Rapport de projet
Programmation Objet Avancée

M1 MIAGE – Initial

Zaher Boudhaouia

Célia Briaire

Ndriana Randrianandrasana

Pour ce projet nous devions développer les générateur, solveur, solution checker et interface graphique pour le jeu infinity loop.

La structure

L'architecture de code nous a été fournie au départ donc nous nous en sommes servie. Cependant, nous n'en avons que partiellement fait usage. Certaines méthodes et classes n'ont pas été utilisées mais nous avons implémenter des méthodes qui nous semblaient plus utiles pour nos idées et démarches dans le code.

Nous avons la structure proposée en séparant les classes en packages pour permettre une meilleure visibilité pendant la programmation. Nos packages finaux sont : checker, generateur, GUI, main, modele, solveur et vue.

Le générateur

Pour pouvoir implémenter la méthode *generateLevel()* on a commencé par implémenter des méthodes *haut(), bas(), gauche(), droite(), hautGauche(), hautDroit(), basGauche(), basDroit()* et *milieu()* renvoyant des tableaux des pièces possibles pour chaque position sur une grille.

Une méthode *generateInitBoard()* qui remplit la grille grâce aux méthodes mentionnées plus haut. On récupère les pièces vides de la grille et on y met des pièces adaptées selon les positions. Pour vérifier s'il manque une pièce est manquante dans la grille – et donc qu'il faut en mettre une – on utilise les méthodes booléennes de la classe Piece.

Pour s'assurer du caractère aléatoire des orientations de chaque pièce lors de la création de la grille initiale, on parcourt la nouvelle grille en tournant chaque pièce avec la méthode *turn()* de la classe Piece. On effectue ceci dans une méthode *mixed()*.

A partir de ces méthodes intermédiaires, on a pu écrire *generateLevel()* qui génère une grille de niveau avec des pièces choisies et orientées de manière aléatoire.

Le solution checker

Le checker est formé d'un package contenant la classe Checker. Elle contient une grille Grid sur laquelle on peut utiliser deux méthodes booléennes : *isSolution()* et *isConnected().*

La première teste si la grille est une solution directement. Elle permet d'éviter des tests supplémentaires dans le cas où la grille générée est parfaitement orientée et est donc une solution sans avoir besoin de la résoudre.

La seconde prend en paramètre une pièce. Elle est utilisée dans le cas où *isSolution()* retourne false. On teste si la pièce en paramètre est connectée correctement dans tous ses quatre cotés dans le cas quand elle n'est pas en bordure de grille, ses deux cotés quand elle est dans un angle et ses trois cotés quand elle est sur une bordure.

Le solveur

Notre classe Solver est composée d'une pile de pièces, d'une grille et une méthode *solve()* renvoyant un type Grid. Nous avons choisi d'implémenter le solveur dans une approche exhaustive. On part d'une pile de pièces. On considère qu'une pièce et son voisinage. Ce voisinage peut être fixé ou non. C'est-à-dire que les pièces voisines sont dans les seules orientations compatibles avec la case et ses voisins considérés. On teste une pièce jusqu' à ce que ce qu'elle soit fixée (on la remet dans la pile si elle n'est pas fixée). On dépile pour passer à la pièce suivante et on réalise le même processus ainsi de suite jusqu'à vider la pile. En fin de boucle, toutes pièces sont fixées et le jeu est résolu ou la grille est insolvable. On renvoie la grille résolue, on affiche "la grille est SOLVED" ou on renvoie null si elle ne l'est pas.

Après tests, il ressort que notre méthode ne fonctionne pas dans tous les cas. Il semble que nous n'ayons pas pris en compte tous les cas de figure possibles et certaines grilles solvables ne sont pas résolue complètement. La méthode est néanmoins laissée en commentaire pour pouvoir la continuer si l'on en a envie.

L'interface graphique

On a créé une interface graphique permettant au joueur d'interagir avec l'interface et cliquer sur les pièces pour les roter et résoudre le niveau. Pour permettre un débogage simplifié et une facilité pour suivre la progression de résolution, la ligne commune entre deux pièces est colorée en rouge lorsque les deux pièces ne sont pas connectées correctement et en vert lorsqu'elles le sont.

Pour ce faire, nous avons créé une classe FrmLoop qui hérite de JFrame et implémente l'interface ActionListener. Trois méthodes sont implémentées : *drawGrid(), addColor(), actionPerformer().*

La première récupère la grille et créer des boutons cliquables pour chaque pièce. La méthode *addColor()* est appelée pour colorer les connexions de la nouvelle grille dessinée.

La deuxième utilise le checker et met en rouge les lignes entre deux pièces mal connectées et met en vert les lignes entre deux pièces bien connectées.

La dernière prend en paramètre un ActionEvent qui dans notre cas sera un clic gauche de la souris et tourne la pièce dont ce sont les coordonnées et la tourne de 90°. La méthode *addColor()* est appelée à la fin pour mettre à jour la couleur des lignes de connexion qu'il faut.

Nous n'avons pas pu faire la récupération d'arguments par la ligne de commande. Pour contourner le problème, il est possible de fixer les valeurs grâce à entrées clavier qui sont demandées à l'utilisateur dans le générateur.

Déroulement du projet

Concernant le travail d'équipe, nous avons utilisé un répertoire Github seulement comme ressource partagée. En effet, nous avons préféré ne pas faire de branches individuelles parce que on a rencontré des difficultés avec les différents commit que l'on effectuait au fur et à mesure de l'avancement du projet. Lorsqu'un membre faisait un commit, celui-ci n'était pas visible par les autres membres du groupe sur leurs ordinateurs respectifs. Ceci bien qu'ils soient collaborateurs sur le répertoire. Suite à cela, on a décidé de se partager les codes par mail pour éviter toute mauvaise surprise.

Nous avons rencontré des soucis techniques au cours de ce projet. En effet, nous n'avons pas pu faire de tests sur nos trois machines mais sur une seule. Le code fonctionnait sur Eclipse sur le PC d'un membre et ne marchait pas sur celui des autres membres du groupe, ce qui rendait les tests difficiles d'autant plus que nous ne pouvions pas nous rencontrer en présentiel pendant les congés de noël. Aussi, tous les tests ont été réalisés sur cette seule machine. Nous soupçonnons un problème sur les versions de Eclipse mais ne pouvons en être sûrs. De plus, le même problème s'est également posé sur les machines de Dauphine.