DONZEL Quentin
AILLAUD Romain
Delaveaud Quentin

RAPPORT PROJET SUDOKU
2024-2025 3A FISE

Sommaire

1. Introduction

Notre projet est un jeu de Sudoku. L'architecture choisie vise à séparer clairement l'interface utilisateur du noyau du jeu afin de garantir modularité et maintenabilité. Ce rapport présente les choix de conception grâce auxquels vous pourrez voir notre réflexion adoptée.

Concernant la méthodologie, nous avons travaillé à l'aide d'un git.
Une personne s'est concentrée sur l'UI (Interface Utilisateur) pendant que les 2 autres ont développé le noyau du Sudoku. Nous avons convenu d'utiliser le même environnement de travail (même IDE, IntelliJ).

Nous avons également réalisé une javadoc pour mieux comprendre le fonctionnement de chaque méthode.

Lien du git : https://github.com/R0-M1/sudoku

2. Choix de conception

Nous avons structuré notre conception autour de plusieurs classes principales : `Sudoku`, `Grille`, `Bloc`, et `Case` pour la modélisation de la grille, ainsi que `Solveur` pour la résolution du Sudoku. (Voir les diagrammes en annexes ou dans le dossier /doc)

Case

Nous avons conçu la classe `Case` pour représenter chaque cellule de la grille de Sudoku.

- **Attributs principaux** :
  - `valeur` : représente la valeur actuelle de la case (0 pour une case vide).
  - `blocId` : identifie le bloc auquel appartient la case.
  - `valeursPossibles` : un `Set<Integer>` qui stocke les valeurs possibles pour cette case, ce qui aide à l'implémentation des règles de déduction.

Ce modèle permet une gestion efficace des contraintes du Sudoku et facilite les algorithmes de résolution.

Bloc

La classe `Bloc` représente un sous-ensemble de la grille contenant un groupe de cases.

- **Attributs principaux** :
  - `taille` : la taille du bloc.
  - `cases` : une liste de cases contenues dans ce bloc.
- **Méthode clé** : `validerBloc()`
  - Vérifie qu'aucune valeur ne se répète dans le bloc.

    - Utilise un `HashSet<Integer>` pour détecter les doublons, en ignorant les cases vides.

Grille

La classe `Grille` représente la structure globale du Sudoku.

- **Attributs principaux** :
    - `grille` : un tableau 2D de `Case`.
    - `blocs` : une liste de `Bloc` regroupant les cases en sous-unités logiques.
    - `taille` : définit la taille de la grille.
- **Fonctionnalités clés** :
    - `isValid()`: vérifie si la grille respecte les règles du Sudoku (unicité des valeurs par ligne, colonne et bloc).
    - `isComplete()`: vérifie si toutes les cases sont remplies et que la grille est valide.

Sudoku

La classe `Sudoku` sert d'interface principale entre la `Grille` et le `Solveur`.

- **Attributs principaux** :
    - `grille` : la grille de Grille.
    - `solveur` : un objet de type `Solveur`.
- **Fonctionnalités clés** :
    - `solve(List<MethodeResolution> methodes)`: tente de résoudre la grille en appliquant les méthodes spécifiées.
    - `generer(Difficulte difficulte)`: génère une grille de Sudoku valide en fonction du niveau de difficulté.

Résolution

Nous avons implémenté plusieurs techniques pour résoudre le Sudoku :

1. **Règles de déduction** :
    - **Élimination directe** : suppression des valeurs impossibles en fonction des contraintes de la grille.
    - **Placement forcé** : déduction d'une valeur unique lorsqu'elle est la seule option possible.
    - **Paires** : identification de couples de valeurs possibles pour réduire les options dans une ligne, colonne ou bloc.
2. **Backtracking** :
    - Implémenté pour compléter la résolution lorsque les règles de déduction ne suffisent pas.
    - Explore les solutions possibles récursivement en revenant en arrière si une impasse est détectée.

Cette approche hybride permet une meilleure efficacité.

3. Difficultés rencontrées

La réflexion sur les diagrammes a été compliquée , nous avons longtemps réfléchi à la manière dont nous allions faire ce sudoku et avons donc repris a plusieur fois le diagrammes de classe afin d'avoir une version plus optimale qui nous permettrait de facilement le coder en java.
Les méthodes de résolution ont été particulièrement dures , notamment les règles de déduction .
Cependant le backtracking s'est avéré plus simple qu'espérait.

4. Mode d'emploi

Version Console
Une fois le programme compilé et lancé, vous arriverez sur un menu :

1. Entrer une grille

Vous devrez rentrer une taille de grille (grille 3x3 => taille 9)
Et vous devrez entrer manuellement chaque valeur de la grille.

2. Résoudre la grille

Un second menu apparaît, vous aurez le choix entre un ensemble de règles de deduction et/ou le backtracking

3. Afficher la grille

Affiche la grille résolu ou non résolu.

4. Afficher les étapes de résolution

Afficher dans la console l'historique des opérations effectuées pour résoudre ou partiellement résoudre la grille.

5. Générer une grille

Permet de générer une grille en fonction d'une taille et d'un niveau de difficulté

6. Utiliser des grilles de test

Vous aurez le choix parmi un ensemble de grilles utilisé pour des tests.

7. Modifier les caractères affichés

Permet de modifier les caractères associés à chaque valeur de case pour l'affichage.

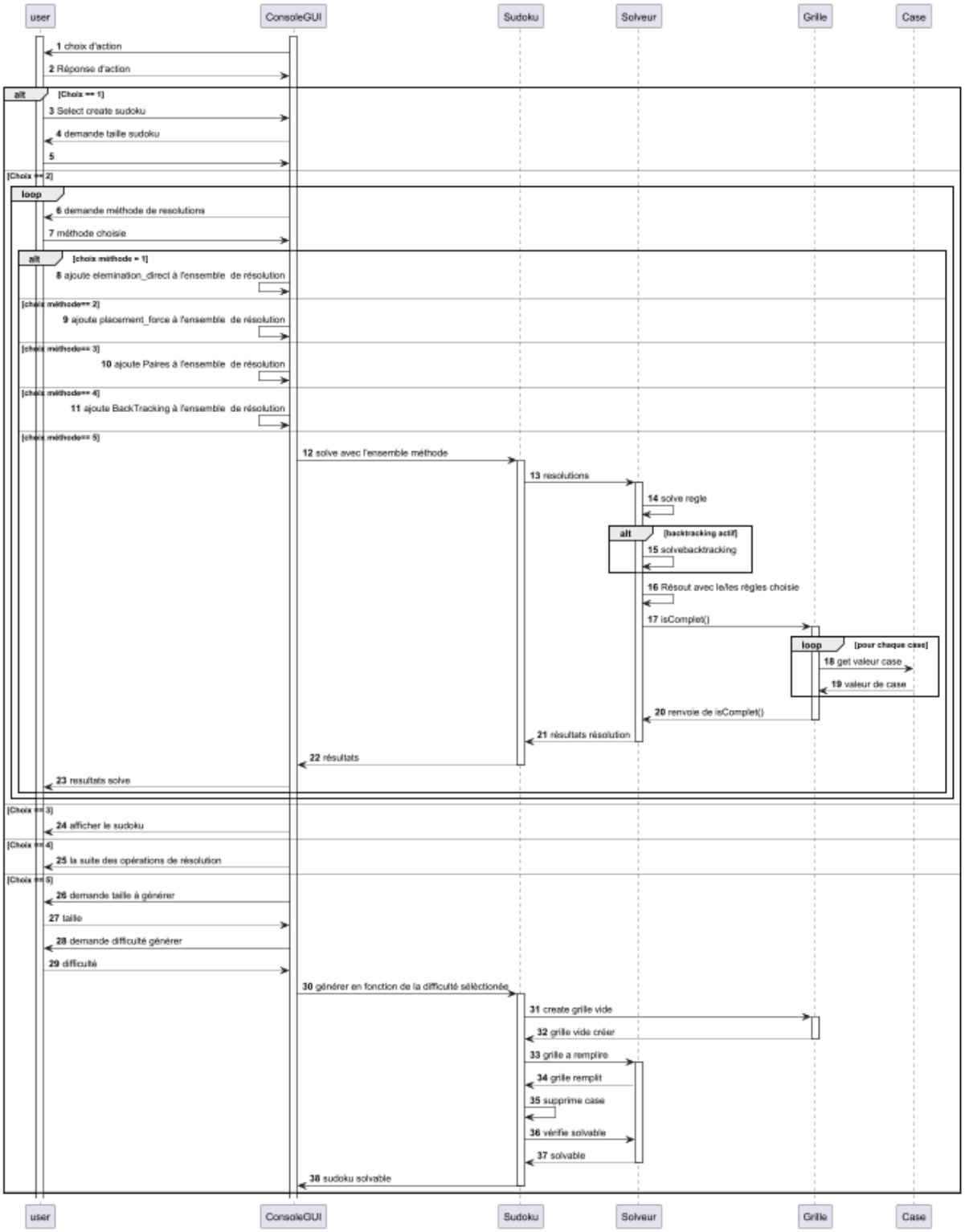
8. Quitter le programme

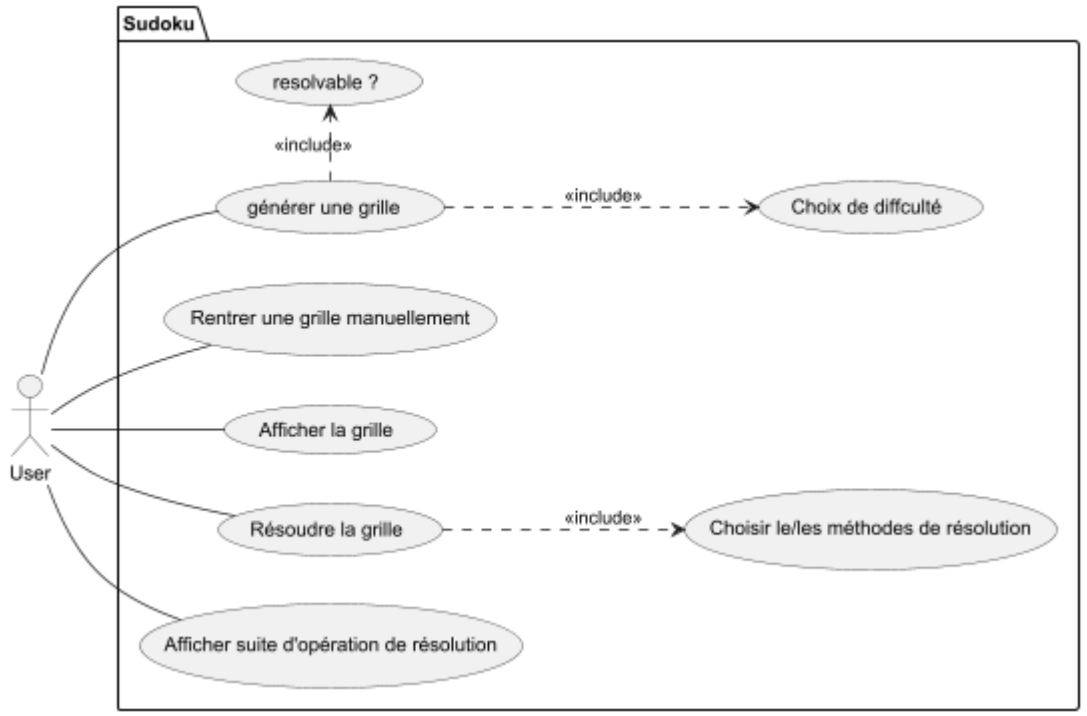
5. Conclusion

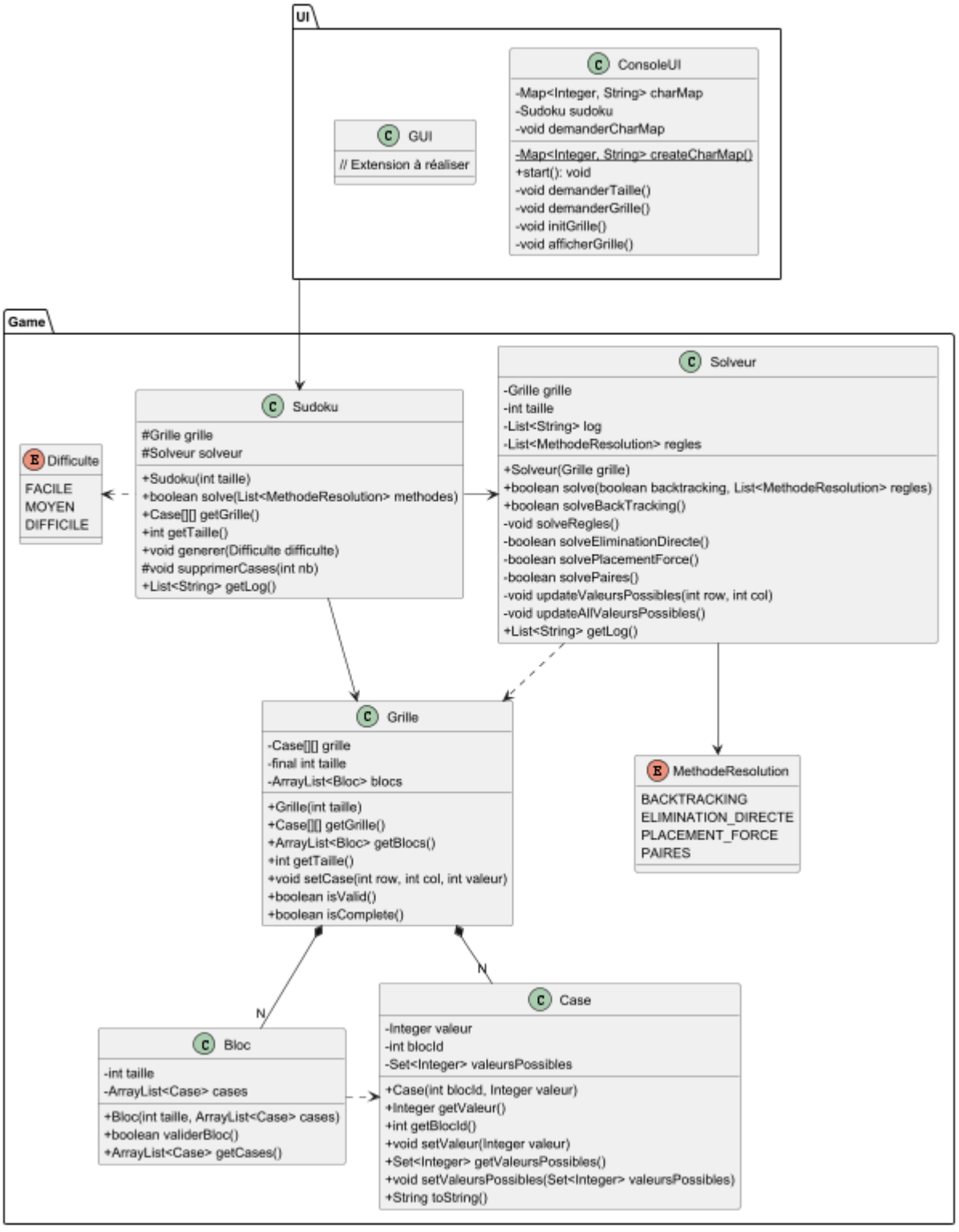
Le Projet à été très intéressant car ce dernier nous a permis d'appliquer des notions du cours tout en développants de nouvelle, ce qui fut enrichissant pour chacun

6. ANNEXE (DIAGRAMMES)

(voir image dans le dossier /doc pour une meilleure qualitée)


user
ConsoleGUI
Sudoku
Solveur
Grille
Case
1 choix d'action
2 Réponse d'action
alt [Choix == 1]
3 Select create sudoku
4 demande taille sudoku
5
[Choix == 2]
loop
6 demande méthode de resolutions
7 méthode choisie
alt [choix méthode = 1]
8 ajoute elemination_direct à l'ensemble de résolution
[choix méthode== 2]
9 ajoute placement_force à l'ensemble de résolution
[choix méthode== 3]
10 ajoute Paires à l'ensemble de résolution
[choix méthode== 4]
11 ajoute BackTracking à l'ensemble de résolution
[choix méthode== 5]
12 solve avec l'ensemble méthode
13 resolutions
14 solve regle
alt [backtracking actif]
15 solvebacktracking
16 Résout avec le/les règles choisie
17 isComplet()
loop [pour chaque case]
18 get valeur case
19 valeur de case
20 renvoie de isComplet()
21 résultats résolution
22 résultats
23 resultats solve
[Choix == 3]
24 afficher le sudoku
[Choix == 4]
25 la suite des opérations de résolution
[Choix == 5]
26 demande taille à générer
27 taille
28 demande difficulté générer
29 difficulté
30 générer en fonction de la difficulté sélectionée
31 create grille vide
32 grille vide créer
33 grille a remplire
34 grille remplit
35 supprime case
36 vérifie solvable
37 solvable
38 sudoku solvable


Sudoku
resolvable ?
«include»
générer une grille
«include»
Choix de diffculté
Rentrer une grille manuellement
Afficher la grille
User
Résoudre la grille
«include»
Choisir le/les méthodes de résolution
Afficher suite d'opération de résolution


UI
ConsoleUI
-Map<Integer, String> charMap
-Sudoku sudoku
-void demanderCharMap
-Map<Integer, String> createCharMap()
+start(): void
-void demanderTaille()
-void demanderGrille()
-void initGrille()
-void afficherGrille()
GUI
// Extension à réaliser
Game
Sudoku
#Grille grille
#Solveur solveur
+Sudoku(int taille)
+boolean solve(List<MethodeResolution> methodes)
+Case[][] getGrille()
+int getTaille()
+void generer(Difficulte difficulte)
#void supprimerCases(int nb)
+List<String> getLog()
Difficulte
FACILE
MOYEN
DIFFICILE
Solveur
-Grille grille
-int taille
-List<String> log
-List<MethodeResolution> regles
+Solveur(Grille grille)
+boolean solve(boolean backtracking, List<MethodeResolution> regles)
+boolean solveBackTracking()
-void solveRegles()
-boolean solveEliminationDirecte()
-boolean solvePlacementForce()
-boolean solvePaires()
-void updateValeursPossibles(int row, int col)
-void updateAllValeursPossibles()
+List<String> getLog()
Grille
-Case[][] grille
-final int taille
-ArrayList<Bloc> blocs
+Grille(int taille)
+Case[][] getGrille()
+ArrayList<Bloc> getBlocs()
+int getTaille()
+void setCase(int row, int col, int valeur)
+boolean isValid()
+boolean isComplete()
MethodeResolution
BACKTRACKING
ELIMINATION_DIRECTE
PLACEMENT_FORCE
PAIRES
N
N
Bloc
-int taille
-ArrayList<Case> cases
+Bloc(int taille, ArrayList<Case> cases)
+boolean validerBloc()
+ArrayList<Case> getCases()
Case
-Integer valeur
-int blocId
-Set<Integer> valeursPossibles
+Case(int blocId, Integer valeur)
+Integer getValeur()
+int getBlocId()
+void setValeur(Integer valeur)
+Set<Integer> getValeursPossibles()
+void setValeursPossibles(Set<Integer> valeursPossibles)
+String toString()