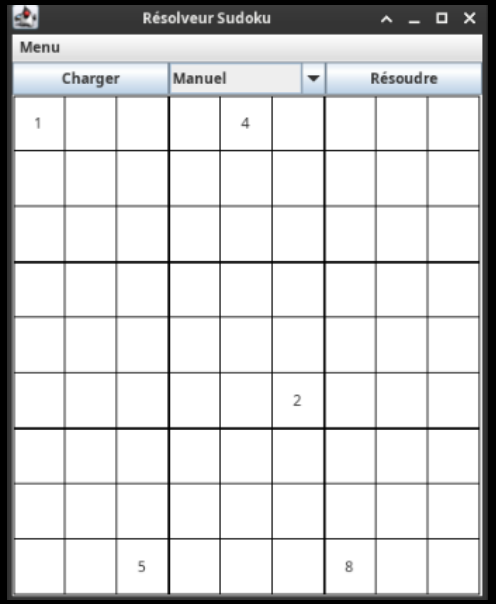


Sudoku

Rapport de projet

Garabajiu Danyil

Evan Heindryckx

Sommaire

Le projet consiste à coder un jeu de Sudoku en Java.

Le programme permet de créer, sauvgarder, charger et bien sur de résoudre une grille.

Fichiers :

MenuSudoku

SudokuBuilder

SudokuResolveur

MenuSudoku

Le menu est le premier fichier a ouvrir.

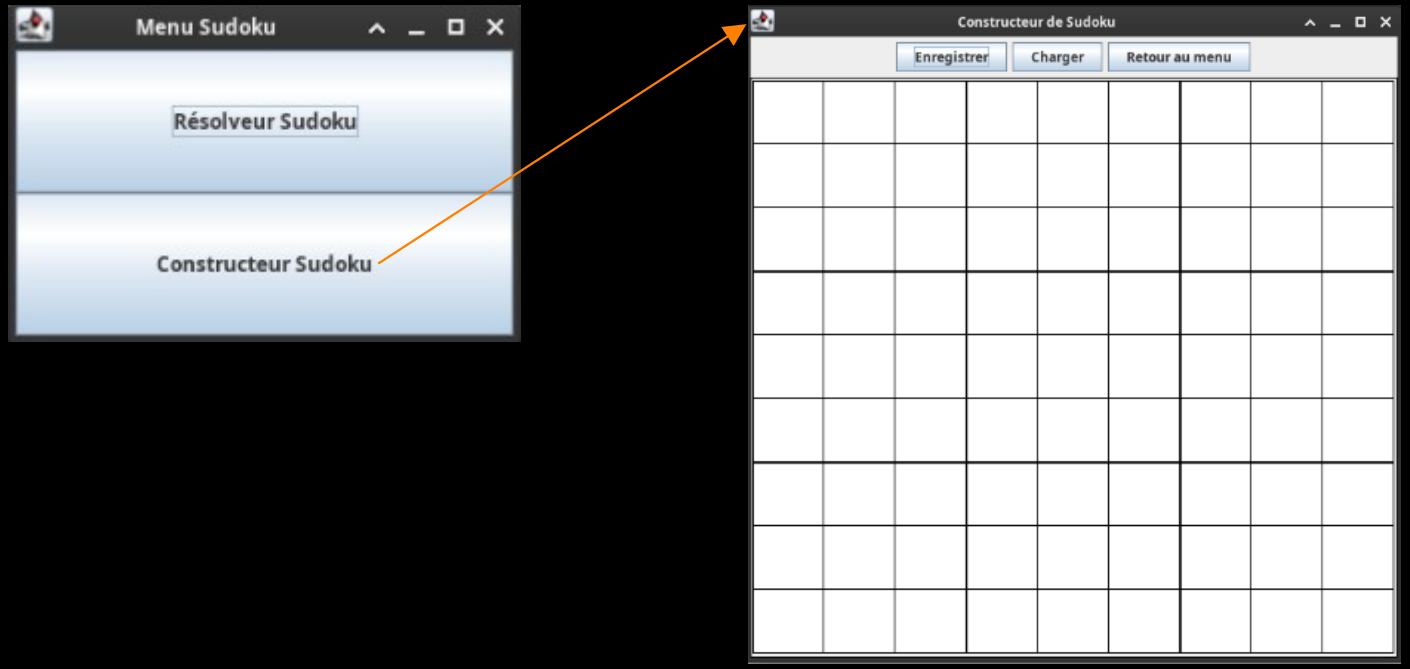
Il permet de choisir entre ouvrir le résolveur ou le constructeur du Sudoku. C'est un GridLayout de deux lignes et une colonne.

```
private static void ouvreréso() {
    SudokuResolveur.main(null);
    menuFrame.dispose();
}

private static void ouvrebuild() {
    SudokuBuilder.main(null);
    menuFrame.dispose();
}
```

```
JButton openSolverButton = new JButton("Résolveur Sudoku");
openSolverButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ouvreréso();
    }
});
panel.add(openSolverButton);

JButton openBuilderButton = new JButton("Constructeur Sudoku");
openBuilderButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ouvrebuild();
    }
});
panel.add(openBuilderButton);
```

SudokuBuilder

Le Builder permet comme son nom l'indique de créer une grille.



SudokuBuilder

Il se présente sous forme de grille, ainsi que de boutons qui permettent d'enregistrer la grille actuelle, de charger un fichier, ou de revenir au menu.

```
private void saveGrid() {
    if (!isValidGrid()) {
        JOptionPane.showMessageDialog(this, "La grille contient des valeurs invalides !");
        return;
    }

    if (!checkGrid()) {
        JOptionPane.showMessageDialog(this, "La grille contient des doublons dans les lignes ou les colonnes !");
        return;
    }

    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle("Enregistrer la grille");
    int userSelection = fileChooser.showSaveDialog(this);

    if (userSelection == JFileChooser.APPROVE_OPTION) {
        File fileToSave = fileChooser.getSelectedFile();
        String filePath = fileToSave.getAbsolutePath();
        if (!filePath.endsWith(".gri")) {
            fileToSave = new File(filePath + ".gri");
        }

        try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(fileToSave))) {
            outputStream.writeObject(getGrid());
            JOptionPane.showMessageDialog(this, "Grille enregistrée avec succès !");
        } catch (IOException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Erreur lors de l'enregistrement de la grille !");
        }
    }
}
```

```
private void loadGrid() {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileFilter(new FileNameExtensionFilter("Fichiers de grille", "gri"));
    int returnValue = fileChooser.showOpenDialog(this);
    if (returnValue == JFileChooser.APPROVE_OPTION) {
        File selectedFile = fileChooser.getSelectedFile();
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(selectedFile))) {
            int[][] loadedGrid = (int[][]) inputStream.readObject();
            setGrid(loadedGrid);
            JOptionPane.showMessageDialog(this, "Grille chargée avec succès !");
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Erreur lors du chargement de la grille !");
        }
    }
}
```

SudokuBuilder

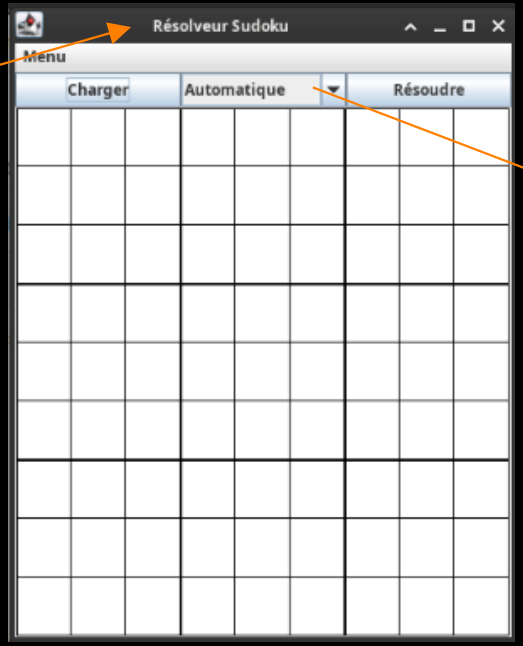
Les fonctions suivantes sont utilisées pour valider la grille; c'est-à-dire de voir si elle respecte toutes les règles du Sudoku.

```
private boolean isValidGrid() {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            String value = gridCells[i][j].getText().trim();
            if (!value.isEmpty()) {
                try {
                    int num = Integer.parseInt(value);
                    if (num < 1 || num > 9) {
                        return false;
                    }
                } catch (NumberFormatException e) {
                    return false;
                }
            }
        }
    }
    return true;
}
```

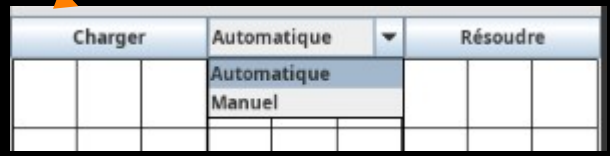
```
private boolean checkGrid() {
    for (int i = 0; i < 9; i++) {
        boolean[] rowCheck = new boolean[10];
        for (int j = 0; j < 9; j++) {
            String value = gridCells[i][j].getText().trim();
            if (!value.isEmpty()) {
                try {
                    int num = Integer.parseInt(value);
                    if (rowCheck[num]) {
                        return false;
                    }
                    rowCheck[num] = true;
                } catch (NumberFormatException e) {
                    return false;
                }
            }
        }
    }

    for (int j = 0; j < 9; j++) {
        boolean[] colCheck = new boolean[10];
        for (int i = 0; i < 9; i++) {
            String value = gridCells[i][j].getText().trim();
            if (!value.isEmpty()) {
                try {
                    int num = Integer.parseInt(value);
                    if (num < 1 || num > 9) {
                        return false;
                    }
                    if (colCheck[num]) {
                        return false;
                    }
                    colCheck[num] = true;
                } catch (NumberFormatException e) {
                    return false;
                }
            }
        }
    }

    return true;
}
```

SudokuResolveur

Lorsqu'on a créé une grille valide, on peut commencer à la résoudre. Cela se fait en ouvrant le résolveur.



On peut ici choisir le mode de résolution;
automatique qui résout pour nous,
et manuel qui nous permet de jouer.




```
comboModeResol = new JComboBox<>(new String[]{"Automatique", "Manuel"});
panneauHaut.add(comboModeResol);
```

SudokuResolveur

On peut charger une grille à l'aide du bouton prévu pour.

```
private static void chargerGrilleDepuisFichier() {
    JFileChooser selecteurFichier = new JFileChooser();
    selecteurFichier.setDialogTitle("Choisir un fichier de grille");
    selecteurFichier.setFileFilter(new FileNameExtensionFilter("Fichiers de grille Sudoku (*.gri)", "gri"));
    int choixUtilisateur = selecteurFichier.showOpenDialog(null);
    if (choixUtilisateur == JFileChooser.APPROVE_OPTION) {
        try {
            FileInputStream fichier = new FileInputStream(selecteurFichier.getSelectedFile());
            DataInputStream dataStream = new DataInputStream(fichier);
            int[][] grilleInitiale = new int[9][9];
            for (int i = 0; i < 9; i++) {
                int valeurLigne = dataStream.readInt();
                for (int j = 8; j >= 0; j--) {
                    int valeurCellule = valeurLigne % 10;
                    valeurLigne /= 10;
                    grilleInitiale[i][j] = valeurCellule;
                    cellulesGrille[i][j].setText(valeurCellule == 0 ? "" : String.valueOf(valeurCellule));
                    cellulesGrille[i][j].setEditable(valeurCellule == 0);
                }
            }
            dataStream.close();
            grilleSudoku = new GrilleSudoku(grilleInitiale);
        } catch (IOException | NumberFormatException ex) {
            ex.printStackTrace();
        }
    }
}
```

Le fichier de sauvgarde contient une grille de numéros qu'on lit ici avec un readInt() en mettant petit à petit les valeurs dans notre grille.
Lorsqu'on a lu toute la grille, on ferme le fichier.

SudokuResolveur

Pour résoudre une grille manuellement, on vérifie la grille de l'utilisateur, puis on test si elle est finie. Si c'est le cas, on lui dit qu'il a résolu la grille. Sinon, on lui signale qu'il y a des erreurs.

```
private static void resoudreGrille() {
    String modeSelectionne = (String) comboModeResol.getSelectedItem();
    if (modeSelectionne.equals("Automatique")) {
        if (grilleSudoku != null && grilleSudoku.resoudre()) {
            afficherGrille();
        }
    } else if (modeSelectionne.equals("Manuel")) {
        verifierGrille();
    }
}
```

```
private static void verifierGrille() {
    if (grilleSudoku != null) {
        if (grilleSudokuEstValide()) {
            JOptionPane.showMessageDialog(frame, "Félicitations ! Vous avez résolu le Sudoku !");
        } else {
            JOptionPane.showMessageDialog(frame, "Il y a des erreurs dans la grille. Veuillez corriger !");
        }
    }
}
```

SudokuResolveur

La vérification de la grille est plutôt simple. On vérifie si, pour chaque case, il n’y a aucun doublon sur la ligne, sur la colonne et sur le carré.

```
private static boolean grilleSudokuEstValide() {
    if (grilleSudoku == null) {
        return false;
    }

    int[][] grille = grilleSudoku.getGrille();

    // Vérifier les lignes
    for (int i = 0; i < 9; i++) {
        if (!estLigneValide(grille, i)) {
            return false;
        }
    }

    // Vérifier les colonnes
    for (int j = 0; j < 9; j++) {
        if (!estColonneValide(grille, j)) {
            return false;
        }
    }

    // Vérifier les carrés
    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 9; j += 3) {
            if (!estCarreValide(grille, i, j)) {
                return false;
            }
        }
    }

    return true;
}
```

```
private static boolean estLigneValide(int[][] grille, int ligne) {
    boolean[] chiffresUtilises = new boolean[10];
    for (int j = 0; j < 9; j++) {
        int chiffre = grille[ligne][j];
        if (chiffre != 0) {
            if (chiffresUtilises[chiffre]) {
                return false;
            }
            chiffresUtilises[chiffre] = true;
        }
    }
    return true;
}

private static boolean estColonneValide(int[][] grille, int colonne) {
    boolean[] chiffresUtilises = new boolean[10];
    for (int i = 0; i < 9; i++) {
        int chiffre = grille[i][colonne];
        if (chiffre != 0) {
            if (chiffresUtilises[chiffre]) {
                return false;
            }
            chiffresUtilises[chiffre] = true;
        }
    }
    return true;
}

private static boolean estCarreValide(int[][] grille, int ligneDebut, int colonneDebut) {
    boolean[] chiffresUtilises = new boolean[10];
    for (int i = ligneDebut; i < ligneDebut + 3; i++) {
        for (int j = colonneDebut; j < colonneDebut + 3; j++) {
            int chiffre = grille[i][j];
            if (chiffre != 0) {
                if (chiffresUtilises[chiffre]) {
                    return false;
                }
                chiffresUtilises[chiffre] = true;
            }
        }
    }
    return true;
}
```

SudokuResolveur

Enfin, le mode automatique de résolution permet de résoudre une grille à l'aide d'un algorithme qu'on pourrait qualifier de brute-force. On essaye de mettre un numéro dans la case et on vérifie les conditions.

Si tout va bien, alors on passe à la case suivante, sinon on reessaye avec un autre nombre.

```
public boolean resoudre() {
    for (int ligne = 0; ligne < 9; ligne++) {
        for (int colonne = 0; colonne < 9; colonne++) {
            if (grille[ligne][colonne] == 0) {
                for (int nombre = 1; nombre <= 9; nombre++) {
                    if (estPlacementValide(ligne, colonne, nombre)) {
                        grille[ligne][colonne] = nombre;
                        if (resoudre()) {
                            return true;
                        } else {
                            grille[ligne][colonne] = 0;
                        }
                    }
                }
                return false;
            }
        }
    }
    return true;
}
}
```

Conclusion

Danyil:

Je pense que cette SAE était un bon moyen de devenir plus à l'aise avec l'utilisation du Java. Bien qu'on ait eu des difficultés à certains moments, comme par exemple avec le chargement et sauvgarde de la grille qui créait et lisait mal les fichiers, on a su corriger les bugs et donner un bon résultat. Elle m'a permis egalement de me rendre compte de l'utilité et "praticité" de Java. Avec le C, même en utilisant des bibliotheques graphiques, je pense que la mise en place de la grille, le sélécteur du mode, les boutons, etc. serait beaucoup plus pénibles qu'en Java.

J'ai aussi compris que le Sudoku n'était pas un jeu d'intellectuels comme on pourrait penser mais un jeu de bourrin où le brute-force marche très bien...

Conclusion

Evan:

J’ai énormément appris avec cette SAE. Déjà que j’aimais beaucoup le Java, ça m’a permis en plus d’apprendre énormément de nouvelles fonctions, logiques de programmation, en plus de savoir comment fonctionne informatiquement un sudoku. Même si le fonctionnement des fichier .gri était une horreur à faire fonctionner, et mon idée de menu qui a plus complexifier qu’autre chose le code, on a finalement réussi à le faire en travaillant en équipe.

Je peux clairement dire que même si je trouve le java plus compliqué que le C, je le préfère largement. Il permet de faire beaucoup plus de choses graphiques, et à mon sens, plus facilement.

Fin de la SAE.