trémaux** : Une méthode basée sur le marquage des chemins, idéale pour la résolution de labyrinthes complexes et imparfaits.
    - **Hand on Wall (Main Gauche)** : Une stratégie simple mais efficace, souvent appelée "règle de la main sur le mur", qui permet de trouver une sortie si le labyrinthe est simplement connexe (sans cycles bloquants).

- **Visualisation Pas à Pas de la Résolution** : Pour chaque algorithme de résolution, un mode "pas à pas" est également proposé. L'utilisateur peut observer la progression de l'algorithme, case par case, ce qui permet de mieux comprendre la logique de recherche et de cheminement de chaque méthode.

Personnalisation et Interactivité

L'application offre des fonctionnalités d'édition et de personnalisation post-génération :

- **Modification Manuelle du Labyrinthe** : L'utilisateur a la possibilité d'ajouter ou de supprimer des murs intérieurs après la génération du labyrinthe, permettant de transformer un labyrinthe parfait en imparfait ou de créer de nouveaux défis.
- **Choix de l'Entrée et de la Sortie** : Les points d'entrée et de sortie du labyrinthe peuvent être définis par l'utilisateur, offrant une flexibilité pour des scénarios de résolution variés.
- **Zoom de l'interface** : L'interface graphique permet un zoom sur le labyrinthe, améliorant l'expérience utilisateur pour la visualisation de grands labyrinthes ou pour une inspection détaillée des cellules.)

Sauvegarde et Chargement

Afin de permettre la persistance des labyrinthes et leur réutilisation :

- **Sauvegarde des Labyrinthes** : L'application permet de sauvegarder l'état actuel d'un labyrinthe (y compris sa topologie et la graine utilisée pour sa génération initiale).
- **Chargement des Labyrinthes** : Les utilisateurs peuvent charger des labyrinthes précédemment sauvegardés à partir d'une liste de "seeds" enregistrées, permettant de reprendre une exploration ou une modification.
- **Gestion des Sauvegardes** : La suppression des sauvegardes obsolètes est également possible, offrant une gestion complète des labyrinthes stockés.

4.2 Captures d'écran de l'interface

4.3 Tests et validation

La robustesse et la fiabilité de l'application ont été assurées par une série de tests rigoureux.

Nous avons systématiquement testé l'application sur différentes plateformes, notamment **Windows** et **Ubuntu**, et vérifié son comportement tant en mode terminal qu'à travers l'environnement de développement **IntelliJ IDEA**. Des tests quotidiens ont été menés pour identifier rapidement tout

potentiel crash ou comportement inattendu.

Par ailleurs, nous avons exploré les **limites de l'application** en soumettant le système à des scénarios extrêmes : des tailles de labyrinthe très grandes, des résolutions complexes, et des variations importantes de la vitesse d'animation. Ces **tests extrêmes** ont permis de valider la stabilité de l'application face à des charges importantes et d'identifier les seuils de performance.

Conclusion et perspectives

Bilan du projet

Le projet **CY-Napse** a abouti au développement d'une application Java fonctionnelle et interactive, répondant pleinement aux exigences du cahier des charges. Nous avons implémenté avec succès la génération de labyrinthes (parfaits et imparfaits via Kruskal), leur résolution (BFS, Trémaux, Hand on Wall) et leur visualisation graphique et terminale. L'application permet également la modification manuelle et la persistance des labyrinthes (sauvegarde/chargement).

L'équipe a démontré une organisation efficace et une communication fluide tout au long du projet. Les défis techniques rencontrés ont été surmontés, et la robustesse de l'application a été validée par des tests réguliers sur diverses plateformes. Ce projet représente une réussite dans la conception et l'implémentation d'une solution logicielle complexe.

Amélioration possibles

Au-delà des objectifs initiaux atteints, plusieurs pistes d'évolution pourraient enrichir ce projet :

- **Rendu Visuel** : Introduire des textures plus détaillées pour les murs (par exemple, "murs en pierre") afin d'améliorer l'esthétique générale du labyrinthe.
- **Expérience Utilisateur** : Affiner l'interface graphique pour la rendre encore plus intuitive et agréable à utiliser, notamment par une meilleure réactivité et des retours visuels plus clairs.
- **Diversification des Algorithmes** : Intégrer un quatrième algorithme de résolution (comme A*) pour offrir plus de méthodes d'exploration et de comparaison.
- **Mode "Jeu" Interactif** : Développer un mode où l'utilisateur pourrait contrôler un personnage pour naviguer manuellement dans le labyrinthe, transformant l'application en une expérience ludique.

Ces axes représentent des prolongements naturels pour faire évoluer l'application vers des fonctionnalités plus avancées et une immersion accrue.

Annexe

Ressources du projet

- **Dépôt Git :** Github
- **Diagramme de Gantt :** Diagramme de Gantt
- **Fichier UML pour les diagrammes :** Diagramme UML

Bibliographie

- **Patrons de conception :** W Patrons de conception
- **Sérialisation :** W Sérialisation
- **Connexion à une base de données :** JDBC
- **Algorithmes de génération de labyrinthe :** W Maze generation algorithm
- **Algorithmes de résolution de labyrinthe :** W Maze solving algorithm
- **Documentation JavaDoc :** JavaDoc
- **Exemples BFS en Java :**
  - Vidéo BFS 1
  - Vidéo BFS 2