Romain FRANCHI

**Rapport : Développement d’un solveur de Sudoku en Python**

I/ Introduction

Le but de ce projet était de développer un solveur de Sudoku capable de résoudre des grilles en appliquant plusieurs règles de déduction. Pour ma part, j’ai choisi de concevoir cela en Python.

Ce projet a été réalisé en suivant les exigences définies, notamment l’utilisation de trois règles de déduction (DR1, DR2, et DR3), dérivées de la classe abstraite DeductionRule.

L'algorithme devait appliquer ces règles pour compléter la grille. Si la déduction ne permettait pas de remplir une grille, une intervention utilisateur était nécessaire pour fournir une valeur manquante.

A noter que la grille peut-être facilement modifiée, elle est notée en fichier texte, se trouvant à l’emplacement ./src/sudoku.txt .

II/ Conception et Algorithme

1. Représentation de la Grille

La grille de Sudoku est représentée dans la classe SudokuGrid qui se trouve dans le fichier sudoku\_grid.py comme une liste de 81 éléments. Chaque élément représente une cellule de la grille, avec la valeur 0 indiquant une cellule vide.

1. Définition des Règles de Déduction

Les règles de déduction sont implémentées dans deduction\_rule.py. Trois règles sont prévues : DR1, DR2, et DR3. Elles dérivent toutes de la classe abstraite DeductionRule, qui suit le design pattern Patron de Méthode. Ce pattern permet de définir une structure d'algorithme dans une classe mère (DeductionRule), laissant les sous-classes (DR1, DR2, DR3) implémenter les détails spécifiques de chaque règle​.

- DR1 : Si une cellule a 8 voisins définis (dans la ligne, colonne ou sous-grille), la 9e valeur est déduite automatiquement​.

- DR2 : Si une valeur n'apparaît qu'une seule fois parmi les candidats possibles d’une ligne, colonne, ou sous-grille, cette valeur doit être placée dans la cellule correspondante.

- DR3 : "Paire cachée" : Si une paire de candidats apparaît uniquement dans deux cellules d’une ligne, colonne ou sous-grille, toutes les autres valeurs peuvent être éliminées de ces deux cellules.

1. Algorithme de Résolution

L'algorithme de résolution est défini dans la classe SudokuSolver (fichier solver.py). Il utilise l’ensemble de règles de déduction (DR1, DR2, DR3) appliquées successivement pour tenter de compléter la grille. S'il reste des cellules vides après avoir appliqué les règles, l'utilisateur est sollicité pour entrer une valeur.

L'algorithme utilise les design patterns suivants :

- Patron de méthode : Utilisé pour la classe DeductionRule et ses sous-classes (DR1, DR2, DR3), où chaque règle implémente une méthode spécifique, permettant une flexibilité dans l'ajout de nouvelles règles sans modifier la structure existante.

- Singleton : Utilisé pour gérer la grille de Sudoku avec une instance unique, garantissant que toutes les parties du programme interagissent avec la même grille, évitant des incohérences dans les données​.

- Observateur : Utilisé pour notifier les règles de déduction lorsque des valeurs sont ajoutées dans la grille. Cela permet à chaque règle d'évaluer si une nouvelle déduction peut être faite suite à l'ajout d'une valeur​.

- Stratégie : Permet de sélectionner dynamiquement la stratégie de déduction à utiliser, en fonction des besoins du solveur à chaque étape de la résolution. Cela rend le système flexible et adaptable à des situations variées​.

III/ Développement du Code

1. Principales Classes

SudokuGrid : Gère la représentation de la grille, la validation des mouvements, et l'affichage. La grille est initialisée avec une valeur par défaut de 0 pour toutes les cellules. La méthode load\_grid() permet de charger une grille depuis un fichier​.

DeductionRule : Classe abstraite définissant la méthode apply\_rule() que toutes les règles de déduction doivent implémenter. Chaque sous-classe (DR1, DR2, DR3) implémente des stratégies spécifiques pour remplir les cellules de la grille​.

SudokuSolver : Cette classe coordonne l'application des règles et permet l'interaction avec l'utilisateur. Elle gère la progression de la résolution et inclut des fonctionnalités permettant de sauvegarder et restaurer des états antérieurs en cas d'erreur​.

1. Interactions avec l’Utilisateur

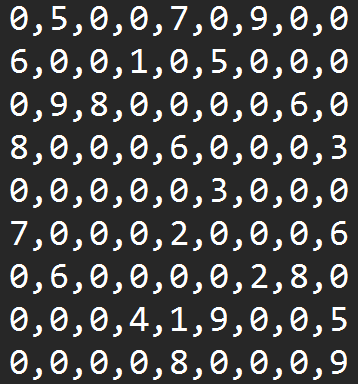
Lorsque les règles de déduction ne peuvent plus progresser, l'utilisateur est invité à entrer une valeur pour une cellule sélectionnée par lui-même en indiquant la ligne et la colonne de celle-ci.

Pour être visuellement plus visible la cellule ciblée est notée avec un « ? » vert et clignotant à la place d’un simple « 0 ».

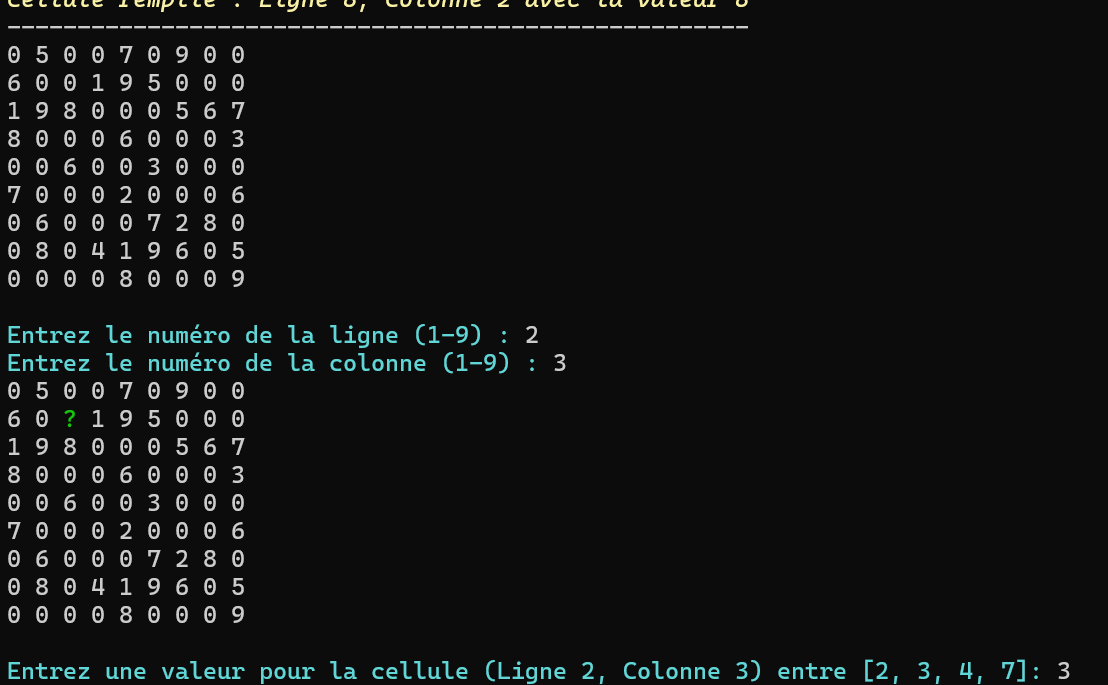
Le solveur propose alors une liste de chiffres valides pour cette cellule, en s'assurant que l'utilisateur entre une valeur correcte​. Ce mécanisme permet de compléter les grilles quand les règles de déductions ne suffisent pas.

IV/ Résultats

Le solveur a été testé sur une grille de Sudoku fournie dans le fichier sudoku.txt​.



La grille initiale comporte plusieurs cellules vides, représentées par des 0. Après application des trois règles de déduction, certaines grilles peuvent être résolues automatiquement si les déductions suffisent. Dans le cas contraire, l'utilisateur est sollicité pour compléter les cellules restantes.



V/ Conclusion

Ce projet a permis de développer un solveur de Sudoku robuste. Les design patterns utilisés (Patron de méthode, Singleton, Observateur, Stratégie) ont permis de structurer le projet de manière assez claire. Le solveur fonctionne sur des grilles de difficultés différentes, bien que certaines grilles plus difficiles nécessitent bien évidemment l'intervention de l'utilisateur.

Pour faciliter le lancement du programme, j’ai utilisé PyInstaller pour créer un fichier .exe de mon fichier python main.py . Evitant de devoir utiliser des lignes de commande pour l’utilisateur.