UFR – Sciences et Techniques Algorithmique 3 Osmont Alexis
Pires Théo

Rapport de projet

Résolution d’un jeu d'Hashiwokakero ou Hashi. Pour résoudre ce jeu nous avons utilisé les algorithmes vus en cours. L’un d’eux est l’algorithme de BackTracking ou Rebroussement que nous avons explicité en java pour nous permettre de résoudre les jeux donnés en paramètre, il est implémenté dans classe et l’interface, Solver.java et StdSolver.java. Nous avons aussi mis en place un algorithme de création aléatoire de jeu de Hashi dans l’interface et classe, HashiCreation.java et StdHashiCreation.java. Java nous semblait être le meilleur langage pour répondre au problème en utilisant les graphes. De plus cela nous a permis de faire une interface graphique pour avoir un affichage.

Notice d’utilisation

Le fichier README.txt.

Types utilisés

Line

Modélise 4 coordonnées et un booléen qui sont respectivement : les coordonnées de deux îles possiblement voisine donc reliable et un indicateur pour savoir si cette liaison est double ou non.

Island

Modélise une île du jeu avec des coordonnées une valeur initiale et modifiable nécessaire a sa résolution, et un indicateur booléen pour savoir si l’île a bien été créée.

Neighbor

Modélise les voisins d’une île et par conséquent ses possibles futures connections.

Hashi

Modélise un jeu de Hashi sous forme de listes, d’îles, de lignes résolues liées a sa génération et de ligne courante pour sa résolution. Le tout ayant pour clef des coordonnées formatées sous forme de chaîne de caractère.

Graph

Modélise une matrice à partir d’un fichier pour ensuite le transformer en jeu de type Hashi pour permettre sa résolution.

HashiCreation

Modélise un jeu de Hashi aléatoirement en créant une liste d’îles et de coordonnées nous permettant de le résoudre.

Explications

StdHashi

Nous récupérons des dimensions pour pouvoir créer un jeu de type Hashi avec des listes d'îles et des listes de lignes de solution finale et courante.
Dans cette classe nous avons uniquement des fonctions utiles à la résolution et à la création des jeux de Hashi tel que : l'ajout de liaison d'île, la création d'îles au sein d'un jeu etc..

StdHashiCreation

Nous récupérons des dimensions permettant la création d'un jeu aléatoire qui est déjà résolu à sa création, en effet nous créons un jeu qui admet donc une solution et des chemins possibles avec une liste SolLine (liste de lignes entre les îles qui représentent une solution pour le jeu généré). Le jeu est créé de la façon suivante :
- Nous commence par des coordonnées aléatoires.
- Nous modélisons les chemins possibles (les voisins possibles).
- Nous choisissons les chemins vers lesquels nous allons étendre le jeu.
- Nous étendons la création d'îles vers les chemins possibles non bloquants (non occupés par des lignes ou des îles préexistantes).
- Nous répétons l'opération avec des nœuds aléatoires et une quantité aléatoire tant que le jeu est solvable.Cette classe nous retourne donc un jeu aléatoire admettant une solution.

StdSolver

StdSolver est donc notre classe de résolution du jeu, celle-ci commence par explorer toutes les îles en les reliant en tant que voisin. En effet nous allons tout d'abord donner aux îles quels sont leurs voisins. Une fois cela fait, nous allons appliquer l'algorithme de rebroussement pour chercher chaque chemin possible reliant toutes les îles et en les associant grâce à des listes des listes de ligne courante (pas totalement résolu) pour nous permettre de rebrousser chemin si celui-ci n'est pas une solution. En détail :

La résolution se passe en plusieurs étapes, la première est de résoudre les cas faciles tels que les 1 et les 2 isolés n'ayant qu'une unique solution par exemple. La deuxième est le test de toutes les îles en suivant des chemins et en effectuant des Backup des chemins précédemment explorés pour retourner sur nos pas si l'algorithme fait fausse route. Tant que le jeu n'est pas résolu ou que toutes les îles n'ont pas été explorées l'algorithme continue d'explorer en testant toutes les possibilités. On peut voir cela comme un arbre de possibilité qui grandit en cherchant l'unique solution. La troisième étape consiste à détruire les connexions erronées qui ne mène pas à la solution ou qui créaient des lignes sans pouvoir les faire se relier aux autres graphes courants. Cet algorithme a une complexité de $O(n+1 + 4x)$ avec n étant le nombre de nœud et x étant le nombre de neighbor (voisin) possible.

StdDisplayModel

Cette classe nous permet de récupérer le contenu du fichier de jeu de type matrice. En effet cette classe récupère le contenu puis le transforme en matrice d'entier pour ensuite nous permettre plus facilement de transformer cette matrice d'entier en jeu de type Hashi. Cette action est nécessaire pour résoudre le jeu du fichier grâce à notre classe précédente StdSolver. Nous avons

fixé la taille maximale de la matrice à 10 pour éviter de potentiel problèmes de dépassement d'affichage du jeu et de sa solution a l'intérieure de l'interface graphique.

Display

Display est la création et la gestion de l'interface graphique de l'application et des actions qui lui sont liées.

GraphicHashi

Cette classe nous permet d'afficher un jeu de type Hashi puis sa solution après avoir cliqué sur les boutons générer ou parcourir et résoudre.

Difficultés rencontrés

Nous avons rencontré des difficultés au niveau de l'algorithme de backtracking et de création de jeu de Hashi aléatoire. En effet nous avions commencé par vouloir résoudre et créer un jeu en partant de matrice et des arbres mais nous n'avons pas réussi à appliquer l'algorithme de backtracking sur ces types de graphe nous nous sommes donc tournés vers des listes pour plus se rapprocher de la structure des graphes. Nous avons donc gardé une partie de cette structure de graphe pour résoudre le fichier et le transformer en liste pour pouvoir le résoudre.

Améliorations possibles

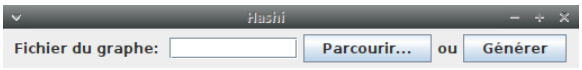
Nous pensons que nous pouvons améliorer la complexité et la fiabilité de nos créations de jeu aléatoires en déléguant certaines taches à des algorithmes adaptés plutôt que tester toutes les possibilités.

Jeux d'essais

Prenons le fichier test.txt sous forme d'une matrice 4x4 se finissant pas un point-virgule:

```
1030
1072
0052
0010;
```

Exécuter le programme (java application), la fenêtre ci-dessous apparaît :

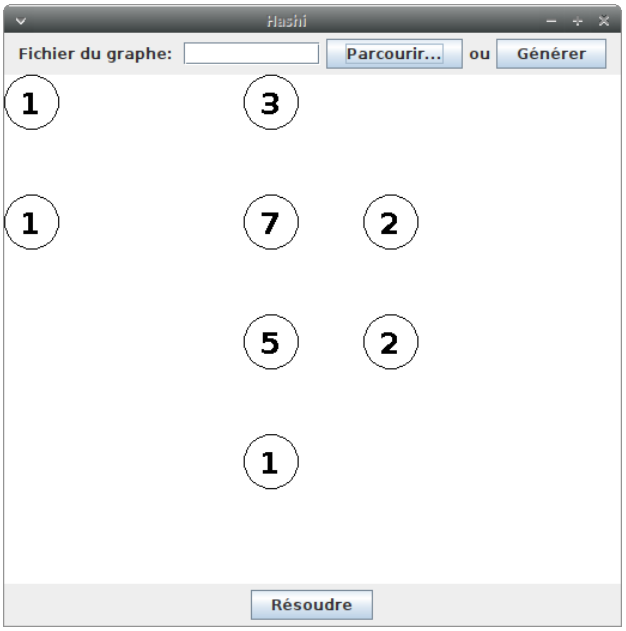


Sélectionner le fichier de jeu test.txt créé au préalable en cliquant sur parcourir ou en donnant le chemin de celui-ci dans l'encadré désigné par « fichier du graphe ». (1)

OU

Cliquez sur générer pour générer aléatoirement un jeu de Hashi. (2)

(1)



(2)