Youcef ANNAB

William-Arno CLEMENT

Projet Tuteuré

Le Sudoku

Langage JAVA

Le Sudoku

Dans le cadre de nos études en IUT Informatique, nous avons eu l'opportunité de réaliser deux programmes permettant la création et la résolution de grilles de Sudoku. Le travail a été réalisé à l'aide du langage JAVA et de l'API officielle.

Notre objectif pour la réalisation de ce travail est multiple :

- Poursuivre notre apprentissage pour une bonne maîtrise du langage JAVA, de ses subtilités de langage orienté objet
- Approfondir nos connaissances sur l'utilisation et la manipulation des flux d'octets, et de la lecture/écriture de fichiers
- Développer une interface utilisateur pratique et conviviale
- Comprendre et anticiper les actions de l'utilisateur

Dans ce document, nous allons vous expliquer notre démarche de développement et vous faire découvrir l'évolution de notre travail.

Nous vous souhaitons une agréable lecture.

Sommaire

Le projet et ses contraintes

Le jeu du Sudoku consiste à compléter entièrement une grille préalablement remplie en respectant les contraintes d'unicité: un chiffre ne peut être disposé qu'une seule fois par ligne, par colonne et par région.

Nous devons donc réaliser l'interface permettant à l'utilisateur de jouer en important une grille, mais également mettre en place un système de résolution automatique de la grille.

Notre travail doit donc s'articuler sur deux programmes, à savoir le jeu, mais aussi l'interface de création de Grille. Nous décidons donc de scinder dès le début notre programme en deux parties:

- **GridMaker**, l'interface de création de Grille
- **GridSolver**, l'interface rassemblant le jeu et le système de résolution automatique

Nous avons commencé par le développement de GridMaker, c'est pourquoi nous commencerons par évoquer ce programme. Dans un premier temps, nous avons dû procéder à une lecture du cahier des charges dont voici ci-dessous les contraintes imposées pour la réalisation de notre projet.

GridMaker

* partir d'une grille vide
* partir d'une grille existante

} gestion de la sauvegarde

GridSolver

* resolution manuelle
* resolution automatique - afficher temps de resolution

Contraintes globales (GridMaker et GridSolver)

- une grille de Sudoku est composée de neuf ligne et de neuf colonnes
- elle est également divisée en neuf régions couvrant chacune trois ligne et trois colonnes
- chaque case peut accueillir un chiffre compris entre 1 et 9
- un même chiffre ne peut apparaitre plusieurs fois sur une même ligne/colonne/région

Contraintes spécifiques à GridMaker

- pour construire une grille, on peut partir d'une grille préexistante via un import de fichier
- On doit pouvoir ajouter et enlever des numéros dans la grille librement
- Le programme doit empêcher les placements contradictoires
- GridMaker doit mettre à disposition un système de sauvegarde de grille dans un fichier
- On doit user du format employé pour les fichiers qui contiennent la représentation canonique Java des entiers composant la grille les uns à la suite des autres

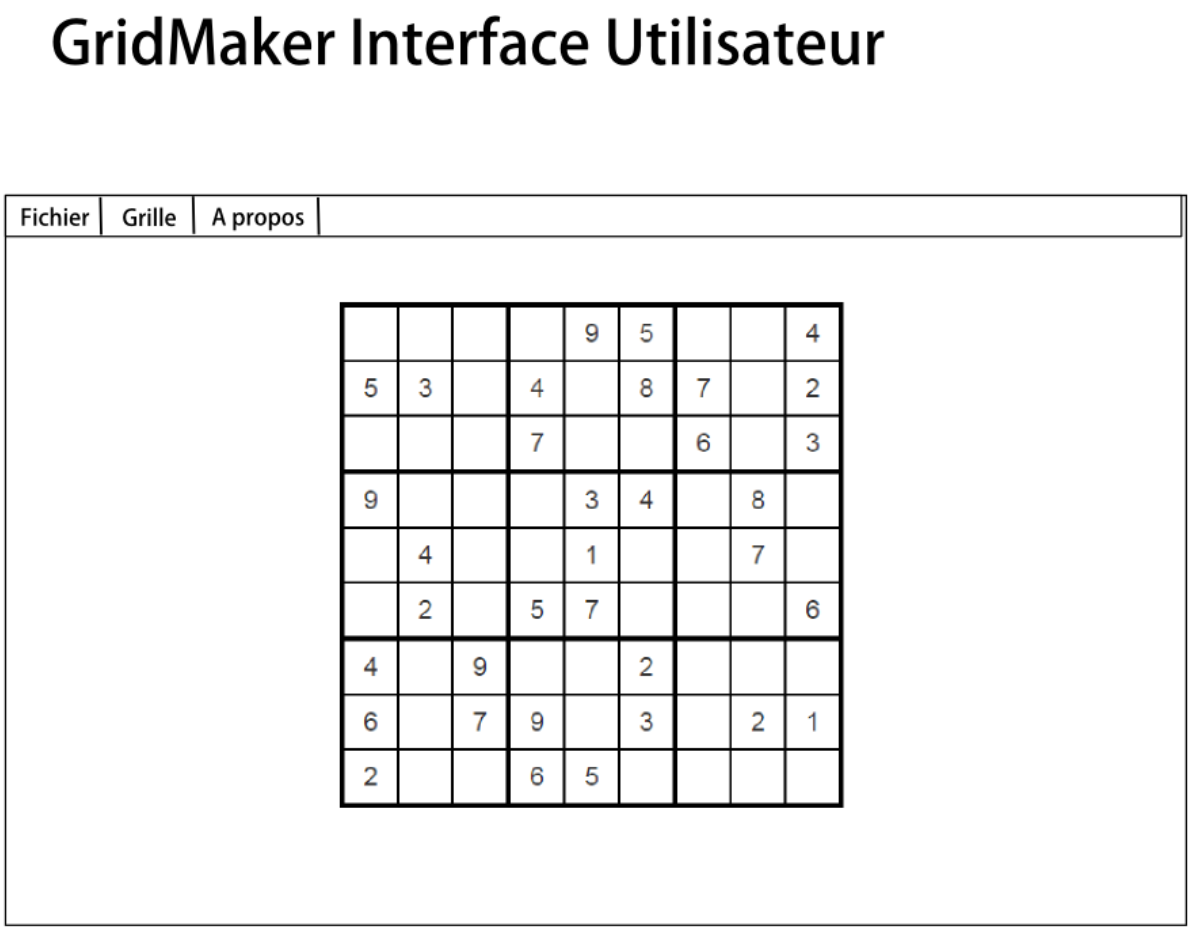
Contraintes spécifiques à GridSolver

- On peut charger une grille depuis un fichier
- on doit pouvoir choisir la résolution automatique ou manuelle de la grille chargée
- On doit afficher dans les deux cas le temps nécessaire à la résolution
- on affichera également à la fin la grille résolue
- en mode manuel, le joueur pourra ajouter des chiffre sauf si cela interfère avec la contrainte d'unicité

- Le joueur pourra enlever des chiffres parmi ceux qu'il a ajouté
- il pourra également faire cohabiter jusqu'à 4 chiffres par cases
- Le joueur sera félicité par le programme lorsque la grille sera remplie correctement

Les fonctionnalités

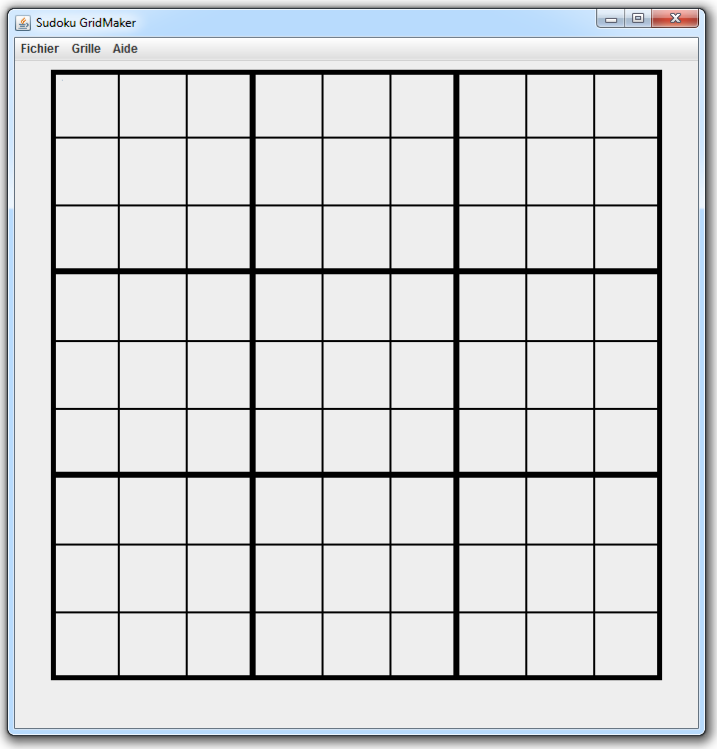
GridMaker



première ébauche de l'interface de GridMaker

GridMaker est pour nous le socle de notre travail. De nombreuses classes et méthodes vont servir pour la construction de GridSolver plus tard. Il est donc important de partir sur de bonnes bases.

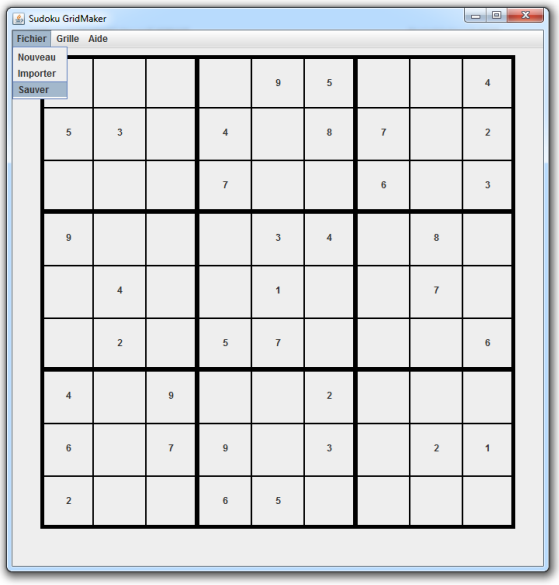
L'interface propose au lancement une grille vide ainsi qu'un menu en haut. Dessus sont disposés des menus déroulants proposants plus d'options. Cette interface est ergonomique et c'est pourquoi nous l'avons choisie.

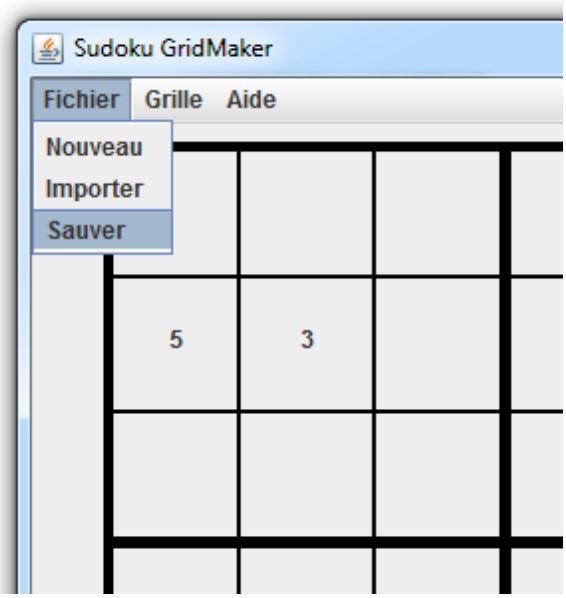


interface définitive de GridMaker

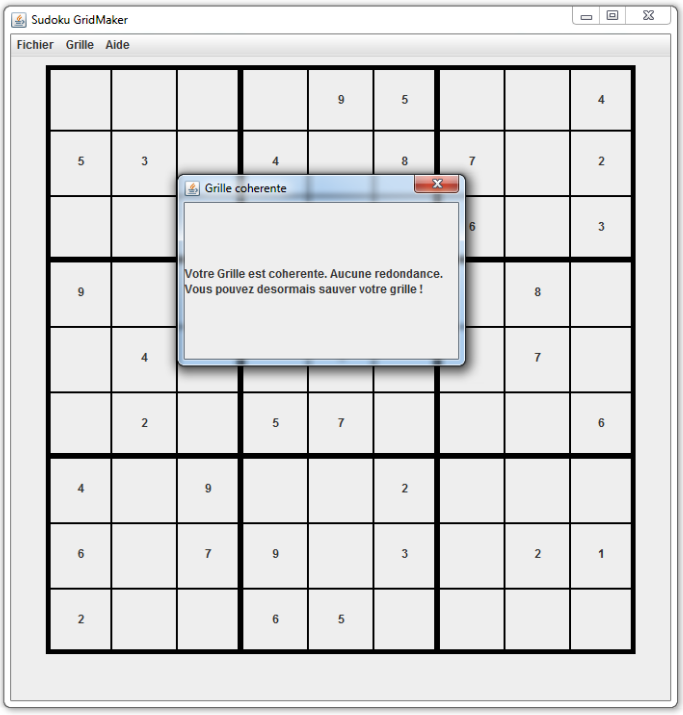
L'interface est constitué de trois partie : le menu fichier qui permet :

- de réinitialiser la grille en mettant toutes le valeurs à zéro (et donc les cases à vide)
- d'importer une grille préexistante dans un fichier au format décrit par l'annexe 1 du sujet
- de sauvegarder la grille courante dans un fichier (nouveau ou préexistant).





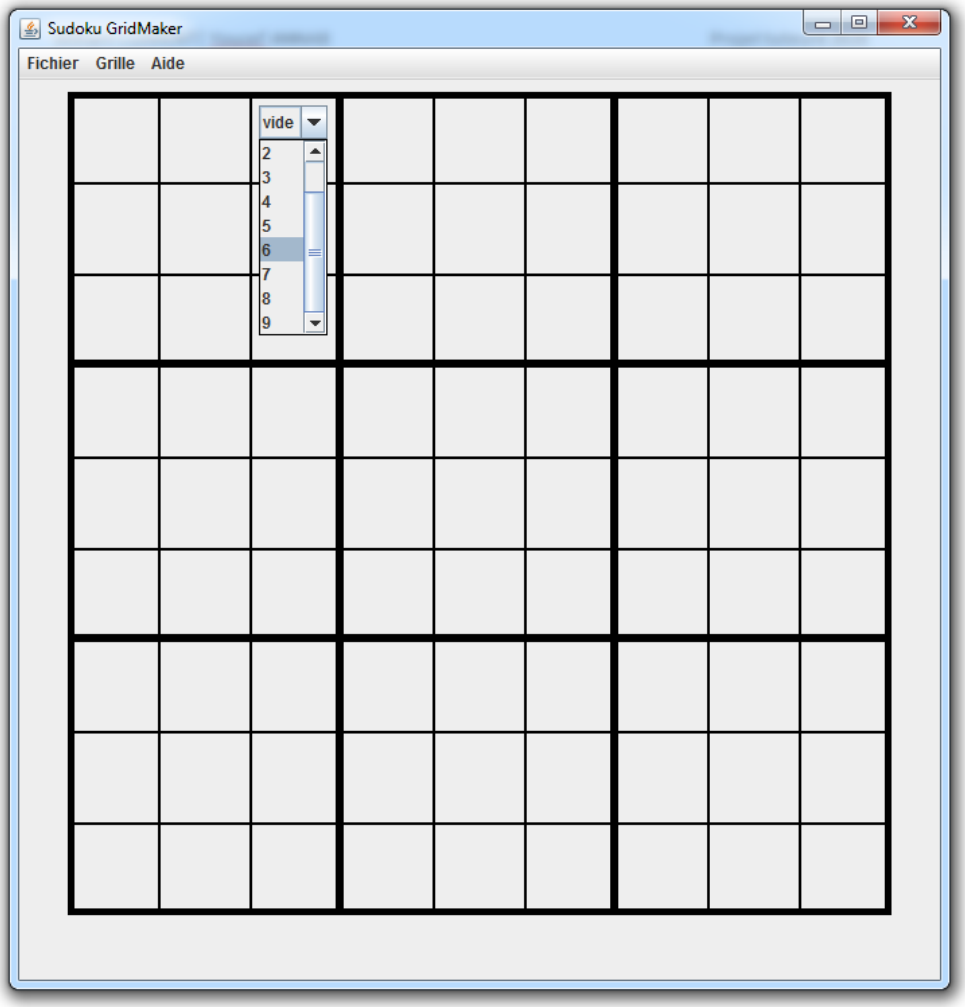
Le menu Grille, qui permet de Vérifier à tout moment si la grille créée est cohérente.



Un message est affiché si la grille est cohérente

Enfin, le menu Aide permet de faire connaissance avec les créateurs du logiciel.

Enfin, pour le choix des valeurs, nous avons choisi une technique de liste déroulante, simple d'utilisation et d'aspect professionnel.



On peut choisir la valeur de la case à l'aide d'un menu déroulant

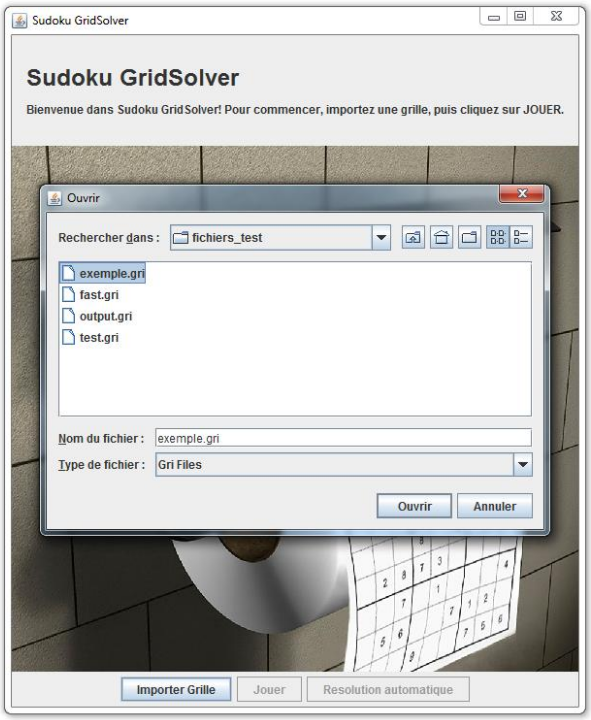
GridSolver

Pour GridSolver, nous avons choisi une interface redessiné, plus conviviale, plus rapide à prendre en main. L'interface devait être dépouillée de tout composant non indispensable. C'est pourquoi nous avons retiré le menu et les liste déroulantes. Désormais, les valeurs sont entrées au clavier: c'est un gain de temps considérable pour le joueur.

Pour refaire le design au mieux, nous avons séparé le menu de l'interface de Gameplay c'est à dire du jeu lui même.

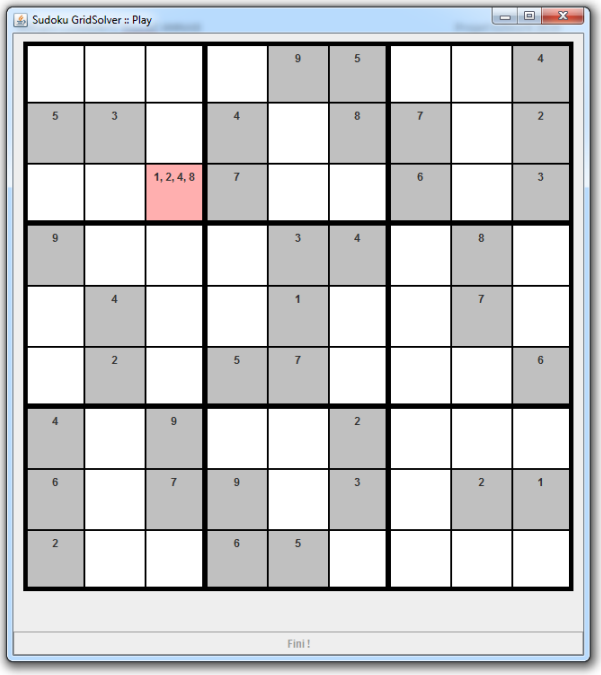
On doit d'abord importer une grille avant de pouvoir résoudre cette dernière

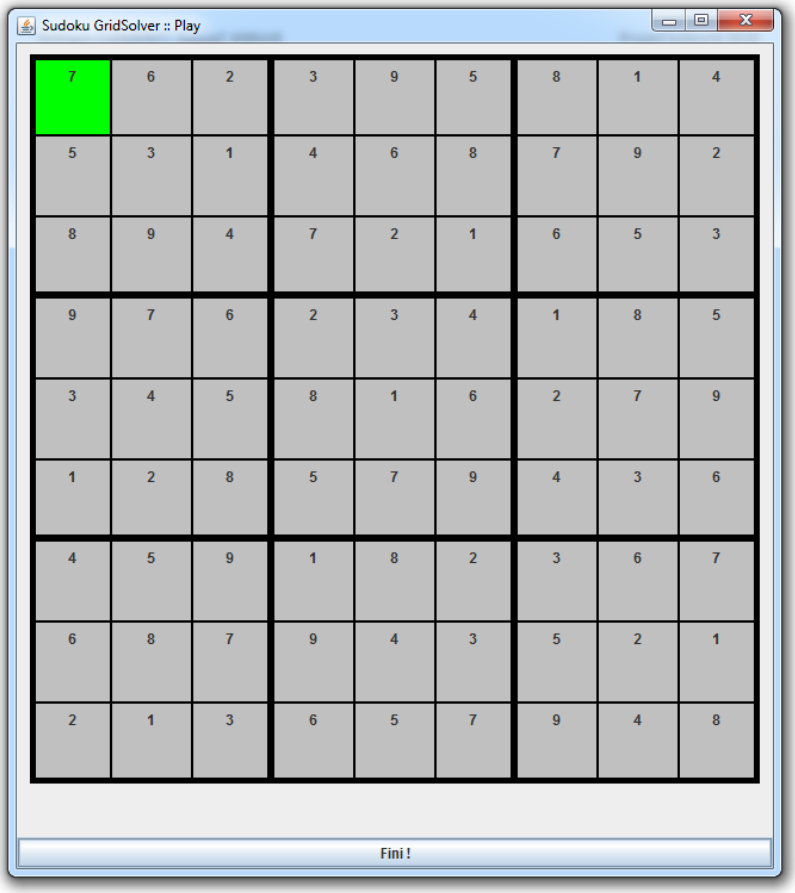
Une fois la grille importée, nous pouvons lancer la résolution manuelle ou automatique.



L'importation de fichier est réalisée à l'aide de la classe JFileChooser

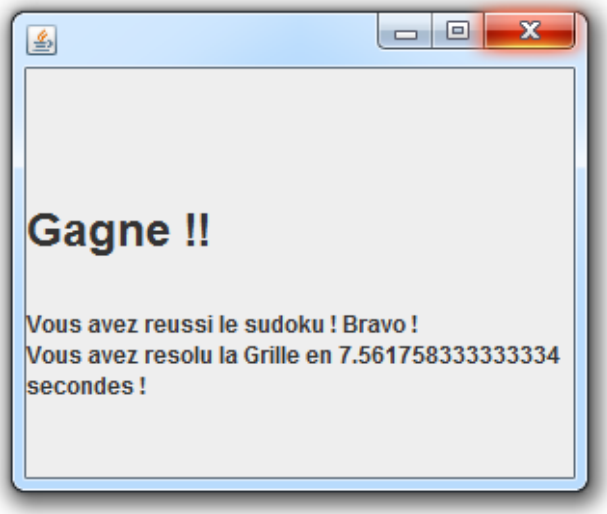
On peut ajouter jusqu'à 4 valeurs par cases en cas de doute, en résolution manuelle.



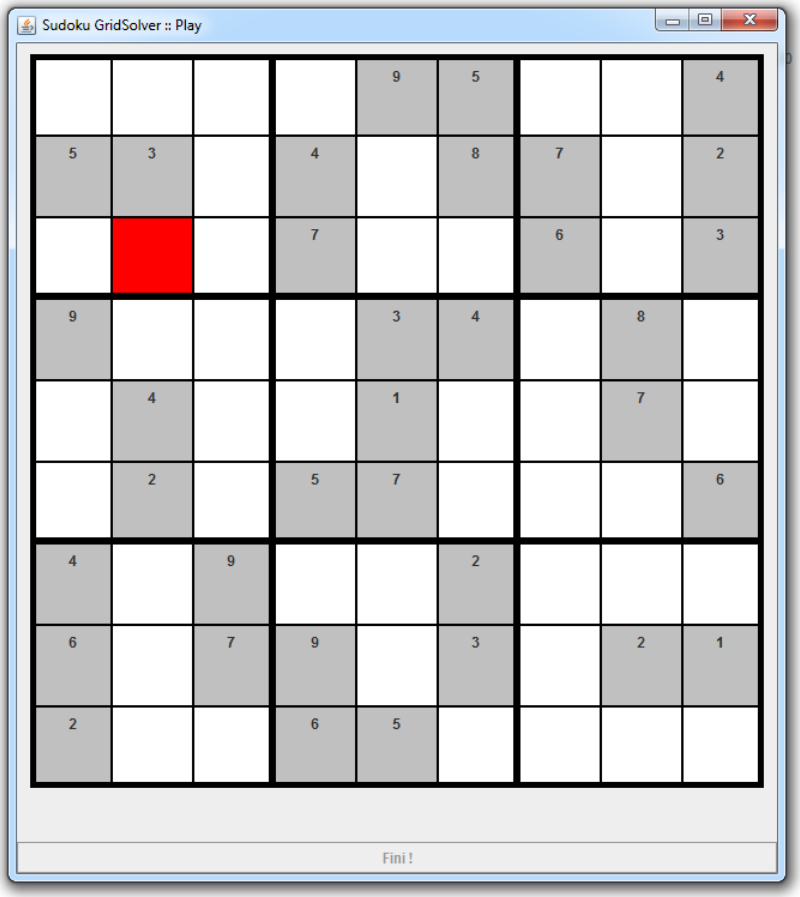


Le bouton "Fini !" devient interactif lorsque toutes les cases sont correctement remplies.

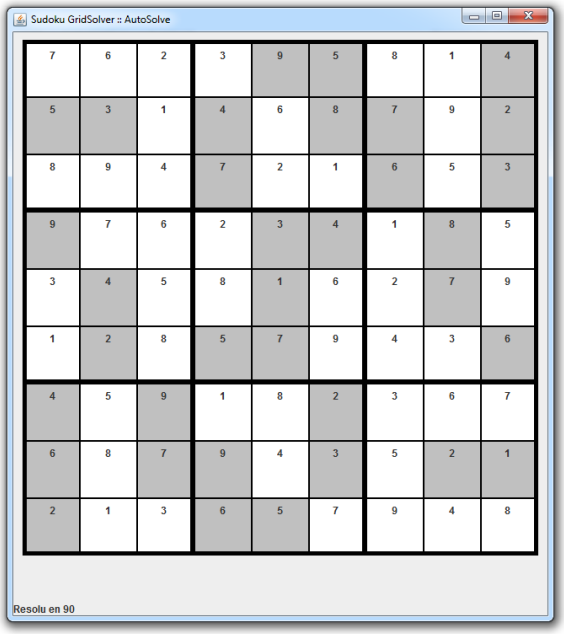
Une fenêtre de félicitations du joueur s'affiche lorsqu'il a résolu le Sudoku en entier. On affiche également le temps nécessaire à sa résolution.



On notera également que la case devient rouge dans le cas ou le joueur entre une valeur erronée, c'est à dire non cohérente avec la grille.

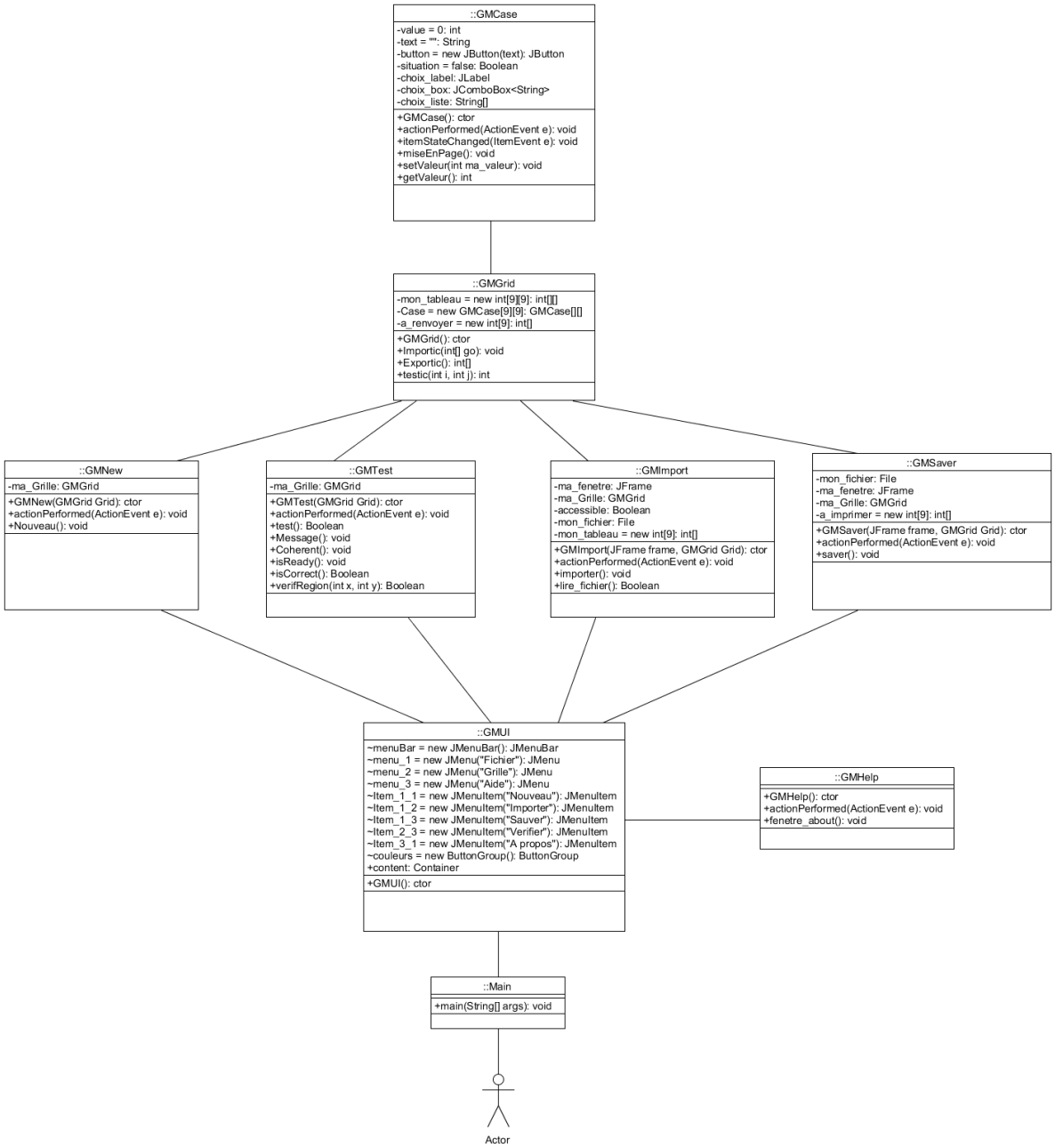


Dans le cas de la résolution automatique, le bouton "Fini !" disparait au profit du temps de résolution de l'algorithme de Backtracking.

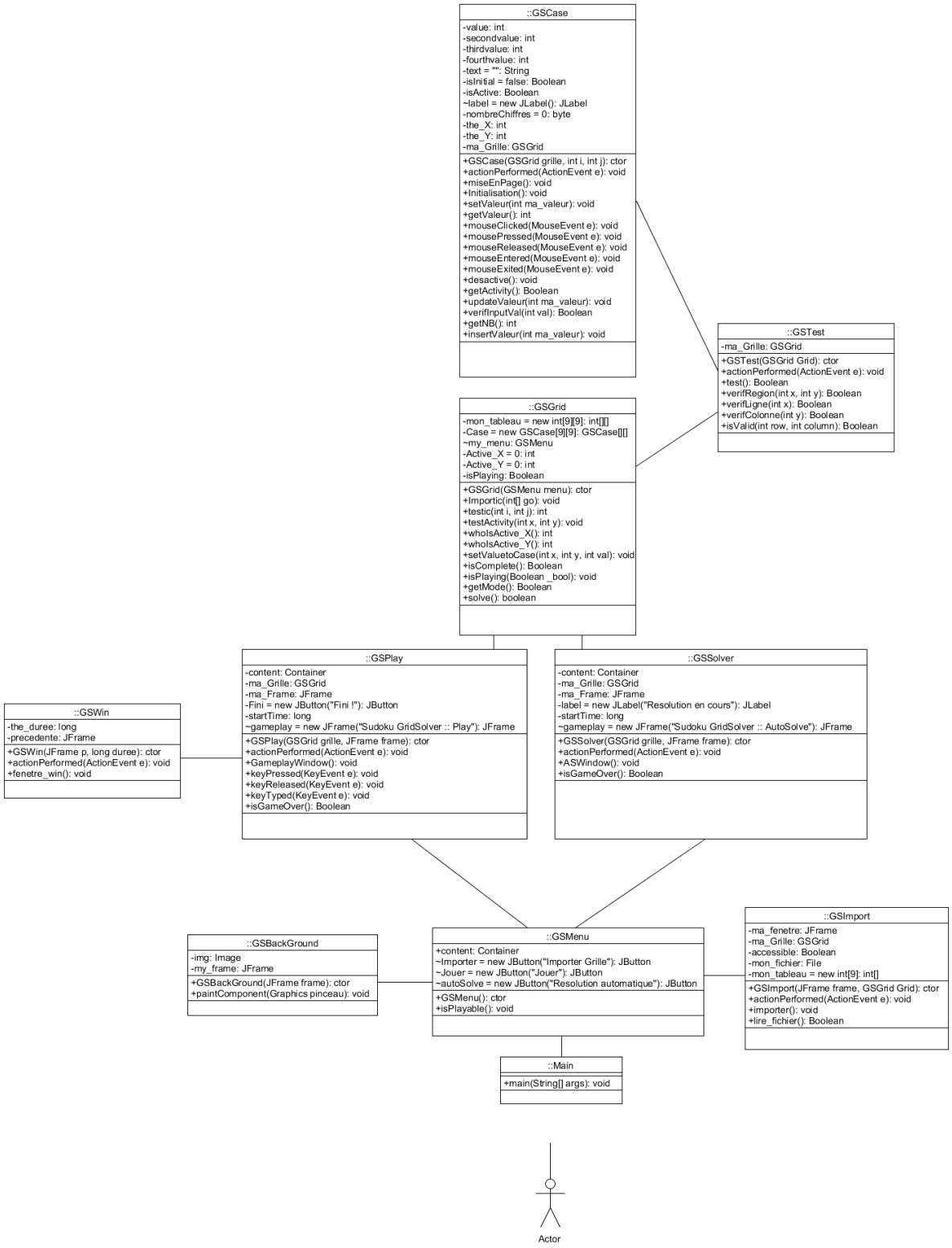


Structures des programmes

GridMaker



GridSolver


::GSCase
-value: int
-secondvalue: int
-thirdvalue: int
-fourthvalue: int
-text = "": String
-isInitial = false: Boolean
-isActive: Boolean
~label = new JLabel(): JLabel
-nombreChiffres = 0: byte
-the_X: int
-the_Y: int
-ma_Grille: GSGrid
+GSCase(GSGrid grille, int i, int j): ctor
+actionPerformed(ActionEvent e): void
+miseEnPage(): void
+Initialisation(): void
+setValeur(int ma_valeur): void
+getValeur(): int
+mouseClicked(MouseEvent e): void
+mousePressed(MouseEvent e): void
+mouseReleased(MouseEvent e): void
+mouseEntered(MouseEvent e): void
+mouseExited(MouseEvent e): void
+desactive(): void
+getActivity(): Boolean
+updateValeur(int ma_valeur): void
+verifInputVal(int val): Boolean
+getNB(): int
+insertValeur(int ma_valeur): void
::GSTest
-ma_Grille: GSGrid
+GSTest(GSGrid Grid): ctor
+actionPerformed(ActionEvent e): void
+test(): Boolean
+verifRegion(int x, int y): Boolean
+verifLigne(int x): Boolean
+verifColonne(int y): Boolean
+isValid(int row, int column): Boolean
::GSGrid
-mon_tableau = new int[9][9]: int[][]
-Case = new GSCase[9][9]: GSCase[][]
~my_menu: GSMenu
-Active_X = 0: int
-Active_Y = 0: int
-isPlaying: Boolean
+GSGrid(GSMenu menu): ctor
+Importic(int[] go): void
+testic(int i, int j): int
+testActivity(int x, int y): void
+whoIsActive_X(): int
+whoIsActive_Y(): int
+setValuetoCase(int x, int y, int val): void
+isComplete(): Boolean
+isPlaying(Boolean _bool): void
+getMode(): Boolean
+solve(): boolean
::GSPlay
-content: Container
-ma_Grille: GSGrid
-ma_Frame: JFrame
-Fini = new JButton("Fini !"): JButton
-startTime: long
~gameplay = new JFrame("Sudoku GridSolver :: Play"): JFrame
+GSPlay(GSGrid grille, JFrame frame): ctor
+actionPerformed(ActionEvent e): void
+GameplayWindow(): void
+keyPressed(KeyEvent e): void
+keyReleased(KeyEvent e): void
+keyTyped(KeyEvent e): void
+isGameOver(): Boolean
::GSSolver
-content: Container
-ma_Grille: GSGrid
-ma_Frame: JFrame
-label = new JLabel("Resolution en cours"): JLabel
-startTime: long
~gameplay = new JFrame("Sudoku GridSolver :: AutoSolve"): JFrame
+GSSolver(GSGrid grille, JFrame frame): ctor
+actionPerformed(ActionEvent e): void
+ASWindow(): void
+isGameOver(): Boolean
::GSWin
-the_duree: long
-precedente: JFrame
+GSWin(JFrame p, long duree): ctor
+actionPerformed(ActionEvent e): void
+fenetre_win(): void
::GSBackGround
-img: Image
-my_frame: JFrame
+GSBackGround(JFrame frame): ctor
+paintComponent(Graphics pinceau): void
::GSMenu
+content: Container
~Importer = new JButton("Importer Grille"): JButton
~Jouer = new JButton("Jouer"): JButton
~autoSolve = new JButton("Resolution automatique"): JButton
+GSMenu(): ctor
+isPlayable(): void
::GSImport
-ma_fenetre: JFrame
-ma_Grille: GSGrid
-accessible: Boolean
-mon_fichier: File
-mon_tableau = new int[9]: int[]
+GSImport(JFrame frame, GSGrid Grid): ctor
+actionPerformed(ActionEvent e): void
+importer(): void
+lire_fichier(): Boolean
::Main
+main(String[] args): void
Actor

Les programmes GridSolver et GridMaker sont conçus à partir des classes représentant les Cases et les Grilles.

D'autres classes s'articulent autour de la classe représentant les Grilles : les tests sur les grilles des classes GSTest et GMTest, mais aussi l'interface utilisateur et les différentes actions disponibles influant sur la grille.

L'algorithme de résolution automatique

Une résolution récursive

L'algorithme de résolution automatique de GridSolver est un algorithme dit de Backtracking. Il se situe dans la méthode récursive **solve()** dans la classe **GSGrid**. Cette dernière renvoie une valeur Booléenne.

Pour une grille donnée, la fonction va parcourir chaque case du tableau via deux boucles testant chaque ligne et chaque colonne de la grille. Dans cette dite case, si la valeur est égale à zéro, on va tester chaque valeur de 1 à 9, et vérifier si elle est valide ET si les autres appels à **solve()** de manière récursive revoient TRUE. Autrement dit, on vérifie de manière récursive si les valeurs postées dans chaque cases sont valides.

La validation des valeurs

La méthode **solve()** va à chaque appel lancer une instance de la classe **GSTest**, qui va lancer sa méthode **isValid()** dans laquelle sont placés deux arguments: la ligne et la colonne de la case courante. A son tour, la méthode va vérifier si il n'y a pas de redondance de valeur au niveau de la ligne via la méthode **verifLigne()** , au niveau de la colonne via la méthode **verifColonne()** et au sein d'une même région via la méthode **verifRegion()** .

De ce fait, l'algorithme va tester systématiquement l'ensemble des affectations de valeurs potentielles dans la grille du Sudoku, puis le remplir de manière automatique.

Conclusion personnelle

William CLEMENT

Travailler sur ce projet en JAVA a été pour moi un vrai plaisir. Partir à la recherche d'informations sur la résolution des Sudoku ainsi que su le fonctionnement du langage JAVA est un travail excitant. J'ai voulu apporter ma touche au travail en créant deux interfaces utilisateurs différentes dans le cadre des deux programmes car je pense que deux contextes différents doivent posséder deux expériences utilisateur différentes.

De plus, mettre en place le logiciel et ses contraintes m'ont donné du fil à retordre mais j'ai vraiment apprécié la partie dédiée à la connexion des méthodes créées en amont.

Enfin, je dois dire que je suis plutôt fier d'avoir réalisé ce projet car l'année dernière je n'ai pas pu travailler sur le projet tuteuré. C'est pourquoi ce projet marque un tournant dans ma courte vie de développeur. Une page se tourne et je peux désormais travailler sur quelque chose de nouveau. J'ai hâte d'être en deuxième année de DUT Informatique pour enfin me lancer dans de nouvelles choses à apprendre.

Car comme le disent certains adeptes du logiciel libre: la route est longue, mais la voie est libre...