Malleret Maxence
Adnet Paul

2022-2023

Projet Labyrinthe Java

Institut des sciences du Digital, Management Cognition
License 3 Miashs

L’architecture du projet

Pour ce projet, nous avons décomposé le logiciel en différentes partie / classes :

- **Main.java**, qui n’est autre que la Frame pour le logiciel complet et la classe contenant la méthode main permettant de tout lancer.

- **Case.java**, la classe représentant l’objet “Case”. Cette objet est au centre du labyrinthe puisque ce dernier n’est autre qu’un tableau à 2 dimensions de cases. Une case est composée de coordonnées, de 4 murs (nord, sud, est et ouest) et d’une valeur.

- **Jeu.java**, la classe dite “back-end” puisque c’est dans cette classe que nous créons à proprement parler le labyrinthe avec ce fameux tableau 2D de cases. Plusieurs méthodes sont présentes dans cette classe pour permettre de créer et résoudre ce labyrinthe.

- **GUIMaze.java** qui est une classe qui étend JPanel, ce n’est autre que la visualisation du labyrinthe fait dans Jeu.java. Il affiche les différents murs et les différentes cases en fonction des valeurs.

- **Visuel.java**, enfin, est le principal panel de la frame, c’est ici que tous les boutons sont présents avec leurs différentes fonctions.

l’algorithme de génération de labyrinthe

Pour l’algorithme de création du labyrinthe, nous avons hésité entre 2 différents : L’algorithme Depth First Search (ou parcours en profondeur en français) et l’algorithme de fusion aléatoire de chemin (nous avions d’ailleurs anticipé le fait de faire les deux en même temps, puisqu’une Combobox avait été préparé pour pouvoir créé le labyrinthe soit par l’un soit par l’autre). Finalement nous avons choisi de partir sur l’algorithme Depth First Search.

Les choix techniques

Pour les données à stocker, un tableau à 2 dimensions de **Case** à suffit pour représenter notre labyrinthe. Pour l’algorithme Depth First Search, la récursivité n’a pas été utilisée mais à la place une **Arraylist** stockant tous les cases visités dans l’ordre a permis de se souvenir des mouvements effectués (une Queue aurait pû être utilisé ici, mais le choix a été fait de rester sur une Arraylist étant plus à l’aise avec les méthodes de l’interface List).
Pour les technologies de l’IHM, nous avons utilisé **Java Swing** pour tout afficher grâce à la fonctionnalité de dessin possible et aux différents composants déjà existant tels que les boutons, les checkboxs ou encore les combobox tous très utiles.

Les difficultés rencontrées

Maxence a rencontré un problème durant le début du projet, après avoir commencé à implémenter l'algorithme de fusion aléatoire de chemin, il n'arrivait pas à cerner les détails de ce dernier. C'est pourquoi il a préféré partir sur l'algorithme Depth First Search qu'il comprit mieux et plus rapidement.
Paul, quant à lui, a rencontré des difficultés à cerner la logique pour la résolution du labyrinthe. Heureusement ça n'a pas eu tant d'impact que cela sur le projet puisque Maxence a pu finir sa partie à temps pour finaliser le projet.

Les ressources utilisées

Nous avons principalement utilisés les ressources du cours ainsi que certains site web tel que :

Ce blog m'a permis de trouver une fonction simple pour passer un JPanel Swing en BufferedImage pour permettre la bonne sauvegarde en png du labyrinthe.
http://blog.gtiwari333.com/2012/04/java-capturesave-image-jframe-jpanel.html

Cette page wikipédia expliquant la logique derrière l'algorithme Depth First Search pour la création et la résolution du labyrinthe.
https://en.wikipedia.org/wiki/Depth-first_search

La répartition des tâches

Paul s'est chargé de la résolution du labyrinthe, alors que Maxence s'est occupé de la création de ce labyrinthe, de l'Interface Homme-Machine du projet en Swing ainsi que ce document final.