

# RAPPORT

## Création jeu du sudoku

### Sommaire :

Partie 1 : Introduction du jeu	p.2
Partie 2 : Description des fonctionnalités	p.3
Programme Concepteur	p.3
Grille de sudoku	p.3
Bouton de la grille	p.4
Gestion des touches	p.5
Gestion des erreurs	p.6
Programme Joueur	p.8
Grille de sudoku	p.8
Bouton de la grille	p.9
Gestion des touches	p.11
Gestion des erreurs	p.12
Partie 3 : Explication de l'algorithme de résolution automatique	p.15
Partie 4 : Découpage des fichiers	p.16
Fichiers sources	p.16
Diagramme du découpage	p.17
Partie 5 : Avis personnel	p.19
Yann	p.19
Thomas	p.19

## Introduction du jeu

Le sudoku est un jeu de logique qui met à l'épreuve vos compétences en résolution de problèmes. Dans ce jeu, on est sur une grille de 9x9 cases, divisée en neuf régions de 3x3 cases. L'objectif de ce jeu est donc de remplir chaque case de la grille avec un chiffre entre 1 et 9, en faisant attention que chaque ligne, chaque colonne et chaque région contiennent tous les chiffres de 1 à 9, sans répétition.

Au début du jeu, on a devant nous une grille partiellement remplie (moins on a de chiffre dans la grille de départ, plus le niveau est difficile). Grâce au chiffre déjà posé, on peut commencer à résoudre la grille, en utilisant l'élimination par élimination, en prenant en compte les chiffres déjà présents dans la grille.

Pour résoudre cette grille, il faut analyser attentivement les chiffres déjà présents, pour savoir quel chiffre mettre dans une case vide. Ce jeu nécessite de la patience et de la stratégie.

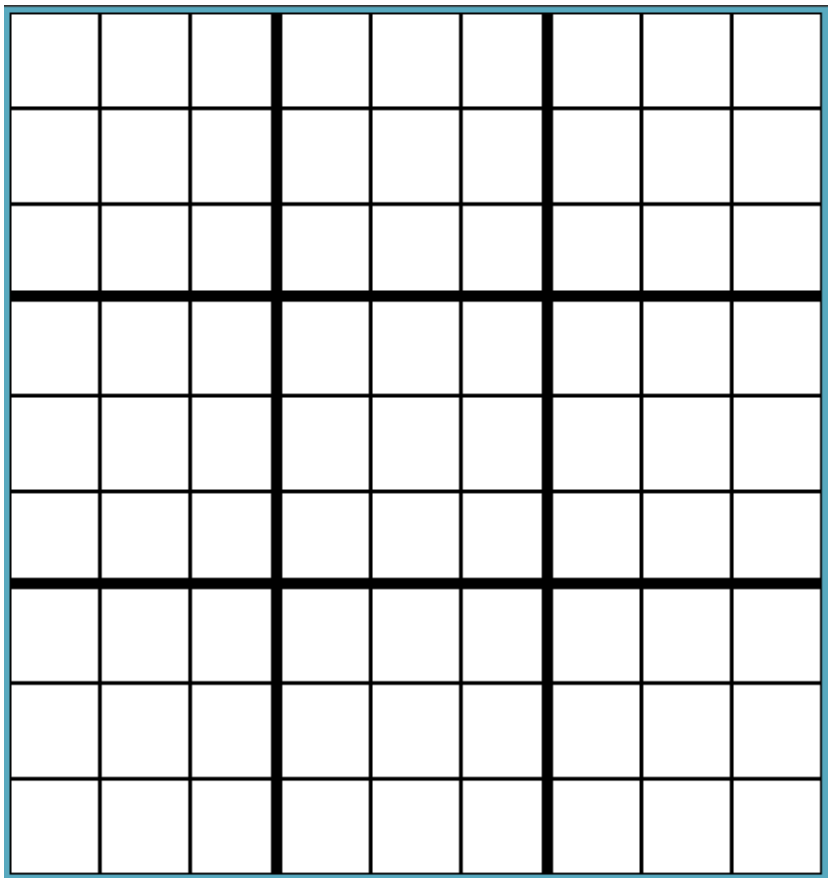
## Descriptions des fonctionnalités

### Le sudoku comporte deux programme :

- Le programme Concepteur, qui va permettre à la conception de la grille de départ  
`make Concepteur`

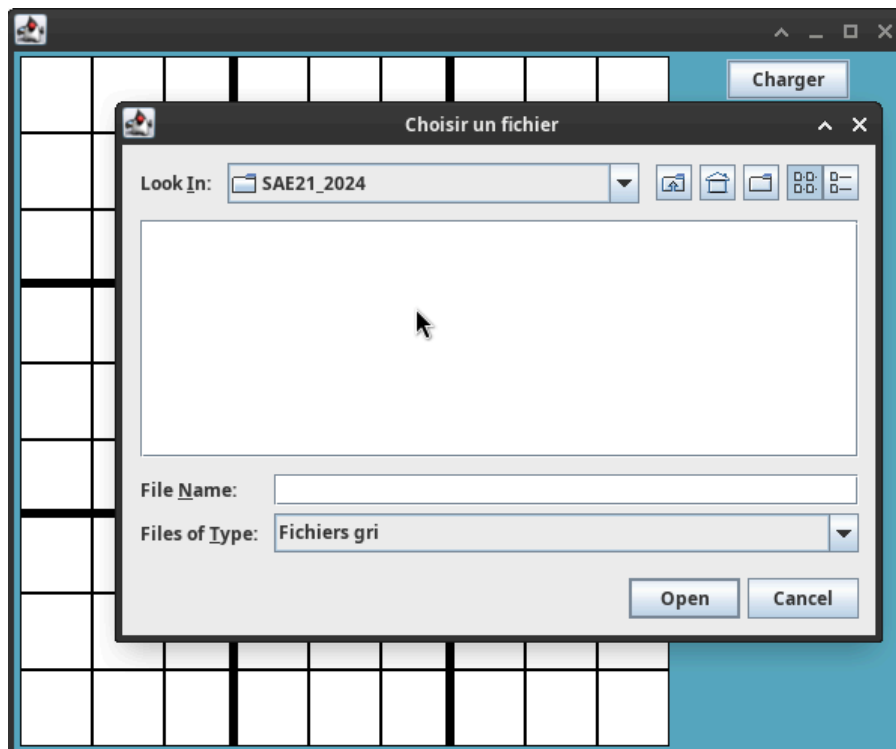
### Grille de sudoku :

Le programme démarre avec une grille de Sudoku au format 9x9 cases, délimitées par des bordures noires. Il y a aussi des régions de 3x3 cases délimitées par des bordures noires plus épaisses. Une case correspond à un JTextField, où l'on ne peut écrire que 1 chiffre **maximum** (de 1 à 9).

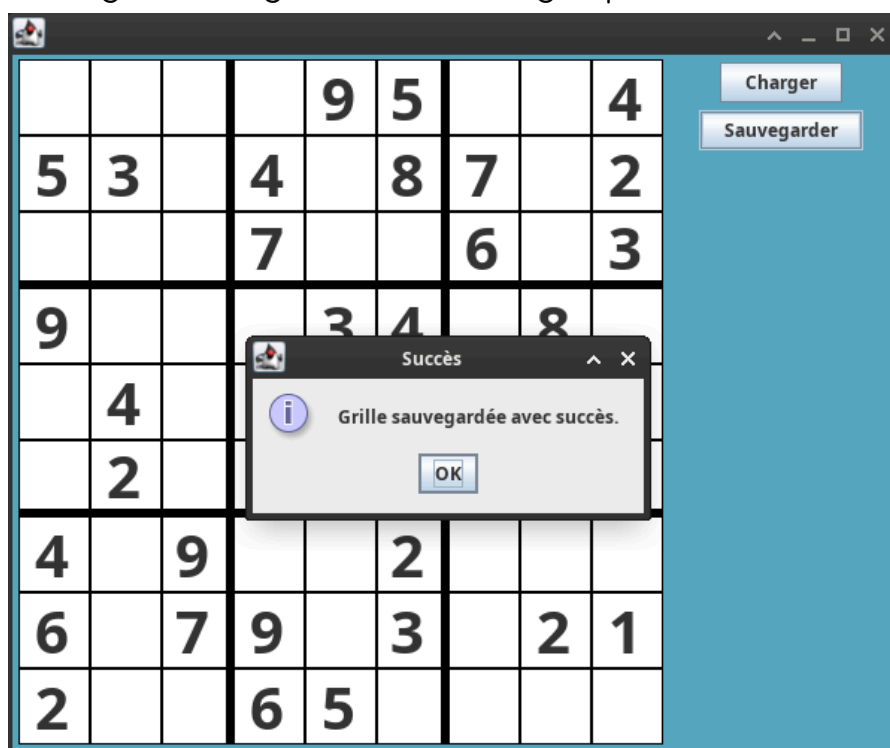


### Bouton de la grille :

La grille est dès le départ vide. Pour le concepteur, libre à lui de créer son propre puzzle (grille) ou de charger une grille existant pour la modifier à sa façon. Un bouton Charger est mis à disposition pour charger une grille (au format .gri).



A la fin de la conception du puzzle, le concepteur doit sauvegarder la grille pour le futur joueur grâce au bouton Sauvegarder, qui va sauvegarder la grille au format .gri qui se nomme GrilleNum1.gri.

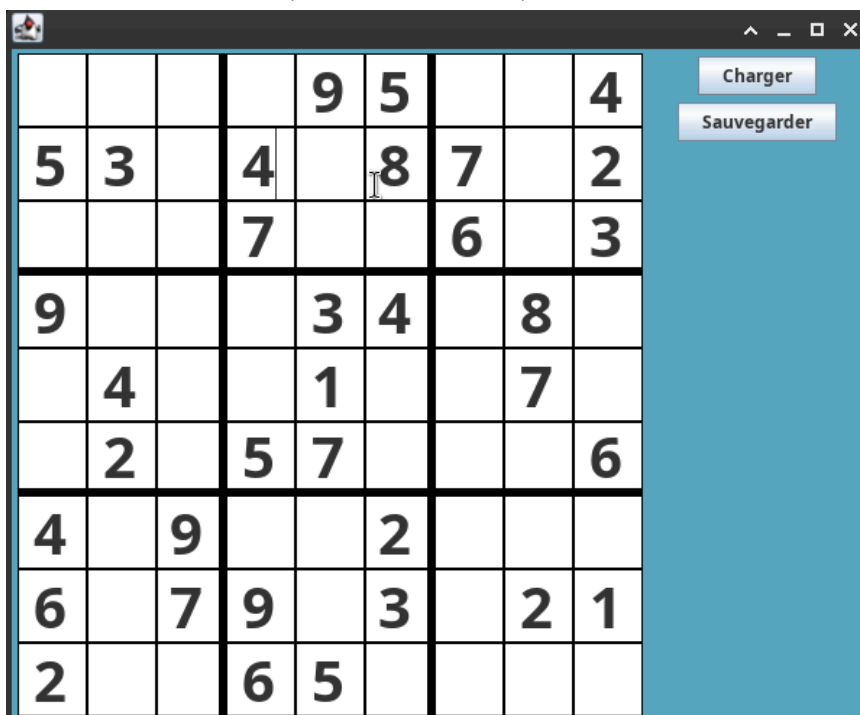


### Gestion des touches :

L'utilisateur peut utiliser le pavé numérique pour pouvoir écrire dans la grille de sudoku.



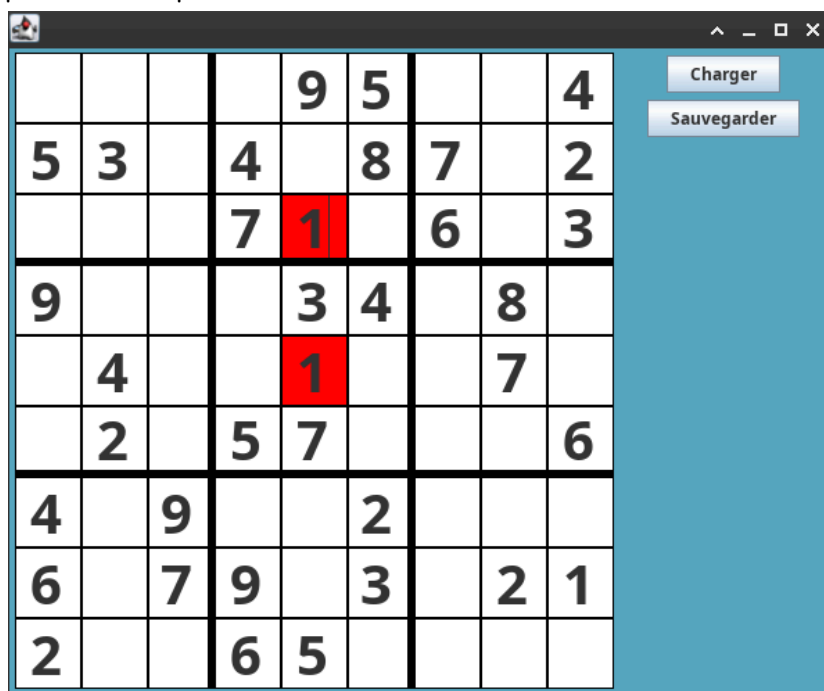
Le programme permet de bloquer le champ de saisie pour que l'on puisse écrire que des chiffres de 1 à 9. Une case peut comporter qu'un chiffre maximum pour la conception de celle-ci.



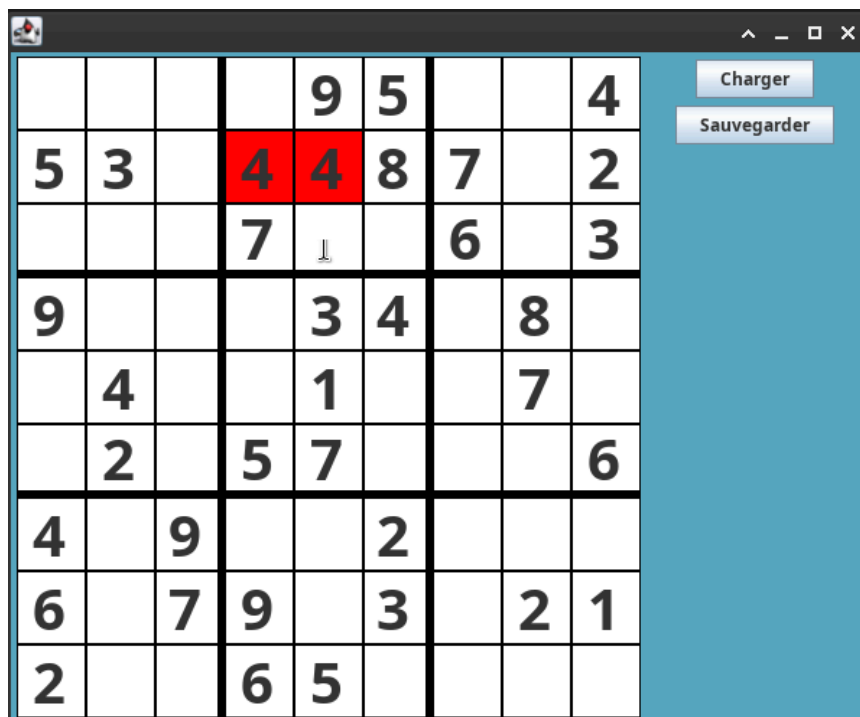
### Gestions des erreurs :

Dans ce programme, la détection des erreurs fonctionne grâce à la grille de sudoku qui est représentée en un tableau à double entrée. La vérification s'effectue

Le programme vérifie d'abord les **lignes et les colonnes**. Cette méthode vérifie si le chiffre nouvellement inséré est déjà présent dans la même ligne ou colonne. Si c'est le cas, le programme colorie les cases en rouge pour marquer l'erreur.

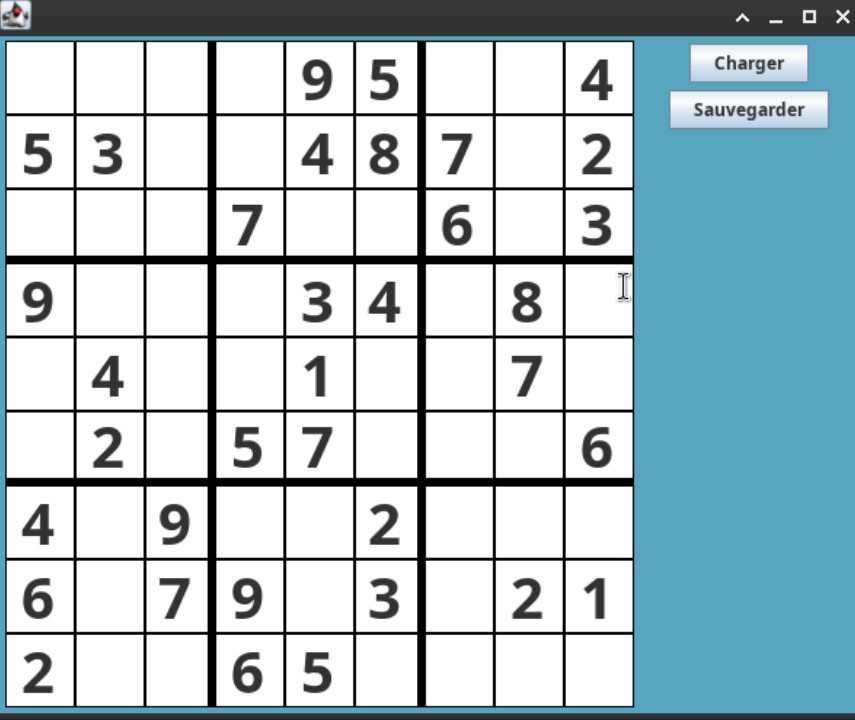


Ensuite, le programme vérifie si le chiffre est également présent dans la même **région**. Si c'est le cas, le programme colorie les cases en rouge pour marquer les erreurs.



La méthode n'arrête pas l'exécution du programme, mais empêche un placement **contradictoire**. Cela permet au concepteur de pouvoir corriger son erreur.

Pour enlever l'erreur, il faut juste **enlever** le chiffre qui a causé l'erreur.



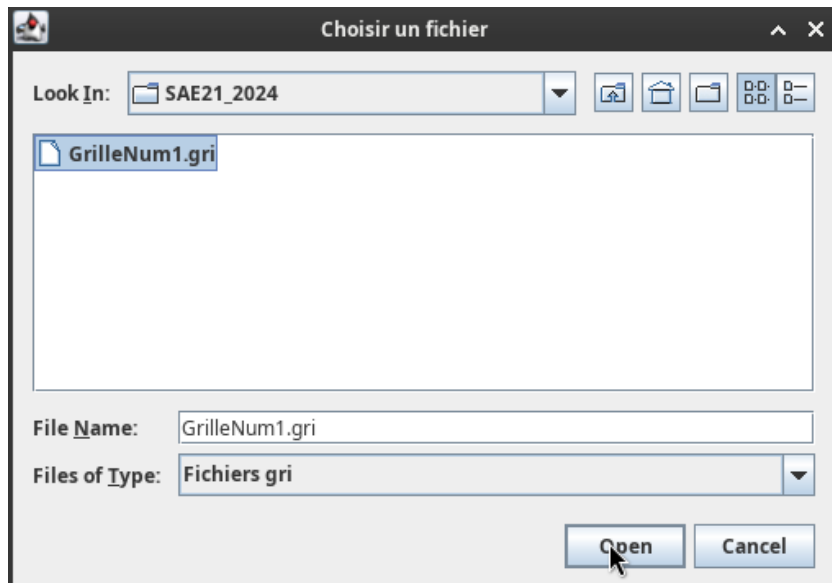
				9	5			4
5	3			4	8	7		2
			7			6		3
9				3	4		8	
	4			1			7	
	2		5	7				6
4		9			2			
6		7	9		3		2	1
2			6	5				

- Le programme Joueur, qui va permettre à un joueur de résoudre la grille.

make Joueur

### Grille de Sudoku :

Le programme commence par le lancement d'un explorateur de fichier, où on choisit un fichier .gri. Ici, il faut choisir le fichier du concepteur, qui se nomme GrilleNum1.gri.



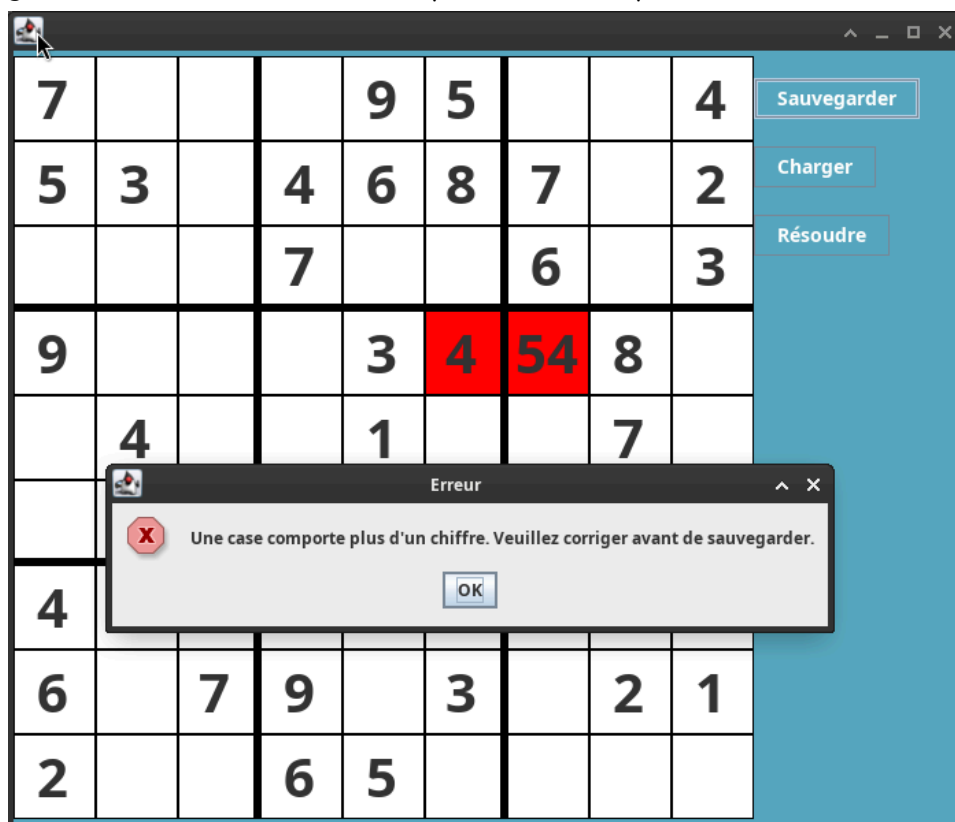
Une fois le fichier choisi, on a donc la grille de sudoku partiellement remplie en fonction du puzzle que le concepteur a choisi. Comme pour le premier programme, la grille est au format **9x9 cases**, délimitées par des bordures noires. Il y a aussi des régions de **3x3 cases** délimitées par des bordures noires plus épaisses. La grille qu'on a chargée n'est pas modifiable.

		I		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				

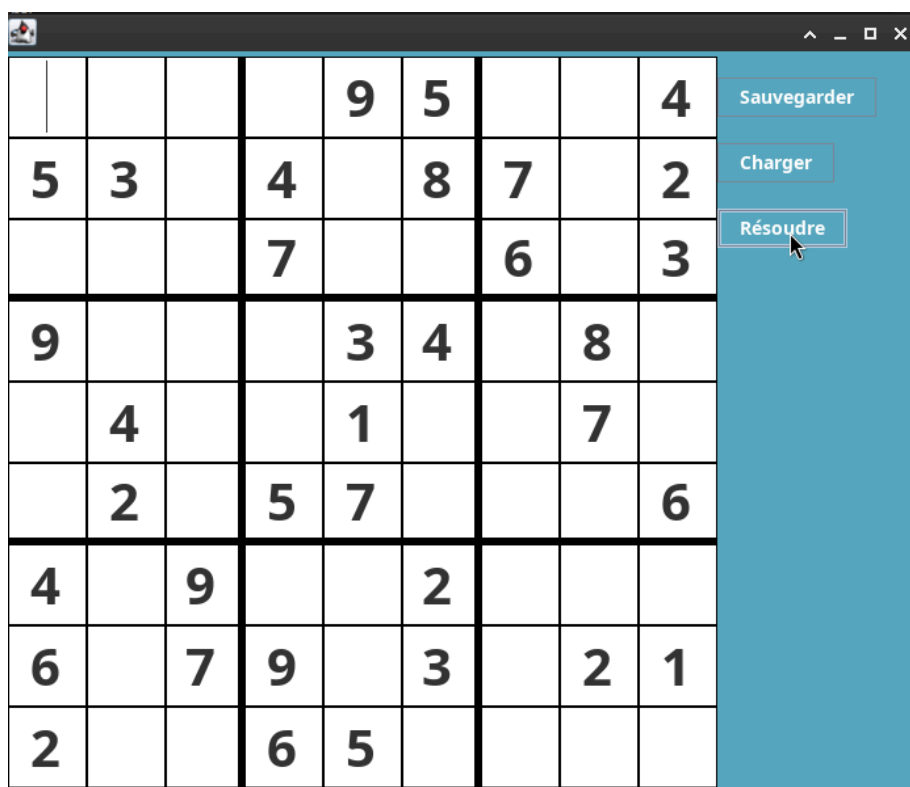


### Bouton de la grille :

La grille contient un bouton **Sauvegarder** pour sauvegarder notre grille si on décide de reprendre le jeu un autre moment. Pour sauvegarder la grille, nous devons avoir qu'un chiffre par case.

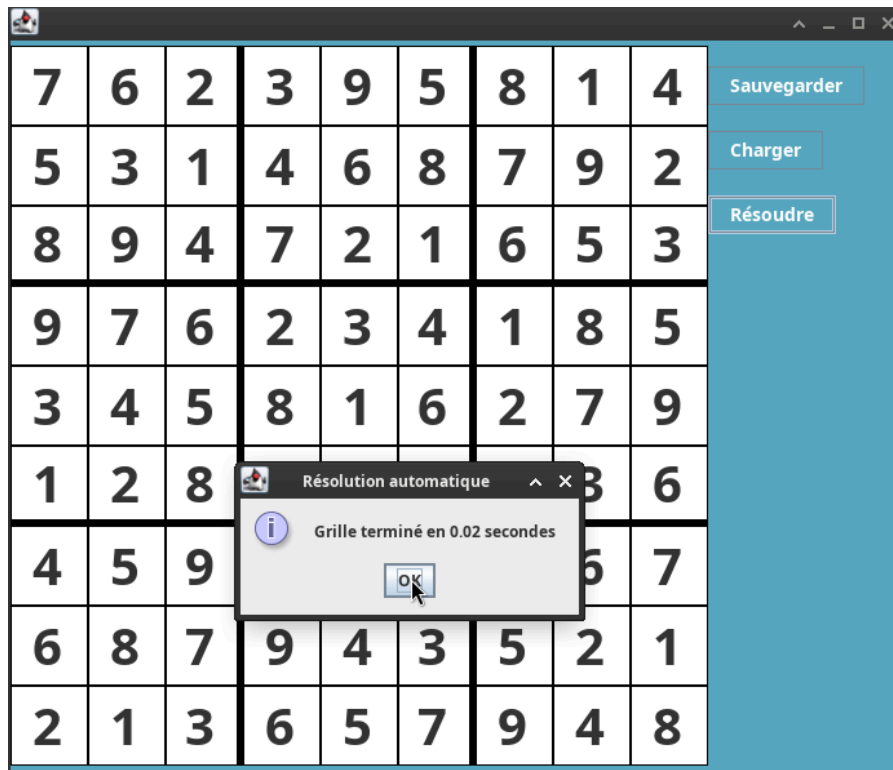


Elle contient aussi un bouton **Charger** pour charger un autre puzzle de grille ou charger votre progression, si auparavant, on a sauvegardé notre grille incomplète (pour la finir plus tard par exemple).  
Un troisième bouton apparaît. C'est le bouton **Résoudre**.



Un redesign des boutons a été effectué pour le joueur.

Le bouton Résoudre permet, si on le souhaite, de résoudre automatiquement la grille de départ, fourni par le concepteur. La grille devient toute remplie et affiche le temps de résolution nécessaire.



### Gestion des touches :

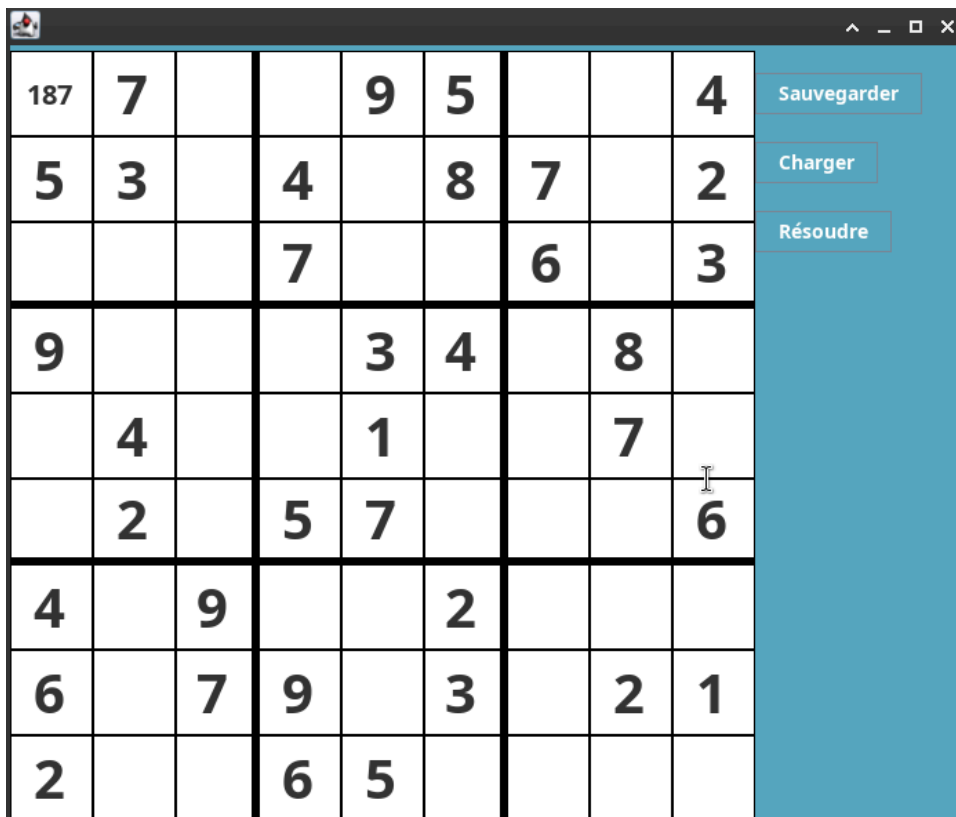
Comme pour le premier programme, l'utilisateur peut utiliser le pavé numérique pour pouvoir écrire dans la grille de sudoku.



Le programme permet de bloquer le champ de saisie pour que l'on puisse écrire que des chiffres de 1 à 9. Une case peut comporter jusqu'à 4 chiffres maximum par cases.

### Gestions des erreurs :

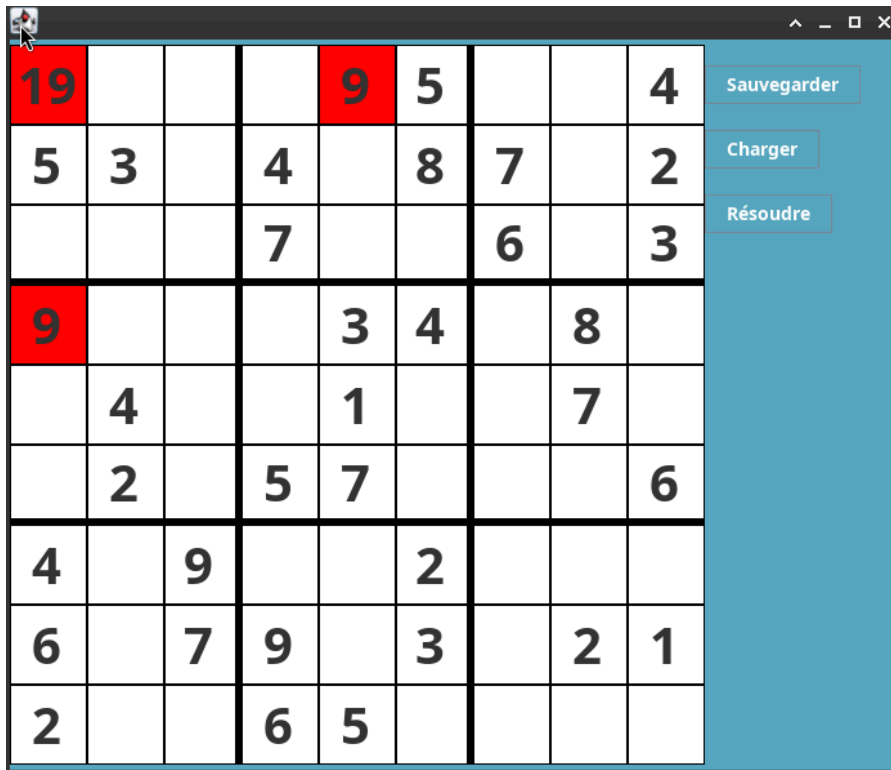
Comme pour le premier programme, la détection des erreurs fonctionne grâce à la grille de sudoku qui est représentée en un tableau à double entrée. La vérification s'effectue. La différence par rapport au premier programme est qu'on peut avoir **plusieurs chiffres** dans une case (4 maximum). Lorsqu'on n'a qu'un seul chiffre dans une case, le chiffre est **limité** en fonction des autres cases et elle **contraint** aussi les autres cases. Lorsqu'on en a plus qu'un chiffre, les chiffres sont eux aussi **limités** en fonction des autres cases mais **ne contraignent pas** les autres cases. Pour finir le sudoku, nous devons avoir qu'un **chiffre maximum** par case.



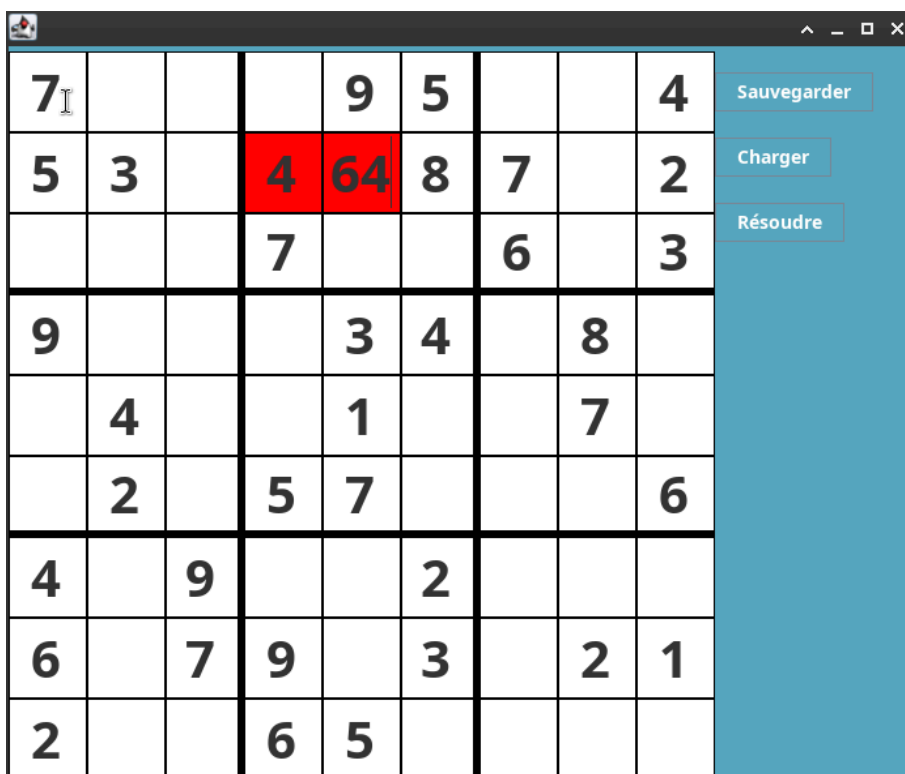
The image shows a web-based Sudoku application. On the left is a 9x9 grid with some numbers filled in. On the right is a sidebar with three buttons: 'Sauvegarder', 'Charger', and 'Résoudre'. The grid is as follows:

187	7			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				

Le programme vérifie d'abord les **lignes et les colonnes**. Cette méthode vérifie si le chiffre nouvellement inséré est déjà présent dans la même ligne ou colonne. Si c'est le cas, le programme colorie les cases en rouge pour marquer l'erreur.

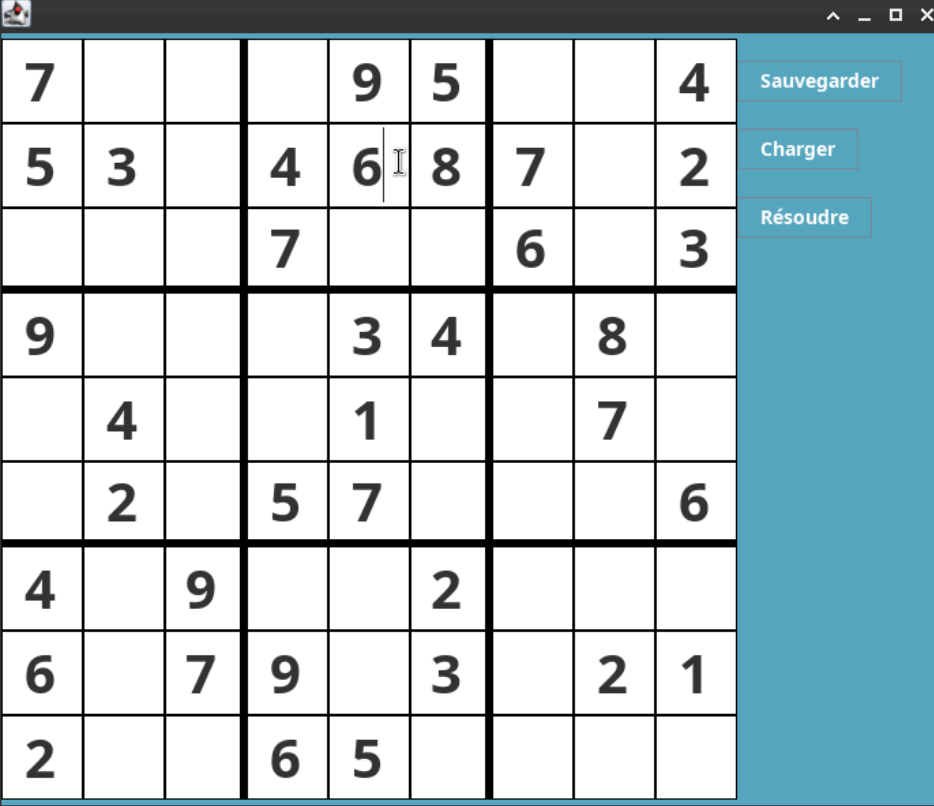


Ensuite, le programme vérifie si le chiffre est également présent dans la même **région**. Si c'est le cas, le programme colorie les cases en rouge pour marquer les erreurs.



La méthode n'arrête pas l'exécution du programme, mais empêche un placement **contradictoire**. Cela permet au concepteur de pouvoir corriger son erreur.

Pour enlever l'erreur, il faut juste **enlever** le chiffre qui a causé l'erreur.



7				9	5			4
5	3		4	6	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				

Sauvegarder

Charger

Résoudre

## Explication de l'algorithme de résolution automatique

Pour effectuer la résolution automatique, le programme s'appuie sur différents points :

- **Initialisation** : La classe '**Resolver**' est utilisée pour résoudre automatiquement une grille de Sudoku. Lorsque le bouton est pressé, l'action exécute la méthode '**resolution()**'.
- **Validation de cases vides** : La méthode '**resolution()**' commence par parcourir la grille pour trouver les cases vides. Si une case est vide, elle stocke la case vide dans une liste nommée '**case\_vide**'.
- **Recherche des chiffres possibles** : Pour chaque case vide, la méthode détermine les chiffres possibles qui peuvent être placés dans cette case sans violer les règles d'unicité. Elle recherche les chiffres déjà présents dans la même ligne, colonne et région, et élimine ces chiffres de la liste de chiffres possibles.
- **Placement des chiffres** : Si la liste des chiffres possibles pour une case vide ne contient qu'un seul chiffre, la méthode le place dans cette case.
- **Répétition** : Ce processeur est répété jusqu'à ce qu'il n'y ait plus de cases vides ou qu'aucun chiffre ne puisse être placé dans les cases vides restant sans violer les règles d'unicité du sudoku.
- **Affichage** : Une fois que toute la grille a été résolue grâce au resolver, qu'aucune autre action est possible, l'algorithme affiche un message de succès, indiquant le temps qu'il a fallu pour résoudre la grille en automatique.

Cet algorithme ne génère pas d'erreurs étant donné qu'il doit la résoudre. Pour la résolution automatique, la grille ne doit pas contenir d'erreur et chaque chiffre dans une case doit être valide et non en cohabitation.

## Découpage des fichiers

Les deux programmes ont été décomposés en plusieurs fichiers afin de rendre le code plus lisible et facile à maintenir. Chaque fichier source a une responsabilité spécifique, contribuant ainsi à une organisation efficace du code.

### Fichiers sources :

#### Joueur.java et Concepteur.java

Ces fichiers sont les points de départ de leur programme. Ils permettent d'appeler les classes qui gèrent l'affichage et les fonctionnalités. Ils contiennent tous les deux la méthode `main()` ; .

#### SudokuGridJoueur.java et SudokuGridConcepteur.java

Ces deux fichiers servent à générer la fenêtre avec la grille du sudoku représenté en `JTextField` pour pouvoir y insérer des chiffres, ils génèrent aussi les boutons cliquables pour accéder à des fonctionnalités. Ces deux fichiers sont très similaires, ils diffèrent uniquement dans le nombre de boutons présents dans la fenêtre.

#### SaveButton.java

Ce fichier sert à sauvegarder la grille affichée à l'écran si elle respecte les règles d'unicité.

#### LoadButton.java

Ce fichier sert à charger une grille de sudoku modifiable ou non modifiable en fonction du programme lancé. Il affiche la grille de sudoku dans notre grille.

#### TextFilter.java

Ce fichier sert à filtrer le texte entré dans les cases du sudoku, il réalise aussi des tests sur l'entrée des chiffres pour vérifier son bon positionnement. Ce fichier s'adapte si il est appelé par le programme joueur ou le programme concepteur.

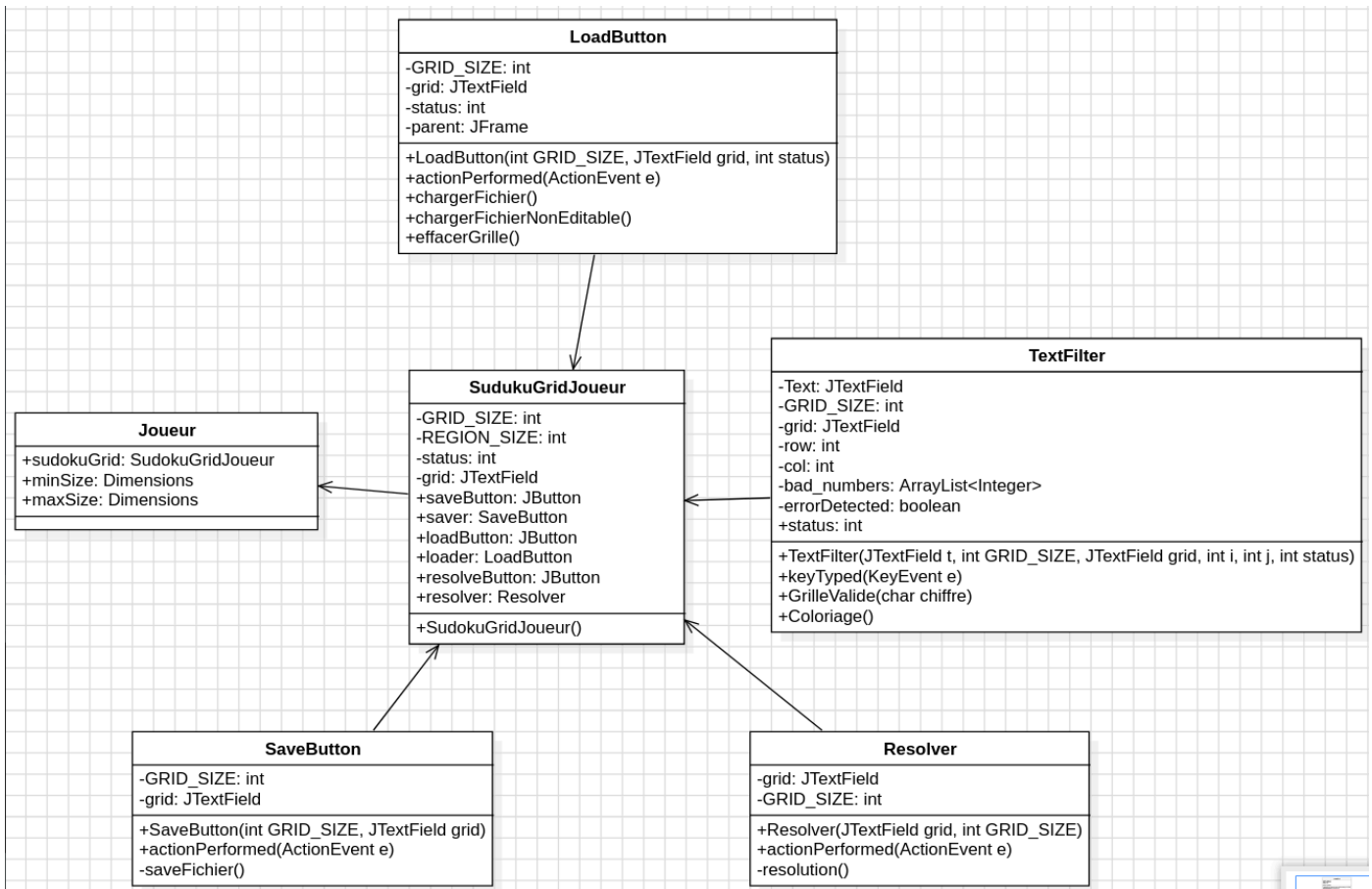
#### Resolver.java

Ce fichier sert à résoudre une grille de sudoku chargée précédemment par l'ordinateur, à la fin il affiche le temps nécessaire à la résolution.

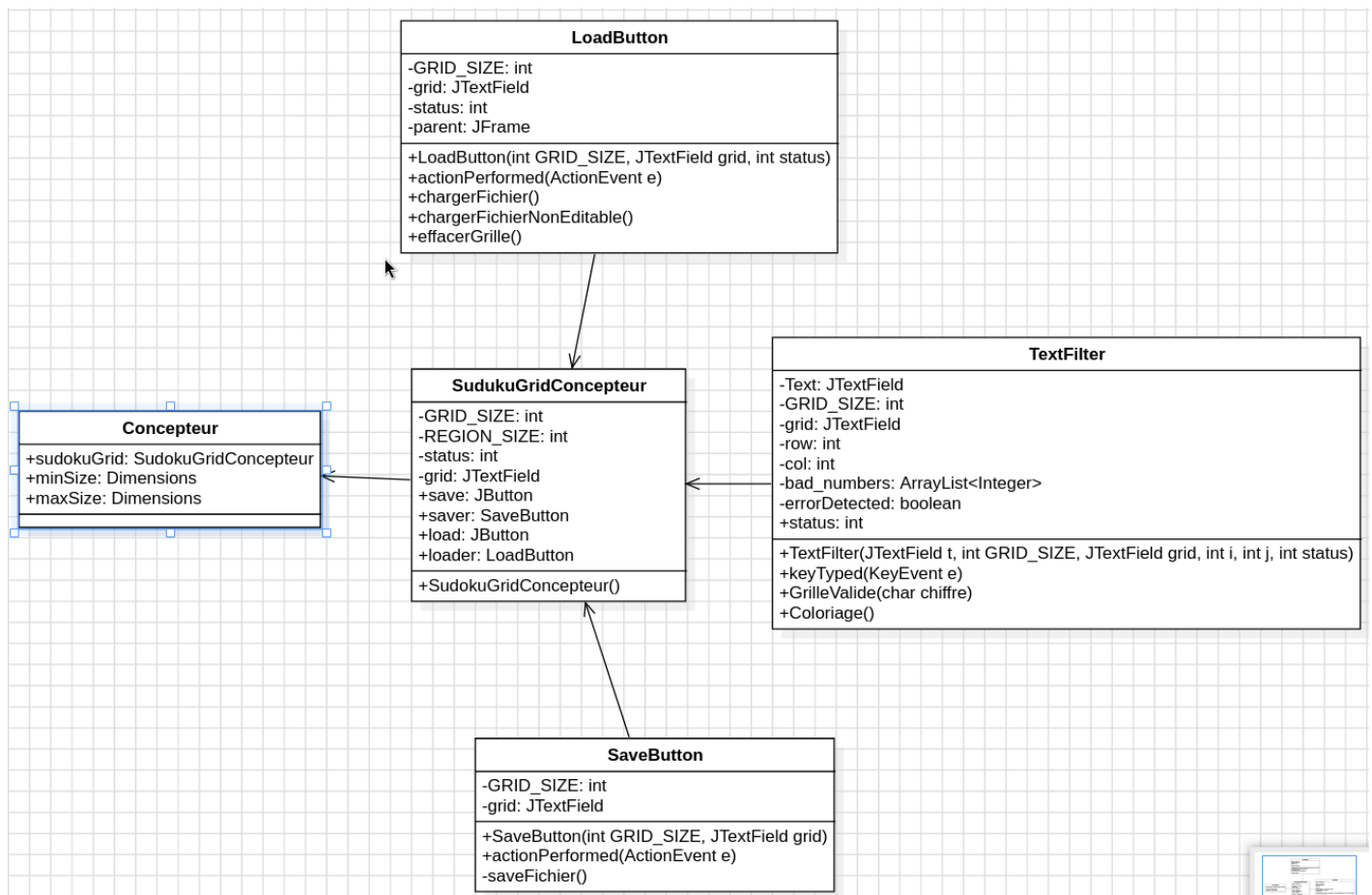


## Diagramme du découpage :

### Premier programme :



## Deuxième programme :



## Avis personnel

### Yann :

Ce projet informatique a été très intéressant, j'ai pu me rendre compte de la puissance d'un langage de programmation orienté objet et la facilité avec laquelle on peut créer des programmes plus harmonieux, plus ergonomiques.

J'ai pu me familiariser avec l'API java qui fait gagner beaucoup de temps.

Pour revenir au projet, à première vue il ne semble pas si compliqué puis quand on commence à se plonger dedans on découvre des aspects assez complexes. Comme le filtrage des caractères qu'on peut rentrer dans les cases de la grille. C'est ce qui a été pour moi la partie la plus compliquée de ce projet.

### Thomas :

J'ai beaucoup aimé faire ce projet. J'ai eu du mal à comprendre au début le sujet, étant donné que je ne connaissais pas du tout les règles du sudoku, et que je n'avais jamais entendu parler de ce jeu (contrairement à la première SAE, où je connaissais déjà le jeu du snake). Le projet a été plus compliqué que ce que je pensais, mais tout de même raisonnable.

Je pensais que cela allait prendre plus de temps, mais étant donné que mon camarade et moi avons bien réparti nos tâches et qu'on a eu une bonne communication entre nous, le projet a été effectué plus facilement. On s'est réparti les tâches entre nous, ce qui m'a permis de me concentrer sur plus de choses ciblées que d'autres. Si j'avais un problème sur une tâche, mon camarade m'aidait (et vice-versa).

J'ai beaucoup aimé le fait d'avoir deux programmes (concepteur et joueur), ce qui permet, pour moi, de voir dans la création d'un jeu les deux côtés.

J'ai tout de même préféré la première SAE comme j'aimais déjà bien le jeu du snake, même si je commence à apprécier le sudoku.

La fabrication du jeu montre l'important impact que l'utilisateur peut avoir sur le jeu.

Le projet a enrichi mes connaissances dans le langage Java.