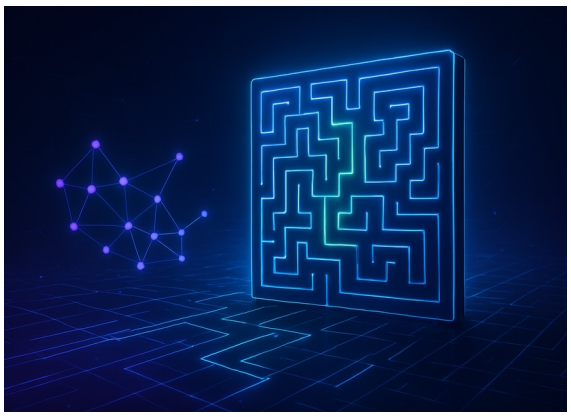

v1.1 Projet CYnapse

FILIERE ING1-GI • 2024-2025
AUTEURS E.ANSERMIN – R. GRIGNON
E-MAILS eva.ansermin@cyu.fr – romuald.grignon@cyu.fr

DESCRIPTION DU PROJET

- Le but de cette application est de créer une interface graphique permettant de générer puis de résoudre des labyrinthes parfaits.
  La génération sera procédurale, ce qui permettra de rejouer la création d'un labyrinthe plusieurs fois.
  La résolution fera appel à plusieurs algorithmes que l'utilisateur pourra choisir.
  Enfin, l'utilisateur pourra sélectionner graphiquement une position du labyrinthe pour qu'une modification puisse être appliquée localement. La résolution devra alors s'adapter dynamiquement à cette modification sans pour autant recalculer l'ensemble du parcours si cela n'est pas nécessaire.
- Ce projet met l'accent sur l'algorithmique des graphes non orientés et acycliques.

INFORMATIONS GENERALES

- **Taille de l'équipe**
  Ce projet est un travail d'équipe. Il est préférable de se réunir en groupe de 5 personnes. Si le nombre d'étudiants n'est pas un multiple de 5 et/ou si des étudiants n'arrivent pas à constituer des groupes, c'est aux chargés de projet de statuer sur la formation des groupes. Pensez donc à anticiper la constitution de vos groupes pour éviter des décisions malheureuses.

- **Démarrage du projet et jalons**
  Vous obtiendrez de plus amples informations quant aux dates précises de rendu, de soutenance, les critères d'évaluation, le contenu du livrable, ..., quand le projet démarrera officiellement.
  Quel que soit le planning initial du projet, vous veillerez à planifier plusieurs rendez-vous d'avancement avec votre chargé(e) de projet. C'est à votre groupe de prendre cette initiative. L'idéal étant de faire un point 1 ou 2 fois par semaine mais cette fréquence est laissée libre en fonction des besoins identifiés avec le tuteur de projet.

- **Versions de l'application**

  Pour éviter d'avoir un projet non fonctionnel à la fin, il vous est demandé d'avoir une version en ligne de commande fonctionnelle. Ceci vous permettra de tester votre modèle de données indépendamment de l'interface graphique. C'est la version avec l'interface graphique JavaFX qui sera bien entendu évaluée mais dans le cas où certaines fonctionnalités ne seraient pas visibles avec cette interface, vous devez pouvoir présenter toutes les fonctionnalités de votre code Java en ligne de commande.

- **Dépôt de code**

  Vous devrez déposer la totalité des fichiers de votre projet sur un dépôt central Git. Il en existe plusieurs disponibles gratuitement sur des sites web comme ***github.com*** ou ***gitlab.com***. La fréquence des commits sur ce dépôt doit être au minimum de 1 commit / 2 jours.

- **Rapport du projet**

  Un rapport écrit est requis, contenant une brève description de l'équipe et du sujet. Il décrira les différents problèmes rencontrés, les solutions apportées et les résultats. L'idée principale est de montrer comment l'équipe s'est organisée, et quel était le flux de travail appliqué pour atteindre les objectifs du cahier des charges. Le rapport du projet peut être rédigé en français ou en anglais.

  Ce rapport contiendra en plus les éléments techniques suivants : un document de conception UML (diagramme de classe) de votre application, et un document montrant les cas d'utilisations.

- **Démonstration**

  Le jour de la présentation de votre projet, votre code sera exécuté sur la machine de votre chargé(e) de TD. La version utilisée sera la **dernière** fournie sur le dépôt Git **avant** la date de rendu. Même si vous avez une nouvelle version qui corrige des erreurs ou implémente de nouvelles fonctionnalités le jour de la démonstration, c'est bien la version du rendu qui sera utilisée.

  En parallèle, il vous faudra obligatoirement une deuxième machine avec votre application fonctionnelle car une partie de votre groupe aura des modifications de code à faire pendant que l'autre partie fera la présentation/démonstration de votre projet.

- **Organisation de l'équipe**

  Votre projet sera stocké sur un dépôt git (ou un outil similaire) tout au long du projet pour au moins trois raisons :

  - éviter de perdre du travail tout au long du développement de votre application
  - être capable de travailler en équipe efficacement
  - partager vos progrès de développement facilement avec votre chargé(e) de projet.

  Il est recommandé de mettre en place un environnement de travail en équipe en utilisant divers outils pour cela (Slack, Trello, Discord, …).

CRITERES GENERAUX

- Le **but principal** du projet est de fournir une **application fonctionnelle** pour l'utilisateur. Le programme doit correspondre à la description en début de document et implémenter toutes les fonctionnalités listées.

- Votre code sera **commenté** de manière adéquate.
- Tous les éléments de **votre code** (variables, fonctions, commentaires) seront écrits **en anglais** obligatoirement.
- Votre code devra être commenté de telle manière que l'on puisse utiliser un outil de génération de documentation automatique de type **JavaDoc**. Un dossier contenant la doc générée par vos soins sera présent dans votre dépôt de code lors de la livraison.

- Votre projet doit être utilisable au clavier ou à la souris en fonction des fonctionnalités nécessaires.

- Votre application ne doit jamais s'interrompre de manière **intempestive** (crash), ou tourner en boucle indéfiniment, quelle que soit la raison.
  Toutes les erreurs doivent être gérées correctement. Il est préférable de d'avoir une application stable avec moins de fonctionnalités plutôt qu'une application contenant toutes les exigences du cahier des charges mais qui plante trop souvent.
  Une application qui se stoppe de manière imprévue à cause d'une exception, par exemple, sera un événement très pénalisant.

- Votre application devra être organisée en modules afin de ne pas avoir l'ensemble du code dans un seul et même fichier par exemple. Apportez du soin à la conception de votre projet avant de vous lancer dans le code.

- Le livrable fourni à votre chargé(e) de TD sera simplement l'**URL** de votre **dépôt Git** accessible **publiquement**.

FONCTIONNALITES DU PROJET

GENERALITES

- La génération d'un labyrinthe devra se faire sur décision de l'utilisateur à partir d'une graine qu'il aura choisie.
  Les dimensions sont aussi des données d'entrée de l'utilisateur (on peut limiter la zone du labyrinthe à un ensemble de cases carrées).
  L'algorithme de génération est laissé libre mais il devra être clairement identifié et documenté (structures, pseudo-code, schémas d'étapes de résolution, ...).

- Le labyrinthe généré peut être :
  - soit **parfait**, c'est-à-dire qu'il n'existe qu'un et un seul chemin possible entre 2 positions quelconques de la zone. Il ne devra pas y avoir avoir de cycle dans ce labyrinthe, ni de zone non atteignable.
  - soit ne pas l'être, et dans ce cas, des cycles peuvent apparaître. Il est également possible qu'il soit sans solution.

  Ce choix de génération sera effectué par l'utilisateur.

GENERATION

- Deux modes de génération doivent coexister :
  - mode **complet** : la génération se fait en tâche de fond et l'affichage sera produit une fois le labyrinthe complètement créé. Pendant la génération, une animation pour faire patienter l'utilisateur et lui indiquer qu'un traitement est en cours peut être utilisée.
  - mode **pas à pas** : les phases de génération et d'affichage ne sont pas séquentielles : au fur et à mesure que le labyrinthe est créé, l'affichage se met a jour.
    Un réglage de la "vitesse" de création (durée d'une étape de modification de la topologie) doit être proposé pour visualiser la création pendant son déroulement (et surtout avoir un visuel du comportement de l'algorithme).

- Si le mode pas à pas est implémenté, une très faible valeur de durée d'étape (voire nulle) peut suffire à implémenter le mode complet. Par contre le mode complet doit tout de même pouvoir être présenté à minima.

RESOLUTION

- On ne peut pas demander une résolution si il n'y a pas eu de génération avant. L'application doit pouvoir permettre le choix entre plusieurs algorithmes de résolution (au moins 3).

- A partir d'un algorithme de résolution de labyrinthe, on souhaite pouvoir visualiser le résultat. Ici aussi les deux modes (**complet** et **pas à pas**) doivent être implémentés (le mode complet à minima).

- Lors de l'affichage, on veut que chaque case du labyrinthe qui a été traitée soit affichée d'une certaine couleur, et les cases qui correspondent au chemin final avec une autre couleur, de manière à identifier le travail effectué par l'algorithme.

- Le nombre de cases du chemin final, le nombre de cases traitées, ainsi que le temps de génération, doivent être visibles également.

MODIFICATION

- Quand un labyrinthe a été créé, on veut pouvoir modifier ponctuellement sa topologie, c'est-à-dire ajouter un mur ou en supprimer un. Cette modification doit permettre de créer un nouveau labyrinthe que l'on peut à nouveau résoudre.

SAUVEGARDE / RESTAURATION

- Lorsqu'un labyrinthe a été généré, que ce soit via une graine, ou après de multiples modifications manuelles, l'application doit pouvoir proposer de le sauvegarder sur un fichier du disque dur.
  Le format du fichier de sauvegarde est laissé libre.

- L'application doit donc permettre de charger un labyrinthe précédemment sauvegardé et de continuer à l'utiliser comme si il venait d'être créé.

- Cette fonctionnalité se sépare en 2 étapes : sauvegarde puis restauration → si elle n'est pas fonctionnelle il faut que vous puissiez tout de même faire valider que l'une ou l'autre de ces étapes (sauvegarde ou restauration) est fonctionnelle.

RESSOURCES UTILES

- **Github**
  https://www.github.com
  https://docs.github.com/en/get-started/quickstart/hello-world
- **Patrons de conception**
  https://fr.wikipedia.org/wiki/Patron_de_conception
- **Sérialisation**
  https://fr.wikipedia.org/wiki/Sérialisation
- **Connexion à une base de données**
  https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html
- **Génération de labyrinthe**
  https://en.wikipedia.org/wiki/Maze_generation_algorithm
- **Résolution de labyrinthe**
  https://en.wikipedia.org/wiki/Maze-solving_algorithm