Rapport – Sudoku Solver

Enzo Ghizzardi & Mehdi Mansour

1. Introduction

Ce projet avait pour objectif de développer un solveur de Sudoku capable de résoudre des grilles en utilisant trois "Deduction Rules", s'appliquant aux cellules de la grille pour calculer et déduire leur valeur, et ainsi résoudre la grille.

2. Problèmes et Défis du Projet

1. **Implémentation des Deduction Rules & difficultés** : le programme doit implémenter différentes règles de déduction qui permettront de résoudre la grille, des plus simples à de plus complexes. Ces règles doivent aussi permettre d'établir un niveau de difficulté pour chaque grille. La règle 1 doit donc pouvoir résoudre des sudokus plutôt faciles, avec la règle 2 en plus on résout des sudokus medium, et avec les 3 règles on peut résoudre des sudokus difficiles. Certains sudokus très difficiles résistent aux 3 règles.
2. **Mise à Jour dynamique de la grille :** Une des caractéristiques essentielles de la résolution d'un Sudoku est la mise à jour continue des « coups possibles » dans chaque cellule en fonction des valeurs déjà résolues dans la ligne, colonne, ou bloc. Cela nécessite un mécanisme de notification pour assurer que les cellules affectées par une modification soient mises à jour automatiquement.
3. **Interaction avec l'Utilisateur :** Dans le cas où les Deduction Rules ne suffisent pas à résoudre la grille, l'interaction avec l'utilisateur est nécessaire. Le programme doit demander à l'utilisateur de fournir une valeur tout en s'assurant que cette valeur n'entraîne pas d'inconsistances dans la grille.
4. **Contraintes de Design Pattern :** Le programme doit implémenter 4 Design Pattern différents.
5. **Format de fichier d'entrée :** Le programme doit implémenter, et donc pouvoir lire un format spécifique de fichier d'entrée. Il doit pouvoir charger depuis une grille de sudoku sous forme d'un fichier txt, selon le format suivant :
   3,8,0,1,0,0,5,9,0
   2,3 …

3. Deduction Rules utilisés

Nous avons donc implémenté 3 règles de déductions différentes, toutes les trois utilisant les “coups possibles” de chaque cellule pour fonctionner. Ces “coups possibles” sont similaires aux notes qu’un humain pourrait utiliser pour compléter une grille de sudoku.

3.1 Singleton nue (Classe NakedSingle)

Elle repose sur le fait que, dans une case spécifique, un seul chiffre (des notes) reste possible.

S'il ne reste plus qu’une valeur dans l’ensemble des coups possibles d’une cellule, alors la règle de déduction va affecter cette dernière valeur à la cellule.

3.2 Singleton caché (Classe HiddenSingle)

Les « singletons cachés » sont une technique de sudoku assez simple. Elle s'appuie sur le fait qu'un chiffre déterminé est présent une seule fois dans les notes d'une ligne, d'une colonne ou d'un bloc.

La règle va vérifier si une valeur de l’ensemble des coups possibles d’une cellule n’apparaitrait jamais dans l’ensemble des coups possibles de sa ligne, de sa colonne ou de son bloc. Si c’est le cas, alors la règle affecte à la cellule cette valeur.

3.3 Paires nues (Classe NakedPairs)

L'idée de base est que vous devez trouver 2 cases avec les mêmes paires de coups possibles dans une ligne, une colonne ou un bloc de 3x3 cases. Cela veut dire que ces paires de notes ne peuvent pas être utilisées dans d'autres cases du même bloc, et que vous pouvez donc les retirer de vos notes. Vous comprendrez plus facilement cette stratégie avec un exemple.

La règle va vérifier pour chaque cellule si son nombre de coups possibles est exactement égal à 2 (donc son nombre de coups possibles est une paire nue). Le cas échéant, si cette paire apparait exactement une autre fois dans les ensembles de coups possibles des cellules de la même ligne, colonne ou bloc, alors la règle pourra enlever de la liste des coups possibles des autres cellules les valeurs appartenant à la paire.

4. Solutions choisies et Design Patterns utilisés

Pour répondre à ces problèmes, plusieurs solutions techniques et concepts de conception ont été mis en place. Ce projet implémente quatre principaux **design patterns** de manière intégrée et collaborative : **Singleton**, **Observer**, **Strategy**, et **Factory**. Ces design patterns permettent de maintenir la structure du code claire, modulable et efficace.

4.1 Singleton Pattern (Classe SudokuGrid)

Le **Singleton Pattern** est utilisé pour la classe `SudokuGrid`. Cela permet de s'assurer qu'une seule instance de la grille est créée et que toutes les modifications apportées à cette instance sont répercutées à travers tout le programme.

On garantit donc l'unicité de l'état de la grille tout au long de la résolution, évitant les problèmes de synchronisation ou de divergence d'état qui pourraient apparaître avec plusieurs instances.

Ce design pattern est idéal pour ce type de problème car la grille est un objet global, qui doit être unique et accessible par plusieurs classes du programme.

4.2 Observer Pattern (Classe Cell)

L'**Observer Pattern** est utilisé pour assurer que chaque fois qu'une cellule est résolue, toutes les autres cellules de la même ligne, colonne, ou bloc soient notifiées de cette modification et puissent ajuster leurs listes de « coups possibles ».

On met à jour dynamiquement et automatiquement les coups possibles lorsque la valeur d'une cellule change. Sans ce pattern, il serait nécessaire de recalculer les coups possibles pour chaque cellule à chaque fois, ce qui serait inefficace.

L'utilisation de l'**Observer Pattern** simplifie grandement la logique de mise à jour de la grille et permet de rendre le programme plus modulable et réactif aux changements d'état.

4.3 Strategy Pattern (Règles de Déduction)

Le **Strategy Pattern** est utilisé pour implémenter les différentes règles de déduction (`NakedSingle`, `HiddenSingle`, `HiddenPairs`). Chaque règle est implémentée comme une stratégie indépendante, permettant de les combiner ou de les utiliser séparément selon le niveau de difficulté de la grille. Chaque règle hérite de la classe "DeductionRules" et implémente donc la méthode "apply()".

Cela permet la sélection et l'application successive de différentes stratégies de résolution sans avoir à modifier la logique de résolution.

Ce pattern facilite l'ajout de nouvelles Deduction Rules sans changer la structure globale du code.

4.4 Factory Pattern (DeductionRuleFactory)

Le **Factory Pattern** est implémenté via la classe `DeductionRuleFactory`. Cela permet de créer des objets de type `DeductionRule` en fonction des besoins du solveur, sans exposer directement le processus de création.

Cela permet de centraliser la logique de création des règles de déduction, permettant ainsi de rendre le code plus modulaire et plus facile à maintenir.

Cette approche permet de découpler la création des objets de leur utilisation, facilitant ainsi la gestion du cycle de vie des objets et réduisant le couplage entre les classes.

5. Processus de Résolution de la Grille

Le processus de résolution commence par charger la grille à partir d'un fichier texte. Ensuite, chaque règle de déduction est appliquée de manière successive jusqu'à ce qu'il n'y ait plus de progression possible. Si aucune règle ne permet de résoudre la grille entièrement, l'utilisateur est invité à entrer une valeur pour continuer la résolution.

L'utilisateur est sollicité seulement lorsque les règles automatiques ne parviennent plus à résoudre la grille.

Avant d'accepter une valeur saisie par l'utilisateur, la validité de la grille est vérifiée pour s'assurer qu'aucune règle du Sudoku n'est violée. Si une inconsistance est détectée, le programme demande de redémarrer la résolution.

6. Conclusion

Ce projet a permis de mettre en pratique plusieurs concepts avancés de programmation orientée objet, en particulier les **design patterns** qui ont grandement contribué à la résolution des problèmes posés par la résolution d'un Sudoku. Chaque pattern a été choisi en fonction des problèmes spécifiques auxquels il répondait, et leur utilisation combinée a permis de produire une solution efficace, modulaire, et facile à maintenir.

Les défis principaux étaient la gestion de l'état de la grille, l'application successive de règles de déduction, et l'interaction avec l'utilisateur. L'utilisation des **Singleton**,

Observer, **Strategy**, et **Factory Patterns** a permis de surmonter ces obstacles et de produire un solveur capable de résoudre des grilles de Sudoku.

Ce projet pourrait être amélioré par l'ajout de nouvelles stratégies de déduction plus avancées, l'optimisation des performances de mise à jour de la grille, ou encore l'ajout d'une interface utilisateur graphique pour rendre l'expérience plus intuitive.