SUDOKU

1. Introduction

Le Sudoku est un jeu de logique populaire qui consiste à remplir une grille de 9x9 cases avec des chiffres allant de 1 à 9, de manière à ce que chaque ligne, chaque colonne et chaque région de 3x3 cases contienne tous les chiffres sans répétition. Ce jeu présente non seulement un défi intellectuel pour les amateurs, mais il soulève également des questions fascinantes dans le domaine de l'informatique et de la résolution de problèmes.

Dans ce contexte, le développement d'un solver de Sudoku constitue une application intéressante des algorithmes de déduction et de l'intelligence artificielle. L'objectif de ce projet est de créer un programme capable de résoudre des grilles de Sudoku en utilisant un ensemble de règles de déduction définies, permettant ainsi d'évaluer la validité de chaque mouvement et d'optimiser le processus de résolution.

Pour atteindre cet objectif, nous avons conçu une architecture orientée objet en Java, où les différentes règles de déduction sont implémentées sous forme de classes dérivées d'une classe de base commune. L'algorithme de résolution suit un processus itératif, appliquant successivement ces règles à la grille jusqu'à ce qu'aucune nouvelle déduction ne puisse être faite. Lorsque des cases demeurent vides, le programme sollicite l'utilisateur pour qu'il entre manuellement des valeurs, permettant ainsi une interaction directe.

Ce rapport présente les défis rencontrés lors du développement du solver, les solutions apportées et les choix techniques effectués, tout en fournissant une vue d'ensemble de l'architecture du code et des classes développées.

2. Architecture du Code

2.1 Classe SudokuSolver

La classe SudokuSolver est une composante essentielle du projet, jouant un rôle crucial dans la gestion et le chargement de la grille de Sudoku. Conçue selon le modèle de conception Singleton, cette classe garantit qu'une seule instance du solver peut exister à tout moment, facilitant ainsi la gestion des ressources et l'accès à la grille. La grille elle-même est représentée par un tableau 2D de dimensions 9x9, permettant de stocker les valeurs des cases. Le constructeur privé initialise cette grille vide, tandis que la méthode statique getInstance() permet d'accéder à l'unique instance de la classe. Pour interagir avec la grille, deux méthodes clés sont fournies : setGrid(), qui permet de définir une nouvelle grille à partir d'un tableau 2D, et getGrid(), qui retourne la grille actuelle. La classe inclut également des fonctionnalités pour charger la grille à partir d'un fichier texte via la méthode readGridFromFile(), qui lit les entiers du fichier et les stocke dans un tableau linéaire, et la méthode convertTo2DGrid(), qui transforme ce tableau linéaire en une structure de données 2D. Le point d'entrée de l'application se trouve dans la méthode main(), où la grille est initialisée, et l'instance de SudokuSolverEngine est créée pour démarrer le processus de résolution. Grâce à cette conception, la classe SudokuSolver permet une gestion fluide de la grille, tout en s'intégrant efficacement au reste du système.

2.2 Classe DeductionRule

La classe DeductionRule constitue la base abstraite pour toutes les règles de déduction appliquées dans le solver de Sudoku. Elle est conçue pour être étendue par des classes spécifiques qui implémenteront des stratégies de résolution particulières. Le constructeur de la classe prend en paramètre un tableau 2D représentant la grille de Sudoku, ce qui permet à chaque règle d'accéder et de manipuler cette grille. La méthode abstraite appliquer() doit être définie dans les sous-classes, garantissant que chaque règle de déduction possède une méthode pour être exécutée. La classe fournit également des méthodes utilitaires qui facilitent la vérification des placements de chiffres. Par exemple, la méthode peutEtrePlace() détermine si un chiffre donné peut être placé dans une case spécifique sans violer les règles du Sudoku. Cette méthode vérifie la présence du chiffre dans la ligne, la colonne et la région 3x3 correspondante, assurant ainsi la validité du placement. La classe inclut aussi des méthodes pour vérifier la présence d'un chiffre dans une ligne (estDansLigne()), une colonne (estDansColonne()) ou une région 3x3 (estDansRegion()), facilitant la mise en œuvre de règles spécifiques dans les classes dérivées. Grâce à cette architecture, DeductionRule

joue un rôle fondamental en fournissant une structure claire et réutilisable pour les règles de déduction qui seront utilisées par le solver de Sudoku.

2.3 Règles de Déduction

DR1

La classe DR1 est une implémentation spécifique de la classe abstraite DeductionRule, qui représente l'une des stratégies de déduction utilisées dans le solver de Sudoku. Conçue pour appliquer une méthode de déduction basée sur la découverte de candidats uniques, DR1 cherche à remplir les cases vides de la grille en analysant les chiffres qui peuvent y être placés. Dans la méthode appliquer(), un processus itératif est mis en place, fonctionnant tant qu'il y a des changements dans la grille. Pour chaque case vide (indiquée par la valeur 0), la méthode trouverCandidats() est appelée pour déterminer les chiffres potentiellement valides qui peuvent être insérés sans enfreindre les règles du Sudoku. Si une case n'a qu'un seul candidat possible, ce chiffre est inséré dans la grille, et une variable changement est mise à true pour indiquer qu'une modification a été effectuée. Ce mécanisme permet de progresser vers une solution en remplissant progressivement la grille avec des valeurs valides. La méthode trouverCandidats() utilise la méthode peutEtrePlace() de la classe parente pour vérifier les règles de placement pour chaque chiffre de 1 à 9. En résumé, DR1 joue un rôle essentiel dans la résolution de la grille en appliquant une stratégie systématique pour identifier et placer des chiffres uniques, contribuant ainsi à la progression vers une solution complète.

DR2

La classe DR2 constitue une autre implémentation de la classe abstraite DeductionRule, mettant en œuvre une stratégie de déduction basée sur l'identification de positions uniques pour chaque chiffre dans les lignes, les colonnes et les régions 3x3 de la grille de Sudoku. Comme pour la classe DR1, cette classe utilise un processus itératif pour continuer les tentatives de placement jusqu'à ce qu'aucun changement supplémentaire ne soit observé. La méthode appliquer() examine d'abord chaque ligne de la grille : pour chaque chiffre de 1 à 9, elle recherche des cases vides où le chiffre peut être placé. Si une seule case est trouvée, ce chiffre est inséré, et une variable changement est mise à jour pour signaler qu'une modification a été effectuée. Ce processus est ensuite répété pour chaque colonne, suivant la même logique. Enfin, DR2 analyse les régions 3x3 de la grille, identifiant également les positions uniques possibles pour chaque chiffre. Si une case unique est trouvée dans une région, le chiffre correspondant est placé dans cette case. L'approche systématique de DR2 permet d'explorer efficacement les possibilités de placement, augmentant ainsi les chances de remplir la grille de manière cohérente. En offrant des vérifications à trois niveaux (lignes,

colonnes et régions), DR2 complète la stratégie de déduction de manière robuste, contribuant à la résolution globale du puzzle de Sudoku.

DR3

La classe DR3 est une autre implémentation de la classe abstraite DeductionRule, apportant une approche plus avancée à la résolution des grilles de Sudoku en utilisant une méthode de déduction axée sur l'obtention et la réduction des candidats possibles pour chaque case vide. Dans la méthode appliquer(), un processus itératif est mis en place pour identifier et placer les chiffres en fonction des candidats disponibles. Pour chaque case vide, la méthode obtenirCandidats() génère un ensemble de chiffres valides, et si cet ensemble ne contient qu'un seul candidat, ce chiffre est directement placé dans la grille. Cette approche garantit que les placements sont effectués efficacement, réduisant ainsi la complexité du puzzle.

En complément de cette première phase, DR3 procède à une réduction systématique des candidats dans les lignes, les colonnes et les régions 3x3. Pour chaque chiffre de 1 à 9, la méthode reductionCandidatsRegion() examine chaque région 3x3 pour vérifier s'il existe des positions uniques où le chiffre peut être placé, permettant ainsi des placements supplémentaires lorsque cela est possible. De manière similaire, les méthodes reductionCandidatsLigne() et reductionCandidatsColonne() effectuent des vérifications similaires dans leurs respectives lignes et colonnes. Si un chiffre peut être placé dans une seule case d'une ligne ou d'une colonne, il est également inséré dans la grille. Cette classe combine donc une identification proactive des candidats avec une stratégie de réduction, ce qui contribue à optimiser le processus de résolution du Sudoku et à augmenter l'efficacité de l'algorithme dans son ensemble.

2.4 Classe DeductionRuleFactory

L’objectif principal est de centraliser la création des différentes règles de déduction (DeductionRule) utilisées pour résoudre la grille. La méthode statique createRule() prend en entrée une chaîne de caractères représentant le type de règle souhaité ainsi qu'une grille de Sudoku sous forme de tableau 2D. En fonction de l'argument ruleType, la méthode instancie et renvoie l'objet correspondant à la règle de déduction appropriée : DR1, DR2 ou DR3. Cette approche permet d'isoler la logique de création des objets, rendant le code plus modulable et facilement extensible. Par exemple, si de nouvelles règles de déduction devaient être ajoutées à l'avenir, il suffirait de créer une nouvelle classe dérivée de DeductionRule et d'ajouter un cas correspondant dans le switch. Si un type de règle non reconnu est fourni, la méthode génère une exception IllegalArgumentException, garantissant ainsi une gestion appropriée des erreurs. En

résumé, DeductionRuleFactory simplifie la gestion des règles de déduction, favorisant un code organisé et extensible.

2.5 Interfaces et Objets Observer

- Interface RegleObserver: Définit un contrat pour les observateurs souhaitant être informés des applications de règles.
- Classe GrilleDisplay: Implemente RegleObserver pour afficher la grille après chaque application de règle.
- Classe ObservableSolver: Gère les observateurs, notifiant les changements dans la grille lors de l'application des règles.

2.6 Commandes

- **Interface Command**: Modèle pour les objets de commande permettant d'exécuter et d'annuler des actions.
- **Classe PlaceNumberCommand**: Implemente l'interface Command pour gérer l'action de placer un chiffre dans la grille, stockant les informations nécessaires pour une manipulation flexible et sécurisée.

2.7 Classe SudokuSolverEngine

La classe SudokuSolverEngine est au cœur de l'application de résolution de Sudoku, manipulant l'ensemble des processus nécessaires pour résoudre une grille de Sudoku donnée. Elle maintient une liste de règles de déduction qui sont appliquées de manière séquentielle pour remplir la grille, et elle utilise une pile de commandes pour gérer les actions effectuées, permettant ainsi des opérations d'annulation. Dans son constructeur, la classe prend une grille de Sudoku initiale et instancie trois règles de déduction à l'aide d'une fabrique dédiée.

La méthode solve() applique chaque règle à tour de rôle et notifie les observateurs de tout changement dans la grille, en affichant l'état de la grille après chaque application. Si la grille est résolue, la classe détermine le niveau de difficulté basé sur la règle qui a permis de résoudre la grille. En cas d'échec, elle invite l'utilisateur à entrer manuellement des chiffres, tout en s'assurant que les valeurs ajoutées respectent les règles du Sudoku. La méthode remplirGrilleManuellement() gère ce processus interactif, permettant à l'utilisateur de remplir la grille jusqu'à sa résolution. En résumé, SudokuSolverEngine sert de médiateur entre la logique de résolution et l'interface utilisateur, facilitant l'interaction et l'affichage des résultats tout en appliquant des stratégies de résolution de manière ordonnée.

3. Conclusion

Le développement du solveur de Sudoku a présenté divers défis, notamment la mise en œuvre des règles de déduction et la gestion de l'interaction utilisateur. Grâce à une architecture orientée objet et à des modèles de conception appropriés, le projet a été réalisé avec succès, offrant un outil interactif pour résoudre des grilles de Sudoku. Les différentes classes et leur structure permettent une extensibilité future, favorisant l'ajout de nouvelles règles et fonctionnalités. Ce projet met en évidence non seulement l'importance des algorithmes de déduction dans la résolution de problèmes, mais aussi l'interaction entre les différentes composantes d'un système logiciel.