1 Problématique

Le Sudoku est un jeu aux règles simples mais qui offre un nombre de combinaisons énormes. Ainsi il existe $9!^9$ grilles possibles (soit 6670903752021072936960) qui correspond au nombre de façons de construire les régions sans tenir compte des contraintes sur les lignes et les colonnes. Il existe plusieurs variantes, mais les règle de base sont :

— chaque ligne ne contient qu'une série de chiffres de 1 à 9

— chaque colonne ne contient qu'une série de chiffres de 1 à 9

— chaque bloc ne contient qu'une série de chiffres de 1 à 9

Peut-on créer un programme pour construire une grille complète et correcte ?

2 Modélisation

Afin de simplifier le problème, concentrons-nous d'abord sur une grille de 4 de côté.

1	4	2	3
2	3	1	4
4	1	3	2
3	2	4	1

FIGURE 1 – Grille 4×4

Un graphe permet de modéliser les contraintes entre les cases. La figure 2 ne montre que les arêtes qui partent de la case en haut à gauche.

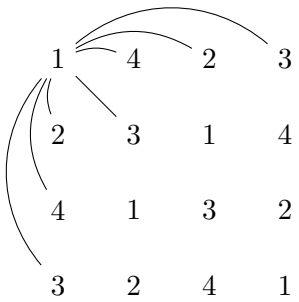

FIGURE 2 – Modélisation sous forme de graphe

3 Implémentation

3.1 Une classe Sudoku

La classe *Graphe* que nous avons déjà construite nous permettra de représenter les contraintes entre chaque case. Chaque sommet sera identifié par *ses coordonnées dans la grille*. En parallèle nous utiliserons un tableau de tableaux pour contenir les chiffres à placer dans la grille.

Christophe Viroulaud

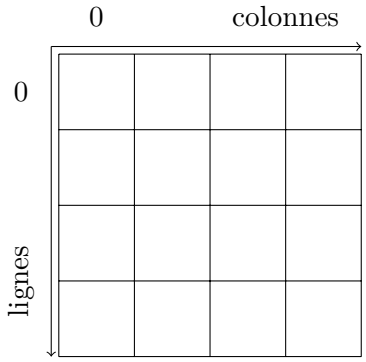

FIGURE 3 – Repérage des cases dans le système de coordonnées choisi.

Activité 1 : Écrire la classe *Sudoku* et son constructeur. Elle possédera trois attributs à initialiser :
- **taille : int** 4 pour un petit Sudoku,
- **grille : list** à construire en fonction de la figure 3,
- **graphe : Graphe**

3.2 Créer le graphe

Comme le montre la figure 2 chaque case doit respecter plusieurs contraintes. Ainsi nous procéderons en plusieurs étapes pour construire les arêtes :
- construire les sommets,
- construire les arêtes verticales,
- construire les arêtes horizontales,
- construire les arêtes de blocs.

Activité 2 : Écrire la méthode **creer_graphe(self)** → **None** qui construit le graphe.

3.3 Méthodes utiles

Il est possible de parcourir le graphe de plusieurs manières. Puisque sa représentation est une grille carrée nous choisirons un balayage simple ligne par ligne.

Activité 3 : Écrire la méthode **case_suivante(self, s : tuple)** → **tuple** qui renvoie les coordonnées, sous forme d'un tuple, de la case suivant celle de coordonnées *s*.

Pour chaque case à remplir, il faudra d'abord vérifier si le chiffre que l'on veut placer n'est pas déjà positionné sur une ligne, une colonne ou un bloc.

Activité 4 : Écrire la méthode **est_possible(self, s : tuple, choix : int)** → **bool** qui renvoie *True* si le chiffre *choix* n'est pas déjà positionné dans une case « adjacente » au sommet *s*.

3.4 Remplir la grille

Au départ, remplir la grille ne pose pas de difficultés car les choix sont nombreux. Cependant il peut apparaître rapidement des impossibilités. Il faut alors revenir en arrière, effacer les chiffres placés précédemment et recommencer. Cette description rapide montre un comportement récursif dans la méthodologie à adopter.
Formalisons cet algorithme :
- Si nous sortons de la grille, elle a été correctement remplie : renvoyer *True*.

Christophe Viroulaud

— Sinon, pour chaque chiffre :
  — S'il peut être positionné :
    — Le placer dans la grille.
    — Remplir récursivement la case suivante et remonter *True* dans la pile d'appel si le placement est correct.

  Tous les chiffres ont été testés et aucun ne fonctionne :
— Réinitialiser la valeur de la case.
— Remonter *False* dans la pile d'appel.

Activité 5 : Écrire la méthode **remplir_rec(self, s : tuple = (0,0)) → None** qui implémente cet algorithme.

4 Affichage

Activité 6 : Écrire la méthode **afficher(self) → None** qui utilise la bibliothèque *tkinter* pour réaliser un affichage de la grille.

5 Aller plus loin

5.1 Construire une partie

Partant d'un grille complète et correcte, il semble aisé de construire une partie. Il suffit de retirer aléatoirement des chiffres. Cependant des difficultés existent. En effet une partie est considérée correcte s'il n'existe qu'une seule solution. En enlevant des chiffres au hasard, nous pouvons sans le vouloir créer une partie qui aurait plusieurs issues.
Pire cela peut créer une situation bloquante où il devient impossible de compléter le jeu. L'expérience montre qu'il est possible de construire des jeux avec seulement 17 chiffres dévoilés. C'est d'ailleurs le nombre minimal d'indices que l'on peut donner. La preuve a été établie il y a quelques années comme le présente cet article du *Monde* : https://tinyurl.com/y4zmj4e7.
En pratique à chaque retrait d'un chiffre, il faut vérifier que la partie est réalisable et la grille correcte.

5.2 Résoudre une partie

En modifiant légèrement la classe *Sudoku*, il est possible de résoudre des parties rapidement. Il faut tout d'abord initialiser la grille avec les indices, puis modifier la fonction récursive pour qu'elle ne touche pas les cases fixées au départ.

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

FIGURE 4 – À vous de jouer !

Christophe Viroulaud