Résolution de grilles de Sudoku

D'après un énoncé proposé au Cnam Poitou Charentes en 2008

On souhaite implémenter un algorithme de résolution du problème du Sudoku.

Une grille de Sudoku est constituée d'un carré de 9 cases de côté. Ce carré est divisé en 9 sous-carrés de 3 cases de côté. Chaque case peut contenir un chiffre compris entre 1 et 9.

Les règles sont les suivantes : chaque ligne, chaque colonne et chaque sous-carré doivent contenir une fois et une seule tout chiffre compris entre 1 et 9.

Initialement, la grille contient des cases remplies selon les règles ci-dessus.

Le but est de compléter la grille en respectant les règles.

Un problème correctement posé doit avoir une solution et une seule.

Exemple :

Grille initiale

6	2			7	5			
	9					5		
				9			1	
		9	6			1	8	4
2	1	8			7	9		
	5			3				
		4					2	
			8	4			5	6

1- Un algorihme récursif de résolution

Un principe récursif simple de résolution d'un problème de Sudoku est le suivant :

- choisir une case libre, fixer le premier nombre possible selon les règles
    - essayer de résoudre le problème à partir de la prochaine case libre
    - si c'est un échec fixer un autre nombre possible et esssayer de résoudre le problème à partir de la prochaine case libre etc…
    - s'il n'y a plus de nombre possible alors c'est un échec.
    - si c'est la dernière case et qu'un nombre convient alors c'est un succès

Un objet(classe) Sudoku peut être implémenté de la manière suivante :

1. Chaque objet contient un champ **grille** de **type byte[][]** matrice de dimensions 9x9, **grille[i][j]=0** si le chiffre n'est pas fixé, un chiffre sinon
2. Écrire un constructeur **Sudoku(byte[][] init)** qui prend une grille **init** en argument et initialise la grille de l'objet
3. Écrire une méthode String **toString()** qui retourne une chaîne de caractères associée à l'objet (mode « texte » de la grille).
4. Écrire une méthode **boolean nApparaitPasLigne(byte nombre, int i, int j)** qui teste si un nombre n'apparaît pas sur la ligne i
5. Écrire une méthode **boolean nApparaitPasColonne(byte nombre, int i, int j)** qui teste si un nombre n'apparaît pas sur la colonne j
6. Écrire une méthode **boolean nApparaitPasCarré(byte nombre, int i, int j)** qui teste si un nombre n'apparaît pas dans le sous-carré associé à la case (i,j)
7. Écrire une méthode **boolean nombrePossible(byte nombre n, int i, int j)** qui teste s'il est possible de mettre le nombre n dans la case de coordonnées (i,j). Écrire une méthode **boolean caseOccupée(int i, int j)** qui teste si la case contient un nombre entre 1 et 9.
8. Écrire une méthode récursive **boolean resolution(int i, int j)** qui retourne true si on a réussi à remplir toutes les cases à partir de la case(i,j), false sinon.

   En supposant qu'on examine les cases ligne par ligne, puis colonne par colonne et les chiffres dans l'ordre 1,…,9, on a trouvé la solution si **boolean resolution(0,0)** retourne true.

2 Implémentation de l'algorithme

- Lire la grille à partir d'un fichier texte et initialiser le champ **grille** de l'objet. Dans le fichier texte, chaque nombre de la grille est une chaîne de caractères pouvant prendre la valeur : « 1 », « 2 » … « 9 » si la case correspondante a une valeur fixée, « 0 » sinon. Le fichier texte est alors une suite de 81 chaînes ou mieux une sutie de 9 lignes de 9 chaînes.
- Utiliser la méthode récursive **boolean resolution**
- Si on a trouvé une solution, afficher la grille solution à l'écran.