Compte rendu : Sudoku solver

Alexis DUBARRY et Shanti NOEL

Objectif du Projet

Le projet vise à résoudre des grilles de Sudoku en appliquant successivement des règles de déduction pour déterminer la valeur correcte des cellules. Chaque règle cible un type particulier de déduction, permettant de réduire le champ des valeurs possibles dans les cellules de manière progressive. Ces règles de déduction sont organisées dans des classes dérivant d'une classe parentale DeductionRule.

Choix des Design Pattern

Pour ce projet de Sudoku, nous avons fait le choix d'implémenter 4 différents design pattern :

1. Pattern Template Method (Méthode Template)

- **Où il est utilisé** : Dans la classe abstraite (DeductionRule).
- **Description du pattern** : Le Template Method est un design pattern comportemental qui définit la structure d'un algorithme dans une méthode, mais laisse certaines étapes spécifiques à des sous-classes. Dans ce projet, la méthode apply de DeductionRule est une méthode template qui impose une structure générale pour toutes les règles de déduction.
- **Pourquoi on l'a utilisé** : Ce pattern garantit que toutes les règles de déduction (DR1, DR2, DR3) suivent une interface commune et une séquence d'actions similaires tout en permettant à chaque règle de définir sa logique spécifique. Grâce à cela, chaque règle peut être traitée uniformément dans le code, ce qui simplifie la gestion des règles et permet de facilement ajouter de nouvelles règles sans modifier les autres.

2. Pattern Strategy

- **Où il est utilisé** : Dans l'organisation des classes de règles de déduction (PrintStrategy).
- **Description du pattern** : Le Strategy est un design pattern comportemental qui permet de définir une famille d'algorithmes (ou de comportements), de les encapsuler et de les rendre interchangeables. Il permet de choisir dynamiquement quel algorithme utiliser.
- **Pourquoi on l'a utilisé** : Ici, on a choisi de l'utiliser pour permettre d'avoir des affichages différents. Dans le contexte d'une éventuelle amélioration du code,

on trouve qu'il est nécessaire de pouvoir choisir entre un affichage simple (pour debug par exemple), et un affichage "complexe" (pour avoir un joli affichage côté utilisateur).

3. Pattern Singleton

- **Où il est utilisé** : Dans la gestion de l'instance unique de la grille de Sudoku (classe SudokuGrid).
- **Description du pattern** : Le Singleton est un design pattern de création qui garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance.
- **Pourquoi on l'a utilisé** : Dans le cas de ce projet, avoir une instance unique SudokuGrid est une bonne idée.

4. Pattern Builder

- **Où il est utilisé** : Dans la construction de la grille de Sudoku (SudokuGridBuilder).
- **Description du pattern** : Le Builder est un design pattern de création qui permet de construire des objets complexes en séparant leur construction de leur représentation. Il est particulièrement utile quand il y a beaucoup de paramètres ou de sous-composants pour un objet complexe.
- **Pourquoi on l'a utilisé** : Dans un projet de Sudoku, le pattern Builder sépare la logique de construction de l'objet de la logique métier de l'objet lui-même. Cela permet de garder le code de la classe SudokuGrid propre et focalisé sur ses responsabilités principales, et d'avoir dans SudokuGridBuilder les étapes nécessaires à l'initialisation de la grille.

Conclusion et avantages des Design Patterns utilisés

Ces patterns apportent plusieurs avantages à la conception du projet :

- **Flexibilité et Extensibilité** : Grâce au Template Method pattern, l'ajout de nouvelles règles de déduction est facilité sans impact sur la structure globale du programme.
- **Réutilisabilité et Maintenance** : Les règles de déduction étant isolées dans des classes séparées, elles peuvent être utilisées dans d'autres projets ou tests unitaires, et le code est plus facile à maintenir.

Ces Design Patterns permettent ainsi de structurer le projet de manière professionnelle, en augmentant sa robustesse, en simplifiant sa maintenance, et en ouvrant la voie à des extensions pour des résolutions plus complexes ou des variantes du Sudoku.

Structure et Composants du Projet

1. Classe Principale : SudokuSolver

La classe SudokuSolver est le cœur du processus de résolution de la grille de Sudoku. Elle est responsable de l'application des différentes règles de déduction pour résoudre la grille de manière progressive et méthodique. SudokuSolver gère également l'interaction avec l'utilisateur, permettant de remplir manuellement les cellules lorsqu'aucune règle de déduction ne peut être appliquée.

Fonctionnalités Principales:

1. **Initialisation de la grille** :
   SudokuSolver utilise un SudokuGridBuilder pour initialiser la grille. Cela permet de séparer la logique de création et d'injection de dépendances de la logique métier, en utilisant le pattern **Builder** pour construire la grille SudokuGrid.
2. **Choix des Règles de Déduction** :
   La méthode choixRules permet à l'utilisateur de choisir dynamiquement jusqu'où appliquer les règles de déduction (DR1, DR2, DR3). Selon le choix, les règles sont ajoutées successivement au solveur grâce au pattern **Strategy**. Cette approche rend la résolution flexible et permet d'adapter la complexité de la résolution en fonction des besoins.
3. **Application des Règles de Déduction** :
   La méthode applyRules applique successivement toutes les règles ajoutées dans l'ordre choisi par l'utilisateur. applyRules utilise la méthode apply de chaque règle (implémentée dans les sous-classes de DeductionRule) pour modifier la grille et afficher les changements après chaque règle appliquée.
4. **Résolution de la Grille** :
   La méthode solve gère le processus global de résolution en appliquant en boucle les règles de déduction jusqu'à ce que la grille soit complètement résolue ou qu'aucune règle ne puisse être appliquée. Elle suit la logique suivante :
   - **Boucle de déduction** : Applique les règles de manière répétée jusqu'à ce qu'aucun changement ne soit détecté ou que la grille soit résolue.

    - **Saisie Utilisateur** : Si la grille n'est toujours pas résolue après l'application de toutes les règles, le solveur demande à l'utilisateur de remplir manuellement une cellule vide.
    - **Vérification de la validité** : Après chaque saisie utilisateur, la méthode grilleValide vérifie si la grille est toujours valide. Si une incohérence est détectée, la résolution s'arrête.
5. **Vérification de la Grille** :
   La méthode grilleValide vérifie l'absence de doublons dans les lignes, colonnes et sous-grilles 3x3, en renvoyant 1 si la grille est valide et 0 en cas de doublons.
6. **Affichage de la Grille** :
   Le solveur affiche la grille après chaque règle appliquée, permettant de suivre la progression de la résolution étape par étape.

2. Classe de la grille: SudokuGrid

La classe SudokuGrid représente la grille de Sudoku, avec ses méthodes de gestion des valeurs possibles pour chaque cellule. Elle est le cœur du projet et se charge des éléments suivants :

- **Représentation de la Grille** : La grille est représentée comme une matrice 9x9.
- **Gestion des Valeurs Possibles** : possibleValues est une matrice 9x9 où chaque cellule contient un ensemble de valeurs (de 1 à 9) possibles pour cette cellule. Ces ensembles se réduisent progressivement en fonction des règles de déduction appliquées.
- **Méthodes Utilitaires** : getPossibleValues(row, col) pour obtenir les valeurs possibles d'une cellule donnée et setValue(row, col, value) pour définir une valeur spécifique dans une cellule.
- **Mise à jour des valeurs possibles** : updatePossibleValue(row, col, value) qui est utilisée à chaque setValue, cette fonction sert à enlever la valeur posée dans la ligne/colonne/carré correspondant à la case.
- Cette classe est donc essentielle pour le fonctionnement des règles de déduction car elle fournit à chaque règle l'accès aux valeurs possibles pour chaque cellule.

3. Classe Abstraite : DeductionRule (template pattern)

DeductionRule sert de classe parentale pour chaque règle de déduction. Elle définit une structure commune via la méthode abstraite apply(SudokuGrid grid), que

chaque sous-classe doit implémenter pour appliquer une règle spécifique sur la grille de Sudoku.

Ce modèle permet :

- **Uniformité** : Toutes les règles de déduction respectent la même interface, ce qui permet de les gérer uniformément dans la logique de résolution.
- **Extensibilité** : De nouvelles règles peuvent être ajoutées sans modifier les règles existantes, grâce au modèle de classe abstraite et à la méthode apply.

Les Règles de Déduction : DR1, DR2 et DR3

Les trois règles de déduction (DR1, DR2, DR3) utilisent des concepts spécifiques du Sudoku pour réduire le champ des valeurs possibles et aboutir à une solution de la grille. Elles se basent toutes sur le **pattern Template Method**, avec apply servant de point d'entrée principal pour l'application de la règle.

1. DR1 : Singleton Nu

La règle DR1, implémentée dans la classe DR1, cible les "singletons nus". Elle détecte les cellules qui ont exactement **une seule valeur possible** et affecte cette valeur à la cellule.

- **Fonctionnement** :
    - Parcours de toutes les cellules de la grille.
    - Si une cellule n'a qu'une seule valeur possible dans possibleValues, elle est définie comme solution pour cette cellule et l'ensemble des valeurs possibles est vidé.
    - La méthode retourne true si un changement a été effectué, false sinon.
- Cette règle basique est nécessaire à la résolution d'un sudoku, c'est la règle la plus intuitive.

2. DR2 : Singleton Caché

La règle DR2, implémentée dans la classe DR2, vise les "singletons cachés", où une valeur n'a qu'une cellule possible dans une ligne, une colonne ou un bloc.

- **Fonctionnement** :
    - Applique la déduction ligne par ligne, colonne par colonne, puis sur chaque bloc 3x3.

    - Pour chaque ligne/colonne/bloc et pour chaque valeur possible (1 à 9), si cette valeur ne peut apparaître que dans une cellule spécifique, cette cellule est fixée à cette valeur.
- **Sous-méthodes** :
    - applyHiddenSingleInRow : Applique le singleton caché dans une ligne.
    - applyHiddenSingleInColumn : Applique le singleton caché dans une colonne.
    - applyHiddenSingleInBlock : Applique le singleton caché dans un bloc 3x3.
- Cette règle est plus complexe que DR1 et réduit les possibilités de manière indirecte, en exploitant les contraintes de positionnement pour les valeurs. C'est aussi une règle intuitive que l'on applique "à l'oeil" et qui est donc importante à implémenter.

3. DR3 : Paires Nues

La règle DR3, implémentée dans la classe DR3, gère les "paires nues". Si deux cellules d'une ligne, colonne ou bloc ont les mêmes deux valeurs possibles, alors ces valeurs peuvent être éliminées des autres cellules de cette même ligne, colonne ou bloc.

- **Fonctionnement** :
    - Pour chaque ligne, colonne et bloc, DR3 cherche des paires de cellules ayant exactement deux valeurs possibles identiques.
    - Si une paire est identifiée, ces deux valeurs sont supprimées des autres cellules de la ligne, colonne ou bloc, car elles sont garanties d'être dans les deux cellules de la paire.
- **Gestion des redondances** : DR3 utilise un ensemble dejaTraite pour éviter de traiter plusieurs fois la même paire.
- **Sous-méthodes** :
    - applyNakedPairsInRow : Applique les paires nues dans une ligne.
    - applyNakedPairsInColumn : Applique les paires nues dans une colonne.
    - applyNakedPairsInBlock : Applique les paires nues dans un bloc 3x3.
- Cette règle permet d'éliminer plusieurs valeurs possibles simultanément, accélérant la résolution dans les situations où des paires nues sont présentes.