RAPPORT PROJET GÉNIE LOGICIEL

CYnapse

ING1-GI3 | Groupe 2

Année universitaire 2024–2025

Encadrant :

Dre.**ANSERMIN**

Réalisateurs:

Abdellah **DJEDIDI**

Melina **GARUFI**

Amâncio **ZANI DA SILVA**

Shawrov **DAS**

Jeremy **PERBOST**

1) Organisation de l’équipe

Dans un premier temps, conformément aux recommandations de notre tutrice, nous avons instauré un système de réunions quotidiennes. Nous avons ainsi organisé plusieurs réunions régulières, notamment :

- Réunion du 11/05 : Vérification que chaque membre de l’équipe dispose d’une version fonctionnelle et compatible du projet.

- Réunion du 12/05 : Répartition des tâches et suivi de l’avancement du projet.

Ces réunions se sont principalement tenues en ligne, via la plateforme Discord. Cependant, nous avons rapidement abandonné ce format pour plusieurs raisons :

- La qualité médiocre du partage d’écran sur Discord,

- Les limites d’interaction et de communication inhérentes au travail à distance.

Nous avons privilégié des rendez-vous en présentiel dans le local Atilla. Ces rencontres avaient essentiellement pour objectif de prioriser les nouvelles tâches nécessaires à l’avancement du projet et permettre la création des moments de collaboration mutuelle où plusieurs membres ont pu collaborer dans la solution d’une tâche ardue. Par ailleurs, les questions techniques relatives au fonctionnement logique et à la programmation étaient discutées sur le canal Telegram.

De plus, Nous avons mis en place un système de TO-DO Liste qui nous permettait d’identifier les tâches à faire et en train d’être réalisées. Cette Liste nous a été fortement utile et nous a permis d’être efficace dans la communication, l'organisation et l’exécution des tâches: À travers le support visuel des tâches majeurs à accomplir, nous avons pu développer le réflexe de segmenter ces dernières en des tâches mineures, plus simples à coordonner et à réaliser. La liste a adopté un format minimaliste : un document partagé Google doc et était donc modifiable par tous les membres en temps réel.

Au départ, chaque membre disposait de sa propre branche sur Github. L’objectif était que chacun réalise ses modifications sur sa branche respective, avec une fusion quotidienne de toutes les branches dans la branche principale (main). Toutefois, cette méthode s’est révélée trop complexe et consommatrice en temps, car des fonctionnalités non achevées finissaient par être intégrés à la branche principale. Nous avons donc opté pour une mise à jour alternée de la branche principale. En cas de modifications simultanées par plusieurs membres sur la branche main, les autres développeurs devaient adapter leur travail en fonction de la dernière version disponible sur la branche principale.

Rôle des membres

- **Jérémy :** Rédaction du rapport, élaboration du fichier README, modification du graphe et du labyrinthe, résolution de divers bugs, dans le projet, ainsi que de la planification et la restructuration de plusieurs fichiers pour atteindre les exigences de l'architecture MVC.
  Créateur de la charte graphique du projet et de même que la création des méthodes graphiques permettant à l'application de représenter l'interface graphique du labyrinthe.
  Ajout des modificateurs de vitesses sur les animations de résolution, de la génération graphique du labyrinthe et s'appuyant sur l'algorithme de Kruskal.

- **Abdellah** : Ajout de la résolution User et A*, élaboration de la version terminal du projet, des diagrammes de classes et leur mise à jour au fil du temps, création du launcher permettant de choisir entre terminal ou l'interface graphique, rédaction de la javadoc, modification et optimisation de l'interface pour agrandir le labyrinthe, édition du fichier de sauvegarde des fichiers et de leur chargement. Génération graphique du labyrinthe en utilisant l'algorithme de parcours en profondeur DFS.

- **Felipe :** Mise en place de l'environnement de build Gradle, création de la base du projet avec les classes Graph.java et Edge.java, accompagnée du développement du module de génération de graphes, incluant une génération parfaite à travers l'algorithme de Kruskal qui crée un graphe en mémoire, ainsi que la génération imparfaite des labyrinthes pour des scénarios plus réalistes. Parallèlement, les fondations du solveur utilisant l'algorithme BFS ont été établies, ainsi que l'interface utilisateur pour l'animation des graphes. Enfin, une fonctionnalité a été ajoutée permettant à l'utilisateur de définir un chemin personnalisé pour le solveur.

- **Shawrov** : Définition des points de départ et d'arrivée et aussi l'algorithme derrière pour trouver leur localisation dans labyrinthe, intégration des algorithmes de génération et de résolution aléatoires, gauches, droites, et de recherche en largeur (BFS), ainsi que l'affichage des étapes de résolution avec ou sans animations. Responsable pour l'optimisation de la génération des chemins de solution du labyrinthe.

2) Problèmes et solutions apportés

Problème de performance au niveau de l'animation de la résolution des graphes :
Nous avons constaté que l'animation de la résolution des graphes provoque des plantages de notre programme. Après analyse, nous avons identifié que ce dysfonctionnement était lié au fait que l'animation était directement intégrée dans les algorithmes de résolution. Pour remédier à ce problème, nous avons séparé la logique de résolution de la logique de visualisation. Autrement dit, le chemin est désormais calculé en amont de manière rapide et sans animation, puis l'animation est lancée ensuite indépendamment de l'algorithme de résolution.

Un autre bug que l'on peut relever sur le logiciel est lors de la résolution du labyrinthe. Lorsque la souris va passer sur le container dédié au labyrinthe durant la résolution, le hover va empêcher et annuler la résolution. La solution apportée est d'effectuer une vérification sur l'état de l'animation. Si l'animation existe, le programme va afficher la résolution tout en effaçant les modifications potentielles de l'utilisateur (hover de la souris).

Des problèmes de structurations au niveau du code ont été identifiés. Notamment des parties de codes extrêmement longues. Par exemple, certains fichiers java avaient des centaines de lignes. Notre solution a été d'utiliser une composition simple du code. Ainsi, nous avons séparé les méthodes en catégories. Ces mêmes méthodes sont toujours appelées par les autres classes. Cette technique a été utilisée avec parcimonie, car elle rajoute du code et des classes dans le projet en général (Toutes les méthodes doivent être écrites au moins deux fois parce que la classe originale est séparée en deux), mais cela permet d'avoir un code plus structuré qui respecte le principe de factorisation en Java.

3) Limites fonctionnelles

Les limites que l'on peut apercevoir dans notre logiciel tout d'abord dans l'espace de l'écran sont la visualisation du labyrinthe. On a arrangé l'espace de sorte que le labyrinthe soit le plus visible possible. Pour des labyrinthes de 75*75 ou de 100*100, le labyrinthe est assez visible, mais placer des sommets sur le labyrinthe devient compliqué, car le hover de la souris va faire la confusion entre les arêtes possiblement modifiables et les sommets de départ et d'arrivée qu'on souhaite choisir. La génération de labyrinthes devient, elle aussi, compliquée pour des valeurs de lignes et de colonnes trop hautes (de l'ordre du 100*100). La génération va marquer un temps de pause d'environ deux secondes avant d'afficher le labyrinthe généré. Dans le cas où on choisit la génération step by step, le logiciel crash.
Enfin, pour la résolution, on peut relever que pour un graphe de 150*150, le logiciel fonctionne correctement avec un temps d'attente de génération minimale.

4) Planning de réalisation

Nous travaillons principalement à distance sur ce projet, avec des sessions ponctuelles en présentiel dans le local Atilla. Ainsi, le projet n'a jamais été interrompu, à l'exception des périodes de repos comprises entre minuit et environ 10 heures du matin.

Chronologie du développement

- **Du 28 avril au 10 mai 2025**
  Mise en place de l'environnement de développement, incluant l'installation de Gradle, Java, ainsi que des outils et dépendances nécessaires à la réalisation du projet.

- **Du 2 mai au 10 mai**
  Début du développement technique :

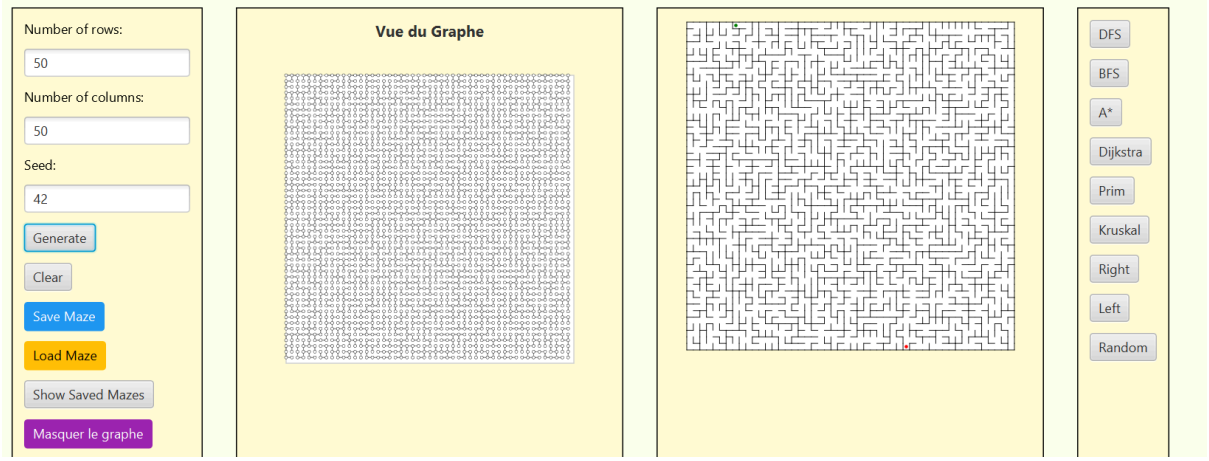
  - Création des classes principales `Graph` et `Edge`,
  - Génération automatique du labyrinthe,
  - Implémentation de la visualisation du graphe et du labyrinthe.

- **Lundi 11 mai**
  Première réunion avec notre tutrice, Mme Eva Ansermin. Cette rencontre d'1h30 a permis de clarifier plusieurs éléments clés du projet et d'établir de nouvelles orientations de travail.

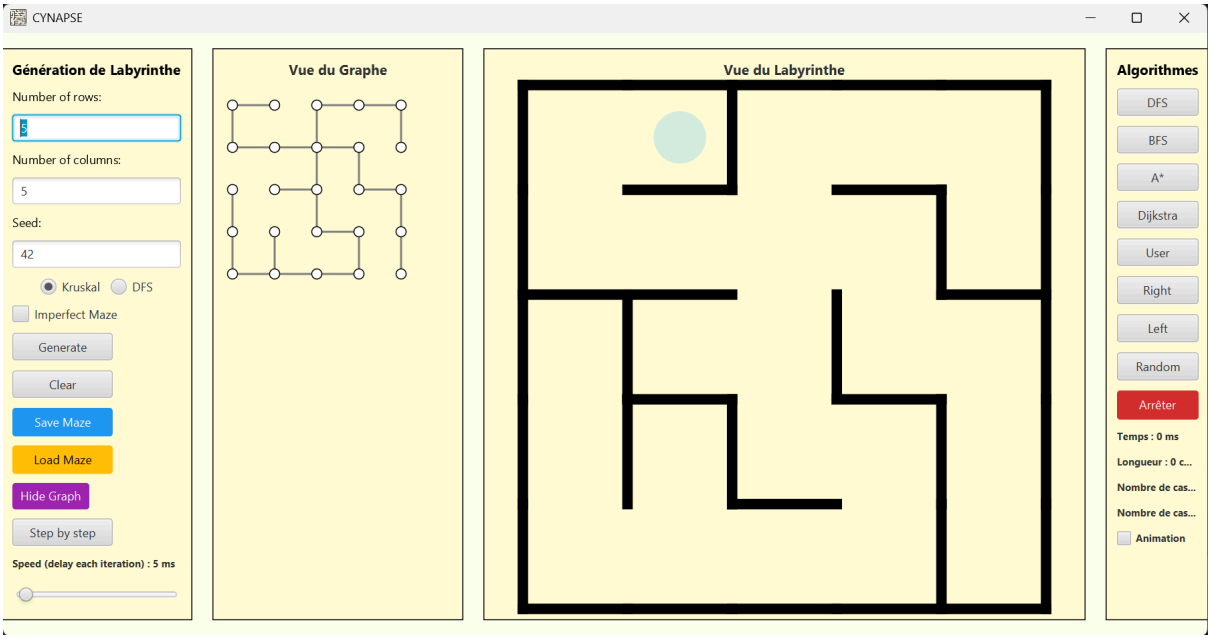
- **Du 11 au 14 mai**
  Intégration des recommandations issues de la réunion :
    - Ajout d'un bouton pour afficher ou masquer le graphe,
    - Mise en place d'un système de sauvegarde du labyrinthe,
    - Gestion dynamique de la taille du labyrinthe,
    - Ajout d'un mécanisme permettant de définir un point de départ et un point d'arrivée.
- **15 mai**
    - Restructuration partielle du code,
    - Implémentation de l'algorithme de résolution **DFS (Depth-First Search)**,
    - Ajout d'un bouton pour supprimer une sauvegarde existante,
    - Développement d'une fonctionnalité de **génération pas à pas** du labyrinthe via **DFS** ou **Kruskal**.
- **16 mai**
    - Intégration de l'algorithme de résolution **BFS (Breadth-First Search)**.
- **17 mai**
    - Ajout de plusieurs algorithmes de résolution supplémentaires :
      A*, **Dijkstra**, **Right-hand rule**, **Left-hand rule**, et **Random walk**,
    - Mise en place de la **modification en temps réel** du graphe et du labyrinthe.
- **21 mai**
    - Améliorations de l'interface utilisateur (UI),
    - Génération de **labyrinthes imparfaits**,
    - Rédaction de la **JavaDoc** pour la documentation du code source.

5)Images d’archives

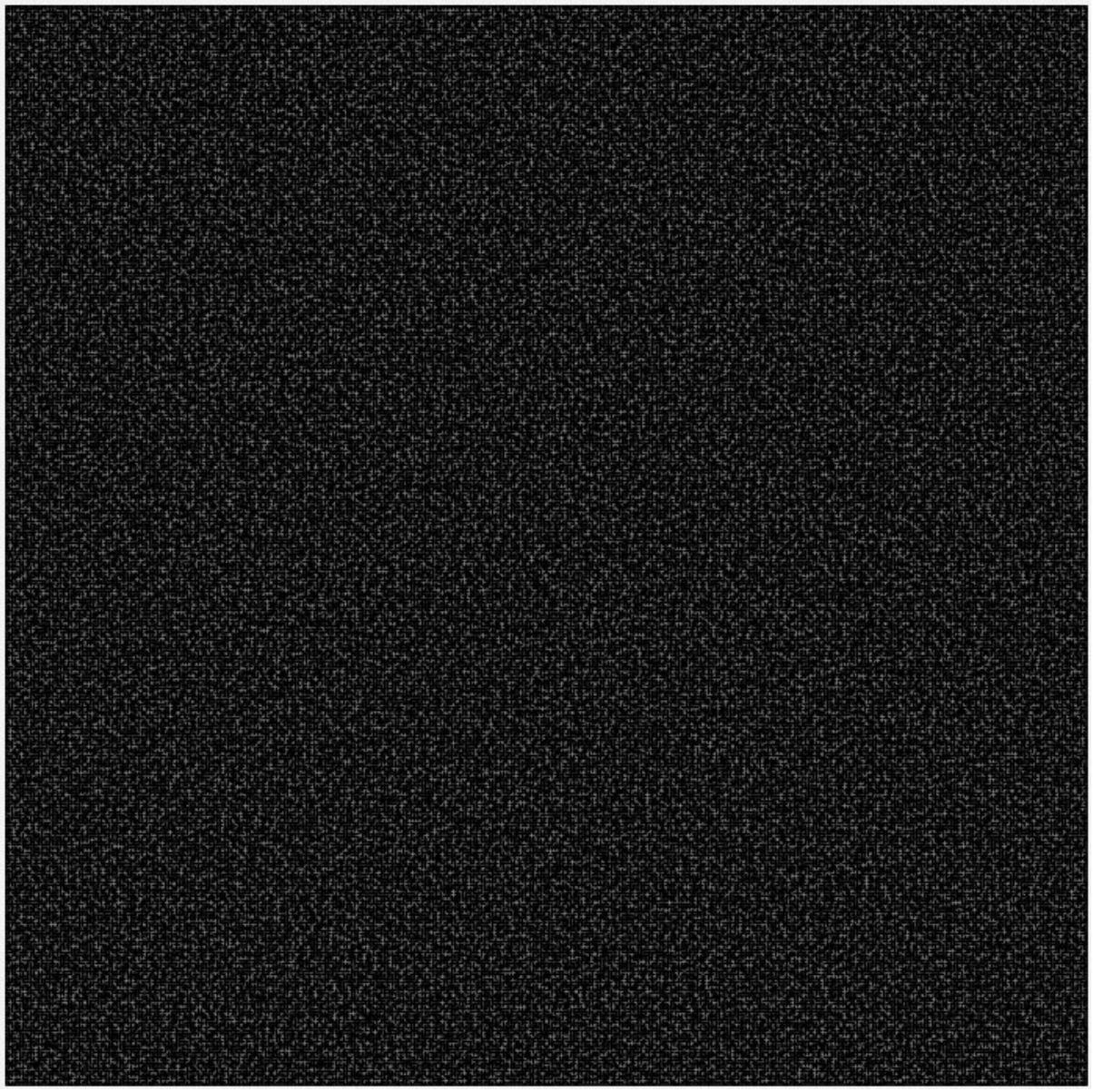
Avancement au 14/05 :



Avancement au 22/05 :



Cette image de Labyrinthe 800*800 témoigne de la stabilité de notre programme face aux lourdes générations. Bien évidemment, le rendu final limite l’utilisateur à la génération de Labyrinthe 100*100.



TO-DO LISTE

LISTE DE CHOSES À FAIRE

- ~~Set up project pour tout le monde~~
- ~~Avoir un Labyrinthe dessiné~~
- Passer le labyrinthe dessiné en diagramme de classes+~~MVC~~
- ~~Regrouper tous les données du labyrinthe dessiné dans une classe~~
- ~~Séparer la partie UI de la partie Logique~~
- Algo pour extraire un graphe du labyrinthe dessiné (récolté les cases vides)
- ~~Saisir Dimensions~~
- ~~Saisir Seed~~
- ~~Saisir vitesse de Création~~
- ~~Ajouter point de départ et d'arrivée~~
- ~~Algorithmes de résolution~~
    - ~~DFS~~
- Statistiques Résolution(temps de résolution)
- ~~Ajouter le dfs pour la création du labyrinthe~~
- ~~Modifications du labyrinthe~~
    - ~~Changer point de départ/arrivé~~
    - ~~Modifier topologie~~
        - ~~Ajouter/enlever Murs~~
        - ~~Resize labyrinthe~~
- ~~Sauvegarde du lab(MVP:On peut sauvegarder seed et c fini)~~
- Tout le terminal retranscris le projet
- ~~Sauvegarde du graph meme quand il est modifier~~
- possibilité de modifier le nom d'une sauvegarde
- ~~Pouvoir générer un graph non parfait (graph aléatoire)~~
- ~~régler le bug quand on lance plusieurs algos de résolutions a la fois~~
- ~~IMPORTANT !!! Patcher la résolution qui rame et fait planter le pc quand il y a trop de cases !~~
- ~~Créer un mode complet pour la résolution du labyrinthe : *C'est à dire que il n'y a pas d'animation lors de la résolution*~~
- ~~Le nombre de cases du chemin final,~~
- Le nombre de cases traitées,

- ~~Créer des arretes et des sommets directement en cliquant dessus~~

Jérémy en cours

Felip

Verification de choses faites :

~~Generation par graine~~
~~saisir dimentions~~
JavaDoc de l'algorithme de generation du graph : nope
JavaDoc de l'algorithme de generation du maze : nope
~~génération labyrinthe parfait~~
~~génération labyrinthe imparfait~~
Saisie imparfaite UI
génération pas à pas
~~génération complet~~
Bloquer demande si pas des points start et end fixés avant
Avoir nombre de cases chemin final, nombre des cases visités au total
~~modification topologie du labyrinthes~~
~~sauvegardes labyrinthe avec ou sans modifs~~
~~load et continuation de modification du labyrinthe~~

Cahier des charges :
JavaDoc de l'algorithme de génération du graph
JavaDoc de l'algorithme de génération du maze
~~Bloquer demande si pas des points start et end fixés avant~~
Avoir nombre des cases visités au total

Bugs trouvés:
~~Pas de point d'arrivé et de départ ça crash~~
~~Checkbox créer maze pas parfait non functional~~
vue du graph non modifiable après avoir tapé sur le bouton "generate"
~~BFS - Felipe il déconne x~~

Bonus:
Essayer de deviner la demande de changement du programme:
tirer la mise de début et arête que sur les bords

6) Diagrammes du système

Les diagrammes qui ont été créés durant le projet pourront être visibles en meilleure qualité dans le dép