Semestre 1

Projet de solveur de sudoku

M1 Informatique

RAPPORT DE PROJET

Matis Chabanat et Raphaël Viaud

Session - 2024/2025

Novembre 2024

S1 : Software Engineering

Enseignant : Monsieur J.C Regin

UNIVERSITÉ CÔTE D'AZUR

Introduction

Dans ce projet, notre objectif était de concevoir un solveur de Sudoku à l'aide de trois règles de déduction que nous avons implémentées. Ces règles ont été choisies pour être de complexité croissante. Chacune d'entre elles est une classe qui découle d'une même classe mère. Nous avons choisi de faire ce projet en Python car l'un comme l'autre nous nous sentions bien plus à l'aise qu'en Java.

Description du code

Avant même de commencer à faire notre code, la première étape fut de choisir nos règles de déduction pour qu'elles aient toutes une complexité différente et soient capables de remplir des valeurs plus ou moins évidentes. Pour cela, nous avons regardé les règles les plus communes que nous avons testées sur quelques grilles pour étudier leur efficacité. Ceci nous a permis de choisir :

- DR1 : Direct Solve
- DR2 : Hidden Single
- DR3 : Naked Pair

La règle DR1 (Direct Solve) recherche les cases où une seule valeur est possible en fonction des valeurs présentes dans la ligne, la colonne et le bloc 3x3. La règle DR2 (Hidden Single) cherche des « singles cachés », c'est-à-dire des chiffres qui, bien que non directement évidents, sont les seuls possibles dans une ligne, colonne ou bloc. La règle DR3 (Naked Pair) identifie les « paires nues », où deux cases dans une ligne ou une colonne partagent les mêmes deux valeurs possibles, permettant de réduire les options pour les autres cases.

Nous avons donc commencé le code par ces trois sous-classes de la classe mère « DeductionRule ». Une fois que nous les avions implémentées, nous devions les tester sur des grilles. Après plusieurs versions et ajustements, nous avons finalisé ces classes.

Une fois que nous avions nos trois règle nous avons commencé à développer la classe Sudoku avec toutes les fonctions nécessaires pour résoudre des grilles en respectant les consignes du projet.

Notre classe « SudokuSolver » finale contient 10 fonctions :

__init__ initialise plusieurs attributs essentiels : il crée une grille vide de 9x9, représentée comme une liste de listes, où chaque case est initialisée à 0. Il instancie aussi les trois règles de déduction DR1, DR2 et DR3.

La méthode possible_values est utilisée dans les classes DR1, DR2 et DR3 ; elle détermine les valeurs possibles pour une case donnée, en fonction des règles de Sudoku.

La méthode load_grid_from_file permet de charger une grille depuis un fichier texte comme demandé dans l'énoncé. Chaque ligne du fichier doit contenir exactement 9 valeurs, séparées par des virgules, et la grille doit comporter précisément 9 lignes.

La méthode is_valid vérifie si une valeur peut être placée dans une case sans violer les règles du Sudoku, car il n'aurait pas de sens de laisser l'utilisateur entrer une valeur qui ne répond pas aux contraintes du Sudoku.

prompt_user_for_value est la méthode qui permet l'interaction avec l'utilisateur lorsqu'il reste des cases vides après application des règles de déduction. Nous avons remarqué que parfois, la grille ne se met pas à jour directement après la saisie d'une valeur, mais nous n'avons pas trouvé la cause.

La méthode get_integer_input est une fonction utilitaire qui est utilisée dans prompt_user_for_value afin de s'assurer que les valeurs sont celles attendues.

La méthode apply_rules coordonne l'application des règles de déduction. Elle exécute chaque règle jusqu'à ce qu'aucun changement ne soit apporté à la grille.

La méthode classify_difficulty utilise cet ensemble de règles appliquées pour déterminer la difficulté de la grille pour la question bonus. Si seule la règle DR1 a été utilisée, la grille est classée comme « Facile ». Si DR2 a également été nécessaire, elle est classée comme « Moyen ». Si DR3 a aussi servi alors la grille est classée « Difficile » et si l'utilisateur a été sollicité alors c'est « Très difficile ».

La méthode solve commence par appliquer les règles pour remplir le plus de cases possible. S'il reste des cases vides après l'application des règles, elle fait appel à prompt_user_for_value. Après chaque saisie de l'utilisateur, elle réapplique les règles à tour de rôle pour essayer de compléter la grille jusqu'à ce qu'elle soit entièrement remplie. Une fois la grille résolue, elle l'affiche pour que l'utilisateur voie le résultat final.

Enfin, la méthode display_grid permet l'affichage visuel de la grille. Elle remplace les cases vides par des points pour une lecture plus claire et affiche la grille dans un format facilement lisible, ce qui permet de suivre l'état de la résolution à chaque étape.

Dans l'ensemble, la classe SudokuSolver est conçue pour gérer l'ensemble du flux de résolution du Sudoku, en appliquant des règles, en facilitant l'interaction avec l'utilisateur, et en offrant un affichage clair de la progression de la grille.

Difficultés

Une des difficultés majeures du projet fut de gérer l'interaction avec l'utilisateur pour remplir des valeurs lorsque le solveur était bloqué. Car le solveur en lui-même n'était pas très dur à concevoir ; en revanche, la question bonus nous a fait réfléchir davantage. Nous avions des problèmes au début dans la fonction prompt_user_for_value que nous avons débuguée à l'aide de « print ».

De plus, intégrer cette interaction tout en assurant que le solveur puisse reprendre l'application des règles automatiquement après chaque saisie a également compliqué la structure du programme.

Un autre problème fut de trouver des grilles de la bonne difficulté pour pouvoir tester le bon fonctionnement du code et des différents niveaux de grilles. Pour valider notre solveur et notre classification des niveaux de difficulté, il était essentiel de disposer de grilles représentatives des différents types de complexité : "Facile", "Moyen", "Difficile" et "Très difficile". Il n'a pas été évident de trouver des grilles déjà catégorisées avec précision. La génération de grilles adaptées a également mis en évidence des limites dans certaines règles de déduction, que nous avons dû affiner pour que le solveur puisse gérer des cas variés sans intervention superflue de l'utilisateur.

Au départ, nous avions choisi la règle "Locked Candidate" pour DR2, car elle permet de trouver des candidats bloqués dans une ligne ou colonne d'une sous-grille 3x3. Cependant, cela nous a posé problème : il était très difficile de trouver des grilles qui pouvaient être résolues avec DR1 et DR2 seulement, sans nécessiter DR3. Cela posait un problème pour la classification des niveaux de difficulté, car nous n'arrivions pas à obtenir des grilles de difficulté "Moyen".

Nous en avons conclu que DR2 et DR3 étaient de complexité similaire. Pour résoudre ce problème, nous avons décidé de remplacer "Locked Candidate" par "Hidden Single" pour DR2. La règle "Hidden Single" est plus simple et permet de remplir un plus grand nombre de cases après l'application de DR1, tout en restant dans un niveau de difficulté modéré. Ce changement nous a permis de mieux distinguer les niveaux de difficulté "Moyen" et "Difficile" et de valider notre solveur sur des grilles variées, chaque niveau correspondant maintenant à l'utilisation des règles prévues.

Nous avons déposé dans notre GitHub des grilles de chaque niveau pour vous permettre de tester son bon fonctionnement si nécessaire.

Conclusion

Ce projet nous a permis de consolider nos compétences en programmation orientée objet, en particulier en Python. Nous avons dû bien manipuler les listes pour naviguer dans une grille et implémenter nos règles de déduction. Ce projet nous a permis de mieux comprendre la logique de résolution de Sudoku et de concevoir une solution structurée, modulaire et facile à utiliser. Le cours de « Problem Solving » nous a aussi aidés au niveau de la réflexion sur les règles de déduction et les contraintes générales du Sudoku.