Sudoku Solver
Documentation

Hadrien Sevel (GM-BA4)

Semestre de printemps 2021

Le programme **Sudoku Solver** permet de reconnaitre un sudoku grâce à un programme d'OCR (Optical Character Recognition) écrit en langage C à partir d'une image au format JPG et de taille 400x400 donnée en entrée. Il peut ensuite le résoudre grâce des fonctions écrites et exécutées avec MATLAB et produire un fichier PDF avec le résultat. Le programme principal est écrit et exécuté avec LabVIEW et fait appel au programme d'OCR et à MATLAB.

Plateforme : Windows 10
Compilateur : GCC 8.1.0 (built by MinGW-W64 project)

1 Fichiers du projet

1.1 VIs LabVIEW

- **SudokuSolver.vi** : VI principal à lancer en premier contenant tous les autres VIs.
- **ReadSudoku.vi** : Vérifie le type de l'image, l'ouvre, vérifie ses dimensions et la convertit en noir & blanc.
- **ComputeCellRect.vi** : Calcule les coordonnées du rectangle pour découper la case en fonction de l'avancement dans les 2 boucles de SudokuSolver.vi.
- **WriteCase.vi** : Découpe une case de l'image avec le rectangle donné par ComputeCellRect.vi et l'enregistre en binaire dans le fichier *cell.bin*.
- **CallC.vi** : Appelle le programme OCR.exe et enregistre *stdout* & *stderr* dans le fil d'erreur.
- **ReadCase.vi** : Ouvre le fichier *CellValue.txt* créé par OCR.exe et lit le chiffre de la case.
- **BuildmScript.vi** : Crée le script *solve.m* avec la matrice obtenue contenant les chiffres du sudoku.
- **MP_LaunchMatlabScript2.vi** : Appelle MATLAB pour exécuter le script *solve.m*.

1.2 Scripts MATLAB

- **solveSudoku.m** : Fonction qui résoud de manière récursive le sudoku (cf. algorithme 1 à la section 4).
- **checkSolved.m** : Fonction qui vérifie si le sudoku est résolu ou non (retourne 1 si une seule possibilité pour chaque case, sudoku résolu ; 0 si au moins une case avec plusieurs possibilités, sudoku non résolu ; -1 si au moins une case sans possibilité, sudoku non résolvable).
- **fillHypothesis.m** : Fonction qui génère la matrice A 9x9x9 qui contient tous les chiffres possibles pour chaque case du sudoku.
- **getChoices.m** : Fonction qui sélectionne une case avec le moins de possibilités et donne ses cordonnées x et y ainsi que le vecteur choices qui contient les possibilités.
- **dispSudoku.m** : Fonction qui crée l'affichage de la grille de sudoku et l'exporte au format PDF. Une partie du code est adapté de la fonction *drawSudoku* prise de la documentation MATLAB (`https://www.mathworks.com/help/optim/ug/solve-sudoku-puzzles-via-integer-programming-solver-based.html#SolveSudokuExample-7`).

1.3 OCR

- **OCR.exe** : Exécutable compilé à partir de *OCR.c*. Ouvre le fichier *cell.bin*, reconnait le chiffre de la case ou si c'est une case vide et l'écrit dans le fichier *CellValue.txt* (cf. description des fonctions à la section 4).
- **OCR.c** : Fichier contenant le code source de *OCR.exe*. Fait appel au fichier header *DigitBitmap.h*.
- **DigitBitmap.h** : Contient les bitmaps sous forme de tableaux d'entiers des chiffres utilisés pour comparaison au contenu du fichier binaire par *OCR.exe* pour reconnaitre le chiffre.

1.4 Fichiers générés par les programmes

Ces fichiers d'exemple ont été générés avec l'image *S7.jpg* des sudokus fournis. *cell.bin* & *CellValue.txt* correspondent à la dernière case du sudoku, soit un 3.

- **cell.bin** : Fichier binaire créé par le VI *WriteCase.vi* contenant les dimensions et les pixels de la dernière case du sudoku.
- **CellValue.txt** : Fichier texte contenant le résultat de la reconnaissance de la case contenue dans *cell.bin* par le programme *OCR.exe*.
- **solve.m** : Script MATLAB généré par le VI *BuildmScript.vi* contenant le sudoku à résoudre.
- **s7.pdf** : Fichier PDF généré par MATLAB lors de l'exécution du script *solve.m* contenant le sudoku résolu.

2 Flow de données

La figure suivante illustre le flow des données au sein du programme. *SudokuSolver.vi* prend en entrée l'image JPG de taille 400x400 pixels. Pour les 81 cases du sudoku, le VI découpe le rectangle de la case dans l'image, l'enregistre en binaire avec ses dimensions dans le fichier *cell.bin* et appelle le programme *OCR.exe*. *OCR.exe* prend en entrée le fichier *cell.bin*, reconnait le chiffre ou si la case est vide et écrit le résultat dans le fichier *CellValue.txt*. Les erreurs sont affichées dans *stderr* et les avertissements dans *stdout* et sont ensuite inclus dans le fil d'erreur du VI. Puis *SudokuSolver.vi* t la valeur de la case dans *CellValue.txt* et l'enregistre dans un tableau. Lorsque toutes les cases du sudoku on été reconnues, le VI génère le script *solve.m* contenant la matrice du sudoku et appelle MATLAB pour l'exécuter.

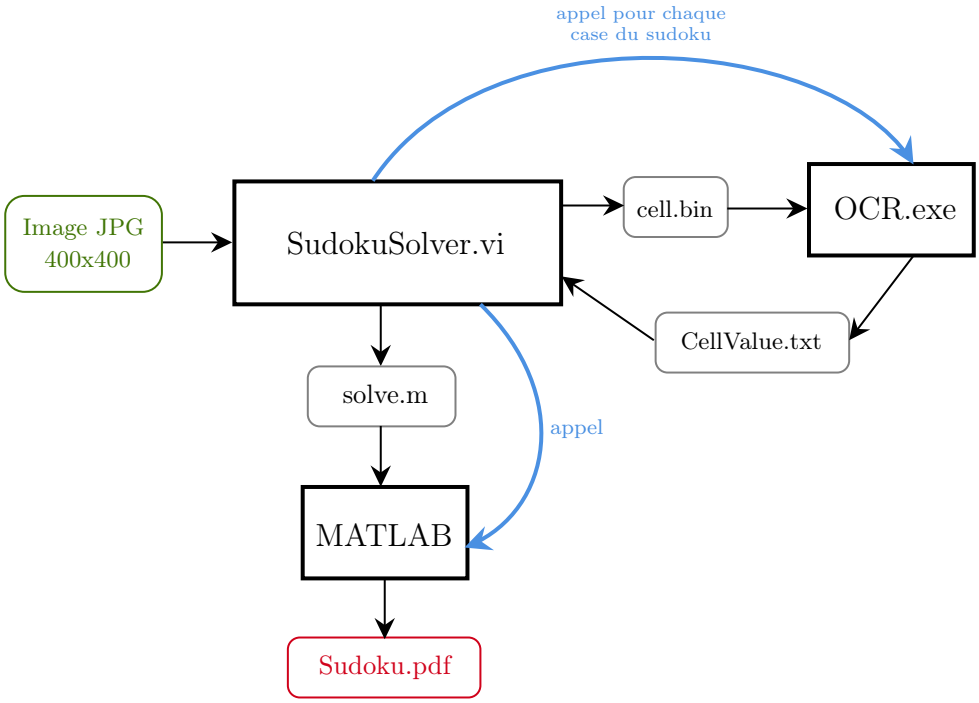

FIGURE 1 – Diagramme illustrant le flow de données

3 Gestion des erreurs

3.1 Au niveau du VI *SudokuSolver.vi* :

- Erreur **lors de l'ouverture de l'image** (*ReadSudoku.vi*) : mise à **true** du fil d'erreur pour arrêter l'exécution du programme et affichage d'une erreur dans l'error out du panel.
- Erreur **l'image n'est pas de type JPG** (*ReadSudoku.vi*) : mise à **true** du fil d'erreur pour arrêter l'exécution du programme et affichage de *"Error : wrong image type."* dans l'error out du panel.
- Erreur **l'image n'est pas de taille 400x400** (*ReadSudoku.vi*) : mise à **true** du fil d'erreur pour arrêter l'exécution du programme et affichage de *"Error : wrong image size."* dans l'error out du panel.

— Erreur **lors de l'écriture de *cell.bin*** (*WriteCase.vi*), **lors de la lecture de *CellValue.txt*** (*ReadValue.vi*) ou **lors de l'écriture de *solve.m*** (*BuildmScript.vi*) : mise à **true** du fil d'erreur pour arrêter l'exécution du programme et affichage d'une erreur dans l'error out du panel.
— Erreur **lors de l'appel de *OCR.exe* (exécutable manquant, pas les droits d'ouvrir le fichier, etc.)** (*CallC.vi*) : mise à **true** du fil d'erreur pour arrêter l'exécution du programme et affichage d'une erreur dans l'error out du panel.

3.2 Au niveau du programme *OCR.exe* :

Note : Tous les avertissements ou les erreurs qui surviennent lors de l'execution de *OCR.exe* après un appel par LabVIEW sont repris dans le fil d'erreur de LabVIEW et affichés dans l'error out du panel. Un avertissement ne change pas le status du fil d'erreur (pas d'arrêt du programme dû à l'avertissement) et une erreur met à **true** le fil d'erreur (interruption de l'exécution du programme).

— Erreur **le nombre d'arguments donnés est différent de 12** : affichage dans *stderr* de *"Wrong number of parameters."* et arrêt du programme.
— Erreur **un ou plusieurs seuils ne sont pas entre les bornes [5,100]** : affichage dans *stderr* de *"Wrong value for one or more thresholds."* et arrêt du programme.
— Erreur **lors de l'ouverture du fichier binaire *cell.bin*** : affichage dans *stderr* de *"Error opening the binary file."* et arrêt du programme.
— Erreur **lors de la lecture des dimensions de l'image** : affichage dans *stderr* de *"Unable to read cell dimensions."*, fermeture du fichier binaire et arrêt du programme.
— Erreur **la hauteur de l'image n'est pas comprise entre les bornes [21,100] et/ou la largeur de l'image n'est pas comprise entre les bornes [16,100]** : affichage dans *stderr* de *"Wrong width and/or height of the cell."*, fermeture du fichier binaire et arrêt du programme.
— Erreur **l'image ne contient pas assez de pixels** : affichage dans *stderr* de *"Not enough pixels in the cell."*, fermeture du fichier binaire et arrêt du programme.
— Avertissement **trop de pixels dans l'image** : affichage dans *stdout* de *"Warning : too many pixels in the image, additional pixels ignored."*, ignore les pixels supplémentaires et continue l'exécution du programme.
— Erreur **lors de la fermeture du fichier binaire** : affichage dans *stderr* de *"Error when closing the binary file."* et arrêt du programme.
— Erreur **lors de l'ouverture ou de la création du fichier *CellValue.txt*** : affichage dans *stderr* de *"Error when opening/creating the file CellValue.txt."* et arrêt du programme.
— Erreur **lors de la fermeture du fichier *CellValue.txt*** : affichage dans *stderr* de *"Error when closing the file CellValue.txt."* et arrêt du programme.

4 Description des fonctions

4.1 Résolution du sudoku avec MATLAB

L'algorithme 1 décrit le fonctionnement de la fonction récursive **solveSudoku** utilisée dans MATLAB pour résoudre un sudoku.

4.2 Reconnaissance des chiffres avec le programme *OCR.exe* :

— Fonction **readCell** : Ouvre le fichier binaire, vérifie les dimensions de l'image, vérifie le nombre de pixels, enregistre la valeur des pixels dans un tableau, ferme le fichier binaire et retourne le pointeur du tableau.
— Fonction **findNumber** : Appelle la fonction *isCellEmpty* pour savoir si la case est vide, sinon boucle sur les chiffres du sudoku, sur les déplacements verticaux et horizontaux que le bitmap doit effectuer sur l'image et appelle à chaque fois la fonction *getRatioCommonPixels* pour obtenir le ratio de pixels communs entre le bitmap et l'image. Enfin, elle enregistre le ratio max et écrit le chiffre correspondant et son ratio dans le fichier *CellValue.txt*. Si *isCellEmpty* n'a pas retourné la case comme étant vide mais que aucun ratio max n'a dépassé le seuil associé, alors la case est considérée vide (notée 0 dans le fichier *CellValue.txt*).
— Fonction **isCellEmpty** : Compte le nombre de pixels blancs dans la case et compare le pourcentage obtenu avec le seuil donnée en argument : retourne 1 si le pourcentage est supérieur au seuil (case vide) et retourne 0 si le pourcentage est inférieur au seuil.
— Fonction **getRatioCommonPixels** : Compte le nombre de pixels communs (noirs & blancs) entre le bitmap et la case et calcule le ratio. Pour cela, elle boucle sur tous les pixels du bitmap et les compare

Algorithme 1 : Fonction récursive **solveSudoku**

Entrées : *Sudoku*
Sorties : *Sudoku*

```
début
    si Sudoku non résolu (i.e. plusieurs chiffres possibles pour au moins une case) alors
        choisir une case avec le moins de chiffres possibles
        pour chiffres possibles de la case choisie faire
            écrire chiffre dans la case
            si Sudoku non résolu alors
                Sudoku = solveSudoku(Sudoku) /* appel de la fonction récursive      */
            sinon si Sudoku résolu (i.e. une possibilité par case) alors
                remplir toute les cases restantes avec leur seul chiffre possible
            fin
            /* bloc exécuté lorsque l'algorithme sort d'un niveau de récursion        */
            si Sudoku résolu alors
                sortir de la boucle
            sinon si Sudoku non résolvable (i.e. au moins une case sans possibilité) alors
                effacer le chiffre de la case
            fin
        fin
    fin
fin
```

avec les pixels de la case en tenant compte du décalage du bitmap sur la case (offsetX & offsetY). Pour obtenir le pixel du bitmap, elle appelle le fonction *getPixelBitmap*.

— Fonction **getPixelBitmap** : retourne la valeur du pixel du bitmap pour le chiffre, la ligne et la colonne donnés.