Brimont Charlotte
Sirko Sophia
Tisseau Clément

Rapport de projet Java Avance

Notre projet propose de jouer à une grille, pour ce faire, il suffit d'ajouter l'option *-p 0* à la ligne de commande du générateur. Amusez-vous bien !

Generator

Il existe deux moyens de créer un plateau de jeu dans notre projet. Le premier qui est un plateau généré complètement aléatoirement et la seconde méthode qui crée au plus nbcc composantes connexes. Une fois crée, la grille est mélangée grâce à la méthode *mixGrid()* puis on l'écrit dans un fichier.

La première façon est implémentée dans la méthode *generateRandomLevel()* et consiste à créer aléatoirement les pièces et à les fixées, de gauche à droite et de haut en bas. Les pièces sont générées aléatoirement dans la méthode *generatePiece()*.

La seconde manière de générer une grille de jeu est d'appeler au plus nbcc fois la méthode *generateCC()*. Cette méthode créée une composante connexe en initialisant les pièces stockées dans une liste d'attente qui regroupe l'ensemble des voisins des pièces déjà ajoutées à la composante connexe, de la même manière que dans la génération aléatoire, les pièces sont générées aléatoirement par *generatePiece()*.

La méthode *generatePiece()* est la méthode principale de notre générateur, elle permet de générée une pièce aléatoirement en fonction des pièces déjà fixée autour d'elle. On choisit d'abord le type aléatoirement parmi les types possibles, c'est-à-dire, en fonction du nombre de pièces fixées autour et du nombre de connecteurs des pièces fixées autour. Une fois le type fixé, on choisit aléatoirement parmi les orientations qui font que la pièce est connectée à toutes les pièces fixées.

La méthode *mixGrid()* parcourt la grille de gauche à droite et de haut en bas, et, pour chaque pièce, choisie un chiffre aléatoirement entre 1 et 4 et tourne la pièce d'autant.

Checker

Le Checker consiste en une seule méthode *isSolution()* qui parcourt l'ensemble des pièces de la grille et qui return *false* si une des pièces n'est pas totalement connectée, donc si *isTotallyConnected()* return *false*, sinon la méthode return *true* car toutes les pièces ont tous leurs connecteurs qui sont bien connectés.

Solver

Le solver utilise une méthode récursive appelée *solveRec()* qui alterne entre les méthodes *solveWaiting()* et *solveAlea()*. Lorsque l'une de ses méthodes donne une grille résolue, on écrit dans le fichier *FileName* la grille puis on arrête l'exécution.

La méthode *solveWaiting()* prend en argument une liste d'attente représentant les voisins des pièces ayant été fixées récemment et pour lesquels les orientations possibles ont donc été modifiées. Lorsque la liste de ses orientations est vide, cela signifie que la configuration initiale est impossible à résoudre. Si la liste est de taille 1, on fixe la pièce et son orientation puis on ajoute ses voisins non fixés dans la liste d'attente pour qu'à leur tour on regarde s'ils peuvent être fixé.

La méthode *solveAlea()* parcourt la grille jusqu'à trouver une pièce qui ne soit pas encore fixée, pour chaque orientation possible de cette pièce, la méthode fixe la pièce et son orientation et essaye de résoudre cette configuration avec *solveRec()* en donnant comme liste d'attente les voisins de la pièce que l'on vient de fixer.

La méthode *initWaiting()* permet de créer la liste d'attente pour une nouvelle grille, c'est-à-dire, parcourir toutes les cases de la grille et pour celle qui n'ont qu'une seule orientation possible, par exemple, les *LTYPE* dans les angles ou les *TTYPE* ou les *BAR* sur les bords, on fixe les pièces et leur orientation puis on ajoute leurs voisins dans la liste d'attente. Cette dernière sera passée en argument de *solveWaiting()*.

Le solver peut gérer une taille de 100, au-dessus, on ne garantit rien.