BUT informatique
Groupe 3

RAPPORT
Sudoku

Clément JANNAIRE
Clémence DUCREUX

2023-2024

SOMMAIRE

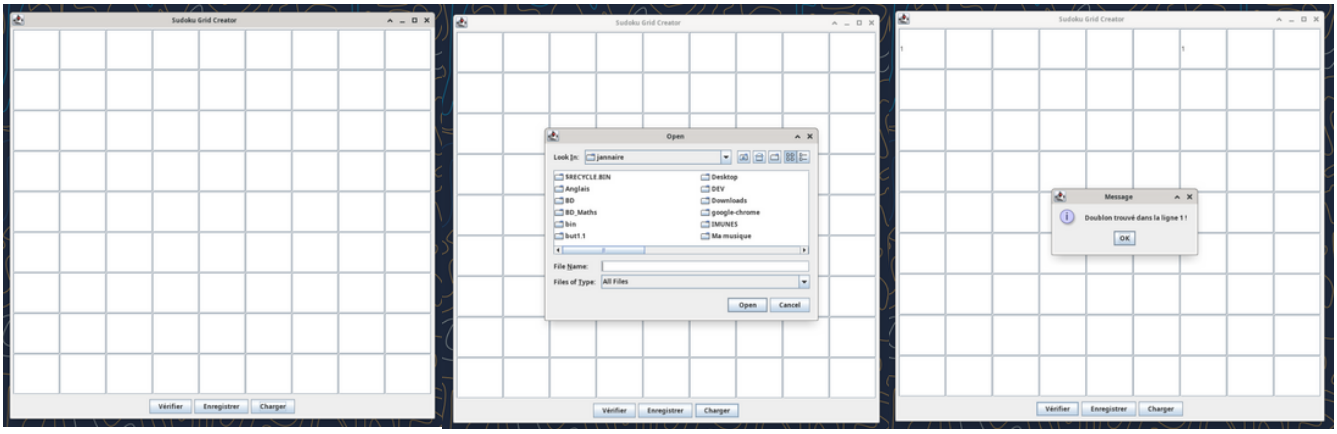
INTRODUCTION

Le projet Sudoku vise à développer deux programmes en Java pour la conception et la résolution de grilles de Sudoku. Le premier programme est dédié à la création de grilles de départ, tandis que le second permet de résoudre les grilles soit manuellement, soit automatiquement.
Ces programmes doivent respecter les règles du Sudoku et offrir une interface conviviale pour les utilisateurs.

FONCTIONNALITÉS

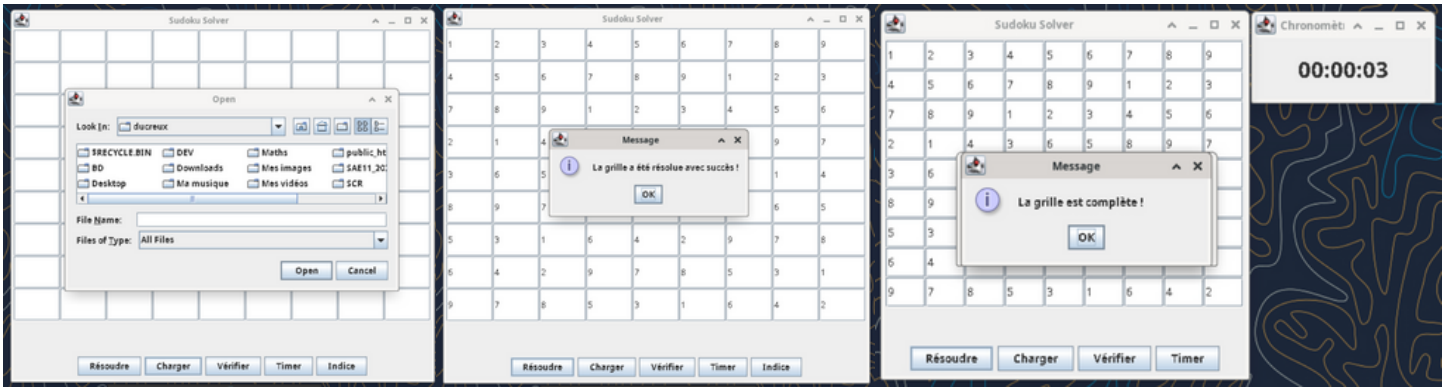
PROGRAMME DE CRÉATION DE GRILLES

-Création de grilles vides ou chargement de grilles existantes depuis un fichier.
-Ajout ou suppression de numéros dans la grille, tout en respectant les contraintes du Sudoku.
-Sauvegarde des grilles dans un fichier au format spécifié.



PROGRAMME DE RÉSOLUTION DE GRILLES

-Chargement de grilles depuis un fichier.
-Résolution automatique ou manuelle des grilles.
-Affichage de la grille résolue et du temps nécessaire à la résolution.
-Les indices ne sont pas fonctionnelle



STRUCTURE PROGRAMME

Les deux programmes partagent une structure similaire, avec plusieurs classes communes pour optimiser le code et la réutilisation.

Le fichier **GridIO.java** contient une classe du même nom qui est responsable de la sauvegarde et du chargement d'une grille de Sudoku à partir et vers un fichier. Voici une brève présentation de sa structure :

- **Importations** : Le fichier commence par l'importation des classes nécessaires.
- **Déclaration de la classe** : La classe **GridIO** est déclarée.
- **Méthode saveGrid :** Cette méthode prend en paramètre une grille de Sudoku sous forme de tableau de champs de texte et enregistre les valeurs dans un fichier. Elle parcourt chaque case de la grille, convertit les valeurs en chaînes de caractères et les écrit dans le fichier.
- **Méthode loadGrid :** Cette méthode charge une grille de Sudoku à partir d'un fichier. Elle utilise un sélecteur de fichier pour permettre à l'utilisateur de choisir le fichier à charger. Ensuite, elle lit chaque ligne du fichier et met à jour la grille en conséquence.

Le fichier **GridValidator.java** contient une classe du même nom qui est responsable de la validation d'une grille de Sudoku. Voici une brève présentation de sa structure :

- **Importations** : Le fichier commence par l'importation des classes nécessaires.
- **Déclaration de la classe** : La classe **GridValidator** est déclarée.
- **Méthode validateGrid** : Cette méthode vérifie si les valeurs saisies dans la grille sont valides, c'est-à-dire des chiffres compris entre 1 et 9. Elle parcourt chaque case de la grille et affiche des messages d'erreur si une valeur est invalide.
- **Méthode verifyDuplicates** : Cette méthode vérifie s'il n'y a pas de doublons dans les lignes, les colonnes et les régions de la grille. Elle utilise les méthodes **verifyRow**, **verifyColumn** et **verifyRegions** pour effectuer ces vérifications.
- **Méthode verifyRow** : Cette méthode vérifie s'il y a des doublons dans une ligne donnée de la grille.
- **Méthode verifyColumn** : Cette méthode vérifie s'il y a des doublons dans une colonne donnée de la grille.
- **Méthode verifyRegions** : Cette méthode vérifie s'il y a des doublons dans chaque région de 3x3 de la grille.
- **Méthode verifySubGrid** : Cette méthode est utilisée par **verifyRegions** pour vérifier les doublons dans une région de 3x3 spécifique.

Le fichier **SudokuGridCreator.java** contient une classe du même nom, qui représente l'interface graphique pour la création et la manipulation de grilles de Sudoku. Voici une présentation de sa structure :

- I**mportations** : Le fichier commence par les importations nécessaires des packages Swing et IO pour l'interface graphique et la manipulation de fichiers.

- D**éclaration de la classe** : La classe **SudokuGridCreator** est déclarée. Elle contient les éléments nécessaires à la création de l'interface graphique.

- C**onstantes** : Une constante **GRID_SIZE** est définie pour déterminer la taille de la grille de Sudoku.

- A**ttributs** : Les attributs **frame**, **panel** et **grid** sont déclarés pour représenter respectivement la fenêtre principale, le panneau de la grille et la grille elle-même.

- C**onstructeur** : Le constructeur initialise la fenêtre principale, crée le panneau de la grille et initialise la grille en créant des champs de texte pour chaque cellule.

- M**éthodes d'action des boutons** : Des méthodes sont définies pour réagir aux actions des boutons. **verifyButton** vérifie la validité de la grille, **saveButton** sauvegarde la grille dans un fichier, et **loadButton** charge une grille depuis un fichier.

- M**éthodes de sauvegarde et de chargement de la grille** : Ces méthodes permettent respectivement de sauvegarder et de charger une grille depuis un fichier.

- M**éthode main** : La méthode **main** crée une instance de **SudokuGridCreator** dans l'Event Dispatch Thread (EDT) en utilisant **SwingUtilities.invokeLater**.

Le fichier **SudokuSolver.java** contient une classe du même nom, qui implémente l'algorithme de résolution d'un Sudoku. Voici une présentation de sa structure :

- **Importations :** Le fichier commence par l'importation de **Arrays** pour l'affichage de la grille.

- **Déclaration de la classe** : La classe **SudokuSolver** est déclarée. Elle contient les méthodes nécessaires pour résoudre un Sudoku.

- **Attributs** : L'attribut **grid** est déclaré pour stocker la grille de Sudoku à résoudre.

- **Constructeur** : Le constructeur initialise la grille avec la grille fournie en paramètre.

- **Méthode solve :** Cette méthode implémente l'algorithme de résolution récursive du Sudoku. Elle cherche d'abord une case vide, puis essaie de placer un nombre valide dans cette case. Si cela conduit à une solution valide, elle récursivement essaie de résoudre le reste de la grille. Si aucune solution n'est trouvée, elle annule les modifications précédentes.

- **Méthode findEmptyCell :** Cette méthode recherche la première case vide dans la grille et retourne ses coordonnées.

- **Méthode isValidMove** : Cette méthode vérifie si placer un certain nombre dans une certaine position de la grille est valide selon les règles du Sudoku.

- **Méthode printGrid** : Cette méthode affiche la grille actuelle dans la console.

- **Méthode main :** La méthode **main** démontre l'utilisation de la classe en résolvant une grille de Sudoku prédéfinie.

- **Méthode getGrid** : Cette méthode permet d'accéder à la grille résolue.

Le fichier **SudokuSolverGUI.java** contient une classe du même nom, qui implémente une interface graphique pour résoudre des grilles de Sudoku. Voici une brève présentation de sa structure :

- **Importations** : Le fichier commence par l'importation de différentes classes Swing et d'autres classes nécessaires.

- **Déclaration de la classe** : La classe **SudokuSolverGUI** est déclarée. Elle contient les composants Swing nécessaires pour créer l'interface graphique.

- **Attributs** : Les différents composants Swing tels que JFrame, JPanel, JButton et JTextField sont déclarés comme attributs de classe.

- **Constructeur** : Le constructeur initialise la fenêtre JFrame et les différents composants de l'interface graphique. Il les dispose dans un arrangement de GridLayout.

- **Méthodes de gestion des événements** : Il y a des méthodes pour gérer les événements des boutons, telles que **solveSudoku**, **loadGrid**, **verifyGrid**, **startTimer** et **provideHint**. Chacune de ces méthodes est invoquée en réponse à un clic sur un bouton spécifique et effectue une action correspondante.

- **Méthode solveSudoku** : Cette méthode récupère la grille saisie par l'utilisateur et utilise la classe **SudokuSolver** pour résoudre le Sudoku.

- **Méthode loadGrid** : Cette méthode permet à l'utilisateur de charger une grille de Sudoku à partir d'un fichier.

- **Méthode verifyGrid** : Cette méthode vérifie si la grille actuelle est valide et ne contient aucun doublon.

- **Méthode startTimer** : Cette méthode démarre un chronomètre pour mesurer le temps mis par l'utilisateur pour résoudre le Sudoku.

- **Méthode provideHint** : Cette méthode fournit un indice à l'utilisateur en remplissant aléatoirement une case vide avec un nombre valide.

- **Méthode getRandomValidNumber** : Cette méthode génère un nombre valide pour une case donnée. Dans l'exemple donné, elle retourne simplement "5" à titre d'exemple.

- **Méthode main** : La méthode **main** initialise l'interface graphique en créant une instance de **SudokuSolverGUI** dans l'EDT (Event Dispatch Thread).

Le fichier **SudokuSolverGUI.java** contient une classe du même nom, qui implémente une interface graphique pour résoudre des grilles de Sudoku. Voici une brève présentation de sa structure :

- **Importations** : Le fichier commence par l'importation de différentes classes Swing et d'autres classes nécessaires.
- **Déclaration de la classe** : La classe **SudokuSolverGUI** est déclarée. Elle contient les composants Swing nécessaires pour créer l'interface graphique.
- **Attributs** : Les différents composants Swing tels que JFrame, JPanel, JButton et JTextField sont déclarés comme attributs de classe.
- **Constructeur** : Le constructeur initialise la fenêtre JFrame et les différents composants de l'interface graphique. Il les dispose dans un arrangement de GridLayout.
- **Méthodes de gestion des événements** : Il y a des méthodes pour gérer les événements des boutons, telles que **solveSudoku**, **loadGrid**, **verifyGrid**, **startTimer** et **provideHint**. Chacune de ces méthodes est invoquée en réponse à un clic sur un bouton spécifique et effectue une action correspondante.
- **Méthode solveSudoku** : Cette méthode récupère la grille saisie par l'utilisateur et utilise la classe **SudokuSolver** pour résoudre le Sudoku.
- **Méthode loadGrid** : Cette méthode permet à l'utilisateur de charger une grille de Sudoku à partir d'un fichier.
- **Méthode verifyGrid** : Cette méthode vérifie si la grille actuelle est valide et ne contient aucun doublon.
- **Méthode startTimer** : Cette méthode démarre un chronomètre pour mesurer le temps mis par l'utilisateur pour résoudre le Sudoku.
- **Méthode provideHint** : Cette méthode fournit un indice à l'utilisateur en remplissant aléatoirement une case vide avec un nombre valide.
- **Méthode getRandomValidNumber** : Cette méthode génère un nombre valide pour une case donnée. Dans l'exemple donné, elle retourne simplement "5" à titre d'exemple.
- **Méthode main** : La méthode **main** initialise l'interface graphique en créant une instance de **SudokuSolverGUI** dans l'EDT (Event Dispatch Thread).

Ce fichier est un fichier **Makefile** qui définit les règles pour la compilation et l'exécution des différents fichiers source de votre programme Sudoku Solver. Voici une brève présentation de sa structure :

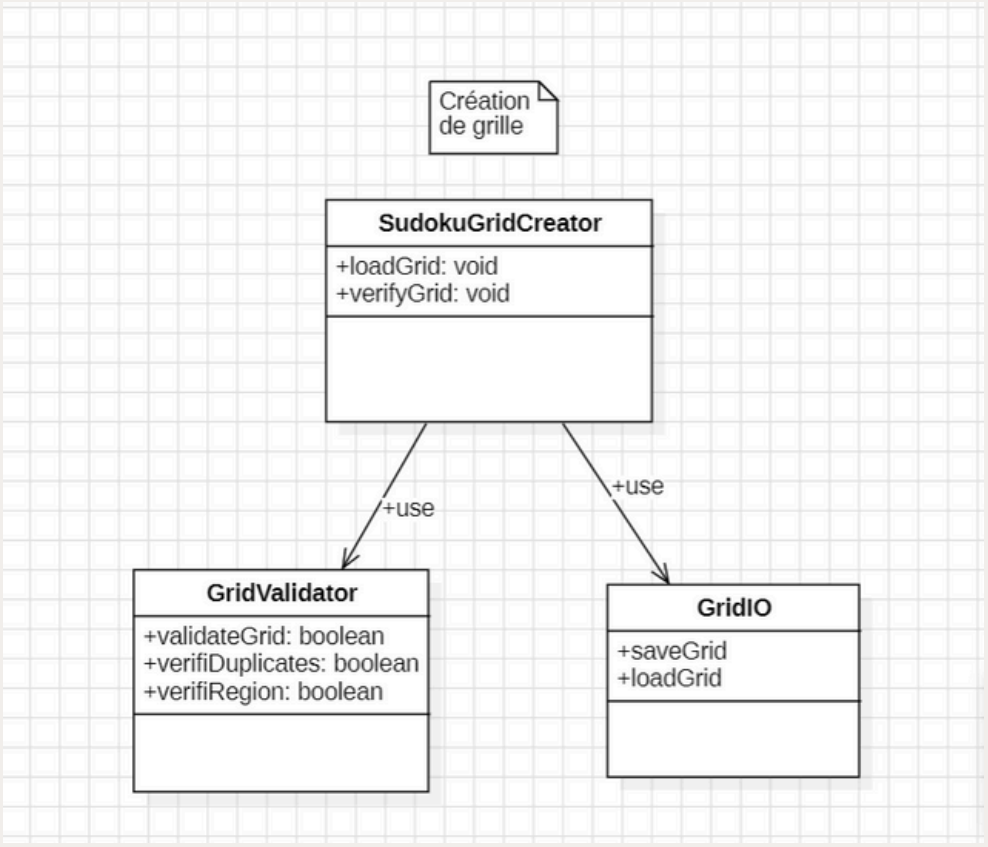
- **Variables** : Il définit des variables pour les commandes de compilation, les répertoires source et binaire, ainsi que les noms des classes principales et des fichiers source.

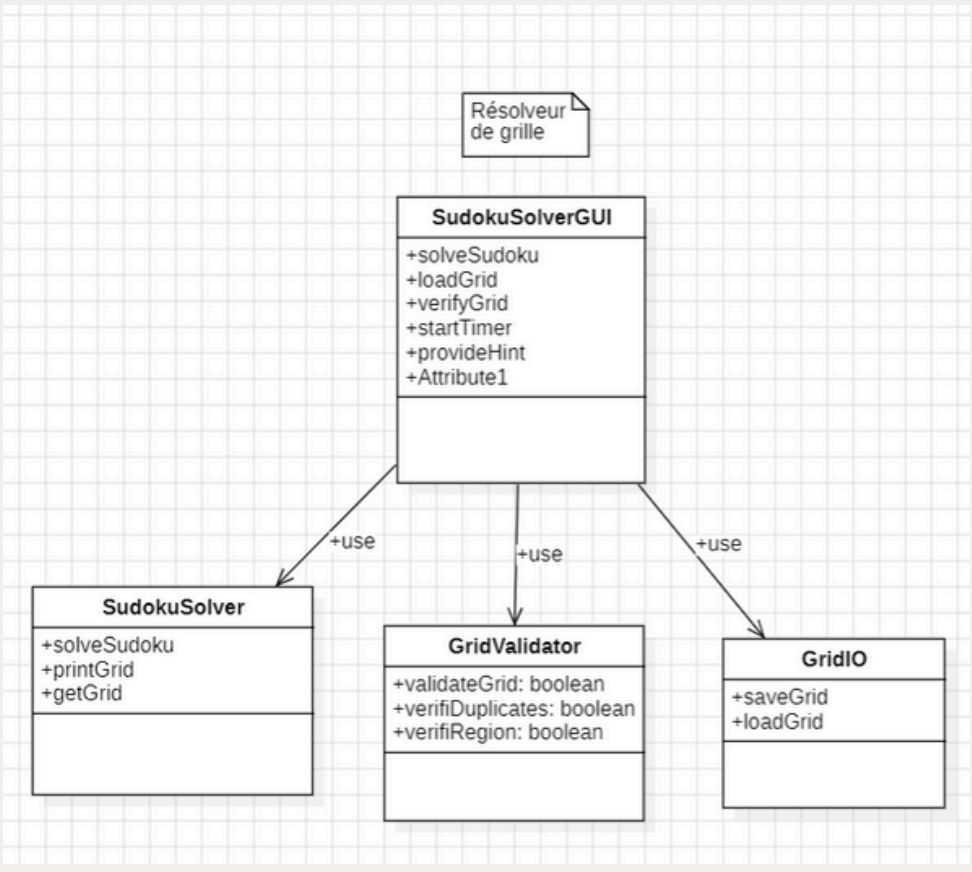
- **Cibles par défaut et règles de construction** : La cible par défaut est **all**, qui compile tous les fichiers source. Chaque règle de construction utilise **javac** pour compiler un fichier source spécifique en un fichier **.class** dans le répertoire binaire.

- **Cibles d'exécution** : Il y a deux cibles d'exécution définies : **run_creator** pour exécuter la classe **SudokuGridCreator**, et **run_solver** pour exécuter la classe **SudokuSolverGUI**. Ces cibles utilisent **java** pour exécuter la classe correspondante à partir du répertoire binaire.

- **Cible de nettoyage** : La cible **clean** est définie pour supprimer tous les fichiers **.class** dans le répertoire binaire.

DIAGRAME DE CLASSE



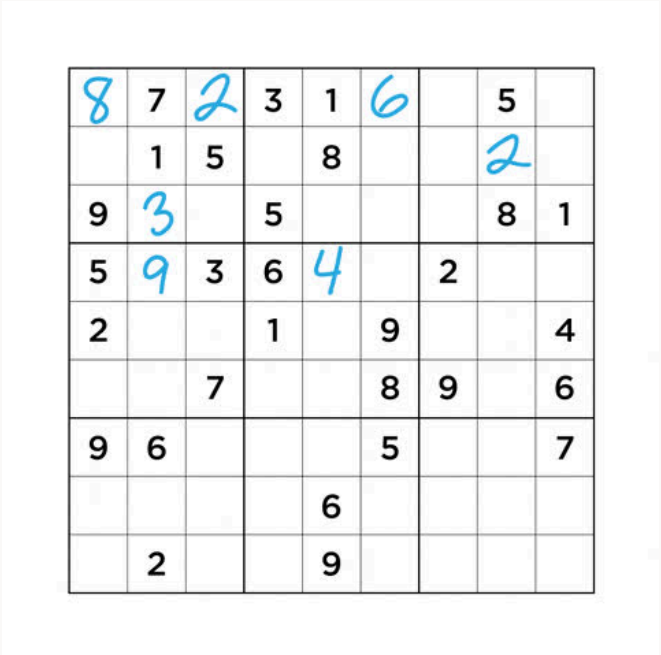


ALGORITHMES

L'algorithme utilisé dans la classe **SudokuSolver** est une approche de résolution de Sudoku basée sur la méthode de récursivité et le backtracking.

1. **Méthode solve()** :
   - La méthode **solve()** est la méthode principale qui tente de résoudre le Sudoku en appelant récursivement la méthode **solve()** jusqu'à ce que toutes les cases soient remplies.
   - Elle commence par rechercher la première case vide dans la grille en appelant **findEmptyCell()**.
   - Si aucune case vide n'est trouvée, cela signifie que la grille est résolue et la méthode renvoie **true**.
   - Sinon, pour chaque nombre de 1 à 9, elle vérifie si ce nombre peut être placé dans la case vide actuelle en appelant **isValidMove()**.
   - Si un nombre valide est trouvé, il est placé dans la case et la méthode **solve()** est rappelée récursivement.
   - Si la méthode **solve()** retourne **true**, cela signifie que la grille est résolue, sinon, la modification est annulée en remplaçant la valeur par 0.
2. **Méthode findEmptyCell()** :
   - Cette méthode parcourt la grille pour trouver la première case vide (représentée par un 0).
   - Elle renvoie un tableau d'entiers contenant les coordonnées de la première case vide trouvée (ligne, colonne).
3. **Méthode isValidMove(int row, int col, int num)** :
   - Cette méthode vérifie si un nombre donné peut être placé dans une certaine position de la grille sans violer les règles du Sudoku.
   - Elle vérifie d'abord la validité du nombre dans la ligne, la colonne et la région 3x3 correspondante.
   - Si le nombre peut être placé sans violer les règles, la méthode renvoie **true**, sinon elle renvoie **false**.

L'algorithme récursif teste toutes les combinaisons possibles de nombres jusqu'à ce qu'une solution soit trouvée ou que toutes les possibilités aient été épuisées, auquel cas il retourne **false**.



CONCLUSION

Clément:

En résumé, ce projet Sudoku en Java a été une étape importante dans notre parcours en programmation. Bien que ce ne soit pas notre premier projet en groupe, il nous a offert de nouveaux défis et nous a permis de consolider nos compétences déjà acquises. Travailler sur la résolution de grilles de Sudoku a été une expérience gratifiante, nous confrontant à des problèmes concrets et nous incitant à trouver des solutions innovantes. En tant qu'étudiant en première année de BUT informatique, je suis reconnaissant pour cette opportunité d'apprentissage et je suis impatient de mettre en pratique les enseignements tirés de ce projet dans mes futurs projets.

Clémence:

Pour conclure, cette expérience de développement Sudoku a été une étape marquante de notre formation en informatique. Bien que nous ayons déjà collaboré sur plusieurs projets, celui-ci nous a permis d'approfondir nos compétences techniques et de développer notre capacité à résoudre des problèmes complexes. La résolution de grilles de Sudoku a été à la fois stimulante et enrichissante, nous poussant à repousser nos limites et à explorer de nouvelles méthodes de résolution. Je suis reconnaissant pour cette opportunité d'apprentissage et je suis convaincu que cette expérience nous sera précieuse dans notre parcours universitaire et professionnel.