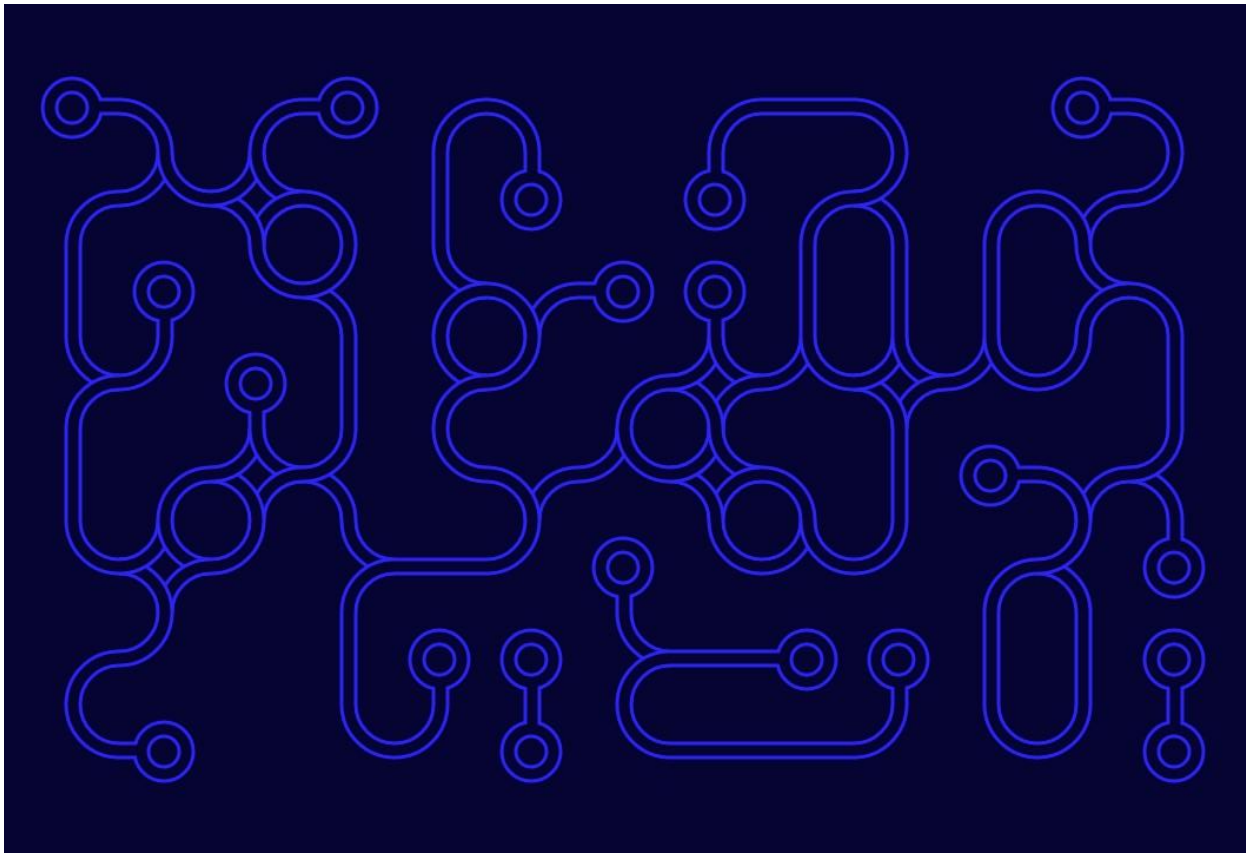
RAPPORT INFINITYLOOP

Projet Java Avancé 2021



Anna Grech
Marina Kayrouz
Martin Le Digabel

INTRODUCTION

Nous avons commencé par modifier les classes dans le package Components. Dans le enum PieceType ou nous avons indiqué les différentes pièces que nous pouvons avoir :

- VOID, ce qui signifie que nous avons un vide et nous avons mis son orientation aléatoirement à North et nous avons initialisé le reste de fonctions.
- ONECONN, qui correspond au rond connecté et qui possède 4 orientations: North, East, West, South.
- BAR, qui correspond au bâton et qui possède 2 orientations seulement comme East et West sont pareilles , ainsi que South.
- TTYPE, ce qui correspond au T et qui possède 4 orientations, nous avons donc initialiser toutes les variables.
- LTYPE, possède aussi 4 voisins et nous avons initialisé toutes les variables.

De plus, un PieceType possède une numéro l'identifiant, et une Orientation suivant la figure 4 du projet.

La classe Piece contient toute les fonctions qui seront plus tard utilisé dans le reste du projet, comme les constructeurs, les getter et setters de toutes les variables, turn qui retourne une pièce de 90° vers la droite et les méthodes pour savoir si les pièces ont des connecteurs de chaque coté (hasRightConnector, hasTopConnector, hasBottomConnector, hasLeftConnector). En plus des fonctions déjà implémentées nous avons ajouté une nouvelle fonction getIntTypeFromPiece qui renvoie le numéro pouvant identifier le type de la pièce suivant le tableau (figure 4) de l'énoncé. Cette méthode sera plus tard utilisée dans le générateur.

Le enum Orientation contient toutes les orientations possibles de chaque pièce. Par exemple, si nous avons une orientation vers le nord, la méthode turn90 tourne la pièce vers l'est, si elle est au sud, la méthode tourne la pièce vers l'ouest. De plus, la méthode getOpposedPieceCoordinates renvoie les coordonnées de la pièce qui est de l'autre côté, par exemple si l'orientation est vers le nord, la méthode renvoie les

coordonnée de la pièce qui a la position de y-1 en y et les coordonnée de l'ancien x en x, car les l'orientation opposée ne change pas de x mais de y quand l'orientation est vers le nord, car son orientation opposée est le sud. Dans cette logique nous avons continué à écrire le reste des méthodes des orientations. Enfin, nous avons implémenté une dernière méthode pour chaque cas d'orientation nommée getOpposedOrientation qui nous indique le sens de l'orientation opposée, donc si la pièce est vers le nord notre orientation opposée est le sud.

GUI

Dans la classe GUI, nous avons créé une frame et un panel avec une liste de boutons qui représente les différentes pièces d'une grille. Pour chaque pièce dans la grille, nous retrouvons son image et la mettons dans le panel pour afficher les boutons. De plus, nous avons ajouté chaque bouton dans un array, comme ca dans le event quand on clique sur une pièce nous pouvons retrouver à quel type appartient le bouton sur lequel nous avons cliqué et nous pouvons le tourner.

GENERATOR

La méthode qui génère de nouvelles grid est generator. Elle prend en paramètre le nom du fichier sur lequel nous voulons générer la grid et une grid vide créée. Cette méthode consiste a générer une grid qui est déjà résolu et puis de tourner toutes les pièces aléatoirement. Notre générateur est divisé en deux méthodes : la première generateLevel1 consiste a générer aléatoirement différentes input levels pour chaque ligne et colonne de la grid. La seconde méthode generateLevel consiste en une boucle sur les différentes lignes et colonnes de la grid et pour chaque pièce l'algorithme voit combien de connexions cette pièce a avec ses voisins, en fonction de quoi nous choisirons la pièce adéquate. Par exemple, si la pièce a 3 connections, nous choisirons TTYPE et si c'est 2 connections, ce sera BAR ou LTYPE qui vont être choisi en fonction des connections (si nous avons une connection vers le bas et une autre vers le haut donc la pièce ne peut etre que LTYPE). Et nous avons continué l'algorithme dans cette logique. A la fin, quand notre solved grid est créé nous allons bouger toutes les pièces pour avoir une grid aléatoire.

Il y a aussi dans cette classe une fonction writeGrid qui écrit la grille sous la forme donnée dans l'énoncé dans un fichier dont le nom est passé en argument.

CHECKER

La classe checker contient un tableau de booléens a deux dimensions isPieceChecked, qui est initialisé au début à false. La méthode isSolved vérifie si la grille est résolue, pour cela nous allons parcourir chaque pièce de la grille et verifié si les pièces à coté de ses connectors ont bien un connecteur opposé. Par exemple si notre pièce est un TTYPE orienté West, nous allons vérifier si la case de gauche a bien un connecteur East, si la case au dessus a bien un connecteur South et la case d'en dessous un connecteur North. Si la case est parfaitement connecté le tableau isPieceChecked est mis a true dans la case de sa coordonnée et on passe à la pièce suivante. Si un pièce que l'on vérifie n'est pas connectée alors la grille n'est pas résolue, notre méthode se termine et retourne false. Sinon true est retourné.

La méthode isWellConnectedAfterMove vérifie si après un clic sur une case qui en change l'orientation, la grille est résolue ou non. On vérifie les connecteurs que cette pièce a après une rotation, on vérifie alors aussi les quatres voisins par récursivité pour mettre à jour notre tableau isPieceChecked et s'ils sont toujours connectés également aux pièces d'à côté. Si la grille est toujours résolue alors la méthode renvoie true, sinon false.

Il y a aussi dans cette classe une fonction builGrid qui lit un grille depuis un fichier au bon format (donné dans l'énoncé) dont le nom est passé en paramètre.

SOLVER

Notre Solver n'est malheureusement pas fonctionnel, nous n'avons pas réussi à implémenter cet algorithme, nous avons laissé dans notre projet des ébauches de codes, qui ne donne pas la solution (si ce n'est que par chance parfois). Nous avons essayé de parcourir la grille de case en case et de créer une sorte de parcours d'arbre avec toutes les combinaisons de grille possibles. Par exemple, si la première pièce avait 4 orientations possibles, nous avons créé quatre grilles clonées ayant chacune une

orientation différente, puis nous avons parcouru ses quatre grilles et créé pour chacunes des copies avec la case suivante sous toutes ses orientations possibles etc…

TESTS

Pour les tests se référer aux tests unitaires JUnit

CONCLUSION

Finalement notre projet permet bien de générer aléatoirement des grilles solvables du jeu InfinityLoop. L'interface graphique (GUI) permet ensuite au joueur de visualiser la grille et de pouvoir jouer au jeu. Les méthodes de la classe Checker assure un suivi de l'orientation des pièces du jeu et permettent de connaître le moment où le joueur gagne la partie. Enfin, nous avons tenté d'implémenter une classe Solver avec différentes méthodes algorithmiques pour résoudre notre problème (Méthode exhaustive, AC-3 algorithm, etc…). Cependant, pris de court par le temps et certains problèmes auxquels nous avons dû faire face durant le projet, notre solveur n'est finalement pas fonctionnel. De plus, nous avons généré un jar file qui peut être retrouvé dans le folder artifact.