RAPPORT DE PROJET : SOLVER DE SUDOKU

REPUBLIQUE FRANCAISE

LIBERTE-EGALITE-FRATERNITE

******

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR

******

ENSIATE

REPUBLIC OF FRENCH

LIBERTY-EGALITY-FRATERNITY

******

MINISTRY OF HIGHER EDUCATION

******

ENSIATE

DOSSIER TECHNIQUE : REALISATION D'UN SOLVER DE SUDOKU

Rapport effectué du 15 mars au 15 avril 2024

Filière : **Internet des Objets**

Rédigé et présenté par :

DZO FOTSO JOSEPH CYRILLE

Classe et niveau : **ING3 IoT**

Matière : **Algorithme et programmation C**

Sous l'encadrement de :

M. LISEMBARD T.

ANNÉE ACADÉMIQUE 2023 - 2024

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

PLAN DE TRAVAIL

Table des matières

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

INTRODUCTION

Le Sudoku, initialement appelé 'Carré latin', a été popularisé au Japon avant de devenir un incontournable des jeux de réflexion dans le monde. Ce qui m'attire dans ce jeu, c'est la combinaison de règles simples et de défis complexes qu'il propose.

Mon but avec ce projet était de construire un programme capable de résoudre n'importe quelle grille de Sudoku, tout en offrant une interface simple pour les utilisateurs souhaitant tester leurs propres solutions. C'est dans cette optique je propose une application permettant de résoudre une grille de sudoku de niveau facile dans un premier temps et propose et proposer d'éventuelle solution permettant d'améliorer l'efficacité de celui-ci.

OBJECTIFS :

L'objectif principal de ce projet était de développer un programme en C capable de résoudre des grilles de Sudoku en utilisant une approche logique et systématique. Le programme devait inclure la **gestion d'une grille logique** pour suivre les contraintes et les possibilités pour chaque case de la grille de Sudoku. L'algorithme devait également utiliser une méthode de retour sur **trace (backtracking)** pour explorer toutes les solutions possibles et revenir en arrière si une impasse était atteinte. En outre, le programme devait être interactif, permettant à l'utilisateur de saisir une grille initiale, et fournir une solution correcte et complète tout en garantissant la validité de la solution selon les règles du Sudoku.

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

ANALYSE FONCTIONNELLE :

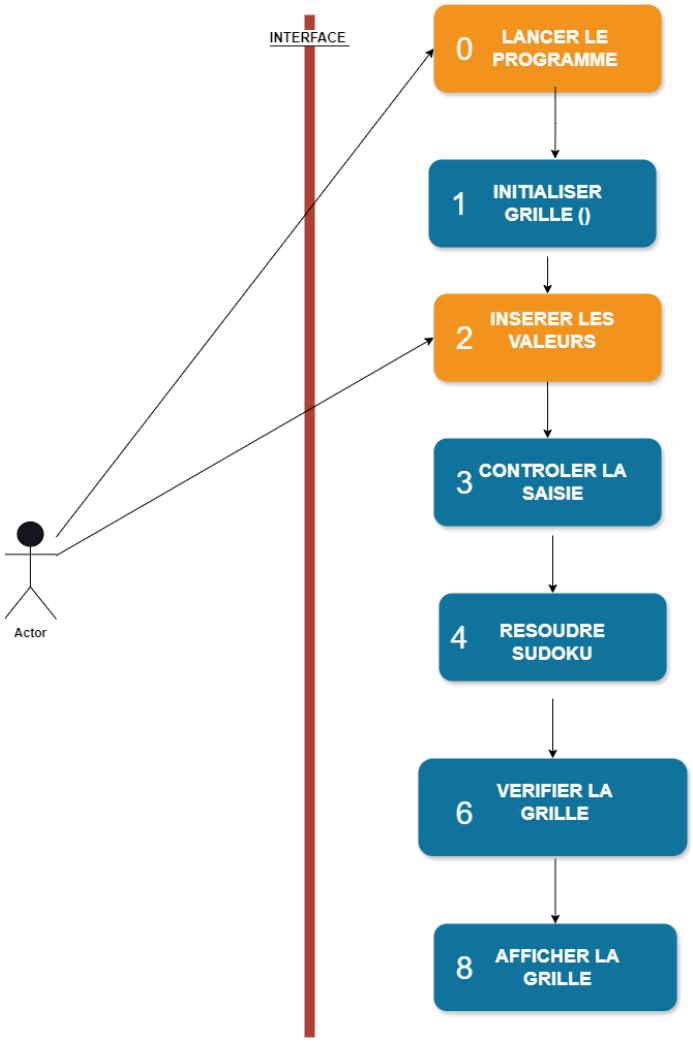
I. METHODE UTILISE :

L'approche adoptée pour résoudre le Sudoku repose sur l'utilisation d'une grille logique associée à la grille principale du jeu, permettant de gérer dynamiquement les possibilités de placement des numéros et d'assurer le respect des règles du Sudoku tout au long du processus de résolution. La grille logique **(logiqueTab)** est initialisée pour considérer toutes les valeurs comme possibles pour chaque case vide. À chaque étape, lors de la tentative de placement d'un numéro, la fonction **updateConstraints** met à jour cette grille logique en excluant les valeurs qui violeraient les contraintes du Sudoku pour la ligne, la colonne et le bloc 3x3 correspondants. La fonction **resoudreSudoku** utilise une approche récursive de retour sur trace (**backtracking),** où elle place un numéro dans une case vide et met à jour la grille logique. Si ce placement ne conduit pas à une solution valide, le numéro est retiré et la grille logique est réinitialisée, permettant d'explorer d'autres possibilités. Cette méthode assure une vérification systématique et exhaustive des solutions possibles, tout en maintenant une mise à jour continue des contraintes grâce à la grille logique. Les fonctions clés sont : **initialiserGrilles** pour préparer les grilles, **afficherGrille** pour visualiser l'état de la grille, **updateConstraints** pour ajuster les possibilités dans la grille logique, **ControlLogic** pour vérifier la validité des placements, **resoudreSudoku** pour l'algorithme de résolution récursive, et **verifierGrille** pour valider la solution finale. Chaque fonction joue un rôle essentiel dans la gestion des contraintes et la validation des étapes de la résolution, garantissant une solution correcte et conforme aux règles du **Sudoku.UML** :

II. UML :

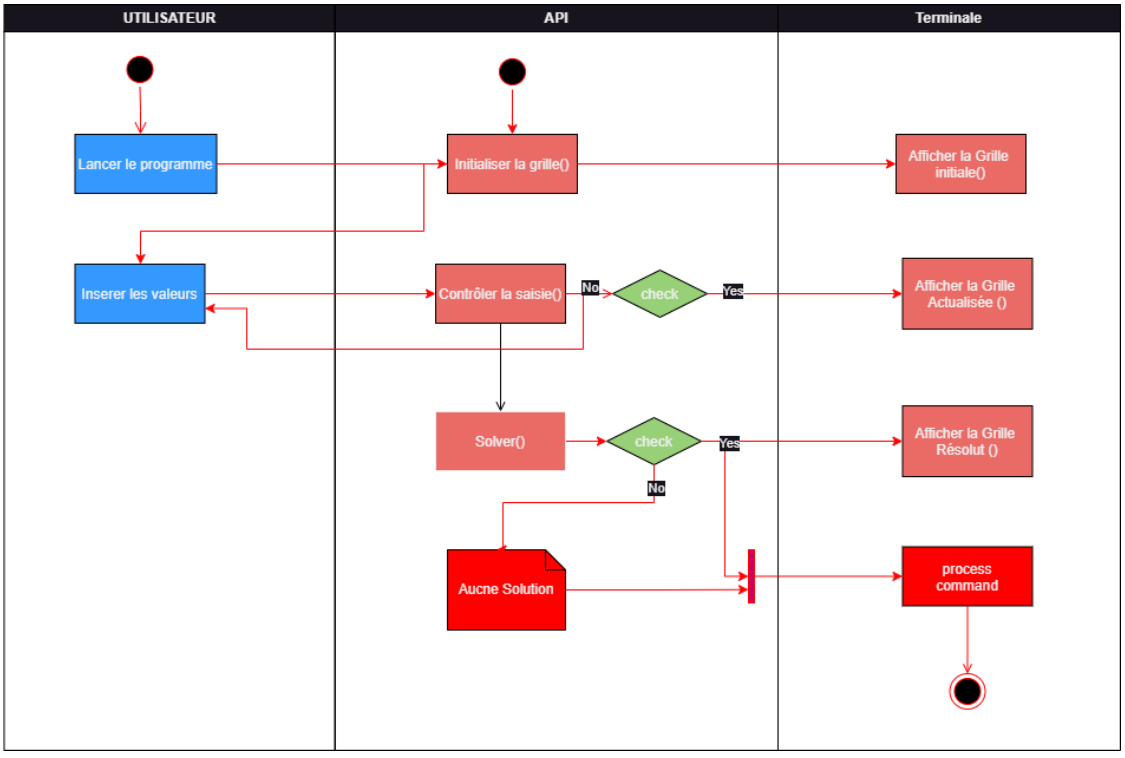
1) Diagramme de Cas d'Utilisation :

Le diagramme de cas d'utilisation pour notre programme de résolution de Sudoku illustre les interactions entre les utilisateurs (les étudiants, les enseignants, ou tout utilisateur intéressé) et le système. Les principaux cas d'utilisation incluent l'initialisation de la grille de Sudoku, la saisie des valeurs par l'utilisateur, la résolution automatique de la grille par le programme, l'affichage de la grille à différents stades (initiale, en cours et finale), et la vérification de la validité de la solution. Chaque interaction met en évidence les actions possibles que l'utilisateur peut effectuer et les réponses correspondantes du système, permettant une compréhension claire des fonctionnalités disponibles



2) Diagramme de Séquence :

Le diagramme de séquence décrit l'ordre des opérations pour résoudre un Sudoku en utilisant notre programme. Il commence par l'initialisation de la grille, suivie par l'appel de la fonction de résolution. Le programme vérifie chaque case vide et essaie les valeurs possibles en utilisant la grille logique pour restreindre les options. Si une valeur valide est trouvée, elle est insérée dans la grille et la grille logique est mise à jour en conséquence. Le processus continue récursivement jusqu'à ce que la grille soit complètement remplie ou qu'une impasse soit atteinte, entraînant un retour en arrière (backtracking). Le diagramme montre également comment le programme affiche la grille à différents moments et vérifie la solution finale, assurant ainsi que toutes les étapes de la résolution sont couvertes.



CONCEPTION DU SYSTÈME

1. Architecture générale du système :

J'ai structuré le programme en plusieurs modules principaux : la gestion de la grille de Sudoku, l'interface utilisateur pour les interactions, et le moteur de résolution utilisant un algorithme basé sur l'élimination de candidat à l'aide d'une grille logique d'une dimension plus grande que la grille de sudoku initiale.

2. Structures de données :

Pour représenter la grille, j'ai utilisé un tableau 2D de taille 9x9, permettant de manipuler facilement les différentes sous-grilles et valeurs et une grille 3D de taille 9x9x9 pour pouvoir renseigner les différents candidats possibles pour chaque élément de la grille.

3. Démonstration de l'Interface Utilisateur

Le système est basé sur une interaction avec l'utilisateur. À travers une interface en ligne de commande, les utilisateurs peuvent entrer leurs chiffres dans la grille, demander de l'aide pour un coup ou résoudre entièrement la grille. Exemples de fonctionnement : Voici un exemple où l'utilisateur remplit quelques cases et demande ensuite la solution complète. L'interface est conçue pour être intuitive et réactive.

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

ALGORITHME DE RÉSOLUTION

I. Algorithme de fonctionnement global :

Mon algorithme de fonctionnement global est basé sur l'appel successif de fonction dans un ordre précis et logique. Chaque fonction effectue un traitement les unes ou deux grilles déclarées globalement, ce qui permet l'actualisation en temps réel de celles-ci.

1) Description et rôles des fonctions utilisées :

a) Initialiser la Grille (InitialiserGrille) :

Cette fonction a pour but d'initialiser notre grille logique et notre grille de sudoku. Dans le plus de respecter les concepts du langage C, nous initialiserons notre Grille de Sudoku à 0 et notre Grille Logique à True (1). Nous l'initialiserons à 1 car dans un premier temps nous supposons que tous les nombres sont des candidats possibles pour une case initialement vide.

b) Insérer les valeurs (Insertvaleur) :

Cette fonction propose deux alternatives à l'utilisateur.

- La grille de Sudoku peut être remplit par l'utilisateur de manière dynamique et simple. Celle-ci fait également appelle à une seconde fonction qui est le contrôle de saisie (ControlSaisie). Elle a pour but de guider l'utilisateur lorsqu'il remplit la grille. Elle permet à celui d'insérer des valeurs sans toutes fois violer les règles de Sudoku. Elle assure également le respect des contraintes sur le type et le format de la variable à saisir.
- L'utilisateur peut aussi choisir d'utiliser une grille déjà préremplit et le programme se chargera de résoudre celle-ci et de lui retourner un résultat conforme.

c) Contrôle de Saisie (ControlSaisie) :

Vérifie si la valeur entrée par l'utilisateur est valide selon les règles du Sudoku, sans violer les contraintes de ligne, de colonne et de sous-grille.

d) Résoudre Sudoku (Solver) :

Fonction récursive qui essaie de résoudre le Sudoku en plaçant les chiffres de 1 à 9 dans la grille, tout en respectant les contraintes de Sudoku.

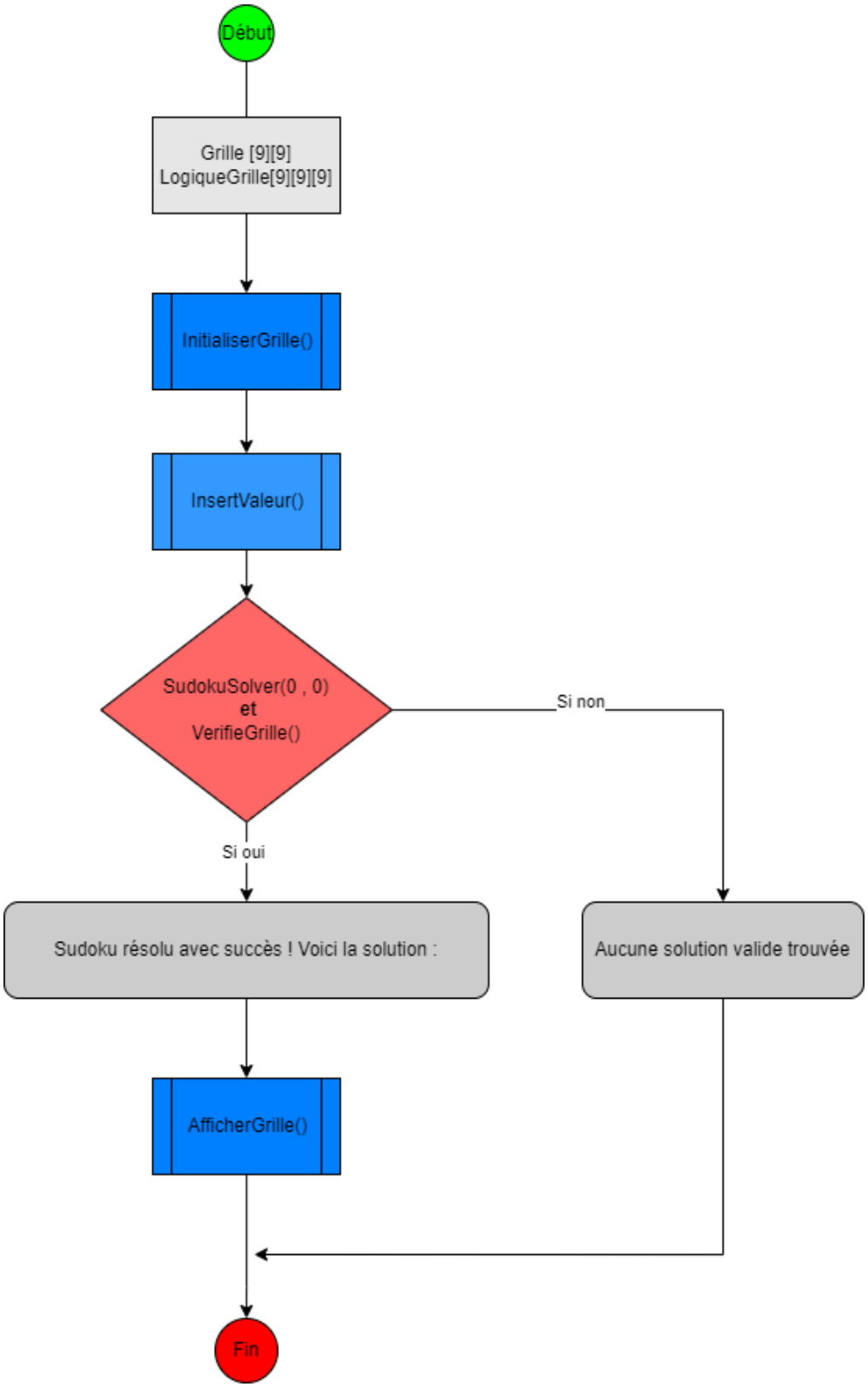
e) Vérifier si grille valide (VerifieGrille) :

Vérifie que la grille complétée est correcte, en s'assurant que toutes les valeurs respectent les règles du Sudoku.

f) Mise à jour grille Logique (UpdateConstraints) :

La fonction est utilisée pour mettre à jour la grille logique (logiqueTab) en fonction des contraintes imposées par un placement donné dans la grille principale (grille). Cette mise à jour est cruciale pour assurer que les règles du Sudoku sont respectées lors de la tentative de résolution.

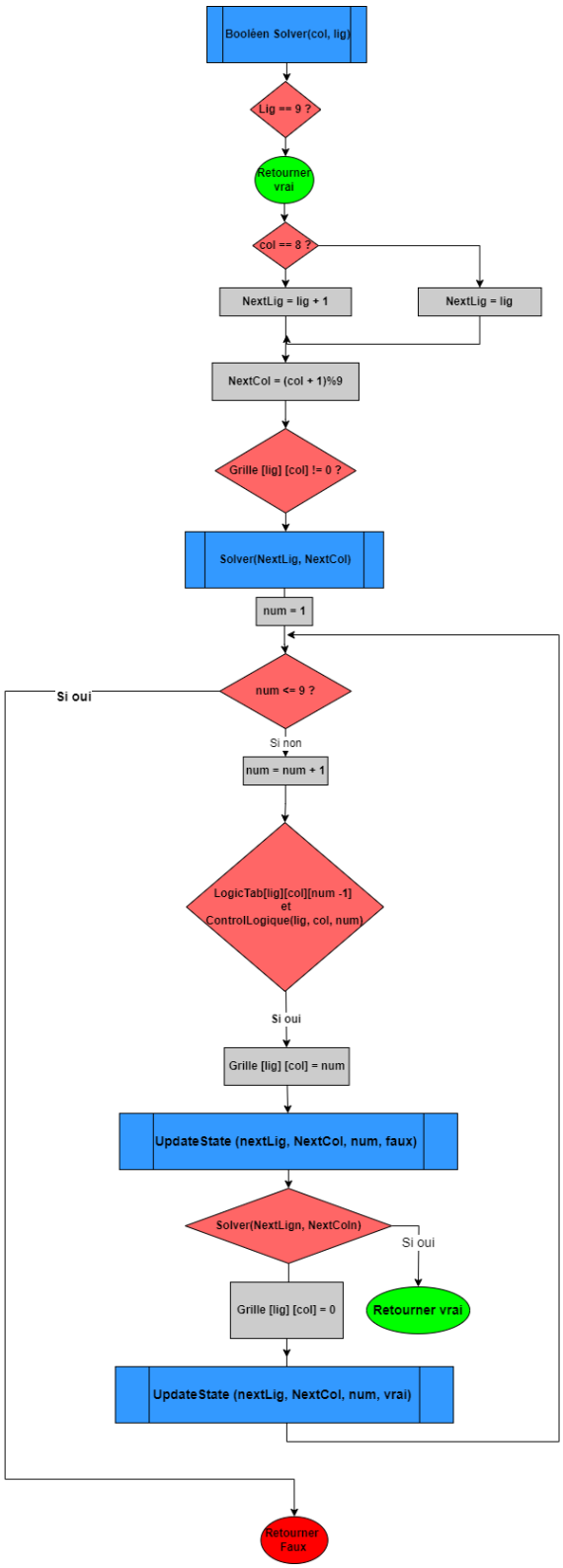
2) Schéma de l'algorithme de fonctionnement global :



Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

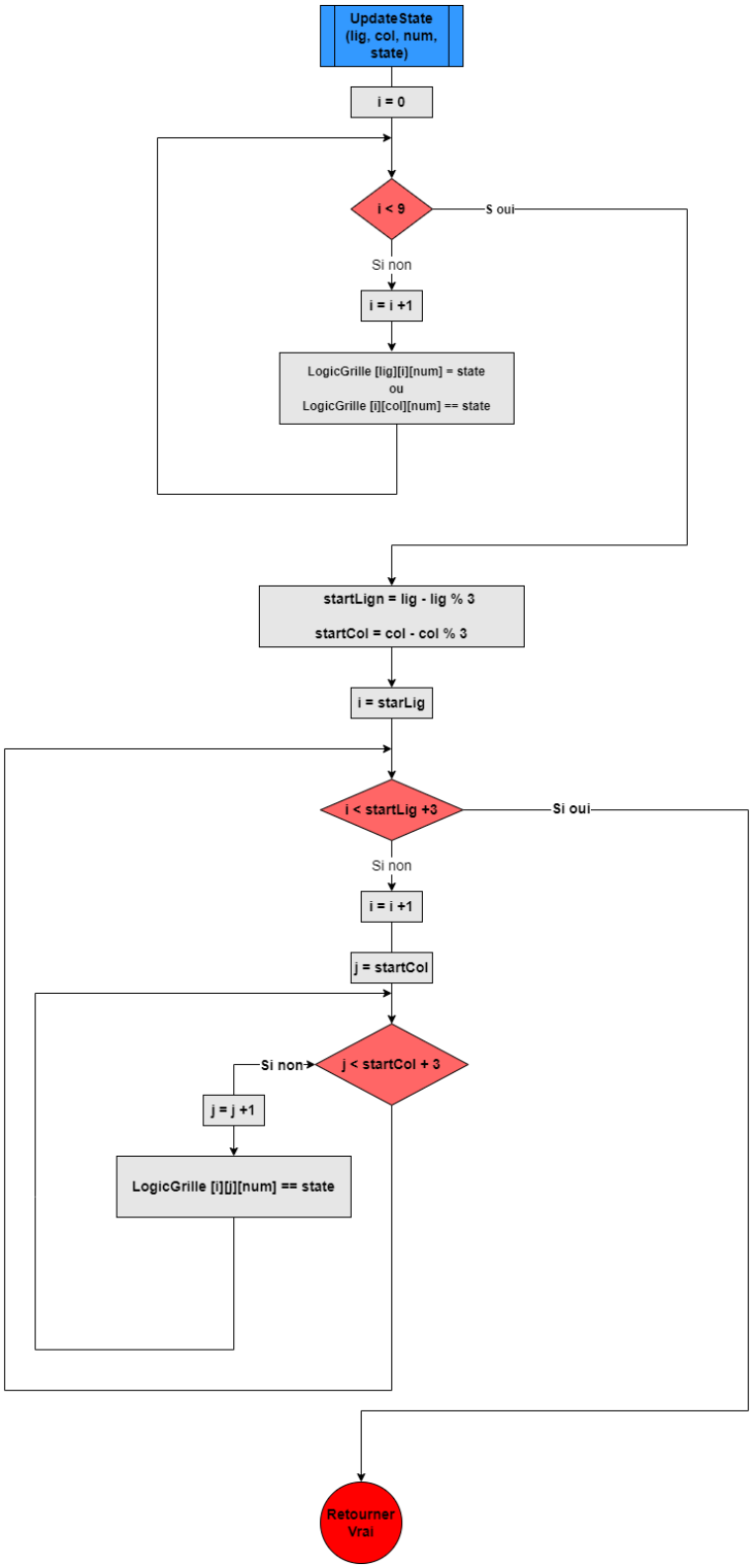
II. Algorithme de fonctions Utilisés :

1) Fonction Solver :



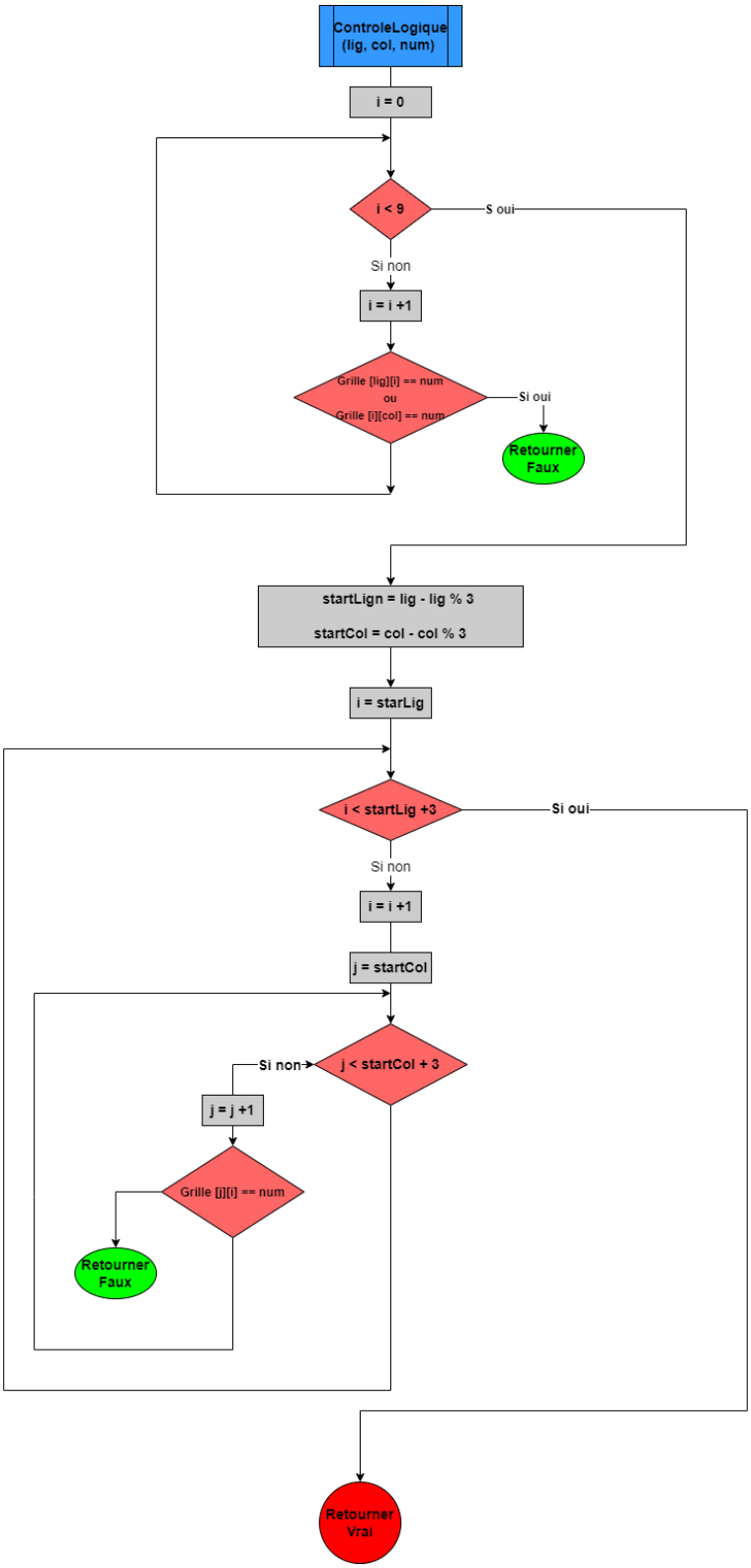
Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

2) Fonction UpdateContraints :



Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

3) Fonction ControlLogic :



Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

4) Fonction vérifie grille :

```
fonction verifierGrille()
    pour chaque ligne de 0 à TAILLE-1
        pour chaque colonne de 0 à TAILLE-1
            num = valeur à la position (ligne, colonne) dans la grille
            si num n'est pas égal à 0 ET ControleLogique(ligne, colonne, num)
                retourner faux
    retourner vrai
```

5) Fonction Vérifie Ligne

```
fonction verifieLigne(grille, ligne)
    créer un tableau 'presente' de taille TAILLE+1 initialisé à faux
    pour chaque colonne de 0 à TAILLE-1
        num = valeur à la position (ligne, colonne) dans la grille
        si num est inférieur à 1 OU supérieur à 9 OU si 'presente[num]' est vrai
            afficher "Ligne ligne+1: Erreur"
            retourner faux
        'presente[num]' = vrai
    afficher "Ligne ligne+1: OK"
    retourner vrai
```

6)Fonction Vérifie Colonne :

```
fonction verifieColonne(grille, colonne)
    créer un tableau 'presente' de taille TAILLE+1 initialisé à faux
    pour chaque ligne de 0 à TAILLE-1
        num = valeur à la position (ligne, colonne) dans la grille
        si num est inférieur à 1 OU supérieur à 9 OU si 'presente[num]' est vrai
            afficher "Colonne colonne+1: Erreur"
            retourner faux
        'presente[num]' = vrai
    afficher "Colonne colonne+1: OK"
    retourner vrai
```

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

7) Fonction Vérifie Sous Grille :

```
fonction verifieSousGrille(grille, startLigne, startColonne)
    créer un tableau 'presente' de taille TAILLE+1 initialisé à faux
    pour chaque ligne de 0 à 2
        pour chaque colonne de 0 à 2
            num = valeur à la position (startLigne + ligne, startColonne + colonne)
            si num est inférieur à 1 OU supérieur à 9 OU si 'presente[num]' est vrai
                afficher "Sous-grille (startLigne, startColonne): Erreur"
                retourner faux
            'presente[num]' = vrai
    afficher "Sous-grille (startLigne, startColonne): OK"
    retourner vrai
```

8) Fonction test Final :

```
fonction TestFinal(grille)
    afficher "Le Test commence..."
    valide = vrai

    pour chaque ligne de 0 à TAILLE-1
        si verifieLigne(grille, ligne) renvoie faux
            valide = faux

    pour chaque colonne de 0 à TAILLE-1
        si verifieColonne(grille, colonne) renvoie faux
            valide = faux

    pour chaque ligne de 0 à TAILLE-1 avec un pas de 3
        pour chaque colonne de 0 à TAILLE-1 avec un pas de 3
            si verifieSousGrille(grille, ligne, colonne) renvoie faux
                valide = faux

    afficher "Sudoku resolu avec succes ! Voici la solution : OK"
    retourner valide
```

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

9) La fonction insert valeur :

```
fonction insertValeur(grille)
    nombre = 0

    pour chaque i de 0 à 8 // Ligne
        pour chaque j de 0 à 8 // Colonne
            début: // Étiquette associée à la boucle extérieure

            afficher "Ligne i+1 et colonne j+1 : "

            si la saisie est un entier (nombre) valide
                si ControleLogique(i, j, nombre) renvoie vrai
                    grille[i][j] = nombre // Affectation du nombre saisi à la grille
                    appeler afficherGrille()
                sinon
                    afficher "Saisie invalide dans ce champ. Reessayer et eviter les
                    aller à début // Retourner à l'étiquette pour réessayer la saisi
            sinon
                afficher "Veuillez saisir une valeur entiere :-("
                vider le flux d'entrée si la saisie était incorrecte
```

IMPLEMENTATION

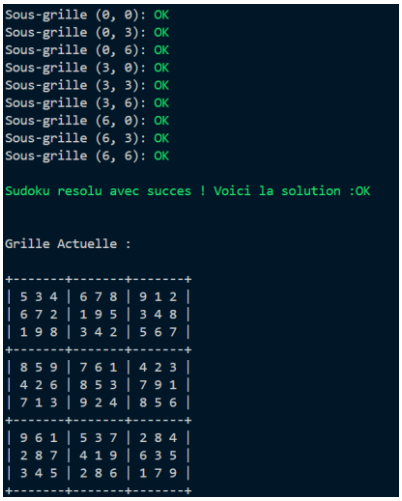
L'implémentation du projet de résolution de Sudoku a suivi une approche méthodique et structurée. Le programme a été développé en langage C, en mettant l'accent sur la clarté et la modularité du code. La grille principale de Sudoku et la grille logique ont été déclarées comme des variables globales pour faciliter leur accès et leur mise à jour par les différentes fonctions du programme. L'initialisation des grilles a été réalisée en définissant une grille d'exemple partiellement remplie, puis en mettant à jour la grille logique pour refléter les contraintes initiales. L'algorithme de résolution utilise une méthode de retour sur trace, en vérifiant systématiquement les possibilités pour chaque case vide et en revenant en arrière si une impasse est rencontrée. La mise à jour des contraintes est effectuée à chaque insertion de nombre pour garantir la validité de la solution. Cette approche a permis de développer un programme capable de résoudre efficacement et correctement des grilles de Sudoku de différentes complexités.

Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

TEST ET VALIDATION :

Pour assurer la fiabilité du programme, diverses grilles de Sudoku ont été testées, allant des grilles faciles à des grilles plus complexes. Mais nous accentuerons sur l**es tests de grille facile conformément à la consigne demandée.** Chaque grille a été soumise à la procédure de résolution automatique, et les solutions ont été vérifiées pour s'assurer qu'elles respectent les règles du Sudoku. Des tests unitaires ont été réalisés sur les fonctions clés telles que la vérification des contraintes et la mise à jour de la grille logique. Les résultats ont montré que le programme peut résoudre efficacement une variété de grilles de Sudoku, confirmant ainsi la validité et l'efficacité de l'algorithme implémenté.

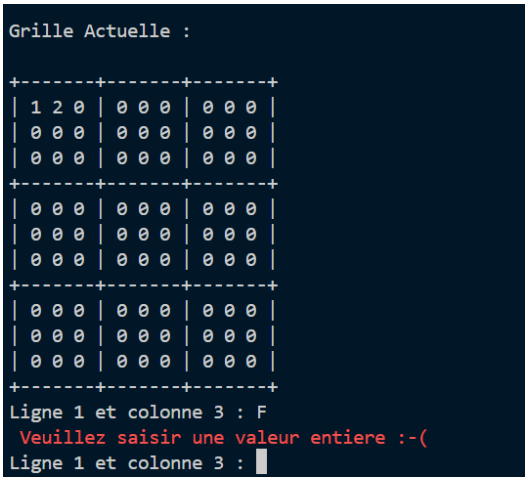
1) Avec une grille initialement choisie :

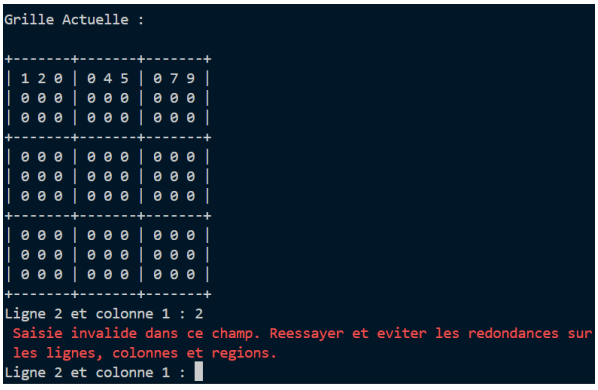


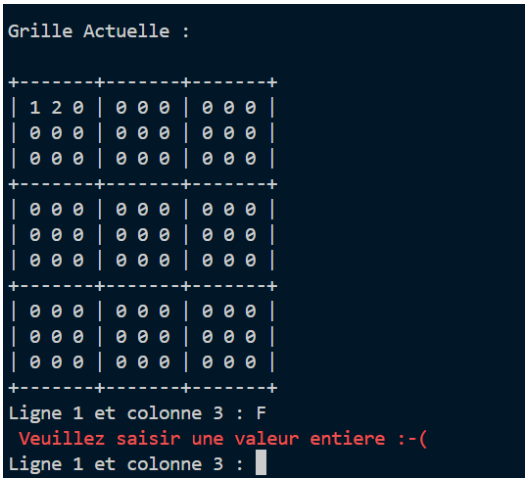
Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**

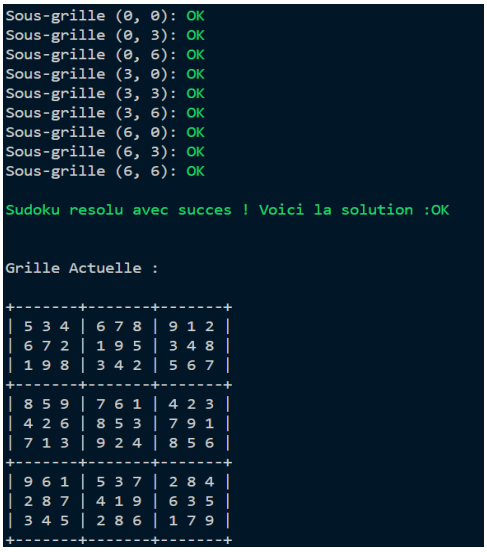
2)Avec une grille saisie manuellement :

Nous allons proposer à plusieurs tests. Dans un premier temps les tests liés au contrôle de saisie et dans un second ceux liés à la résolution.









Rédigé et présenté par : **DZO FOTSO JOSEPH CYRILLE**