Document de conception :

Les cas d’utilisation :

Charger une grille :

Ce diagramme illustre le processus de chargement d’une grille de Sudoku par l’utilisateur. Celui-ci peut choisir de charger une grille à partir d’un fichier ou de la saisir manuellement. Une fois la grille récupérée, le système effectue une validation pour vérifier qu’elle respecte les règles du Sudoku. Si la grille est correcte, elle est affichée dans l’interface utilisateur, soit en mode texte, soit en mode graphique. En revanche, si la grille est invalide (par exemple, si elle contient des doublons ou une taille incorrecte), un message d’erreur est affiché pour informer l’utilisateur, comme indiqué par l’extension `<<extend>> Message d’erreur si grille non valide`.

Générer une grille :

Ce diagramme montre les étapes permettant de générer une nouvelle grille de Sudoku. L’utilisateur commence par sélectionner un niveau de difficulté (facile, moyen ou difficile). Le système crée ensuite une grille complète en respectant les règles du Sudoku avant d’effectuer une validation (`<<include>> Validation de la grille`). Une fois la grille validée, le programme supprime certaines valeurs afin de la rendre jouable en fonction du niveau de difficulté choisi. Enfin, la grille est affichée, et l’utilisateur peut soit l’enregistrer pour plus tard, soit commencer à jouer immédiatement. Ce diagramme met en évidence l’importance de la validation avant l’affichage de la grille pour éviter des erreurs dans la génération.

Résolution :

Ce diagramme décrit le processus de résolution d’une grille de Sudoku en tenant compte des différentes méthodes disponibles. L’utilisateur peut choisir entre une résolution par **Backtracking**, qui utilise une approche exhaustive, ou une résolution par **Déduction**, qui applique des règles logiques pour contraindre les valeurs possibles. Dans le cas des grilles **Multidoku**, les mêmes méthodes sont appliquées en tenant compte des contraintes supplémentaires. Une fois l’algorithme sélectionné, le système l’exécute et affiche la solution. Cependant, si la grille est impossible à résoudre (par exemple, en raison d’un manque d’indices ou d’une erreur dans la grille), un message d’erreur est affiché (`<<extend>> Afficher un message d’erreur`). Ce diagramme met en avant la modularité des méthodes de résolution et la gestion des erreurs lorsque la résolution échoue.

Le diagramme de classe :

Le diagramme UML du projet Sudoku représente une architecture modulaire bien structurée autour de la classe centrale `Grille`, utilisée par plusieurs composants pour la génération, la résolution et l'affichage des grilles. Les méthodes de résolution incluent `Backtracking`, qui explore toutes les combinaisons possibles, et `Deduction`, qui applique des règles logiques pour restreindre les choix, tandis que `ResolveurCombine` combine ces deux approches pour une résolution plus efficace. L'interface utilisateur est gérée par `SudokuModeGraphique` et `SudokuModeTexte`, offrant respectivement une interaction en mode visuel et textuel. `SudokuGenerator`, qui hérite de `SudokuModeGraphique`, permet de créer des grilles complètes ou incomplètes selon un niveau de difficulté. La classe `Logger` est utilisée pour suivre les étapes de la résolution et faciliter le débogage. Le programme est orchestré par `Main`, qui demande à l'utilisateur de choisir la taille de la grille, le type de jeu (`Sudoku` ou `Multidoku`), et le mode d'affichage, avant d'instancier la classe correspondante. `Main` veille également à fermer proprement les fichiers de log pour éviter toute erreur d'écriture. Cette conception assure une séparation claire des responsabilités, une grande flexibilité dans l'utilisation des solveurs et une interface adaptable aux besoins des utilisateurs.

Diagramme de séquences :

Résolution d'une grille avec `ResolveurCombine` :

Ce diagramme illustre le processus de résolution d'une grille en combinant deux méthodes complémentaires. L'utilisateur commence par demander la résolution via `ResolveurCombine`, qui tente d'abord d'appliquer la méthode **de déduction** (`Deduction`). Cette dernière analyse les contraintes de la grille et applique des règles logiques pour restreindre les possibilités, retournant une version partiellement complétée. Si la grille n'est pas entièrement résolue, `ResolveurCombine` passe alors à la méthode **de backtracking** (`Backtracking`), qui teste systématiquement toutes les combinaisons possibles pour compléter la grille. Une fois la grille totalement remplie, `Backtracking` retourne la solution finale à `ResolveurCombine`, qui l'affiche ensuite à l'utilisateur.

Génération d'une grille avec `SudokuGenerator:`

Ce diagramme représente le processus de génération d'une grille de Sudoku jouable. L'utilisateur envoie une requête à SudokuGenerator pour obtenir une nouvelle grille. Ce dernier commence par créer une **grille complète** en remplissant toutes les cases selon les règles du Sudoku (genererGrilleComplete()). Ensuite, une version **incomplète** est générée en

retirant certaines valeurs en fonction du niveau de difficulté sélectionné (genererGrilleIncomplète(niveau)). Une fois la grille adaptée au jeu, elle est retournée à l'utilisateur, qui peut alors la résoudre. Ce processus garantit que la grille générée est **jouable et possède une solution unique,** assurant ainsi une expérience de jeu équilibrée.