Université Paris Dauphine

Programmation Java Avancé

Infinity Loop

Etudiant
Alexy Abi Hanna Daher

Professeur
M. Hossein Khani

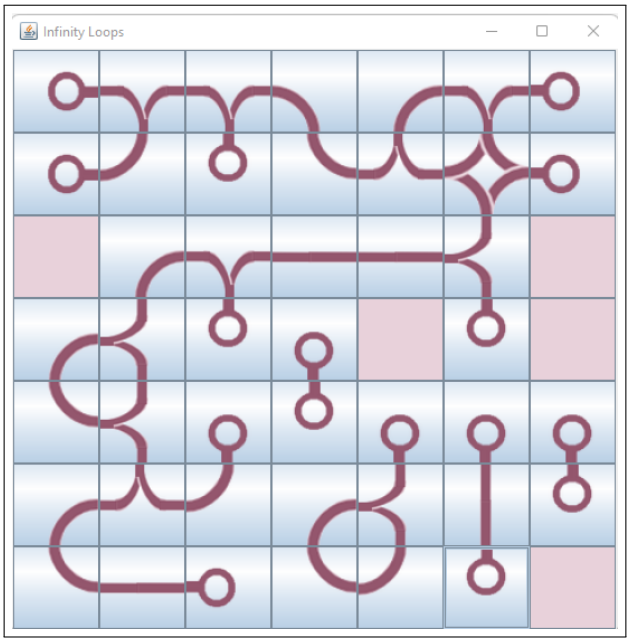


Table des matières

1 Introduction

Dans ce projet, on implémente le jeu Infinity Loop. On utilisera Jframe pour la représentation avec des boutons pour chaque pièce, et une fois un bouton est pressé, cette pièce tournera 90 degrée à droite. Le but du jeu est d'avoir toutes les pièces connéctées.
On pourra genérer un level, le stocker dans un fichier, jouer le jeu ou appliquer un solver pour obtenir l'état final du jeu, ou checker si un fichier qui contient un level est déjà résolu.

2 Utilisation

Générer un grid : Toutes les fichiers qui seront lu ou générés doivent être situés dans le répertoire 'Levels'.

3 Components

Tout d'abord on a commmencé par remplir les classes dans le package 'Components'. Dans la classe PieceType, on a défini les différents types de pièces qu'on peut avoir, et pour chaque pièce on implémente les différentes méthodes :

- getOrientation : renvoi l'orientation de la pièce.
- setConnectorsList : renvoie la liste des connections par rapport à l'orientation de la pièce
- getListOfPossibleOri : renvoie la liste des connections possibles

Pour la classe orientation, on a aussi implémenté quelques méthodes :

- turn90 : renvoie l'orientation tourné à droite
- getOpposedPieceCoordinates : renvoie la position de la pièce opposée
- getOpposedOrientation : renvoie l'orientation opposée à l'orientation initiale.

4 Genérer un level

La classe Generator est composée de plusieurs fonctions :

- La première copyGrid prend en paramètre un grid et renvoie une copie.
- La deuxième méthode est nommée writeGrid qui prend en paramètre le nom d'un fichier avec un grid et consiste à écrire le grid dans ce fichier.
- La troisième méthode est generateLevel, on expliquera son fonctionnement ci-dessous.

Pour générer un level, on doit spécifier d'abord dans les arguments le nombre de lignes et le nombre de colonnes voulus.
Dans le générateur, on commence par créer une liste de height*2 lignes et width colonnes, puis on rempli aléatoirement la liste avec des 1 ou 0. 1 si on ajoute une ligne horizontal si i est impair et vertical si i est pair. Comme ça on a les connections des pièces. Il faut juste ajouter les pièces qui conviennent aux nombre de connections.
Une fois le level est généré, on le sauvegarde dans le fichier fourni en paramètre.
Cette fonction marche de cette façon :

- On fait une liste de liste [height*2 -1][width] height*2 - 1 c'est le nombre de niveaux dans notre grid (horizontale et verticale).
- On met aléatoirement des tirets(1) horizontale ou vertical dans notre grid ou on laisse vide(0)
- Pour chaque position dans notre grid : on calcule le nombre de connections qu'il a :
  - 0 connection → VOID
  - 1 connection → ONECONN
  - 2 connections → si deux orientations qui s'opposent alors BAR, sinon LTYPE
  - 3 connections → TTYPE
  - 4 connections → FOURCONN
- Après avoir choisi la bonne pièce, il suffit de mettre une orientation aléatoire

Avec cette méthode, on est sûr que le grid est solvable.

5 Checker

Il suffit d'itérer sur toutes les pièces du grid et vérifier que toutes les connections d'une pièce soient connectées à une pièce voisines. Cette fonction marche de cette façon :

- On itère sur toutes les pièces du grid
- On récupère les connections à l'aide de la méthode getConnectors() de la pièce
- On vérifie qu'à chaque orientation il y a une pièce voisine qui est connectée

6 Solver

Pour le solver, on a fait la méthode la plus basique : on essaie toutes les possibilitées.
Cette fonction marche de cette façon :

- Si le grid est solved, on le renvoie
- Si i ou j dépasse les limites du grid, on renvoie null
- sinon on appelle 4 fois la fonction en changeant l'orientation de la pièce (i, j) dans les 4 directions, et on teste si une de ses 4 combinaisons renvoie une solution valide.

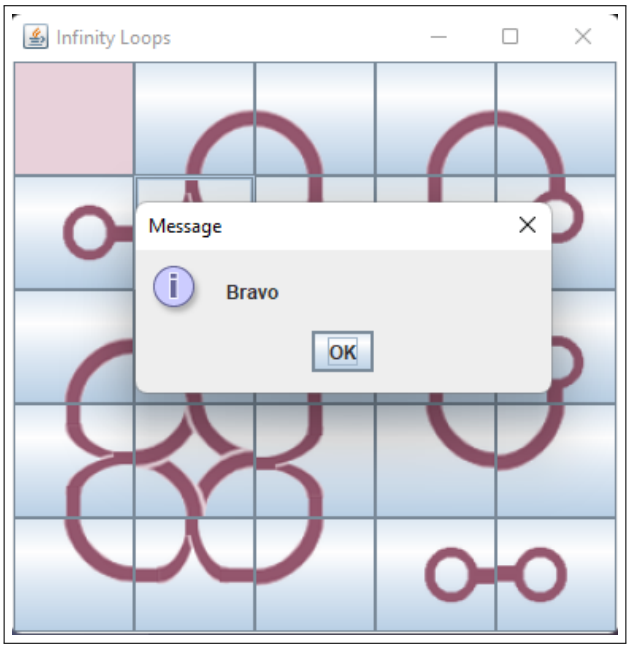
On a trouvé qu'avec cette méthode, on peut facilement trouvé la solution d'un grid 3x3 mais avec des grid plus grands, on a trouvé un problème de performance.

7 GUI

La classe GUI représente l'interface graphique, et elle est composée de plusieurs fonctions :

- initialize(Grid grid) permet d'afficher le grid
- solve(Grid grid) permet de solver la grid en version live (elle est plus lente que la version sans affichage)

Pour le GUI, on a ajouté un bouton pour chaque pièce et le bouton fait tourner la pièce de 90 degré. Le jeu se termine quand le grid est solved et on affiche un message de victoire.



8 Difficultées rencontrées

En conclusion, on n'a pas pu utiliser les threads pour faire une version plus avancée du solver.