Blanche Antoni
Costandi Adrien

Groupe 6
10/04/2020

Projet Sudoku

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

Table des matières

Description du projet :

Le projet de ce deuxième semestre est un jeu de Sudoku. Celui-ci est à réaliser en Java, langage orienté objet qui a fait l'objet de quelques mois de cours dédiés.

Le Sudoku est un jeu nécessitant une grille de 9 par 9 cellules contenant des valeurs allant de 0 (vide) à 9. Cette grille est subdivisée en trois régions de 3 par 3 cellules.

Le but du jeu est de remplir la grille avec des valeurs en respectant quelques règles :

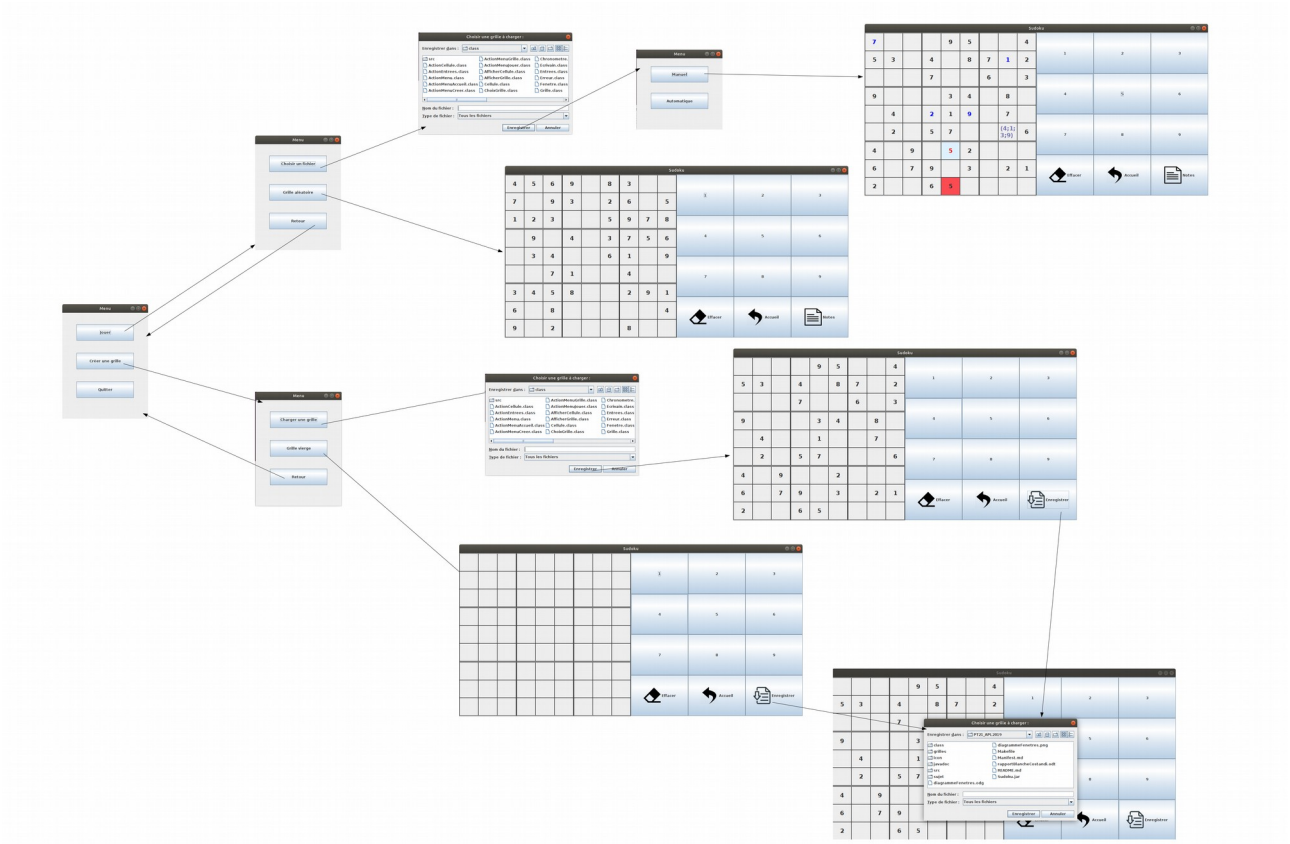
1. Une valeur ne dois être présente qu'une seule et unique fois dans une ligne
2. Une valeur ne dois être présente qu'une seule et unique fois dans une colonne
3. Une valeur ne dois être présente qu'une seule et unique fois dans une région

Ainsi chaque ligne, chaque colonne et chacune des régions doit contenir tous les chiffres de 1 à 9 en un seul exemplaire.

Chaque grille est stockée dans un fichier au format *.gri*.

Fonctionnalités :

Au début du jeu, l'utilisateur a le choix entre charger une grille pour y jouer, ou éditer sa propre grille.



Note : Cette image est disponible au format PNG dans le répertoire du projet

Si l'utilisateur choisi l'option « Jouer », il aura le choix entre charger une grille dans ses fichiers, ou jouer sur une grille générée aléatoirement.

Lorsque le joueur a choisi sa grille, il peut choisir de la résoudre lui-même, ou bien peut faire appel à un solveur.

En mode manuel, le joueur doit sélectionner une case et y insérer une avaleur à l'aide des boutons sur la droite de la fenêtre. En cas de doute, il peut utiliser les notes qui lui sont mises à disposition, grâce auxquelles il peut stoker jusqu'à 4 valeurs temporaires.

Les coups de l'utilisateur doivent respecter les règles précédemment établies pour être validés dans la grille, sinon les cases qui entrent en conflit avec une des règles sont mises en évidence,

pour aider le joueur à situer son erreur. En mode notes, les valeurs contradictoires sont même bloquées par le programme.

Les valeurs insérées dans la grille sont bloquantes : elles sont prises en compte lors des vérifications en cas d'insertion d'une nouvelle valeur sur la même ligne, colonne ou région. Les notes, elles, sont influencées pas les valeurs des autres cellules mais ne sont pas bloquantes. Il est donc possible de mettre dans deux cases adjacentes les mêmes valeurs dans les notes.

En mode automatique, le programme fait appel à un solveur, qui s'occupe de résoudre la grille pour l'utilisateur.

L'utilisateur peut à tout moment, revenir au menu d'accueil en cliquant sur le bouton dédié.

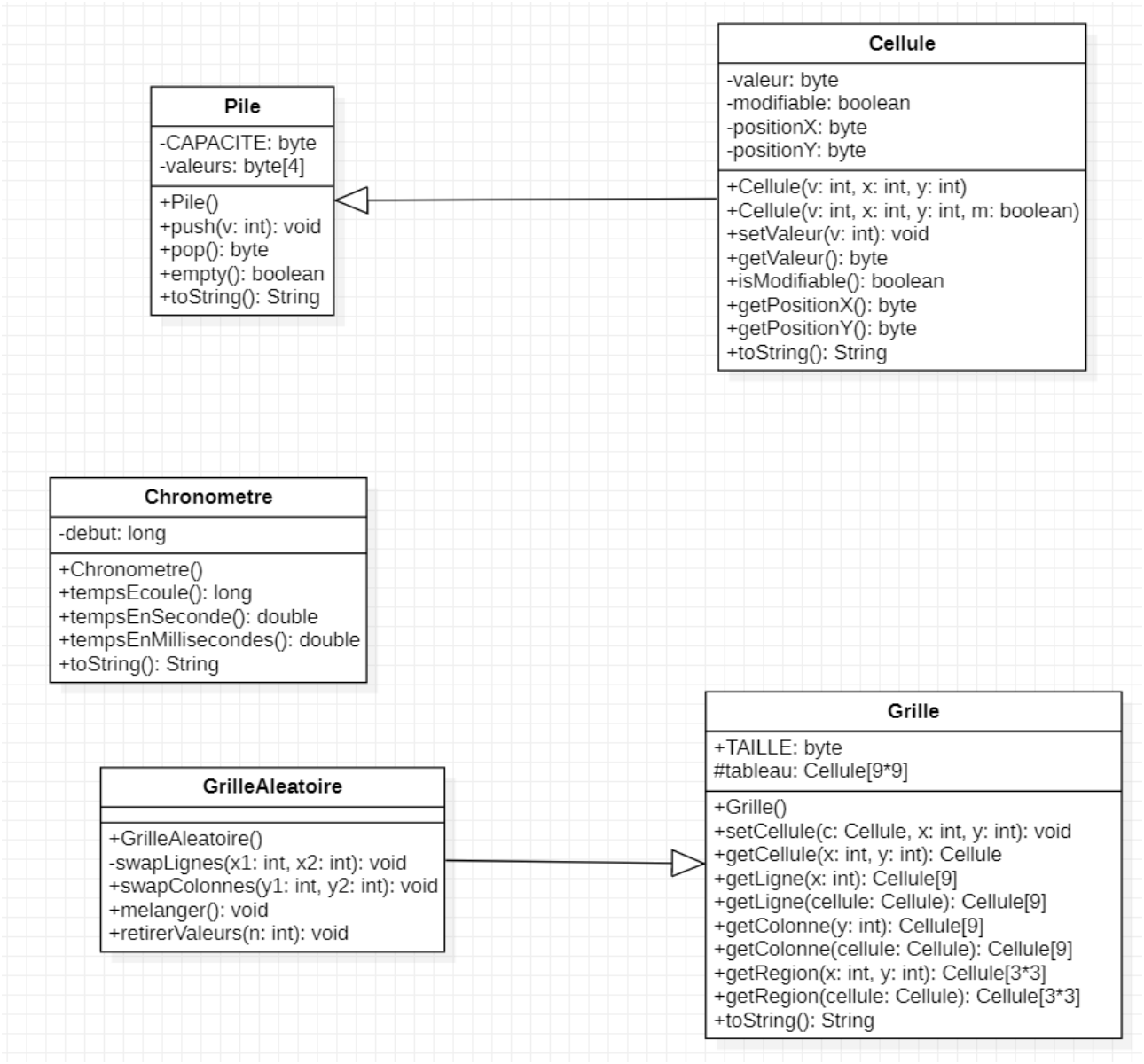
Nous avons aussi conçu un éditeur de grille, dans lequel le joueur pourra charger et modifier des grilles existantes ou créer une nouvelle grille.

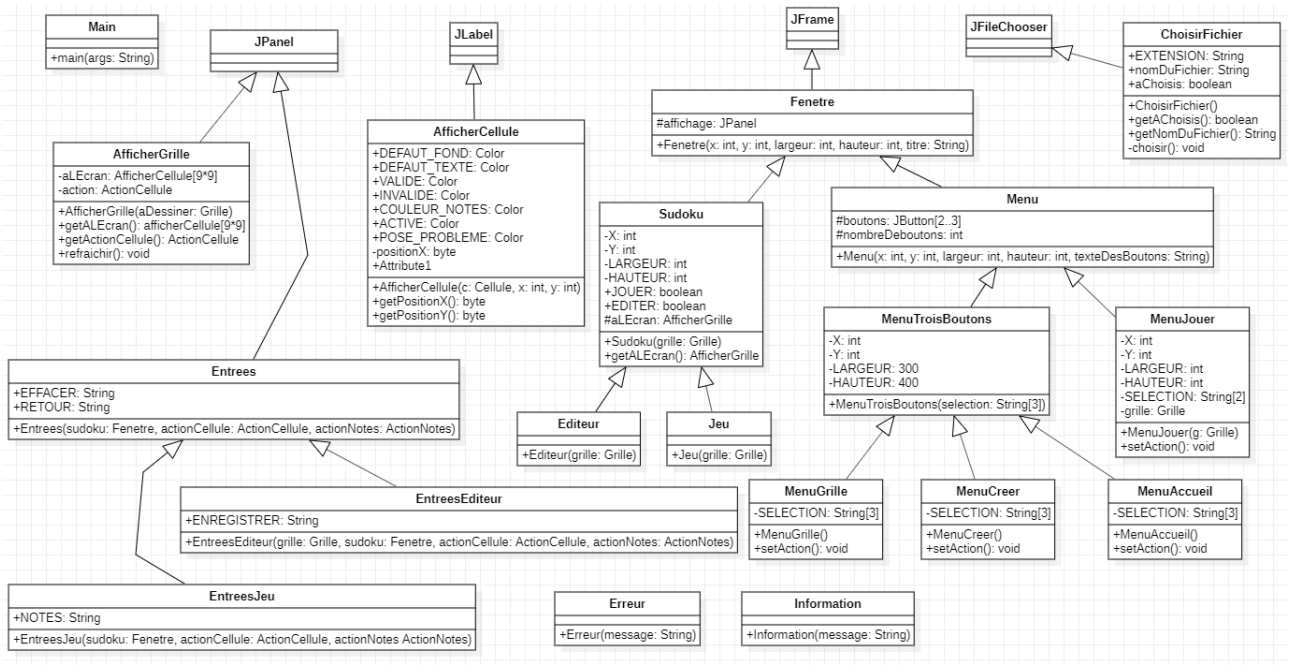
L'édition est similaire au mode de jeu, à la différence que le joueur peut effacer ou modifier les valeurs déjà présentes dans la grille chargée. Un nouveau bouton est venu remplacer celui des notes dans l'interface à droite : c'est le bouton d'enregistrement de la grille. À tout moment, si l'utilisateur est satisfait, il peut enregistrer sa grille dans un fichier à l'extension *.gri*.

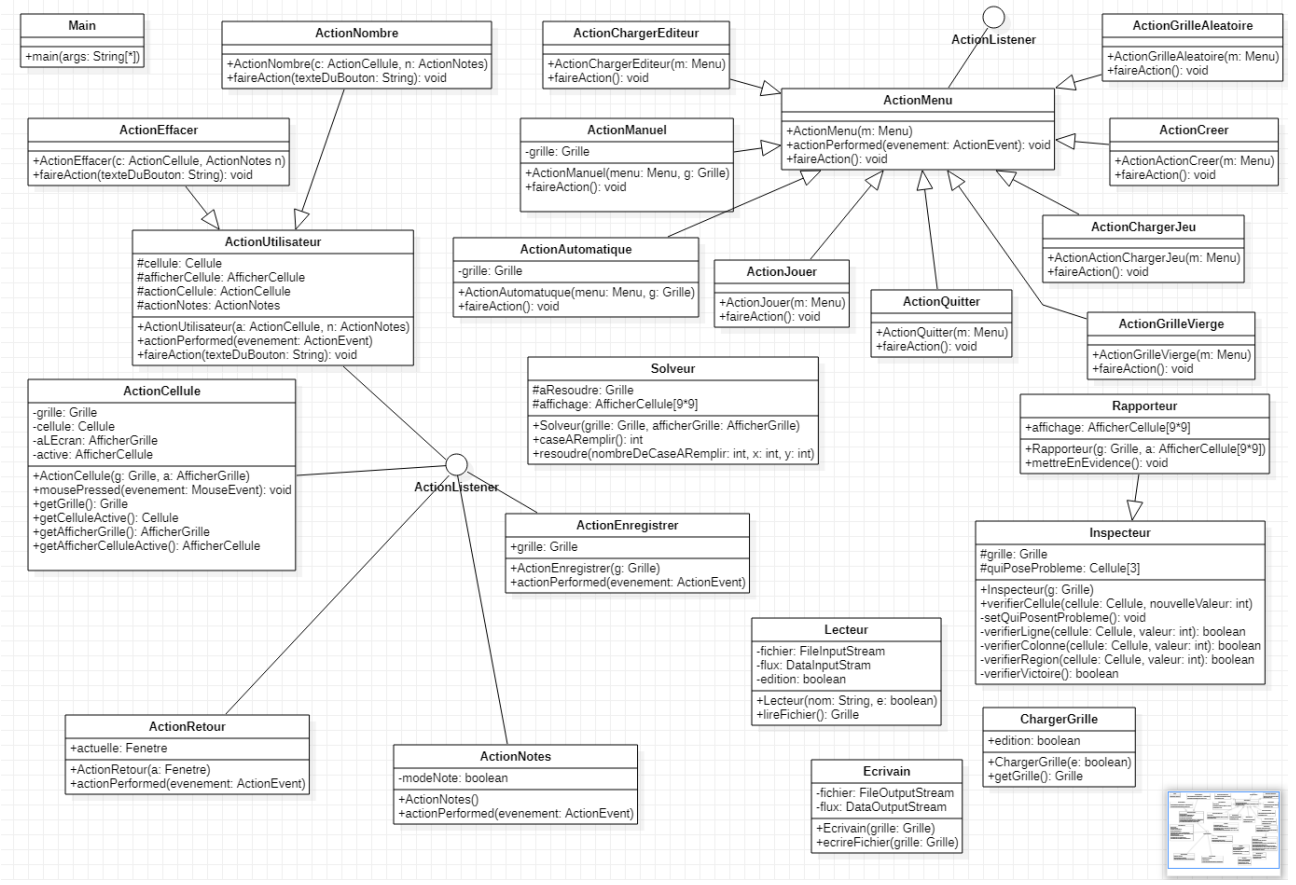
La vérification des coups est toujours présente, afin d'éviter de créer une grille illégale. Toutefois, l'utilisateur peut créer une grille qui ne possède pas de solution.

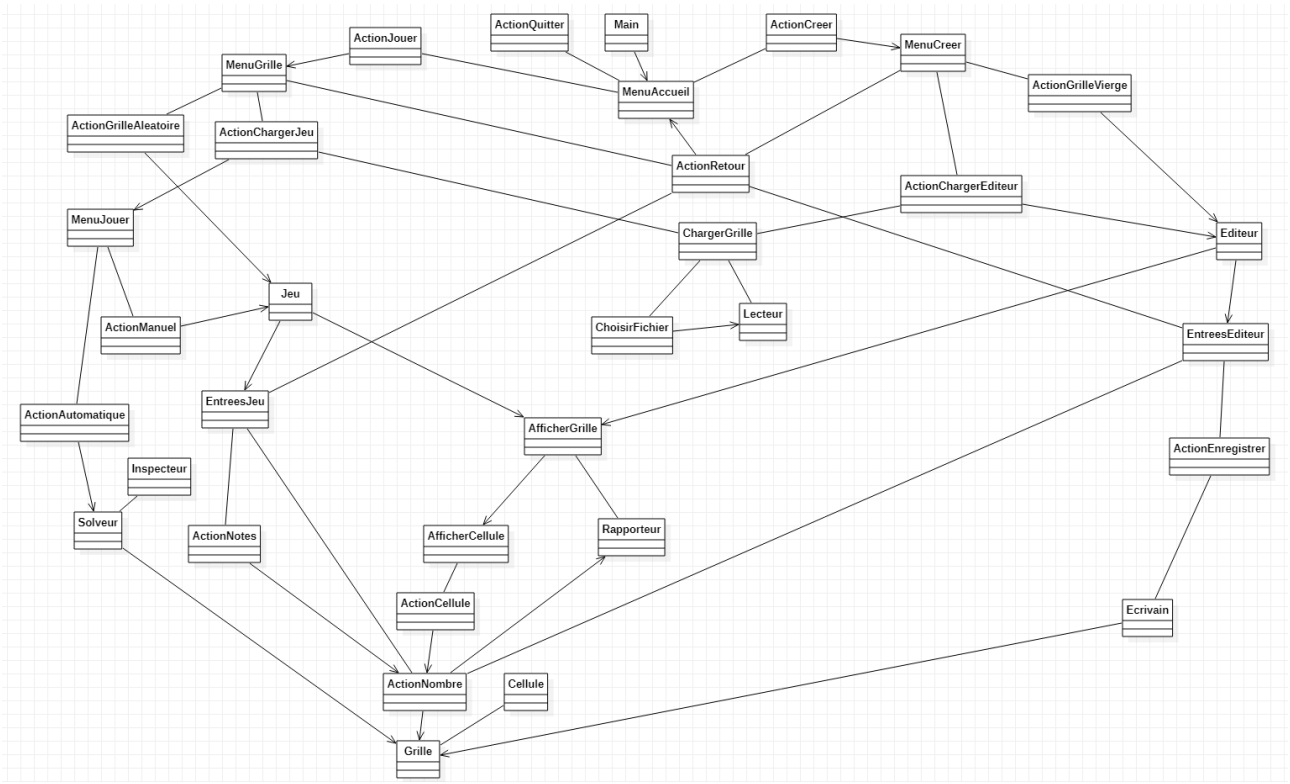
Organisation du code :

Le programme est divisé en 3 catégories, suivant le MVC. Les informations sont donc stockées dans des modèles, affichées par des vues que les contrôleurs gèrent.
Voici les diagrammes représentant la structure de notre projet :




Main
+main(args: String)
JPanel
JLabel
JFrame
JFileChooser
ChoisirFichier
+EXTENSION: String
+nomDuFichier: String
+aChoisis: boolean
+ChoisirFichier()
+getAChoisis(): boolean
+getNomDuFichier(): String
-choisir(): void
AfficherGrille
-aLEcran: AfficherCellule[9*9]
-action: ActionCellule
+AfficherGrille(aDessiner: Grille)
+getALEcran(): afficherCellule[9*9]
+getActionCellule(): ActionCellule
+refraichir(): void
AfficherCellule
+DEFAUT_FOND: Color
+DEFAUT_TEXTE: Color
+VALIDE: Color
+INVALIDE: Color
+COULEUR_NOTES: Color
+ACTIVE: Color
+POSE_PROBLEME: Color
-positionX: byte
+Attribute1
+AfficherCellule(c: Cellule, x: int, y: int)
+getPositionX(): byte
+getPositionY(): byte
Fenetre
#affichage: JPanel
+Fenetre(x: int, y: int, largeur: int, hauteur: int, titre: String)
Sudoku
-X: int
-Y: int
-LARGEUR: int
-HAUTEUR: int
+JOUER: boolean
+EDITER: boolean
#aLEcran: AfficherGrille
+Sudoku(grille: Grille)
+getALEcran(): AfficherGrille
Menu
#boutons: JButton[2..3]
#nombreDeboutons: int
+Menu(x: int, y: int, largeur: int, hauteur: int, texteDesBoutons: String)
MenuTroisBoutons
-X: int
-Y: int
-LARGEUR: 300
-HAUTEUR: 400
+MenuTroisBoutons(selection: String[3])
MenuJouer
-X: int
-Y: int
-LARGEUR: int
-HAUTEUR: int
-SELECTION: String[2]
-grille: Grille
+MenuJouer(g: Grille)
+setAction(): void
Entrees
+EFFACER: String
+RETOUR: String
+Entrees(sudoku: Fenetre, actionCellule: ActionCellule, actionNotes: ActionNotes)
Editeur
+Editeur(grille: Grille)
Jeu
+Jeu(grille: Grille)
EntreesEditeur
+ENREGISTRER: String
+EntreesEditeur(grille: Grille, sudoku: Fenetre, actionCellule: ActionCellule, actionNotes: ActionNotes)
MenuGrille
-SELECTION: String[3]
+MenuGrille()
+setAction(): void
MenuCreer
-SELECTION: String[3]
+MenuCreer()
+setAction(): void
MenuAccueil
-SELECTION: String[3]
+MenuAccueil()
+setAction(): void
EntreesJeu
+NOTES: String
+EntreesJeu(sudoku: Fenetre, actionCellule: ActionCellule, actionNotes ActionNotes)
Erreur
+Erreur(message: String)
Information
+Information(message: String)


Main
+main(args: String[*])
ActionNombre
+ActionNombre(c: ActionCellule, n: ActionNotes)
+faireAction(texteDuBouton: String): void
ActionChargerEditeur
+ActionChargerEditeur(m: Menu)
+faireAction(): void
ActionListener
ActionGrilleAleatoire
+ActionGrilleAleatoire(m: Menu)
+faireAction(): void
ActionEffacer
+ActionEffacer(c: ActionCellule, ActionNotes n)
+faireAction(texteDuBouton: String): void
ActionManuel
-grille: Grille
+ActionManuel(menu: Menu, g: Grille)
+faireAction(): void
ActionMenu
+ActionMenu(m: Menu)
+actionPerformed(evenement: ActionEvent): void
+faireAction(): void
ActionCreer
+ActionActionCreer(m: Menu)
+faireAction(): void
ActionUtilisateur
#cellule: Cellule
#afficherCellule: AfficherCellule
#actionCellule: ActionCellule
#actionNotes: ActionNotes
+ActionUtilisateur(a: ActionCellule, n: ActionNotes)
+actionPerformed(evenement: ActionEvent)
+faireAction(texteDuBouton: String): void
ActionAutomatique
-grille: Grille
+ActionAutomatuque(menu: Menu, g: Grille)
+faireAction(): void
ActionJouer
+ActionJouer(m: Menu)
+faireAction(): void
ActionQuitter
+ActionQuitter(m: Menu)
+faireAction(): void
ActionChargerJeu
+ActionActionChargerJeu(m: Menu)
+faireAction(): void
ActionGrilleVierge
+ActionGrilleVierge(m: Menu)
+faireAction(): void
Solveur
#aResoudre: Grille
#affichage: AfficherCellule[9*9]
+Solveur(grille: Grille, afficherGrille: AfficherGrille)
+caseARemplir(): int
+resoudre(nombreDeCaseARemplir: int, x: int, y: int)
ActionCellule
-grille: Grille
-cellule: Cellule
-aLEcran: AfficherGrille
-active: AfficherCellule
+ActionCellule(g: Grille, a: AfficherGrille)
+mousePressed(evenement: MouseEvent): void
+getGrille(): Grille
+getCelluleActive(): Cellule
+getAfficherGrille(): AfficherGrille
+getAfficherCelluleActive(): AfficherCellule
ActionListener
Rapporteur
+affichage: AfficherCellule[9*9]
+Rapporteur(g: Grille, a: AfficherCellule[9*9])
+mettreEnEvidence(): void
ActionEnregistrer
+grille: Grille
+ActionEnregistrer(g: Grille)
+actionPerformed(evenement: ActionEvent)
Inspecteur
#grille: Grille
#quiPoseProbleme: Cellule[3]
+Inspecteur(g: Grille)
+verifierCellule(cellule: Cellule, nouvelleValeur: int)
-setQuiPosentProbleme(): void
-verifierLigne(cellule: Cellule, valeur: int): boolean
-verifierColonne(cellule: Cellule, valeur: int): boolean
-verifierRegion(cellule: Cellule, valeur: int): boolean
-verifierVictoire(): boolean
Lecteur
-fichier: FileInputStream
-flux: DataInputStram
-edition: boolean
+Lecteur(nom: String, e: boolean)
+lireFichier(): Grille
ChargerGrille
+edition: boolean
+ChargerGrille(e: boolean)
+getGrille(): Grille
ActionRetour
+actuelle: Fenetre
+ActionRetour(a: Fenetre)
+actionPerformed(evenement: ActionEvent)
ActionNotes
-modeNote: boolean
+ActionNotes()
+actionPerformed(evenement: ActionEvent)
Ecrivain
-fichier: FileOutputStream
-flux: DataOutputStream
+Ecrivain(grille: Grille)
+ecrireFichier(grille: Grille)



Ainsi, dans les modèles on retrouve les éléments essentiels : la Cellule, qui correspond à une case de la grille, la grille, qui correspond à un tableau de 9x9 cellules, on retrouve aussi une pile, qui permet à la cellule de stocker 4 valeurs temporaires. Ces modèles sont utilisés par les contrôleurs et les vues qui leurs sont associés (pour la cellule, le controleur est ActionCellule, et la vue est AfficherCellule).

Dans notre code, les classes dont le nom commence pas ‘Action’ sont souvent associées à un ActionListener ou un MouseListener. Il s’agit des contrôleurs qui servent à l’interaction entre la grille et l’utilisateur.

Les menus occupent une place importante de nos vues, et leurs boutons ont des contrôleurs dédiés afin de rendre le code plus simple à comprendre au premier coup d’œil. Chaque bouton possède son propre contrôleur, car certains boutons sont présents dans plusieurs menus.

Chaque classe est rangée dans un dossier qui la caractérise (dans le dossier vue, on a un dossier menus, dans contrôleurs, on a aussi on dossier menus, ces deux dossiers contiennent les classes servant à l’affichage et l’utilisation des menus et de leurs boutons), ceci afin de catégoriser plus finement encore notre code et améliorer sa lisibilité.

Parmi les contrôleurs, on retrouve aussi nos fidèles assistants :

- l'Inspecteur sert à vérifier si une valeur pose problème dans une case en fonction de la ligne, de la colonne et de la région, comme énoncé dans les règles plus haut.
- Le Rapporteur se sert du travail de l'Inspecteur afin de rapporter les éventuelles erreurs à l'utilisateur en les mettant en évidence.
- Enfin le Solveur, qui se sert lui aussi de l'Inspecteur afin de résoudre une grille lui même.

Le Solveur se résume par méthode récursive qui se répète jusqu'à avoir rempli la grille.
Premièrement, le solveur compte le nombre de case qu'il doit remplir. Ensuite, il passe sur toutes les cases vides en essayant de mettre une valeur comprise entre 1 et 9, grâce à l'Inspecteur qui lui dit s'il peut ou non mettre la valeur dans la cellule. Dès qu'il y parvient, il passe à la case suivante, et réitère en retirant 1 au nombre de cases à remplir. Dès que ce nombre tombe à 0, alors la méthode retourne 'true' jusqu'à revenir sur la première cellule. À chaque cellule, le statut de retour est 'true' si la valeur peut être mise dans le cellule, 'false' sinon, et le statut de la cellule suivante pour les cellules qui ne sont pas à remplir. À la fin de l'exécution de la méthode récursive, si 'false' est retourné, alors le grille n'est tout simplement pas solvable.

Conclusions :

Adrien COSTANDI : ce projet m'a permis de mieux comprendre le java en général, ce n'est toujours pas parfait mais le flux d'octets, les listeners ou autres contrôleurs ne posent plus de problèmes. Il me reste encore à améliorer la qualité de mon code, qui n'est parfois pas toujours optimale mais donne tout de même généralement un bon résultat. Ce travail d'équipe a été un plaisir et le projet était intéressant ! Le MVC me semble être acquis aussi, ce qui n'a pas tout de suite été évident.
S'il y a une chose que j'aimerai améliorer, c'est la génération aléatoire qui ne me plaît pas encore totalement mais, qui n'était pas demandée. (PS : ce dernier commentaire est obsolète car la modification a été implémentée !)

Antoni BLANCHE : lorsque le sujet du projet est tombé, je n'étais clairement pas confiant. Les attentes me semblaient bien supérieures à mon niveau, et je ne savais pas par où commencer. Au final, tout s'est bien passé, et nous avons rapidement trouvé la démarche à suivre. En nous répartissant le travail, nous avons rapidement eu notre première version du projet. J'ai beaucoup aimé ce projet car j'ai l'impression d'avoir compris bien plus de chose en trois semaines que durant le mois passé sur les TP de Java. Le point le moins évident du projet était sûrement de découper le programme selon le principe du MVC, qui nous a posé quelques problèmes.