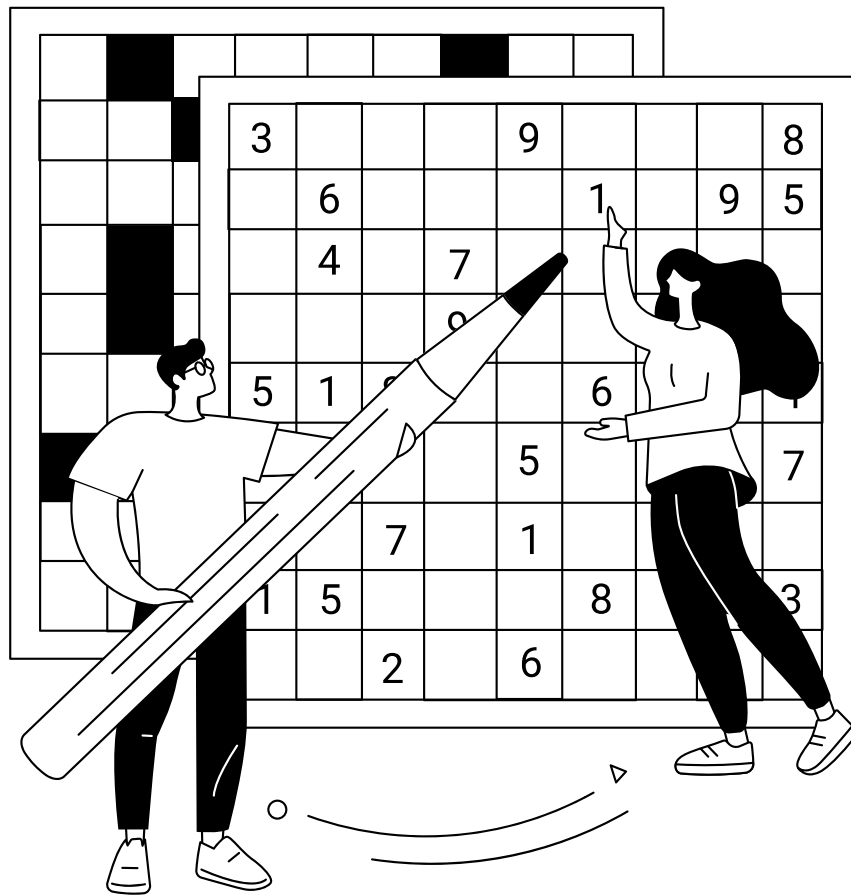


2023-2024

# Rapport de projet

## SAÉ 2.01 - Réalisation d'un Sudoku



**MONCEF STITI**  
**MARCO ORFAO**

# Sommaire

---

## **I. Introduction**

1) Principe du Sudoku.....	3
2) Introduction au sujet.....	4
3) Fonctionnalités du créateur de grille.....	5/6/7/8
4) Fonctionnalités du solveur de grille.....	9/10/11/12/13/14

## **II. Structure**

1) Programme GridSolver.....	15/16/17/18
2) Programme GridMaker.....	19/20/21/22/23
3) Classes communes.....	24/25/26
4) Makefile	
a. Commandes utiles.....	27
b. Structure des programmes.....	28

## **III. Explication de l'algorithme de résolution**

1) La méthode solve().....	29
----------------------------	----

## **IV. Conclusions personnelles**

1) Moncef STITI.....	30
2) Marco ORFAO.....	30

## 1) Principe du Sudoku

Le Sudoku est un jeu de logique et de résolution de problèmes qui repose sur des principes simples mais stimulants. Dans un Sudoku, une grille de 9x9 est divisée en neuf sous-grilles de 3x3.

Le but est de remplir chaque case de la grille avec un chiffre de 1 à 9, en veillant à ce que chaque ligne, chaque colonne et chaque sous-grille contienne tous les chiffres de 1 à 9, sans répétition.



Les principes clés du Sudoku sont la logique et l'élimination progressive des possibilités.

Pour résoudre une grille, les joueurs utilisent des techniques telles que la recherche de chiffres uniques, l'élimination des possibilités dans les cases en fonction des chiffres déjà placés, et la déduction pour identifier les chiffres manquants.

Le Sudoku ne nécessite aucune compétence mathématique avancée, mais plutôt une approche méthodique et logique. Il offre un défi intellectuel captivant, avec des grilles de difficultés variées pour satisfaire les novices comme les experts.

## 2) Introduction au sujet

L'objectif de ce projet est de créer deux programmes en Java : Un programme pour concevoir des grilles de Sudoku (qui respectent les règles) et un programme pour résoudre des grilles de Sudoku automatiquement ou manuellement tout en respectant les règles du jeu.

### 1. Programme de Conception de Grilles de Sudoku :

- Ce programme permet à l'utilisateur de créer des grilles de Sudoku conformes aux règles du jeu.
- Il offre la possibilité de démarrer avec une grille vide ou de charger une grille déjà existante à partir d'un fichier.
- L'utilisateur peut ajouter ou supprimer des chiffres dans la grille en respectant les contraintes d'unicité du Sudoku.
- Une fois la grille achevée, elle peut être sauvegardée dans un fichier .gri pour une utilisation ultérieure.

### 2. Programme de Résolution de Grilles de Sudoku:

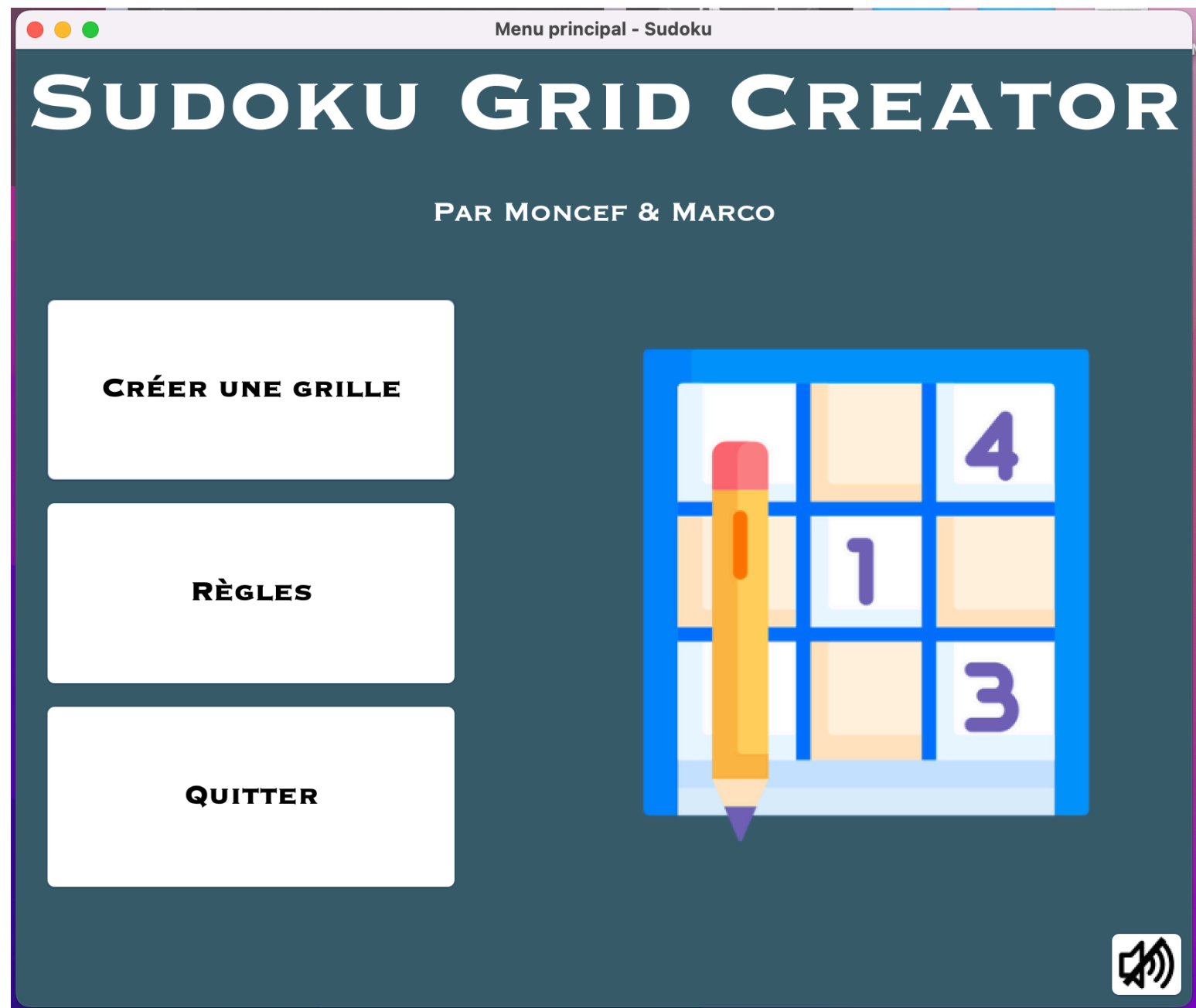
- Ce programme permet à l'utilisateur de résoudre des grilles de Sudoku, soit automatiquement, soit manuellement.
- L'utilisateur peut charger une grille à résoudre à partir d'un fichier .gri.
- En mode automatique, le programme résout la grille et affiche la solution ainsi que le temps de résolution.
- En mode manuel, l'utilisateur peut ajouter ou supprimer des chiffres dans la grille tout en respectant les règles du jeu.
- Le programme vérifie en permanence que les règles du Sudoku sont respectées et félicite l'utilisateur lorsque la grille est entièrement remplie correctement.

# I. Introduction

## 3) Fonctionnalités du créateur de grille

### 1. Menu principal :

Au démarrage du programme, l'écran d'accueil suivant s'affiche. Il permet à l'utilisateur de quitter le programme, d'afficher les règles du Sudoku ou de créer une grille.

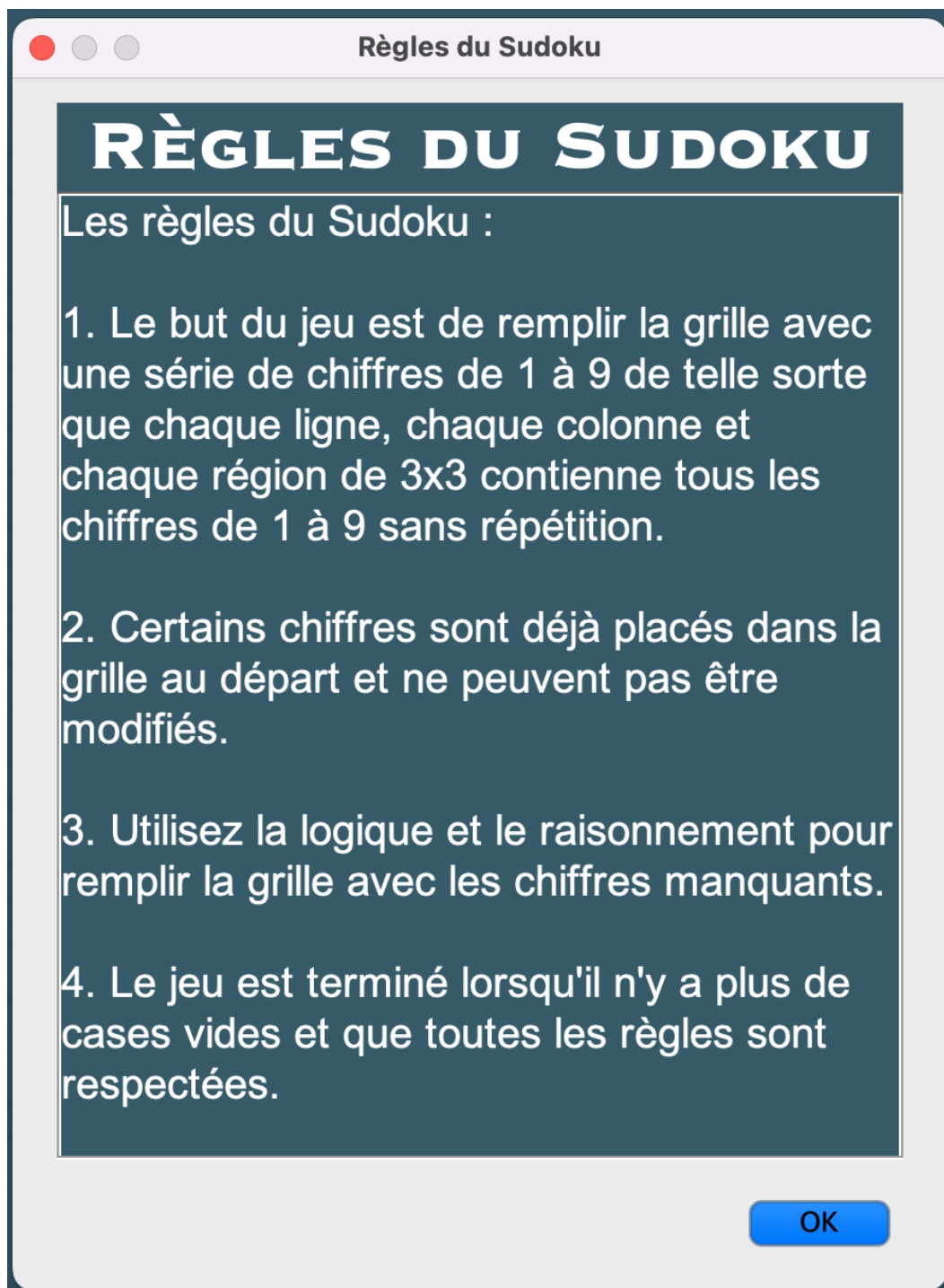


Mais également d'activer ou de désactiver la musique d'ambiance du menu à l'aide du petit bouton de son en bas à droite.

## 3) Fonctionnalités du créateur de grille

### 2. Règles du jeu :

Dans le menu principal, lorsque l'utilisateur clique sur le bouton "**Règles**", la fenêtre de dialogue suivante s'ouvre pour lui expliquer les règles du Sudoku.

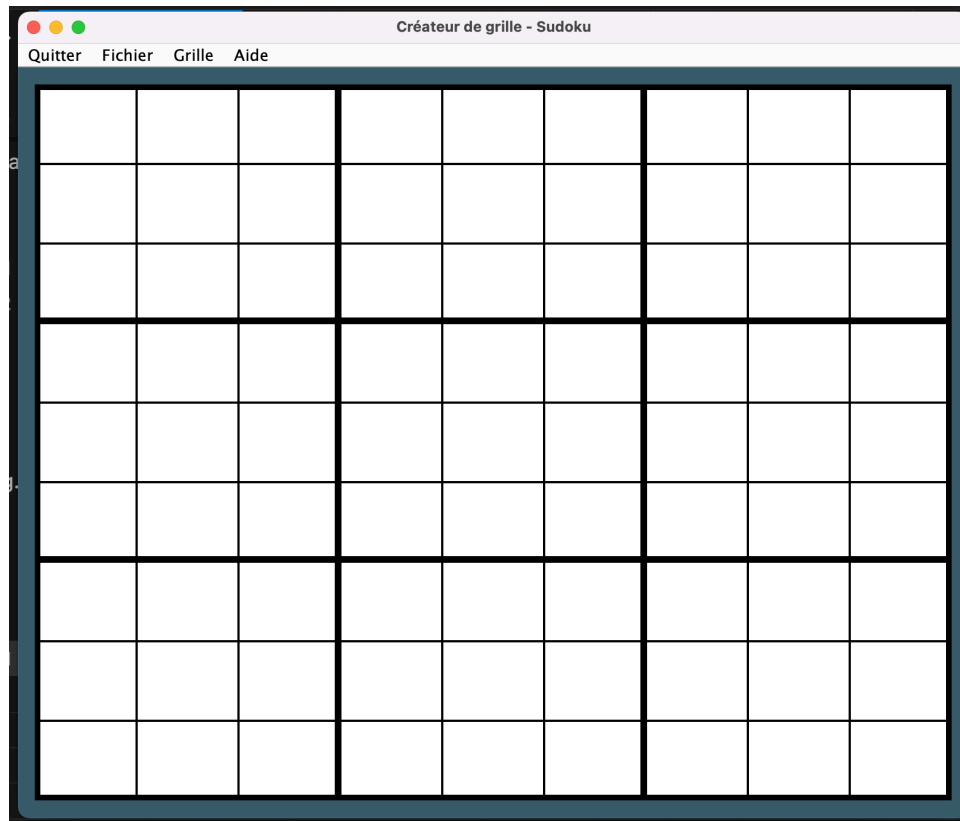


# I. Introduction

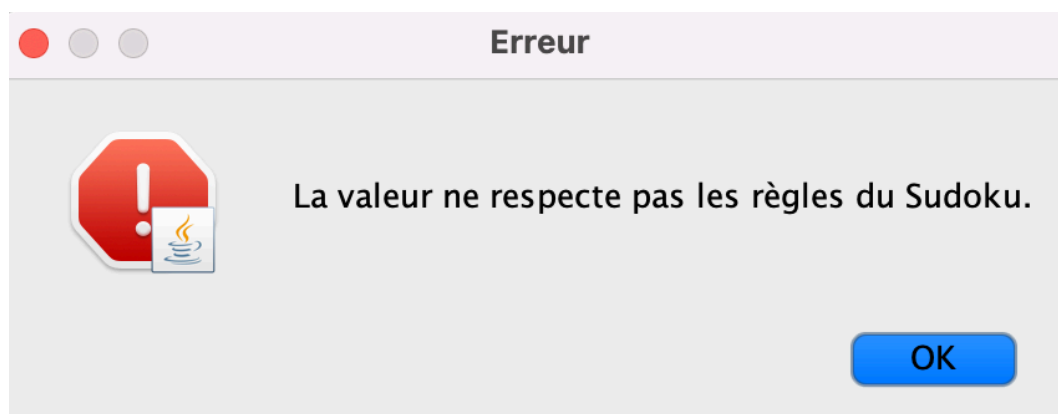
## 3) Fonctionnalités du créateur de grille

### 3. Créateur de grille :

Dans le menu principal, lorsque l'utilisateur clique sur le bouton "Créer une grille", cette interface s'ouvre :



Elle permet à l'utilisateur d'écrire des chiffres dans les cases afin de créer sa grille. Cependant, l'interface bloque à l'utilisateur la possibilité d'entrer des chiffres qui ne respectent pas les règles du Sudoku sous peine de voir ce message d'erreur :



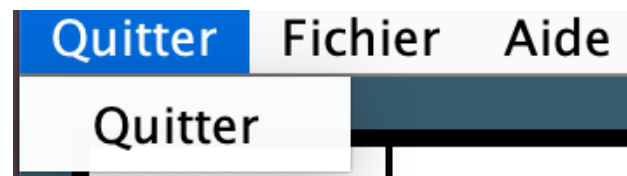
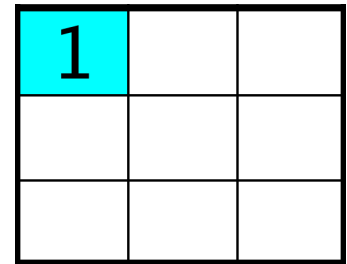
# I. Introduction

## 3) Fonctionnalités du créateur de grille

L'interface bloque également à l'utilisateur la possibilité d'entrer des valeurs autres que des chiffres :

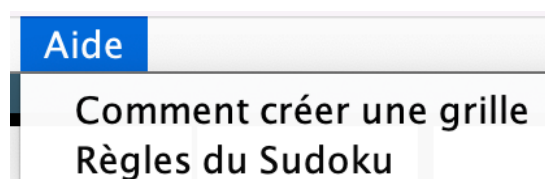
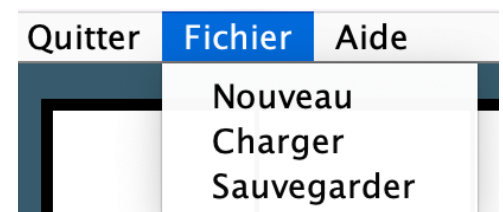


Lorsque que l'utilisateur clique sur une case, cette dernière devient turquoise afin de faciliter l'expérience utilisateur.



L'utilisateur peut également cliquer sur le bouton "**Quitter**" dans le menu déroulant "**Quitter**" afin de fermer le programme.

Dans le menu déroulant "**Fichier**", l'utilisateur peut cliquer sur "**Nouveau**" pour créer une nouvelle grille, "**Charger**" pour charger un fichier .gri d'une grille déjà existante ou alors "**Sauvegarder**" afin d'enregistrer la grille au format .gri.



Dans le menu déroulant "**Aide**", l'utilisateur peut cliquer sur "**Comment crée une grille**" afin d'afficher un tutoriel ou sur "**Règles du Sudoku**" afin d'afficher les règles.



# I. Introduction

## 4) Fonctionnalités du solveur de grille

### 1. Menu principal :

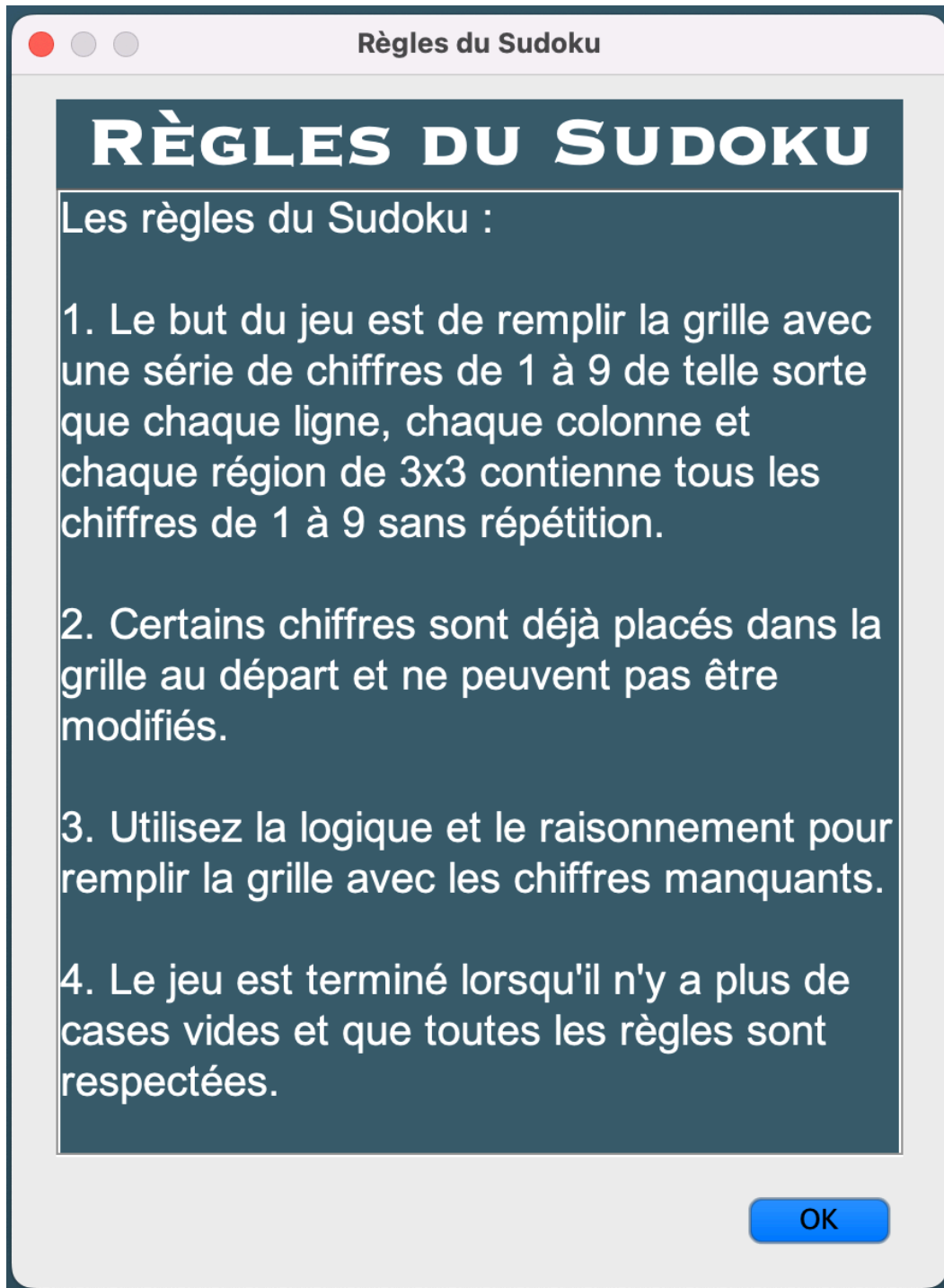
Tout comme le créateur de grille, au démarrage du solveur de grille, l'écran d'accueil suivant s'affiche, il permet à l'utilisateur de quitter le programme, d'afficher les règles du Sudoku, ou de "**Jouer**".



Mais également d'activer ou de désactiver la musique d'ambiance du menu à l'aide du petit bouton de son en bas à droite.

### 2. Règles du jeu :

Dans le menu principal, lorsque l'utilisateur clique sur le bouton "**Règles**", la fenêtre de dialogue suivante s'ouvre pour lui expliquer les règles du Sudoku.



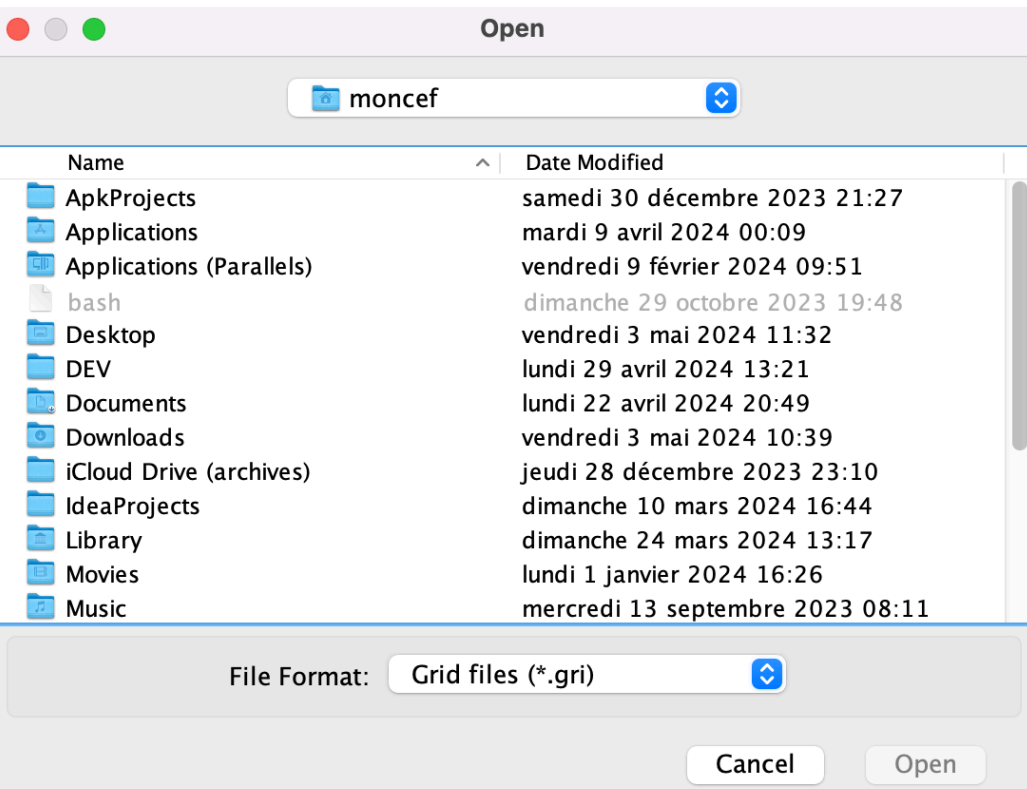
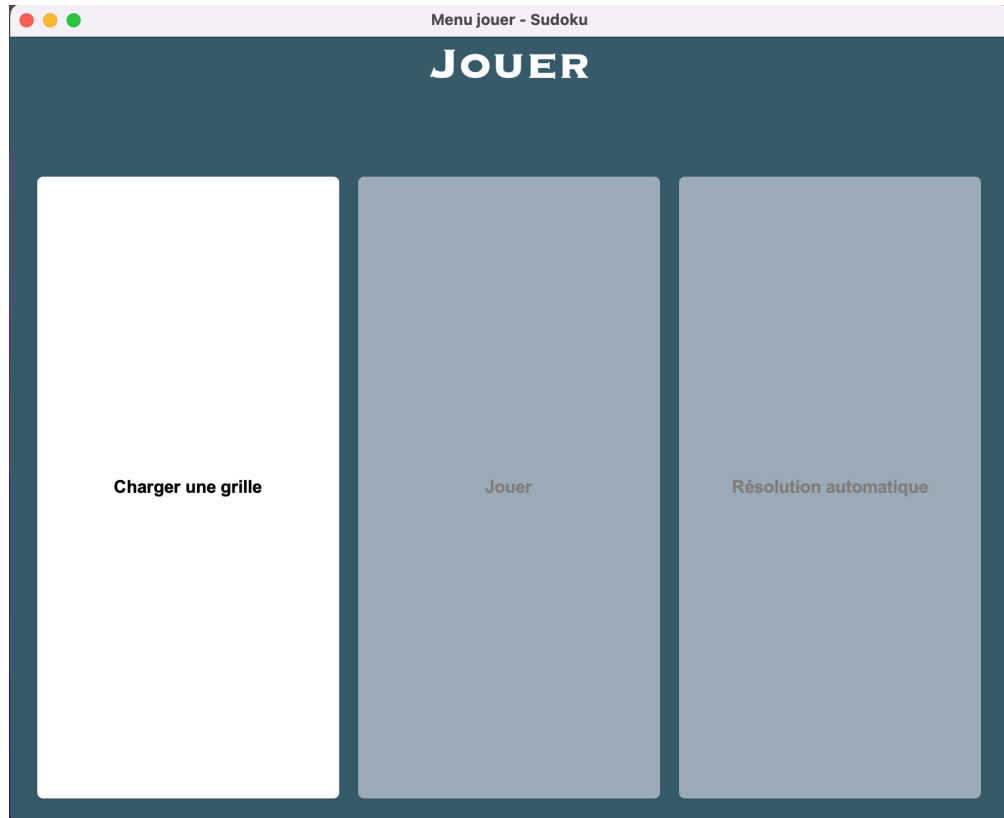
# I. Introduction

## 4) Fonctionnalités du solveur de grille

### 3. Menu jouer :

Dans le menu principal, lorsque l'utilisateur clique sur le bouton "**Jouer**", cette interface s'ouvre :

Elle permet à l'utilisateur de charger une grille afin de jouer. Si ce dernier ne charge pas de grille, il n'a pas la possibilité de cliquer sur les autres boutons.

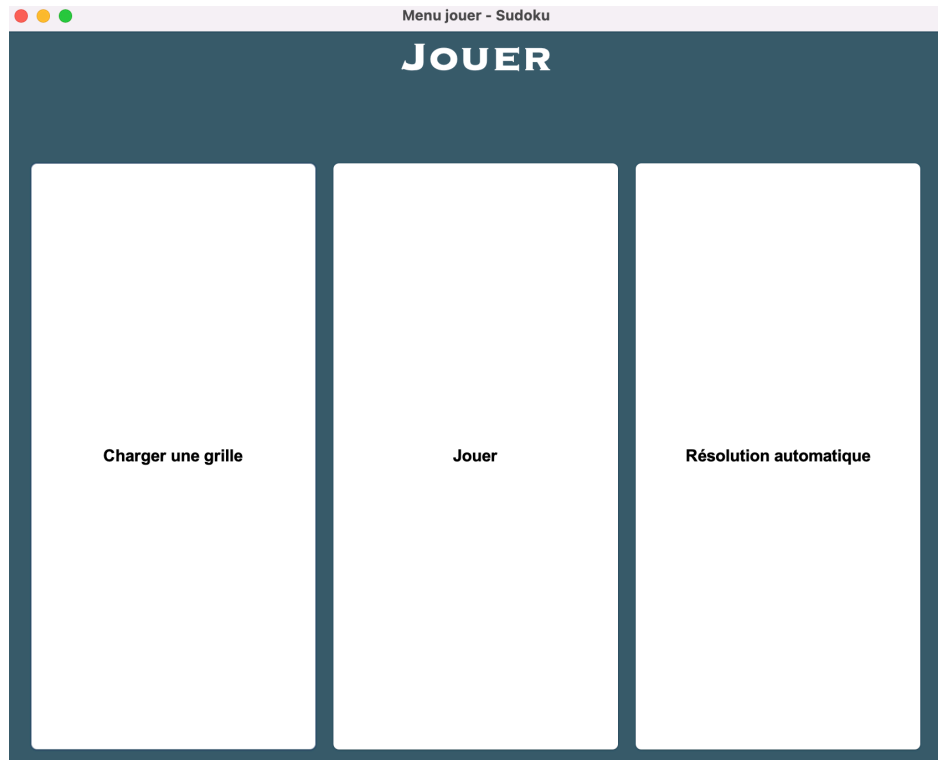


Lorsque l'utilisateur clique sur "**Charger une grille**", il a la possibilité de choisir le fichier de sa grille au format **.gri** dans ses fichiers.

# I. Introduction

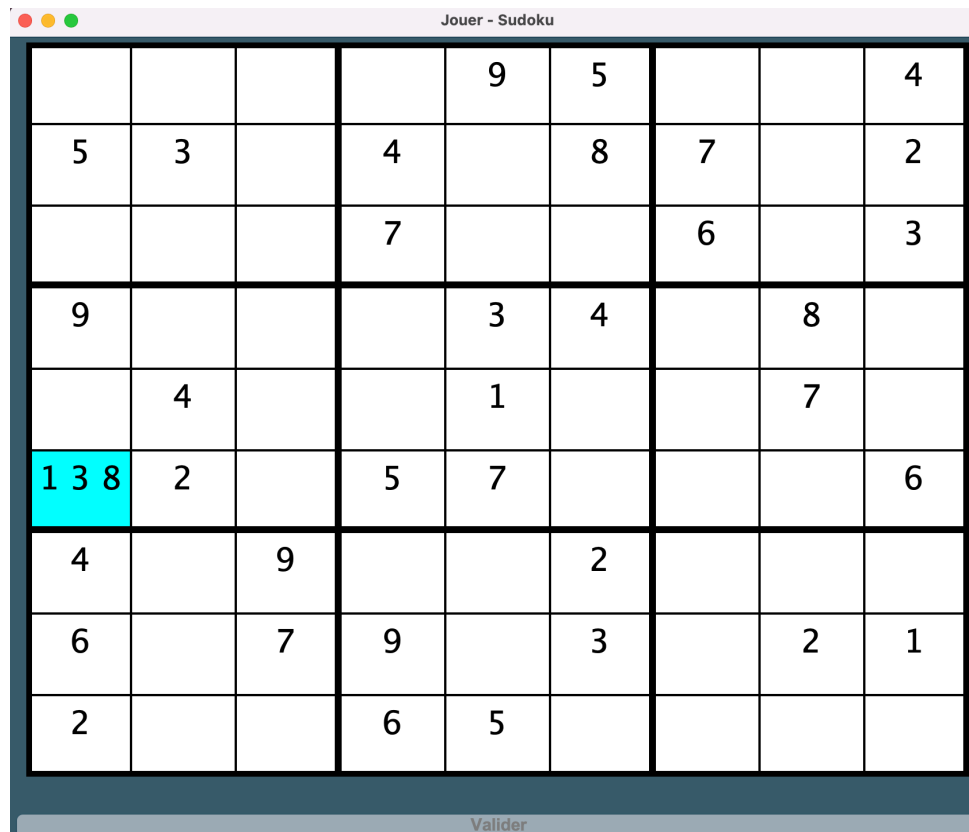
## 4) Fonctionnalités du solveur de grille

Après avoir chargé une grille, l'utilisateur débloque la possibilité de cliquer sur les boutons "**Jouer**" et "**Résolution automatique**".



### 4. Jeu :

Après avoir cliqué sur le bouton "**Jouer**", cette interface s'ouvre et permet à l'utilisateur de remplir la grille :



Lorsqu'une case est cliquée, elle devient **verte**.

Lorsqu'une réponse correcte est entrée dans une case, elle devient **turquoise**. En revanche, si une valeur incorrecte est entrée dans une case, celle-ci devient **rouge**.

Le programme permet également à l'utilisateur d'entrer jusqu'à 4 chiffres temporaires dans une case, agissant comme un bloc-notes.

# I. Introduction

## 4) Fonctionnalités du solveur de grille

Lorsque l'utilisateur a résolu la grille, le bouton "**Valider**" s'active. Lorsqu'il est pressé, un message de félicitations s'affiche, accompagné du temps nécessaire à la résolution de la grille.

The screenshot shows a window titled "Jouer - Sudoku" containing a 9x9 grid. The grid is filled with numbers from 1 to 9, representing a solved Sudoku puzzle. A modal dialog box is displayed in the center of the grid, with the title "Félicitations !" and the message "Félicitations ! Vous avez résolu le Sudoku en 0 minute et 5 secondes." Below the message is a blue "OK" button. At the bottom of the window, a button labeled "Valider" is visible.

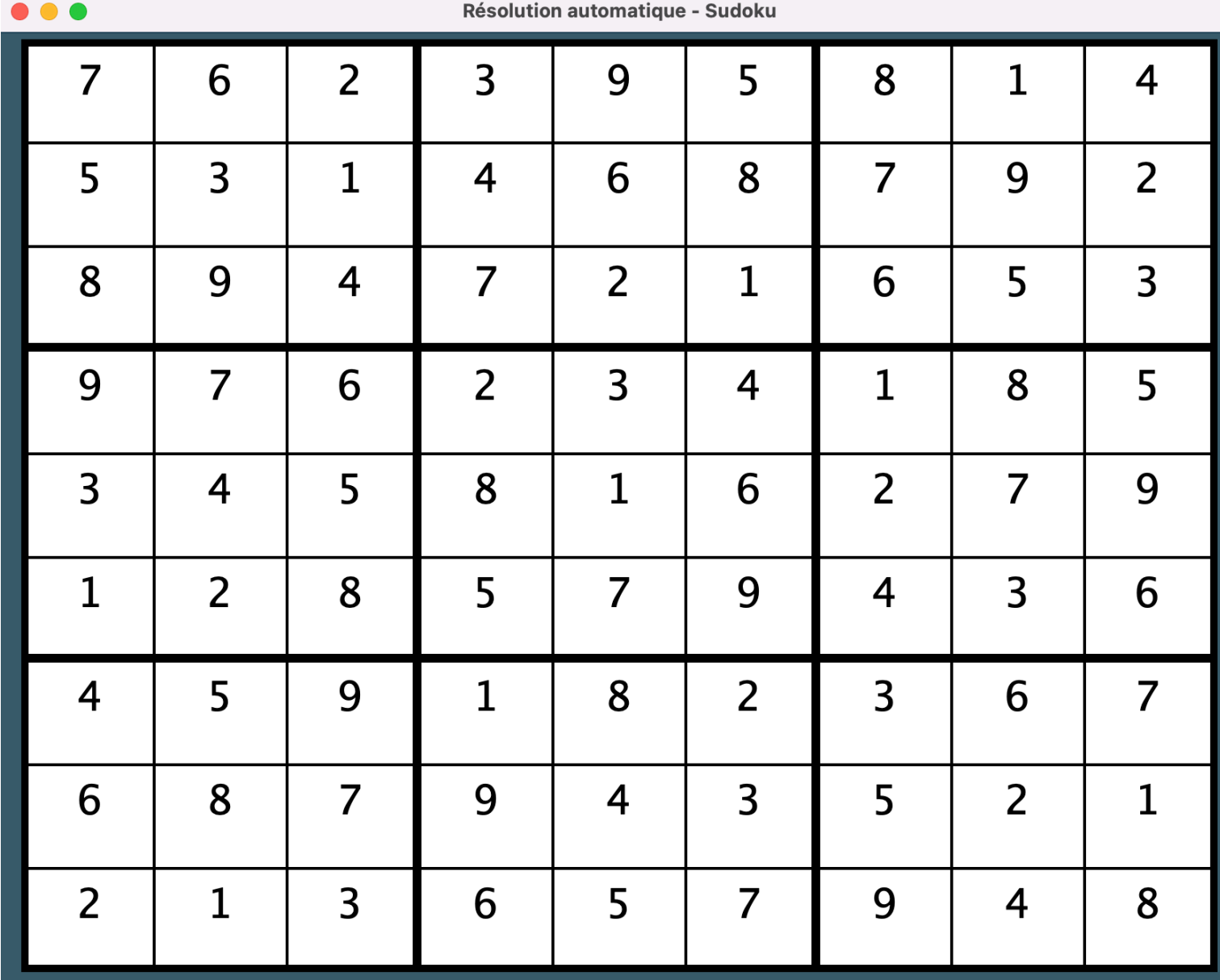
1	2	3	7	8	9	6	5	4
4	5	6	1	2	3	9	8	7
7	8	9	4	5	6	3	2	1
2	3	4	5	1	7	8	6	9
5	6	1	3	6	4	5	4	3
8	9	1	3	6	4	5	7	2
3	4	5	6	7	1	2	9	8
6	7	8	9	3	2	4	1	5
9	1	2	8	4	5	7	3	6

# I. Introduction

## 4) Fonctionnalités du solveur de grille

### 5. Résolution automatique :

Dans le menu jouer, si l'utilisateur clique sur le bouton "**Résolution automatique**", cette interface s'affiche :



The screenshot shows a window titled "Résolution automatique - Sudoku". It displays a 9x9 grid with the following numbers:

7	6	2	3	9	5	8	1	4
5	3	1	4	6	8	7	9	2
8	9	4	7	2	1	6	5	3
9	7	6	2	3	4	1	8	5
3	4	5	8	1	6	2	7	9
1	2	8	5	7	9	4	3	6
4	5	9	1	8	2	3	6	7
6	8	7	9	4	3	5	2	1
2	1	3	6	5	7	9	4	8

Résolu en 0.006543709 secondes.

Elle résout automatiquement la grille chargée précédemment et affiche le temps nécessaire à sa résolution.

## II. Structure

### 1) Programme GridSolver

#### -> **GSWin.java**

La classe **GSWin** est responsable de l'affichage d'une fenêtre de félicitations une fois que l'utilisateur a résolu le Sudoku. Cette fenêtre apparaît pour célébrer la réussite de l'utilisateur. En recevant en entrée le temps nécessaire à la résolution du Sudoku, elle utilise cette information pour afficher un message approprié.

#### -> **GSTest.java**

La classe **GSTest** fournit les fonctionnalités pour tester la validité d'une grille (GSGrid) de Sudoku. Elle vérifie s'il y a des doublons dans les colonnes, les lignes et les régions de la grille. Si des doublons sont trouvés, elle indique que la grille n'est pas valide. Sinon, elle confirme la validité de la grille.

#### -> **GSSolver.java**

La classe **GSSolver** résout une grille de Sudoku et affiche le résultat dans une fenêtre. Elle initialise les composants graphiques de la fenêtre pour afficher la résolution en cours. Ensuite, elle démarre le processus de résolution en ajoutant la grille de Sudoku à la fenêtre et en mettant à jour l'état de résolution. Une fois la résolution terminée, elle affiche le temps écoulé pour résoudre le Sudoku dans la fenêtre. Enfin, elle vérifie si le jeu est terminé en vérifiant si la grille est complète.

#### -> **GSPlayController.java**

La classe **GSPlayController** agit comme un contrôleur pour le jeu. Il réagit aux événements déclenchés par l'utilisateur lors de l'interaction avec la grille de Sudoku. Lorsque l'utilisateur appuie sur une touche du clavier, il met à jour la valeur de la case active de la grille avec la valeur correspondante à la touche appuyée. Il vérifie également si le jeu est terminé et active le bouton de validation si nécessaire. En outre, il gère les actions déclenchées par l'utilisateur, telles que la validation du jeu en cliquant sur le bouton "Valider", en affichant une fenêtre de dialogue pour montrer le temps écoulé depuis le début du jeu.

## II. Structure

### -> GSPlay.java

La classe '**GSPlay**' permet de jouer au Sudoku. Elle gère l'affichage de la grille de jeu dans une fenêtre dédiée. Elle initialise les composants nécessaires tels que la grille, le bouton "Valider" et le contrôleur pour gérer les événements de jeu. Lorsque la fenêtre de jeu est affichée, elle prend en charge les entrées du clavier pour permettre à l'utilisateur de remplir la grille avec les chiffres de 1 à 9 et de supprimer les valeurs. Elle vérifie également si le jeu est terminé en s'assurant que la grille est complète. Enfin, elle permet d'obtenir des informations sur l'état du jeu, comme le bouton "Valider", la grille en cours et le temps écoulé depuis le début du jeu.

### -> GSMenuController.java

La classe '**GSMenuController**' gère les actions déclenchées par les boutons du menu dans une application Sudoku. Elle est chargée de réagir aux événements de clic sur les différents boutons du menu. Lorsqu'un bouton est cliqué, elle effectue une action correspondante :

1. Si le bouton "Importer" est cliqué, elle crée un gestionnaire d'importation de grille, importe une grille, met à jour la grille Sudoku avec les valeurs importées et réactive les options de jeu dans le menu.
2. Si le bouton "Jouer" est cliqué, elle crée un jeu Sudoku, un contrôleur pour le jeu, ajoute un écouteur de touches pour le jeu et affiche le jeu.
3. Si le bouton "Résoudre automatiquement" est cliqué, elle crée un résolveur de grille.

### -> GSMenu.java

La classe '**GSMenu**' représente le menu de jeu du Sudoku. Elle crée et gère les éléments visuels du menu, tels que le titre, les boutons pour importer une grille, jouer et résoudre automatiquement la grille. Lors de sa construction, elle initialise les composants nécessaires, tels que les panneaux pour le titre et les boutons, ainsi que les boutons eux-mêmes avec leur état initial. Elle expose des méthodes pour activer les options de jeu dans le menu et pour obtenir les boutons spécifiques du menu. En résumé, cette classe facilite la gestion du menu de jeu du Sudoku dans une interface utilisateur graphique.

### -> GSImport.java

La classe '**GSImport**' permet à l'utilisateur d'importer une grille de Sudoku à partir d'un fichier externe. Elle offre une interface conviviale en affichant une boîte de dialogue de sélection de fichier où l'utilisateur peut choisir le fichier contenant la grille à importer. Une fois le fichier sélectionné, la classe lit les données à partir de celui-ci et les stocke dans un tableau interne. Elle vérifie également si le fichier est accessible et si la lecture a été effectuée avec succès. Cette fonctionnalité permet à l'utilisateur d'importer facilement des grilles de Sudoku depuis des sources externes, facilitant ainsi la personnalisation et l'extension du programme de Sudoku.



## II. Structure

### -> GSCase.java

La classe '**GSCase**' représente une case dans une grille de jeu de Sudoku. Elle offre plusieurs fonctionnalités pour gérer l'affichage et la manipulation des valeurs dans une case. Voici ce que fait cette classe :

1. **Affichage de la valeur** : La classe permet d'afficher une valeur dans la case. Cette valeur peut être la valeur principale de la case ou une combinaison de plusieurs valeurs optionnelles.
2. **Initialisation de la case** : Elle permet d'initialiser une case avec une valeur par défaut ou une valeur initiale. Cette fonction est utilisée pour définir la valeur de départ d'une case, qui est fournie dans les grilles importés.
3. **Modification de la valeur** : La classe permet de modifier la valeur d'une case en fonction des entrées de l'utilisateur. Elle gère également la validation des valeurs insérées pour s'assurer qu'elles respectent les règles du Sudoku.
4. **Activation et désactivation** : Elle permet d'activer ou de désactiver une case. Une case désactivée est utilisée pour indiquer une case non modifiable dans une grille importée.

### -> GSCaseMouseListener.java

La classe '**GSCaseMouseListener**' implémente l'interface '**MouseListener**' pour gérer les événements de la souris sur une instance de la classe '**GSCase**', qui représente une case dans une grille de jeu de Sudoku. Voici ce que fait cette classe :

1. **Activation de la case** : Lorsque l'utilisateur clique sur une case avec la souris, si la case n'est pas initiale (c'est-à-dire qu'elle n'est pas pré-remplie dans la grille), la case devient active et est mise en surbrillance avec une couleur verte. De plus, toutes les autres cases de la grille deviennent désactivées, empêchant ainsi leur modification.
2. **Surbrillance lors du survol** : Lorsque l'utilisateur survole une case avec la souris, si la case n'est pas initiale et n'est pas déjà active, elle est mise en surbrillance avec une couleur jaune pour indiquer qu'elle peut être activée en cliquant dessus.
3. **Rétablissement de la couleur d'arrière-plan** : Lorsque la souris quitte une case après le survol, la couleur d'arrière-plan de la case est rétablie à sa couleur par défaut (blanche) si elle n'est pas initiale et n'est pas active.

### -> GSGrid.java

La classe '**GSGrid**' gère la grille de jeu pour le Sudoku. Elle initialise la grille, permet l'importation des données, contrôle l'interaction avec les cases, vérifie si la grille est complète, gère le mode de jeu et propose une fonction de résolution automatique.

# II. Structure

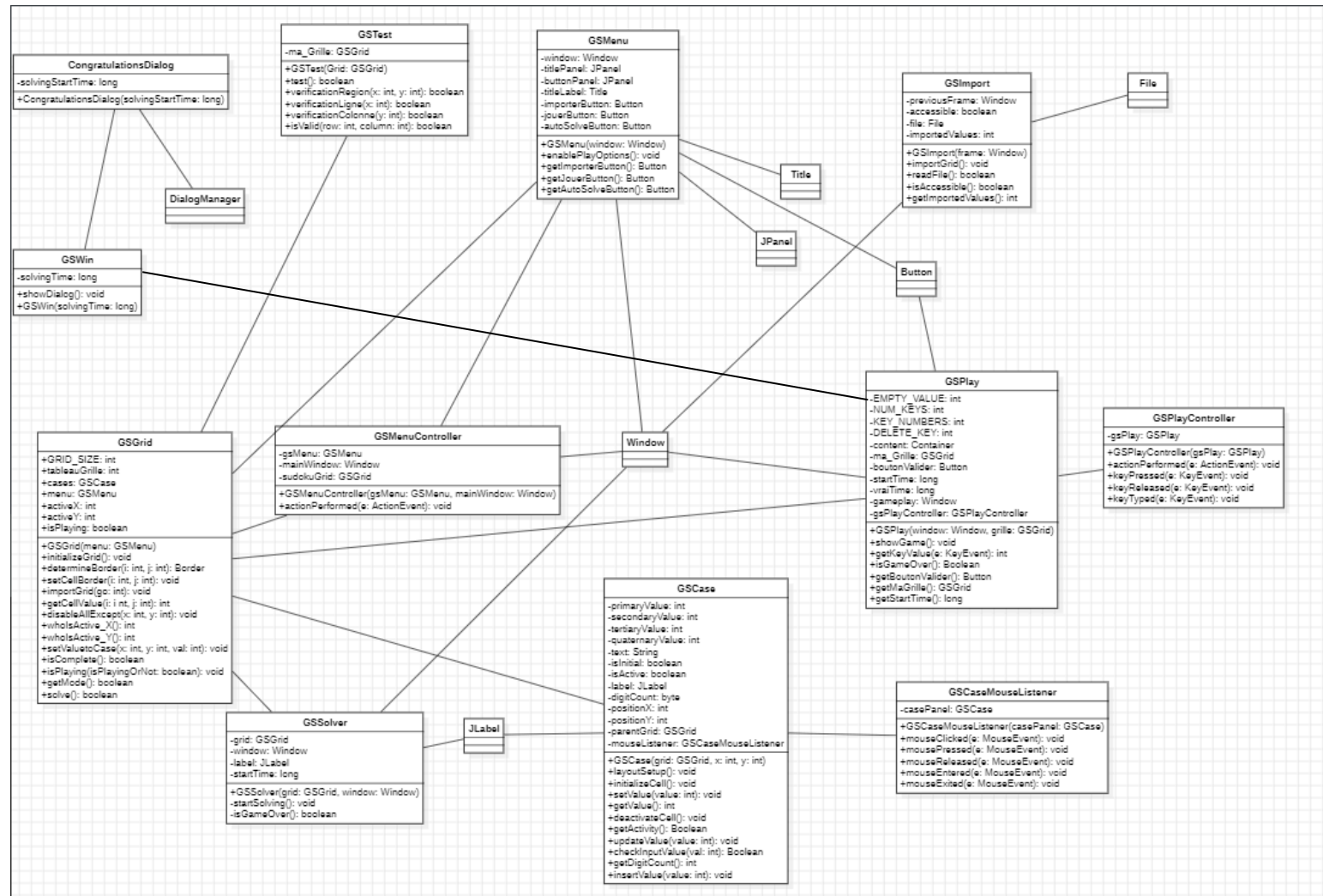
## -> CongratulationsDialog.java

La classe 'CongratulationsDialog' crée une boîte de dialogue de félicitations pour afficher le temps de résolution d'un Sudoku. Elle prend le temps de démarrage de la résolution du Sudoku en nanosecondes comme paramètre dans son constructeur.

En utilisant le temps actuel en nanosecondes, elle calcule le temps de résolution en secondes. Ensuite, elle formate un message de félicitations en fonction du temps de résolution, en prenant en compte les minutes et les secondes. Le message affiché dans la boîte de dialogue informe l'utilisateur du temps de résolution de la grille Sudoku.

Une fois le message de félicitations créé, la boîte de dialogue est affichée avec le message approprié et le titre "Félicitations !".

## -> Diagramme de classe de GridSolver



# II. Structure

## 2) Programme GridMaker

### -> GMCaseKeyListener.java

La classe '**GMCaseKeyListener**' agit en tant que '**KeyListener**' pour écouter les événements de touche associés à une case dans une grille de Sudoku. Elle est utilisée pour vérifier et mettre à jour les valeurs des cases en fonction des entrées de l'utilisateur. Lorsque l'utilisateur appuie sur une touche, cette classe vérifie si la touche correspond à un chiffre entre 1 et 9. Si c'est le cas, elle met à jour la valeur de la case avec ce chiffre. Ensuite, elle vérifie si la grille respecte toujours les règles du Sudoku après la mise à jour. Si oui, elle laisse simplement la valeur mise à jour. Sinon, elle restaure la valeur précédente de la case et affiche un message d'erreur indiquant que la valeur ne respecte pas les règles du Sudoku.

Dans le cas où la touche enfoncée n'est pas un chiffre entre 1 et 9 la classe affiche un message d'erreur pour indiquer que l'entrée de l'utilisateur est invalide. Cette classe permet donc de garantir que les valeurs entrées par l'utilisateur respectent les règles du Sudoku, assurant ainsi l'intégrité de la grille de jeu.

### -> GMCaseMouseListener.java

La classe '**GMCaseMouseListener**' agit en tant qu'écouteur de souris pour les cases de la grille de Sudoku. Son objectif est de gérer l'activation des cases lorsqu'elles sont cliquées par l'utilisateur. Lorsque l'utilisateur clique sur une case, cette classe vérifie d'abord si la source de l'événement est un bouton. Ensuite, elle vérifie si ce bouton est contenu dans une instance de '**GMCase**', qui représente une case de la grille. Si c'est le cas, elle active la case actuelle et désactive éventuellement la dernière case activée si elle est différente de la case actuelle. Cette classe garantit ainsi qu'une seule case est activée à la fois dans la grille, facilitant ainsi la sélection et la saisie des valeurs par l'utilisateur.

### -> GMCase.java

La classe '**GMCase**' permet à l'utilisateur d'interagir avec les cases d'une grille de Sudoku éditable. Voici ce qu'elle fait :

1. **Affichage des cases** : Elle crée visuellement chaque case de la grille avec un bouton qui affiche un chiffre de 1 à 9 ou reste vide.
2. **Saisie des chiffres** : L'utilisateur peut entrer des chiffres en cliquant sur les boutons correspondants ou en les tapant au clavier.
3. **Validation des entrées** : Elle vérifie si les chiffres entrés respectent les règles du Sudoku et affiche un message d'erreur en cas de violation.
4. **Sélection des cases** : L'utilisateur peut sélectionner une case en cliquant dessus, la mettant en surbrillance pour suivre sa progression.

## II. Structure

### -> GMChecker.java

La classe '**GMChecker**' est responsable de vérifier la cohérence d'une grille de Sudoku éditable. Voici ce qu'elle fait :

1. **Vérification de la grille** : Elle examine les lignes, les colonnes et les régions de la grille pour s'assurer qu'elles respectent les règles du jeu.
2. **Affichage des messages** : En fonction du résultat de la vérification, elle affiche un message informatif indiquant si la grille est cohérente ou non.
3. **Validation des entrées** : Elle détecte les doublons dans les lignes, les colonnes et les régions de la grille, signalant ainsi les erreurs à l'utilisateur.
4. **Traitement des erreurs** : En cas de détection d'erreurs, elle fournit des informations détaillées sur les doublons et demande à l'utilisateur de corriger la grille.

### -> GMHowToCreateController.java

La classe '**GMHowToCreateController**' agit comme un contrôleur pour gérer l'affichage de la fenêtre de création de grille dans le Sudoku. Voici ce qu'elle fait :

1. **Initialisation du gestionnaire de dialogue** : Elle crée une instance du gestionnaire de dialogue '**GMHowToCreateDialogManager**' dans son constructeur.
2. **Réaction aux événements de clic sur un bouton** : Elle implémente l'interface **ActionListener**, ce qui lui permet de réagir aux événements de clic sur un bouton.
3. **Affichage de la fenêtre de dialogue** : Lorsqu'un événement d'action est déclenché, comme un clic sur un bouton, la méthode **actionPerformed** est invoquée. Dans cette méthode, elle appelle la méthode **showDialog()** du gestionnaire de dialogue pour afficher la fenêtre de dialogue expliquant comment créer une grille de Sudoku.

### -> GMHowToCreateDialogManager.java

La classe '**GMHowToCreateDialogManager**' est un gestionnaire de dialogue de l'application du Sudoku. Il affiche simplement une boîte de dialogue modale avec des instructions claires sur la création de grille en utilisant **JOptionPane**.

### -> GMHowToCreateView.java

La classe '**GMHowToCreateView**' est une vue de l'application Sudoku qui affiche des instructions pour créer une grille dans une boîte de dialogue. Elle hérite de **JPanel** et dispose les composants selon un **BorderLayout**. Cette vue comprend un titre, affiché en haut de la boîte de dialogue, ainsi que du texte des instructions, affiché au centre de la boîte de dialogue avec une barre de défilement. Les styles de texte, tels que la police, la couleur et la taille, sont définis pour assurer une présentation claire et lisible.

## II. Structure

### -> **GMImport.java**

La classe '**GMImport**' est une classe utilisée pour importer une grille à partir d'un fichier. Elle implémente l'interface **ActionListener** pour réagir aux événements de clic sur un bouton. Cette classe permet à l'utilisateur de sélectionner un fichier contenant les données de la grille à importer.

### -> **GMSaver.java**

La classe '**GMSaver**' facilite l'enregistrement des grilles de jeu dans des fichiers spécifiques pour les utilisateurs. Son objectif est de rendre le processus d'enregistrement simple et intuitif. Lorsque les utilisateurs souhaitent sauvegarder une grille, ils peuvent utiliser cette fonctionnalité pour sélectionner l'emplacement et le nom du fichier où la grille sera stockée.

### -> **GMSaverActionListener.java**

La classe '**GMSaverActionListener**' agit en tant qu'écouteur d'événements de clic sur le bouton de sauvegarde. Lorsqu'un événement de clic se produit, cette classe crée une instance de '**GMSaver**' et appelle sa méthode **saveGridIfPossible()** pour gérer la sauvegarde de la grille si celle-ci est valide.

Son objectif principal est de faciliter l'interaction entre l'interface utilisateur et la sauvegarde des données de la grille. En associant cet écouteur au bouton de sauvegarde, les utilisateurs peuvent déclencher le processus de sauvegarde en un seul clic, offrant ainsi une expérience utilisateur fluide et intuitive.

### -> **GMResetGrid.java**

La classe '**GMResetGrid**' joue un rôle important dans la réinitialisation de la grille de jeu. Son objectif principal est de permettre aux utilisateurs de recréer une grille. Lorsque l'événement de clic sur le bouton de réinitialisation est déclenché, cette classe prend en charge la création d'une nouvelle grille vide en remplaçant les valeurs existantes par des zéros.

### -> **GMUserInterfaceController.java**

Le **GMUserInterfaceController** est le contrôleur de l'interface utilisateur pour le GridMaker. Il gère les actions de l'utilisateur sur l'interface en réponse à des événements. Ce contrôleur est chargé de fermer la fenêtre de l'application lorsque l'utilisateur choisit de quitter. Pour cela, il implémente l'interface **ActionListener** et définit une action pour la méthode **actionPerformed**, qui se déclenche lorsque l'utilisateur interagit avec l'interface. Lorsque cette méthode est appelée, elle invoque la méthode **System.exit(0)** pour fermer proprement l'application. En résumé, le **GMUserInterfaceController** assure une sortie propre de l'application lorsque l'utilisateur choisit de la quitter, offrant ainsi une expérience utilisateur fluide et intuitive.

### -> **GMGrid.java**

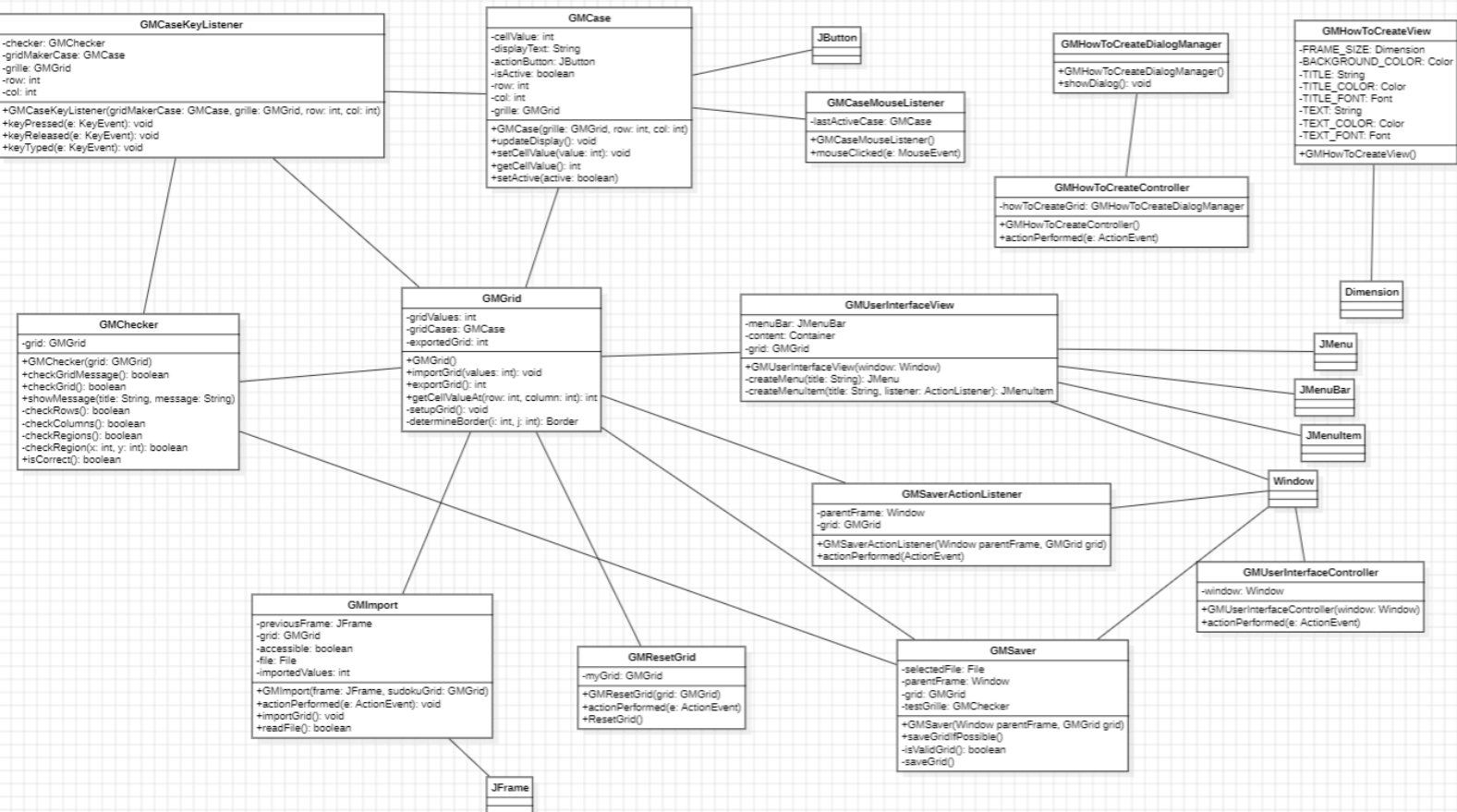
La classe '**GMGrid**' représente une grille de Sudoku héritant de **JPanel**. Elle est chargée de stocker les valeurs de chaque cellule dans la grille, de gérer les interactions avec les cases individuelles, d'exporter les données de la grille et de contrôler son apparence visuelle. Elle initialise la grille, détermine les bordures des cellules et permet l'importation des données de la grille. De plus, elle offre des méthodes pour importer et exporter les valeurs de la grille, ainsi que pour récupérer la valeur d'une cellule spécifique. Cette classe fournit une structure solide pour la représentation et la manipulation d'une grille de Sudoku dans une interface graphique

### -> **GMUserInterfaceView.java**

La classe '**GMUserInterfaceView**' représente la vue de l'interface utilisateur pour le créateur de grille. Elle crée une barre de menu avec des options pour créer, charger et sauvegarder des grilles de Sudoku, ainsi que pour accéder à l'aide et aux règles du Sudoku. Cette classe initialise les différents composants de l'interface utilisateur, notamment la barre de menu et la grille de Sudoku, et les organise dans la fenêtre de l'application. Elle fournit également des méthodes pour créer des menus et des éléments de menu avec des titres spécifiés et des écouteurs d'événements associés. Grâce à cette classe, les utilisateurs peuvent facilement interagir avec l'application pour créer et gérer des grilles de Sudoku de manière intuitive et efficace.

# II. Structure

-> Diagramme de classe



#### -> **Button.java**

La classe '**Button**' contient des paramètres personnalisés pour les boutons Swing. Elle offre plusieurs constructeurs permettant de définir le texte, la dimension, la police de caractères et la couleur d'arrière-plan des boutons. Ces paramètres facilitent la personnalisation et l'intégration harmonieuse des boutons dans une interface utilisateur Swing.

#### -> **DialogManager.java**

L'interface '**DialogManager**' définit la méthode **showDialog()**, qui est utilisée pour afficher une boîte de dialogue. Cette interface ne fournit pas d'implémentation concrète de la boîte de dialogue, mais elle définit simplement la structure pour les classes qui souhaitent implémenter cette fonctionnalité. Cela permet une certaine flexibilité dans la façon dont les boîtes de dialogue sont affichées et gérées dans différentes parties de l'application.

#### -> **MusicButton.java**

La classe '**MusicButton**' fournit un bouton graphique qui permet de basculer entre la lecture et l'arrêt de la musique lorsqu'il est cliqué. Elle utilise deux icônes différentes pour représenter l'état activé et désactivé de la musique. Lorsque le bouton est cliqué, il vérifie s'il y a déjà une musique en cours de lecture. Si c'est le cas, il l'arrête et change l'icône du bouton pour représenter que la musique est désactivée. Sinon, il lance la lecture de la musique et change l'icône du bouton pour représenter que la musique est activée.

Cette classe offre une façon conviviale d'activer et de désactiver la musique dans une application à l'aide d'un bouton visuel. Elle prend en compte la possibilité qu'une seule musique puisse être jouée à la fois.

#### -> **MusicPlayer.java**

La classe '**MusicPlayer**' fournit un lecteur de musique simple qui permet de jouer et d'arrêter la musique à partir d'un fichier audio spécifié. Elle utilise la bibliothèque Java Sound pour lire le fichier audio.

Le lecteur de musique utilise un objet **Clip** pour jouer la musique. Lorsque la méthode **play()** est appelée, la musique démarre à partir du début du fichier audio. La méthode **stop()** permet d'arrêter la lecture de la musique en cours.

La classe contient également une méthode **isPlaying()** qui retourne un booléen indiquant si la musique est en train de jouer ou non.



## II. Structure

### -> Title.java

La classe '**Title**' fournit une manière pratique d'afficher des titres dans une interface graphique Swing. En utilisant cette classe, les développeurs peuvent créer des titres esthétiques et uniformes avec une police et une couleur spécifiées. Cette classe s'occupe automatiquement de centrer horizontalement le texte, ce qui simplifie la mise en page des titres dans les composants Swing tels que les panneaux, les fenêtres, ou les boîtes de dialogue.

### -> RulesDialogManager.java

La classe '**RulesDialogManager**' gère l'affichage des règles du Sudoku dans une boîte de dialogue. Elle fournit une interface pour afficher les règles du jeu à l'utilisateur. En utilisant cette classe, les règles peuvent être facilement affichées dans une fenêtre contextuelle sans nécessiter de gestion supplémentaire de la part du développeur. Cela facilite l'expérience utilisateur en rendant les règles du jeu accessibles à tout moment, ce qui est particulièrement utile pour les nouveaux joueurs qui souhaitent apprendre les règles du Sudoku.

### -> RulesSudoku.java

La classe '**RulesSudoku**' représente un panneau affichant les règles du jeu Sudoku. Elle étend JPanel et définit le contenu des règles à afficher dans une fenêtre dédiée.

Dans son constructeur, elle configure l'apparence du panneau en utilisant un gestionnaire de disposition BorderLayout. Le panneau a une couleur d'arrière-plan spécifique et une taille définie.

### -> Rules.java

La classe '**Rules**' gère les actions liées aux règles du jeu dans l'application Sudoku. En implémentant l'interface '**ActionListener**', elle peut réagir aux événements, tels que les clics sur un bouton.

## II. Structure

### -> **HomeView.java**

La classe '**HomeView**' représente la vue de la page d'accueil de l'application Sudoku. Elle étend JPanel et affiche les éléments de la page d'accueil, y compris le titre, les boutons et les contrôles audio.

Elle utilise les classes '**Title**', '**Button**', et '**MusicButton**' pour créer et organiser les composants de la page d'accueil. Le constructeur prend en paramètre la fenêtre parente, le titre principal, le sous-titre et les textes des boutons à afficher.

Dans sa méthode **createComponents()**, elle crée les panneaux pour le titre et les boutons, configure leur disposition et leur apparence. Elle utilise également des écouteurs de clic pour les boutons.

La méthode **addComponentsToWindow()** ajoute les composants créés à la fenêtre parente en utilisant un gestionnaire de disposition **BorderLayout**.

### -> **HomeButtonClickListener.java**

Le '**HomeButtonClickListener**' est un controller qui écoute les clics effectués sur les boutons dans le menu principal de l'application Sudoku. Son rôle est d'interpréter ces clics et d'initier les actions correspondantes. Par exemple, si l'utilisateur clique sur le bouton "**Jouer**", le '**HomeButtonClickListener**' est chargé de préparer la transition vers l'écran de jeu en supprimant les composants actuels et en initialisant le menu de jeu. De même, si l'utilisateur souhaite créer une nouvelle grille ou consulter les règles, le '**HomeButtonClickListener**' prend en charge ces demandes en lançant les fonctionnalités appropriées de l'application. En résumé, cette classe facilite l'interaction de l'utilisateur avec le menu principal en dirigeant ses actions vers les fonctionnalités correspondantes de l'application.

### -> **Window.java**

La classe '**Window**' représente la fenêtre principale de l'application Sudoku. En étendant JFrame, elle fournit un cadre de base pour afficher les différentes pages de l'application. Cette classe initialise la fenêtre avec une taille minimale spécifiée et une couleur d'arrière-plan par défaut. De plus, elle permet de définir le titre de la page actuelle, ce qui met à jour automatiquement le titre de la fenêtre pour inclure à la fois le titre de la page et le titre du programme.

### -> **GridMaker.java** et **GridSolver.java**

Les classes '**GridMaker.java**' et '**GridSolver.java**' représente les classes principales des deux programmes. En effet, elles permettent de lancer le créateur de grille et le jeu.

### 4) Makefile

#### A) Commandes utiles

##### Compilation et lancement du createur de grille

Utiliser la commande suivante pour compiler et lancer le créateur de grille :

```
make GridMaker
```

##### Compilation et lancement du solveur de grille

Utiliser la commande suivante pour compiler et lancer le solveur de grille :

```
make GridSolveur
```

##### Suppression des fichiers .class et de la documentation

```
make clean
```

##### Générer la documentation

```
make doc
```



# III. Explication de l'algorithme de résolution

## La méthode solve()

La méthode **solve()** permet de résoudre un Sudoku de manière récursif. Elle vise à remplir la grille de Sudoku actuelle en essayant différentes combinaisons de chiffres dans chaque case jusqu'à ce que la solution complète soit trouvée.

Voici une explication détaillée de la méthode **solve()** :

- 1. Boucles Parcourant la Grille :**
  - La méthode utilise deux boucles imbriquées pour parcourir chaque case de la grille. Ces boucles vont de 0 à 8 pour les lignes et les colonnes de la grille 9x9.
- 2. Vérification de la Case Vide :**
  - À chaque itération des boucles, la méthode vérifie si la case actuelle est vide. Une case vide est représentée par la valeur 0 dans le tableau **tableauGrille**.
- 3. Tentative de Remplissage de la Case :**
  - Si la case est vide, la méthode commence à essayer différentes valeurs de 1 à 9 pour la remplir.
- 4. Validation de la Valeur :**
  - Pour chaque valeur testée dans la case vide, la méthode vérifie si cette valeur est valide en appelant la méthode **isValid()** de la classe **GSTest**. Cette méthode vérifie si la valeur à insérer respecte les règles du Sudoku, c'est-à-dire qu'elle ne doit pas déjà exister dans la même ligne, la même colonne ou le même bloc 3x3.
- 5. Récursion pour la Prochaine Case :**
  - Si la valeur testée est valide, la méthode appelle récursivement **solve()** pour passer à la case suivante dans la grille et tenter de la remplir.
- 6. Retour en Arrière :**
  - Si la récursion échoue à trouver une solution, la méthode réinitialise la valeur de la case actuelle à 0, ce qui la rend vide, puis elle revient en arrière pour essayer une autre valeur dans la case précédente.
- 7. Condition de Sortie :**
  - La méthode continue ce processus de manière récursive jusqu'à ce qu'elle remplisse toutes les cases de la grille avec des valeurs valides, ou jusqu'à ce qu'elle atteigne une impasse où aucune valeur ne peut être insérée dans une case donnée.
- 8. Résultat de la Résolution :**
  - Si la méthode trouve une solution valide, elle retourne **true**, indiquant que le Sudoku a été résolu avec succès. Sinon, elle retourne **false**, indiquant que la grille ne peut pas être résolue dans son état actuel.
- 9. Backtracking :**
  - L'utilisation de la récursion et du backtracking permet à l'algorithme d'explorer toutes les combinaisons possibles de chiffres dans chaque case jusqu'à ce qu'une solution valide soit trouvée, ou qu'il soit déterminé qu'aucune solution n'est possible.

En résumé, la méthode **solve()** utilise une approche de “brute force” avec une combinaison de récursion et de backtracking pour résoudre le Sudoku en explorant toutes les possibilités jusqu'à ce qu'une solution valide soit trouvée.

## IV. Conclusions personnelles

### Moncef STITI

Ce projet a été une expérience très formatrice pour moi. Malgré les contraintes de temps (dû aux nombreux contrôles), j'ai pu mettre en pratique mes connaissances en programmation orientée objet et en architecture logicielle en utilisant le modèle MVC en Java. Ce choix s'est avéré judicieux, car il nous a permis de bien séparer la logique, la présentation et la gestion des événements, rendant ainsi le code plus clair, facile à maintenir et à étendre. De plus, cette approche a grandement facilité la collaboration avec les autres membres de l'équipe de développement.

En parallèle du développement java, j'ai également approfondi ma maîtrise des outils de collaboration et de versionnage tels que Git.

Je suis convaincu que les compétences acquises lors de ce projet seront précieuses pour mes projets futurs.

### Marco ORFAO

En conclusion, ce projet a été une expérience enrichissante sur le plan des compétences acquises. Il s'est révélé captivant, nous plongeant dans des défis stimulants tout au long de son développement. De plus, il a été l'occasion d'approfondir notre compréhension des projets de groupe, en nous permettant de mieux appréhender leur déroulement et les conditions nécessaires pour garantir un travail fluide et une répartition efficace des tâches. Nous avons été confrontés à diverses situations qui nous ont permis d'affiner nos compétences en collaboration et en gestion de projet. En somme, cette expérience nous a apporté une précieuse leçon sur l'importance de la communication, de l'organisation et du respect des conditions requises pour mener à bien un travail d'équipe

# Rapport

---

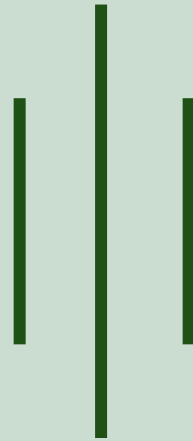
# RÉALISÉE PAR



**Moncef STITI**



**Marco ORFAO**



---

Ce rapport à été réaliser par Moncef STITI et Marco ORFAO dans le cadre de la SAÉ2.01 "Développement d'une application" lors du deuxième semestre de BUT 1 à l'IUT de Fontainebleau.

Lien GitTea du projet : [https://grond.iut-fbleau.fr/stiti/SAE21\\_2023](https://grond.iut-fbleau.fr/stiti/SAE21_2023)

---