

Projet de programmation

Résolution d'un Sudoku

Solveur Sudoku

Grille de taille 12X12 dont bloc : 4X3

6	8	10	7	4	1	11	12	2	3	5	9
2	12	11			3	8	6	10	4		7
9	3				6	1	4	7	2	12	11
7	4	1	12	11	2		9				8
10		8		7		12	2	11			
11	2		3	5	9	6		8			1
1	7	4	2	12				3	8		
5	9	3	6						11	2	12
12	11	2	9	3	5		8	6	1	7	
8	10	6	4			2				9	
3	5		10	6		7	1		12		
4	1	7	11	2					6	8	

Backtraking

Induction

Générer une nouvelle Grille

CHAGNON Sarah - MI1 n°21800911

LEONI Delphine - MI1 n°21800436

NIZARD Alexandre - MI1 n°21803672

SAMAKÉ Attoumassa - MI1 n°21807148

I- Installation et utilisation du programme

A) Installation

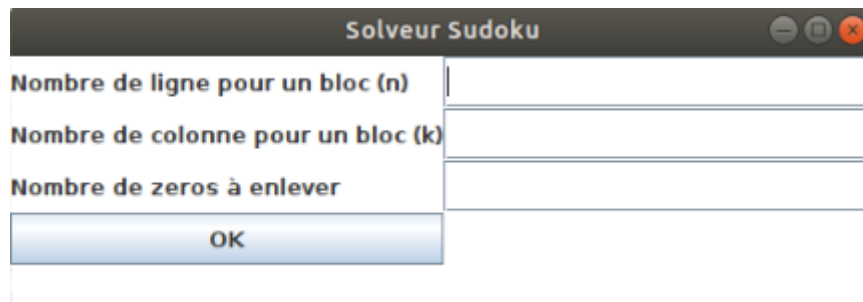
Pour lancer notre projet, il faut tout d'abord se placer à la base, dans le dossier sudoku. Ensuite, il faut lancer la commande *make* dans le terminal. Cette commande va compiler le programme. Puis, pour lancer le programme, il faut taper la commande *java -jar target/projet-0.1.0.jar*. Au lancement, une fenêtre va s'ouvrir avec une grille de Sudoku non-remplie. On a alors le choix entre résoudre cette grille via différentes méthodes implémentées, ou alors de générer une nouvelle grille de Sudoku.

Au début du programme, la grille de Sudoku n'est pas choisie directement par l'utilisateur. S'il veut la modifier, il doit aller dans la classe Test et modifier le nom du fichier qui s'y trouve avec le nom d'un fichier du dossier *FicherSudoku/SudokuGenererZeros*.

B) Utilisation

Pour utiliser le programme, il faut obligatoirement passer par l'interface graphique. Si on résout le Sudoku avec la méthode *backtracking*, il n'y a rien d'autre à faire. En revanche, si on choisit le bouton *induction*, on peut passer sur les cases qui ont été remplies par le programme (chiffre en rouge). On voit alors la règle finale qui a permis de remplir cette case. On y voit également l'ordre, ainsi que le moment où la case a été remplie (pour voir la phrase affichée, il faut agrandir la fenêtre).

Pour la génération d'une grille de Sudoku, une nouvelle fenêtre va s'ouvrir et l'utilisateur devra remplir différents paramètres.



The image shows a Java Swing window titled "Solveur Sudoku". It has a dark title bar with standard window controls. The main area contains three text labels on the left, each followed by a text input field on the right. The labels are "Nombre de ligne pour un bloc (n)", "Nombre de colonne pour un bloc (k)", and "Nombre de zeros à enlever". Below these input fields is a single button labeled "OK".

En effet, celui-ci doit indiquer les dimensions de la grille souhaitée, à savoir n et k . Attention, il y a une condition pour n et k . En effet, $n*k$ doit être inférieur à 25. Nous avons pris cette décision pour éviter que la génération et le backtracking ne soient trop longs. L'utilisateur doit également préciser le nombre de zéros qu'il désire enlever, c'est-à-dire le nombre de cases vides.

Le Sudoku est alors créé, et on peut le résoudre avec la première interface.



II- Description

A) Classes

Nous avons créé trois classes pour implémenter une grille de Sudoku, se présentant comme suit :

- **Grille** : C'est la classe la plus importante, car toutes les fonctions s'y trouvent. Elle permet de créer une grille, mais également de la résoudre. Nous avons décidé de ne pas séparer les résolutions (backtracking et induction) de la classe Grille. En effet, nous avons préféré avoir le code dans une seule et même classe, puisque les éléments sont liés. Mais finalement, cela aurait permis une meilleure lisibilité de notre code, et nous aurait évité une classe atteignant presque les mille lignes.
- **Case** : C'est la classe gérant tout ce qui est en rapport avec une case. On y trouve donc sa valeur, son tableau des solutions possibles, ainsi que la raison expliquant pourquoi la case a été remplie.
- **GenererSudoku** : C'est la classe permettant de créer une grille de Sudoku. On y trouve les dimensions n et k de la grille. Puis, on y trouve également le nombre de zéros qu'a souhaité l'utilisateur. Ces zéros déterminent la difficulté de la grille de Sudoku. En effet, plus il y a de zéros, plus le Sudoku est difficile.

Pour l'interface, nous avons créé deux classes, se présentant comme suit :

- **VueSudoku** : C'est la classe permettant d'afficher une grille de Sudoku, ainsi que les boutons qui permettent de résoudre et de générer une grille.
- **VueChoixGrille** : C'est la classe permettant de sélectionner les dimensions d'une grille de Sudoku, puis de le créer dans le but de le résoudre.

Au début, nous avons une classe **ModelSudoku**. Mais elle n'était pas très utile, car chaque action de l'utilisateur influence directement la vue. Il n'y a pas d'étape intermédiaire à gérer dans un modèle.

B) Algorithmes

Nous avons trois algorithmes importants dans notre projet. Chaque algorithme appelle d'autres algorithmes moins importants :

- **RésolutionSudoku** : Cet algorithme est le plus important de notre projet. En effet, il utilise quasiment toutes les autres fonctions existantes. Cet algorithme fait une boucle en appliquant diverses règles pour résoudre la grille de Sudoku. avec des règles que nous avons implémentées. Il y a huit règles, dont une n'étant pas implémentée.
 1. Ce n'est pas vraiment une règle, elle permet de remplir le tableau de solutions d'une case, en fonction des cases remplies sur la ligne, colonne, bloc.
 2. Le tableau des solutions ne comporte qu'une seule valeur, cette règle va donc remplir la case avec cette valeur.
 3. Elle permet de remplir la case une ligne (colonne, bloc) est presque rempli avec l'unique chiffre manquant.
 4. Si dans une ligne (resp. colonne, bloc) un chiffre ne se trouve que dans un seul tableau de solution, c'est qu'il doit se trouver dans cette case.
 5. Dans un bloc, si une valeur ne se trouve que dans une ligne (colonne), alors cette valeur ne peut pas être dans les autres blocs sur cette ligne (colonne).
 6. Dans un bloc, si sur une ligne (colonne), une valeur ne se trouve que dans ce bloc, alors on peut enlever cette valeur des autres lignes (colonnes) du bloc.
 7. (pas implémentée) Dans un bloc, une paire de deux chiffres ou plus existe, alors on peut enlever ces chiffres des autres cases du bloc.
 8. Mini-backtracking, lorsque aucune règle n'a modifié la grille, un backtracking va se faire sur la première case vide.
- **Backtracking** : Cet algorithme va remplir bêtement le Sudoku, jusqu'à trouver la bonne solution. Il va essayer, à chaque case vide, d'y placer une valeur (qui ne se trouve ni dans la ligne, ni dans la colonne, ni dans le bloc). Puis, il va répéter l'opération pour la case suivante. Si à un moment il y a un problème, l'algorithme retourne à la case précédente et change la valeur qui s'y trouve. L'algorithme ne prend fin que lorsque la grille est entièrement remplie.
- **GénérerSudoku** : Cet algorithme permet non seulement de créer une grille de Sudoku, mais également de l'enregistrer au sein d'un fichier. Grâce aux dimensions fournies par l'utilisateur, l'algorithme remplit une grille, puis mélange les lignes et les colonnes. Ensuite, il utilise la fonction backtracking pour pouvoir vider la grille. Tant

que le backtracking n'obtient qu'une unique grille finale, on enlève des zéros. Enfin, on enregistre cette grille dans un fichier texte.

C) Problèmes rencontrés

Le premier problème que nous avons rencontré a eu lieu lors de l'écriture du backtracking. Nous l'avions correctement implémenté, et celui-ci fonctionnait sur des petites grilles de Sudoku. Par contre, les grandes grilles posaient problème. Nous avons réécrit la fonction maintes et maintes fois, avant de nous rendre compte que l'erreur ne provenait pas du code. En effet, celle-ci venait d'un fichier texte du Sudoku qui avait été mal copié.

Le deuxième souci a été rencontré lors de l'implémentation de la génération d'une grille de Sudoku. Nous avons décidé de générer tout d'abord des grilles de Sudoku pleines, et seulement ensuite de retirer des zéros aléatoirement. Mais notre chargé de TP préférait que les zéros soient enlevés lors de la génération de la grille. Nous avons mis du temps à comprendre cette tâche, et avons changé beaucoup d'éléments dans la génération.

Le troisième problème était lors de l'implémentation de la règle 8, le mini-backtracking. En effet, le principe du backtracking est simple à comprendre et à réaliser manuellement. Malheureusement, le coder est plus difficile. Nous avons donc eu des difficultés à implémenter cette règle, malgré notre algorithme de backtracking déjà fonctionnel.

Enfin, l'utilisation de Git a aussi été un problème. En effet, nous avons découvert Git au premier semestre, mais son utilisation n'était pas encore acquise pour tout le monde. Cela a causé quelques problèmes pour travailler sur des branches, ou pour réaliser des merges request.

III- Exemple

Tout d'abord, il faut choisir la grille de Sudoku que l'on veut dans le dossier *FichierSudoku/SudokuGenererZeros*. Ensuite, il faut lancer le programme en remplaçant un nom de fichier dans la classe Test par celui du fichier correspondant à notre grille. Enfin, il faut choisir entre la résolution par backtracking et la résolution par induction. On peut également générer une grille de Sudoku, en suivant les instructions données par l'interface.