Rapport Soutenance 1
Solver de Sudoku

Victor Tang Olivier Bensemhoun Marc Moutarde
Jayson Vanmarcke

9 Novembre 2023

Résumé

Ce rapport présente l'état d'avancement du projet *Sudoku OCR* à la date du 4 novembre 2023. Les élements fonctionnels du programme sont le solver de sudoku, le chargement d'une image, la détection des contours et le réseau de neurones.

Nous disposons également de petits utilitaires créés pour simplifier divers opérations répétitives (formatage du code, compilation de l'ensemble du projet, ...).

A ce jour, le projet se déroule en respectant les délais demandés et les choses se présentent de façon à ce que le projet soit terminé pour l'échéance requise.

Sommaire

1 Introduction

1.1 Présentation des membres du groupe

1.1.1 Victor Tang

Les sudokus ou les jeux de logiques plus généralement, sont des jeux qui m'ont toujours interessé. Le sudoku particulièrement car les règles sont simples à comprendre, mais il peut rapidement devenir difficile. Cela me fait penser que je prenais pour habitude de noter les possibilités dans un coin de la case et les interdictions dans une autre. En y pensant plus profondément, c'est un travail récursif qu'on fait à la tête car le résultat est obtenu en fonctions des hypothèses émises et que ces hypothèses en engendre d'autres. Je trouve ce sujet très intéressant car il aborde plusieurs sujets comme l'intelligence artificielle et la manipulation d'images.

1.1.2 Jayson Vanmarcke

Depuis mon enfance, je suis passionné de jeux de logiques tels que les mathématiques, les sudokus, les énigmes ou encore des puzzles très complexes. J'ai une attirance particulière pour ces jeux car ils stimulent le cerveau qui est pour moi une fascinante machine. Ainsi, ils permettent d'améliorer différentes fonctions de notre cerveau lorsque que les neurones sont boostés. C'est la raison pour laquelle, réaliser un programme permettant de résoudre un sudoku, est un défit stimulant et intéressant qui me rend particulièrement enthousiaste. Par ailleurs, le projet de l'an passé, qui était de concevoir un jeu vidéo m'a beaucoup plu et j'avais vraiment hâte de refaire un autre projet. Ne connaissant pas le langage C, j'étais pressé de découvrir son univers pour la réalisation de ce projet.

1.1.3 Olivier Bensemhoun

Les neural networks m'ont toujours intéressés. Depuis, ce que j'ai découvert avec ChatGPT et LLama des LLm, j'ai voulu passer du temps pour découvrir les réseaux de neurones avec plus de details. Je me suis donc mis tout naturellement sur la création du réseau de neurones.

1.1.4 Marc Moutarde

Passionné par la programmation, je m'intéresse particulièrement au développement de jeux vidéo et de systèmes d'exploitation. Aussi, je connais quelque peu la famille des languages C. Dans cette famille, le C fait partie de mes préférés du à sa simplicité et sa portabilité : la relative simplicité de sa norme et son usage extrêmement courant en fait un des languages les plus rapidement portés sur un nouveau système d'exploitation. Cette même simplicité en fait un des languages privilégiés pour les systèmes embarqués.

1.2 Répartition des tâches

Tâches	Victor Tang	Jayson Vanmarcke	Olivier Benshemoun	Marc Moutarde
Chargement d'une image et suppression des couleurs				Responsable
Rotation manuelle de l'image	Responsable			
Détection de la grille et de la position des cases	Responsable			
Découpage de l'image				Responsable
Résolution du sudoku		Responsable		
Réseau de neurones			Responsable	

1.3 Etat d'avancement du projet

Tâches	Avancement
Chargement d'une image et suppression des couleurs	Fonctionnel
Rotation manuelle de l'image	Fonctionnel mais fuite de mémoire
Détection de la grille et de la position des cases	Présente des défauts
Découpage de l'image	Fonctionnel
Résolution du sudoku	Fonctionnel
Réseau de neurones	Fonctionnel

2 Les aspects techniques

2.1 Chargement d'une image et suppression des couleurs

Le chargement de l'image s'oppère en utilisant les PixelBuffers offerts par Gdk. Cet objet dispose d'une fonction permettant de charger une image de format courant (png, jpeg, ...) directement, ainsi que de rapidement obtenir un objet pour Gtk.

La suppression des couleurs s'oppère en prenant les cannaux rouges, verts et bleus de l'image et les pondérants avec les poids 0,2126 pour le rouge, 0,7152 pour le vert et 0,0722 pour le bleu. Ces poids sont des poids courants déterminés à partir de la sensibilité de l'oeil humain aux différentes composantes de la lumière blanche.

2.2 Rotation manuelle de l'image

La rotation de l'image est implémentée en utilisant de façon intensive la bibliothèque Cairo sur laquelle est basée Gtk. Cette bibliothèque fournit en effet les fonctions de géometrie courante comme la translation, la rotation mises à l'échelle directement.

De façon plus détaillée, j'ai rencontré différents problèmes. Le premier était d'utiliser GTK3.0+, car ce n'est pas une bibliothèque étudiée en cours. C'etait beaucoup plus compliqué de s'infornmer surtout que la documentation officielle n'est pas très compréhensible et ne donne pas énormement d'informations.

J'ai tout d'abord commencé à réfléchir sur le résultat attendu lors d'une rotation avec un angle précis. Le premier problème était de redimensionner l'image finale car si elle n'est pas redimensionnée, on risque de perdre des informations importantes sur la grille de sudoku (voir image ci-dessous).

Ici, les parties en rouge sont nécessaires pour avoir toutes les informations de l'image originale.

J'ai moi-même essayé d'implémenter la rotation de l'image mais les pixels sont stockés différemment. L'implémentation générale étudiée en classe était un tableau de tableau contenant un élément qui a 3 ou 4 attributs qui étaient RGBA ou RGB. Avec GTK, il y a seulement un tableau donc une implementation en Row-Major order et, où les 3 ou 4 premières valeurs correspondent aux valeurs RGBA du premier pixel et ainsi de suite.

Suite à un temps considérable de recherches, j'ai découvert qu'une fonction intégrée existait déjà avec Cairo, un outils spécialisé dans la manipulation d'image. J'ai donc du passer encore quelques heures sur la faible documentation disponible afin de comprendre son fonctionnement.

A la fin, l'algorithme était assez simple :
— Créer une nouvelle image blanche avec les dimensions finales.
— Coller l'image de base tournée.
Mais, la dernière étape nécessitait une compréhension approfondie du collage de l'image. En effet, pour coller l'image, il fallait s'assurer qu'elle soit bien centrée à la fin en ne perdant aucune information. Cependant, la fonction de collage commençait avec le pixel en haut à gauche de l'image de base. Il a donc fallu calculer les coordonnées de ce pixel à l'image finale afin de bien se placer pour le coller efficacement.

2.3 Détection de la grille et de la position des cases

La détection de la grille se découpe en trois étapes : on commence par aténuer les contours afin de limiter les faux positifs puis, un algorithme de détection de contours est appliqué et enfin, un algorithme permettant de détecter des droites dans les contours précédents. Pour l'aténuation des contours, on fait usage d'un masque discret de 3 par 3 pixels. Les pixels hors de l'image sont systématiquement considérés comme noirs.

Une fois ce filtre appliqué, nous affectons un filtre de Sobel pour détecter les contours. Il est simple à implémenter et donne relativement de bons résultats sur les images testées. Ce filtre consiste en l'application de deux masques sur l'image. Encore une fois, les pixels hors de l'image seront considérés comme noirs et le carré des résultats de ces deux masques est sommé. On récupère ensuite la racine de cette somme que l'on divise par la valeur maximale possible afin d'obtenir un résultat compris entre 0 et 1. Puis, cette valeur est multipliée par 255,9999 puis passée au format d'entier naturel sur 8bits afin d'obtenir une valeur entière entre 0 et 255 sur les trois canneaux rouge, vert et bleu.

Une fois ce filtre appliqué, nous utilisons une transformée de Hough pour détecter les différentes droites qui composent l'image.

2.4 Découpage de l'image

Le découpage de l'image se passe en trois parties : dans un premier temps, la transformée de Hough est appliquée sur l'ensemble de l'image permettant ainsi de détécter les différentes droites composant la grille. Cette opération se fait en passant d'un espace en coordonnées cartésiennes à un espace en coordonnées polaires. Dans cet espace, tous les points composant une même droite sont sur le même point.

Nous pouvons ainsi par des opérations de trigonométrie récupérer les différentes droites composant l'image. Il est à noté que cette technique ne sert pas uniquement à trouver des droites mais également des segments de droites sous leurs formes probabilisées. Le code utilisé pour la transformée de Hough est ici une adaptation de l'implémentation de la célèbre bibliothèque OpenCV. Nous avions commencé avec une implémentation "faite maison". En revanche cette dernière souffrait d'imprécisions ainsi que d'un manque critique de performance. Il fallait de l'ordre de 1 minute et 4Gio de RAM pour détecter les contours d'une grille simple.

Après, nous recherchons les plus petits quadrilatères possibles dans l'image pour ensuite les enregister comme étant les différentes cases du Sudoku recherché pour ce faire, nous trions les droites trouvées entre les droites verticales et les droites horizontales. Ce tri s'opère en appliquant une fonction de seuil sur la composante x du vecteur directeur. En effet plus x se rapproche de la valeur 1, plus la ligne se rapproche de la verticale. Une fois ceci fait, les droites verticales sont triées en fonction de leur valeur de absice et les droites horizontale en fonction de leur hordonée. Une fois ce tris fait les deux listes sont parcourues afin de trouver les cases.

Les cases sont ensuite trouvé en calculant 4 intersections de droite donnant ainsi les coordonée des 4 sommets de la grille.

Le quadrilatère connexe défini par ces 4 points est ensuite extrait de l'image pour être placé sur une image de taille minimal au fond blanc puis sauvegardée. Pour obtenir l'image finale, nous récupérons l'ensemble des pixels de l'image de base compris entre la plus petite et la plus grande ordonnée du quadrilatère ainsi qu' entre sa plus petite et plus grande abscisse.

2.5 Solver sudoku

Hors mis le fait que le sudoku est un jeu de puzzle divertissant, stimulant et particulièrement attrayant une fois qu'on s'y habitue, sa résolution m'a toujours intéressé et intrigué. C'est la raison pour laquelle j'ai pris en charge l'algorithme permettant de le résoudre.

Le principe d'un sudoku est simple. Le joueur a pour tâche de remplir les cases vides avec les chiffres manquants et ainsi compléter la grille en se basant sur la logique. Au début du jeu, la grille de taille 9*9 a des cases qui sont au préalable déjà remplies. Ainsi, la consigne est d'assigner des chiffres compris entre 1 et 9 dans les cases de cette grille qui sont vides. Cependant, il est strictement interdit selon les règles du sudoku d'avoir un doublon sur une ligne, une colonne ou dans un mini carré de taille 3*3. Dans le cas ou un chiffre est déjà présent, on testera avec un autre chiffre si ce dernier est présent. C'est un exercice de réflexion et de logique permettant de faire travailler intensivement le cerveau en stimulant les neurones.

Dès lors, j'ai établi la construction de mon algorithme solver en différentes étapes prédéfinies :

Dans un premier temps, j'ai créé un programme permettant de vérifier si une case est vide dans le sudoku afin d'y inscrire un chiffre compris entre 1 et 9.

```
int find_empty_cell(int* i, int* j, int** sudoku)
```

Ensuite, une fois que nous avons vérifié le nombre de cases vides dans la grille du sudoku, les règles du jeu citées précédemment sont appliquées.

J'ai créé un programme permettant de vérifier si sur chaque ligne, quand je veux ajouter un chiffre, que ce dernier ne soit pas déjà présent. Par exemple, si le chiffre 2 est déjà présent, le programme renverra 0 (faux) sinon 1 (vrai). Si le programme renvoi vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

```
int check_line(int i,int digit, int** sudoku)
```

Par la suite, j'ai créé un programme similaire permettant de vérifier si sur chaque colonne, quand je veux ajouter un chiffre, que ce dernier ne soit pas déjà présent. Par exemple, si le chiffre 2 est déjà présent, le programme renverra 0 (faux) sinon 1 (vrai). Si le programme renvoi vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette colonne.

```
int check_colum(int digit, int j, int** sudoku)
```

Puis, j'ai refait l'étape à l'identique que précédemment mais dans un mini carré (3*3) qui vérifie si sur chaque colonne et chaque ligne lorsque je veux ajouter un chiffre, que ce dernier ne soit pas déjà présent. Par exemple, si le chiffre 2 est déjà présent, le programme renvoi 0 (faux) sinon 1 (vrai). Si le programme renvoi « vrai », alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

```
int check_square(int i, int j, int digit, int** sudoku)
```

Pour finir, j'ai conçu le programme IsValidSudoku qui renvoie vrai si les 3 programmes précédents sont vérifiés. Le sudoku sera alors valide et nous pourrons donc réaliser le solver de

sudoku qui résout le sudoku en entier.

```
int IsValidSudoku(int i, int j, int digit,int** sudoku)
```

```
int solver_sudoku(int** sudoku)
```

Lorsque toutes les fonctions « mathématiques » sont effectuées, il faut prendre en charge la lecture et l'écriture d'un fichier. La lecture permet de récupérer un fichier dans lequel un sudoku incomplet est sauvegardé afin de le lire et le stocker.

```
void readsudo(char* file,int** sudoku)
```

A partir du moment où la lecture est effectuée, je vérifie si le sudoku s'est bien résolu en contrôlant si toutes les cases sont remplies et qu'il y ait bien une solution pour chaque case. S'il s'est bien résolu, j'élabore un nouveau fichier contenant le sudoku résolu en respectant les normes du fichier d'origine. Nous pouvons ainsi trouver un nouveau fichier créé contenant la grille de sudoku remplie. Si le sudoku ne s'est pas résolu, un message d'erreur s'affiche.

Ce nouveau fichier est le fichier d'entrée avec un ajout d'extension « .result ». En affichant le contenu du nouveau fichier, on remarque que les points d'origines ont été remplacés par des chiffres et qu'il n'y ait aucun chiffre en double sur une ligne, une colonne ou un carré de 3*3. Le format de sortie est le même que le format d'origine.

```
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ ls
grid_00  main.c  solver.c  solver.h
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
```

FIGURE 1 – Sudoku initialisé et sauvegardé

```
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ gcc -Wall -Wextra -Werror -pedantic -std=c99 -g3
solver.c main.c -o solver
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ ls
grid_00  main.c  solver  solver.c  solver.h
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ ./solver grid_00
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ ls
grid_00  grid_00.result  main.c  solver  solver.c  solver.h
```

FIGURE 2 – Exécution du programme et du solver permettant d'obtenir un nouveau fichier

```
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ cat grid_00.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
```

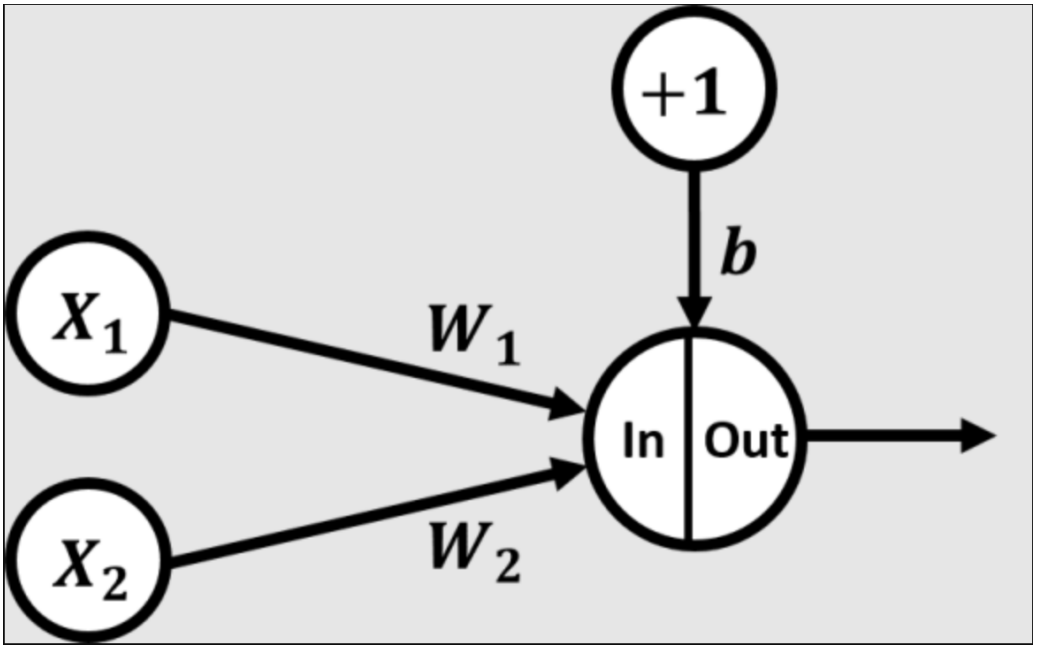
FIGURE 3 – Affichage du nouveau fichier contenant le sudoku rempli et du même format que la première image.

2.6 Réseau de neurones

La réalisation du réseau de neurones se sépare en plusieurs parties :

La première partie est sur la création du réseau de neurones et son code pour l'utiliser.

Nous devons créer le réseau de neurones. Pour cela, je suis passé par un struct qui compose un neurone. Puis, j'ai fait un tableau à deux dimensions qui contient ces neurones qui sont de la forme : (voir image ci-dessous)



Maintenant, nous devons pouvoir utiliser ce réseau de neurones. C'est pour cette raison que j'ai réalisé une fonction. Celle-ci prend un tableau d'entrée et renvoie le résultat du max de la layer une fois softmax appliqué. Pour trouver la valeur des neurones, nous utilisons cette formule :

Somme wi+ai + b avec wi un weight associé au neurone de la layer précédent et le résultat du neurone de la layer précédent. De plus, on rajoute le bias qui est relié à un neurone.

Une fois le calcul réalisé en feedforward, on peut récupérer le résultat.

Pour conclure dans cette partie, nous avons vu comment créer et utiliser le réseau de neurones.

La seconde partie est sur l'apprentissage du réseau de neurones.

Dans cette partie, nous allons voir comment j'ai permis à mon réseau de neurones d'apprendre. Pour ce faire, l'idée principale est de réaliser un coût et de voir à quel point la sortie de notre réseau de neurones est fausse. A cet effet, on utilise cette formule :

Somme 1/2(predicted - Exepted)**2

Le réseau de neurones est bon si cette somme est exacte.

Maintenant, il faut modifier les billets et les poids des réseaux de neurones. Pour ce faire, il faut réaliser un backpropagation sur l'ensemble des réseaux. Ainsi, nous allons effectuer le ratio des dérivées partielles entre l'erreur totale puis le multiplier par le ratio de la sortie et Z et multiplier par le ratio entre Z et le poids ou le billet.

Une fois tous les gradients récupérés qui permettent de savoir à quel point la fonction augmente, nous devons modifier chaque poids et billet par cette formule :

Wi = Wi - learning rate * gradient Wi

Bi = Bi - learning rate * gradient Bi.

Pour diminuer le nombre de calculs, j'ai utilisé la méthode avec des groupes qui permettent de faire la moyenne de plusieurs gradients et de modifier les réseaux de neurones. L'ensemble de ces actions permet d'avoir des réseaux de neurones fonctionnels.

2.7 Utilitaires mis en place

2.7.1 Nix-shell

Un des gros avantages de Nix est de permettre de répliquer entièrement un environnement de développement à l'aide d'un programme appellé "nix-shell". Nous avons donc fait un fichier shell.nix décrivant les bibliothèques requises pour la compilation du code afin de pouvoir, quelque soit le système de compilation (dans la limite des systèmes supportant nix), obtenir une compilation la plus reproductible possible.

Certains utilitaires ont été ajoutés dans ce shell par rapport aux machines de l'école, tel que Bear, un programme permettant de générer des bases de données de compilation à partir des Makefile.

2.7.2 Formatage du code

Le formatage du code est plus que pratique et extrêmement comfortable dans un groupe car l'ensemble des membres du groupe utilise les mêmes conventions. pour le formatage du code. Par ailleurs, nous avons utilisé un fichier clang-format pour décrire le formatage du code souhaité et un script shell qui l'applique à tous les fichiers C du projet. Un fichier éditor config est également mis en place afin d'indiquer quelques règles basiques aux éditeurs de textes utilisés (format de fin de ligne, encodage du fichier, model d'indentation, taille des indentations).

L'ajustement de la taille des lignes pour ne pas excéder les 80 colonnes est effectué par le programme clang-format.

2.8 Problèmes rencontrés au cours du dévellopement

2.8.1 GTK

La bibliothèque GTK est une bibliothèque écrite en C se voulant orientée objet. Le principe de l'orienté objet est très utile. D'ailleurs une surcouche du C, le C++ a été réalisée pour palier à ce manque. En revanche, la bibliothèque GTK ne profite pas de l'implémentation objet du C++ et à la place, il fait une simulation d'héritage à grand renfort de macro et de pointeurs de fonctions.

Les soucis commencent lorsque pour afficher une image, l'image affichée en utilisant GtkImage est trop grande pour la fenêtre et évidemment, GtkImage ne dispose pas d'un système permettant de redimensionner les images à la taille de la fenêtre. Il a donc fallu créer notre propre système qui redimensionne une image. Pour se faire, nous avons derivé la classe GtkDrawingArea pour créer notre class Image, qui a cette propriété de redimensionnement automatique.

Le problème provient de la syntaxe et de la documentation : bien qu'ancienne et progressivement dépréciée pour la version 4, Gtk3 ne dispose que d'une documentation légère sur la façon de procéder pour faire un héritage. De plus, on constate également que la documentation en question n'est pas mise à jour régulièrement car elle utilise des macros dépréciées entre-temps.

En résumé, ce qui en C++ aurait pris 2 voir 3 lignes simples à comprendre, prendra en C plusieurs dizaines de lignes de code sacrifiant ainsi la lisibilité du code.

2.9 Homogénéité du code

Le code est écrit par 4 personnes. Par conséquent, les conventions utilisées par chacun sont différentes. De plus, dans le code d'une même personne, il est également possible de voir des changements majeurs de style, ce qui nuit à la lisibilité du code. Chacune des conventions est raisonnable en revanche leur changement est mauvais.

Pour palier à ce problème, nous avons mis en place les outils de formatage décrits dans la section *2.9 Formatage du code*.

Il est également possible de noter un certains nombres de fonctions qui gagneraient à être séparées en sous fonctions de taille plus raisonnables.

3 Problèmes connus du code

3.1 Fuite de mémoire

Au moment où ce rapport est écrit, il existe une fuite de mémoire importante localisée dans la fonction de rotation de l'image. Il en existe également dans la fonction de transformation de l'image (la fonction fournissant les niveaux de gris, la detection de contours ainsi que la transformée de Hough).

3.2 Usage de fonctions dépréciées

Dans certains morceaux du code, des fonctions dépréciées sont utilisées, faute d'avoir trouver de la documentation indiquant les fonctions "modernes" à utiliser.

Dans ce domaine se trouvent les fonctions *g_type_class_add_private* et *G_ADD_PRIVATE* utilisées lors de la dérivation de la classe *GtkDrawingArea*.

Pour se débarasser de ces avertissements au titre de mesure temporaire, les avertissements relatifs aux fonctions dépréciées ont été désactivés dans le fichier concerné.

3.3 Paramètres non utilisés

Certaines fonctions demandent des paramètres qui ne seront pas utilisés dans le corps de la fonction. Ceci provoque des avertissements inutiles qui ont été désactivés également. Ces fonctions sont dans les fichiers gui/callback.c et gui/img.c. Dans ces deux fichiers, les avertissements relatifs aux fonctions non utilisées ont été désactivés car non significatifs.

3.4 Cases non détéctées

Les cases ne sont que imparfaitement détéctées. La reconnaissance des cases est basée sur la reconnaissance des lignes qui ne fonctionnent pas toujours parfaitement. Donc, sur l'exemple parfaitement net, le programme reconnait actuellement 54 cases sur les 81 composants l'image mais, reconnait parfaitement la grille sur la page de journal. La majorité de la grille est détectée en revanche, la partie située en bas ne passe pas le seuil et n'est donc pas détectée. Sur cette image, le programme repère 45 cases sur les 81 de la grille.

Il est également notable que suivant la rotation de l'image, les cases sont plus ou moins bien détectées.

4 Conclusion

En conclusion, nous constatons que le groupe fonctionne car nous avons su produire une base solide, qui ne demande que des implémentations supplémentaires. De plus, nous sommes particulièrement fiers de notre production commune car le résultat attendu pour cette première soutenance est atteint. Certes, nous rencontrons un petit désagrément concernant la détection de la grille et des cases qui ne fonctionnent pas sur tous les éléments de l'ensemble des tests et les fuites de mémoires mais nous ne considérons pas cela comme un échec mais plutôt comme un petit contre-temps qui sera résolu pour l'ultime soutenance.