



RAPPORT FINAL DE PROJET - PROJET DE S3

SUDOKU SOLVER

2023

CHEF DE PROJET
Lucas ESTRADE.

MEMBRES
Johan BOURDAIS, Arthur FOURCART, Ayoub HAJJI.

TABLE DES MATIÈRES.

1	<i>Introduction</i>	2
2	<i>Présentation de l'équipe SUDO PACMAN</i>	3
2.1	L'Équipe	3
2.2	Répartition des charges	6
3	<i>Nature du projet</i>	7
4	<i>Pré-traitement de l'image</i>	8
4.1	Suppression des couleurs	8
4.2	Rotation de l'image	9
4.3	Filtres	11
5	<i>Traitemet de l'image</i>	13
5.1	Algorithme Carving	14
5.2	Découpage de l'image	26
6	<i>Réseau de neurones</i>	27
6.1	Structure globale	27
6.2	Reconnaissance de chiffre	27
7	<i>Résolveur de sodoku</i>	28
8	<i>Interface graphique</i>	29
8.1	Composition de l'interface	29
8.2	Fonctionnement et construction	31
9	<i>Nos contraintes et nos moyens</i>	35
9.1	Nos contraintes	35
9.2	Outils & Logiciels	36
10	<i>Conclusion</i>	37

1. INTRODUCTION

Dans le cadre de notre seconde année préparatoire à l'école d'Epita, il nous est proposé un projet de trois mois, par groupe de quatre élèves, mettant en pratique nos connaissances et nos compétences acquises lors de nos premières années. La finalité de ce projet est de nous faire réaliser un logiciel de type OCR (Optical Character Recognition ou reconnaissance optique de caractères) ayant pour but de résoudre des grilles de sudoku. Pour y parvenir, le projet a été découpé en quatre catégories. La première catégorie est le pré-traitement d'image, cette partie va nous permettre d'obtenir une image utilisable par notre logiciel à partir d'une image de basse qualité. La seconde catégorie est le traitement de l'image, dans cette catégorie le logiciel va détecter les limites de la grille présente dans notre image et va la découper en plusieurs cases comportant soit un chiffre soit rien. La troisième catégorie est le réseau de neurones, cette partie va nous permettre de faire le lien entre notre image et notre algorithme de résolution de sudoku. Le réseau va lire les cases données par notre traitement et va récupérer l'élément présent à l'intérieur. La dernière partie est le résolveur de sodoku qui est comme son nom l'indique l'algorithme qui est en charge de résoudre les grilles de sudoku.

Pour cette dernière version du rapport de projet, nous débuterons par une présentation des membres du groupe SUDO PACMAN. Ensuite, nous aborderons plus en détail les différentes catégories évoquées précédemment et en quoi elles consistent, puis nous détaillerons chacun le travail fourni pour le projet. Nous continuerons ensuite par rappeler les moyens que nous avons à notre disposition pour réaliser ce projet. Pour finir, nous conclurons par un commentaire global et des commentaires personnels d'expérience pour exprimer notre ressenti.

2. PRÉSENTATION DE L'ÉQUIPE SUDO PACMAN**2.1. L'Équipe****a. Présentation globale**

Notre équipe s'intitule SUDO PACMAN et a pour objectif de réaliser un logiciel de type OCR résolvant des sudokus. Ce groupe est composé de Johan Bourdais, Lucas Estrade, Arthur Fourcart, Ayoub Hajji, quatre élèves provenant tous de la classe C1. Étant dans la même classe l'année dernière, nous avons naturellement décidé de former un groupe tous les quatre.

b. Présentation des membres

Certaines parties ont été conservées du premier rapport.

JOHAN BOURDAIS :

Depuis tout petit, j'ai toujours été curieux du monde qui m'entourait. En grandissant, je me suis rendu compte que l'informatique prenait beaucoup de place dans ce dernier. Je m'y suis donc intéressé et je me suis réellement lancé dans la programmation avec EPITA. J'appréhende beaucoup ce projet car je suis en charge du réseau de neurone et je n'ai aucune connaissance en terme de machine learning. Mais cela ne me démotive pas car c'est un domaine qui m'intéresse beaucoup.

LUCAS ESTRADE :

J'ai commencé la programmation en apprenant le langage HTLM et CSS en seconde, puis j'ai pratiqué un petit peu de python dans ma spécialité de première et terminale, Sciences de l'Ingénieur. Ces pratiques du code m'ont amené à rechercher une école d'informatique pour mes études supérieures, et c'est ainsi que je suis tombé sur EPITA durant un salon étudiant. J'ai candidaté pour l'école, et j'ai été fort-heureusement été admis. Je suis globalement quelqu'un de très passionné et j'aime beaucoup m'investir à fond dans ce qui m'intéresse, la programmation en fait partie, certes, mais les bus, les métros, ou encore le tennis, sont parmi mes plus grandes passions. Cela fait donc quelques années que j'ai un penchant pour l'informatique, et, ce projet, qui représente quelque chose de très nouveau pour moi comparé à tout ce que j'ai pu faire, m'excite et m'intrigue tout particulièrement.

ARTHUR FOURCART :

J'ai toujours voulu construire tout ce qui me passait par la tête. J'ai très vite compris que pour réaliser mes projets les plus amusants, j'allais devoir passer par de la programmation. C'est pour cette raison que j'ai décidé de rejoindre EPITA sans aucune expérience en programmation. Dans ce début de projet de résolution de sudoku, j'ai eu quelques problèmes de mauvaise gestion de mon temps mais j'ai quand même pu aider sur le réseau de neurones. Je ne me démotive pas pour autant la partie du réseau de neurones m'intéresse fortement et je compte bien avancer dessus pour la prochaine soutenance.

AYOUB HAJJI :

Je suis un passionné d'apprentissage, toujours avide d'acquérir de nouvelles compétences et expériences. Mon intérêt pour le travail en équipe m'a conduit à rejoindre l'EPITA, une école qui valorise la collaboration. Dans ce projet Sudoku, j'ai joué un rôle essentiel dans le prétraitement d'images, où j'ai acquis des compétences précieuses liées à la manipulation d'images et à l'optimisation des performances. Mon rôle consistait à préparer l'image d'entrée de manière à ce qu'elle puisse être traitée efficacement par les étapes ultérieures de notre algorithme de résolution

2.2. Répartition des charges

Membre	Lucas	Johan	Arthur	Ayoub
Tâche				
Chargement d'image				RESP.
Algorithmes de prétraitement d'image				RESP.
Rotation de l'image				RESP.
Détection de la grille et des cases	RESP.			
Découpage de l'image	RESP.			
Résolution de sudoku		RESP.		
Réseau de neurones		RESP.		
Interface utilisateur			RESP.	

3. NATURE DU PROJET

L'objectif de ce projet est de créer un logiciel de type OCR pour *Optical Character Recognition* ou Reconnaissance Optique de Caractères en français, qui a pour but de résoudre une grille d'un jeu japonais millénaire mondialement connu, le sudoku.

Ce projet présente différents enjeux, notamment celui de réaliser une intelligence artificielle capable de reconnaître des chiffres, l'OCR donc, également celui du traitement d'image, en incluant les algorithmes de prétraitement et de filtrage d'image, et en incluant de même la reconnaissance de forme sur une image. En outre, une interface graphique viendra chapeauter le tout afin de former un ensemble cohérent.

Parmi toutes ces notions, plusieurs tâches très importantes seront à réaliser, telles que la suppression des couleurs, les différents algorithmes de prétraitement, la détection de la grille de sudoku sur l'image ainsi que ses cases. S'ensuivra la détection des chiffres dans les cases en vue de l'étape suivante, la résolution de la grille. Une fois la grille résolue, la réponse doit être rendue à l'utilisateur.

Le travail de groupe tient une place prépondérante, étant donné le nombre conséquent de tâches à réaliser.

4. PRÉ-TRAITEMENT DE L'IMAGE

Dans le cadre de notre projet visant à développer une application en langage C capable de résoudre des grilles de sudoku à partir d'images, une phase se démarque des autres, pour son importance au bon fonctionnement du projet. Cette phase est la phase de pré-traitement de l'image. Cette phase consiste à préparer l'image d'entrée de manière à ce qu'elle puisse être traitée efficacement par les étapes ultérieures de notre algorithme de résolution. Cette étape est essentielle pour garantir la précision et la rapidité de notre application.

4.1. Suppression des couleurs

a. Filtrage en niveaux de gris

L'étape cruciale du filtrage en niveaux de gris dans notre processus de prétraitement joue un rôle fondamental dans la simplification de la représentation de l'image tout en préservant les caractéristiques essentielles pour la détection de la grille Sudoku. Nous adoptons une approche particulièrement raffinée, utilisant la formule de conversion pondérée des canaux RVB vers niveaux de gris : $p(x,y) = p(x,y)r * 0.3 + p(x,y)g * 0.59 + p(x,y)b * 0.11$. Cette formule, basée sur les contributions relatives des composantes rouge (r), verte (g), et bleue (b) à la perception humaine, permet de calculer la valeur de chaque pixel ($p(x,y)$) de manière équilibrée. En attribuant des poids spécifiques à chaque canal, la formule produit une représentation en niveaux de gris qui préserve les contrastes tout en atténuant les variations de couleur. L'application de cette formule s'inscrit dans une démarche visant à simplifier la complexité de l'image, favorisant ainsi la détection ultérieure des contours de la grille. En choisissant judicieusement les coefficients de pondération, nous avons optimisé la conversion en niveaux de gris pour garantir une représentation fidèle des informations essentielles de l'image. Cette approche sophistiquée contribue à préserver les caractéristiques distinctives du Sudoku, facilitant ainsi les étapes subséquentes du prétraitement et de la résolution automatique du puzzle.

b. Binarisation de l'image

La phase de binarisation, un pilier essentiel de notre processus de prétraitement, repose sur une approche sophistiquée exploitant l'histogramme de luminance. Cette méthode permet de transformer l'image en une version binaire, où les pixels sont classés en deux catégories distinctes : noir ou blanc, facilitant ainsi la détection ultérieure des éléments de la grille Sudoku. L'histogramme de luminance offre une représentation graphique de la distribution des niveaux de luminance dans l'image. Notre approche consiste à analyser cet histogramme afin de déterminer le seuil de binarisation optimal. La méthode repose sur la recherche du point de transition dans l'histogramme, où les pixels de faible luminance (fond) et ceux de luminance élevée (contours de la grille et chiffres) se distinguent nettement. Une fois identifié, ce seuil est utilisé pour diviser l'image en deux catégories, créant ainsi une version binaire où les pixels au-dessus du seuil sont considérés comme blancs et ceux en dessous comme noirs. Cette approche adaptative de la binarisation garantit une meilleure adaptabilité aux variations d'éclairage et aux nuances présentes dans différentes images de Sudoku. Elle permet également de contourner les défis liés à des conditions d'éclairage inégales ou à des contrastes variables. La binarisation par histogramme de luminance offre ainsi une solution robuste, contribuant à la création d'une représentation binaire de la grille Sudoku qui préserve les détails tout en simplifiant l'analyse ultérieure. En résumé, l'utilisation de l'histogramme de luminance pour la binarisation représente une stratégie avancée dans notre processus de prétraitement, visant à optimiser la distinction entre les éléments de la grille et le fond. Cette méthode contribue significativement à la précision globale du système, assurant une qualité optimale de l'image binaire pour les étapes subséquentes de résolution automatique du Sudoku.

4.2. Rotation de l'image**a. Rotation manuelle**

Une étape cruciale de notre processus de prétraitement de l'image consistait à garantir que l'image de la grille de sudoku soit correctement orientée pour faciliter la détection des

cellules de la grille et la reconnaissance des chiffres. À cette fin, nous avons développé un programme dédié permettant aux utilisateurs de faire pivoter manuellement l'image au besoin. Le programme de gestion de la rotation manuelle offre une solution interactive pour les cas où l'image d'origine pourrait ne pas être parfaitement alignée. Les utilisateurs peuvent ajuster la rotation de l'image jusqu'à ce qu'elle soit correctement orientée, ce qui est essentiel pour garantir la précision de la reconnaissance des chiffres et la résolution de la grille. Cette fonctionnalité offre une grande flexibilité et permet à notre application de s'adapter à diverses orientations d'images d'entrée.

b. *Rotation automatique*

La phase de rotation automatique constitue une étape cruciale du processus de prétraitement, visant à aligner correctement la grille Sudoku détectée. Pour atteindre cette orientation optimale, nous avons mis en œuvre une méthode de détection d'angle automatique. Cette approche innovante repose sur l'identification des quatre coins de la grille. Nous calculons ensuite les points médians des côtés inférieurs et supérieurs, déterminant ainsi une ligne reliant ces deux points médians. En utilisant la fonction arctangente (atan), nous calculons l'angle de cette ligne par rapport à l'axe horizontal. La rotation nécessaire est ensuite appliquée à l'image en ajustant l'orientation de la grille Sudoku à un angle optimal. Cette méthode garantit une correction précise de l'orientation, permettant ainsi une détection et une résolution ultérieures précises du Sudoku.

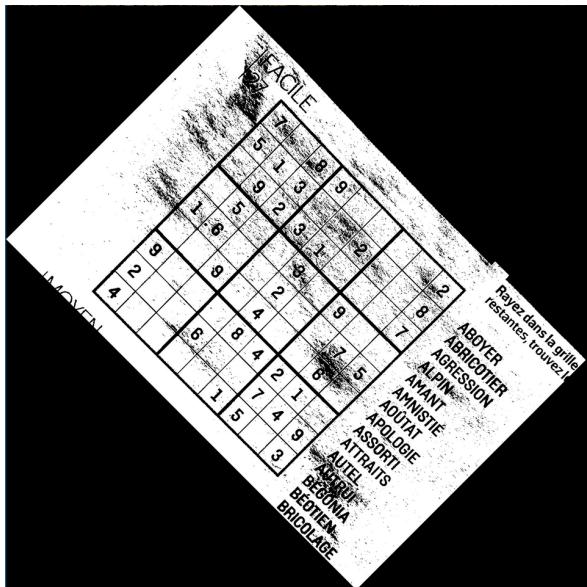


Figure 1: Grille testée avant rotation

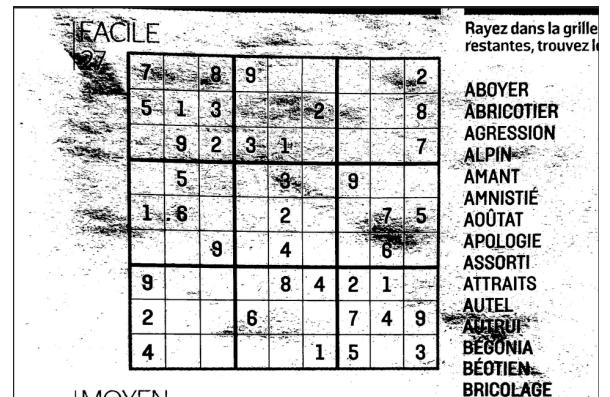


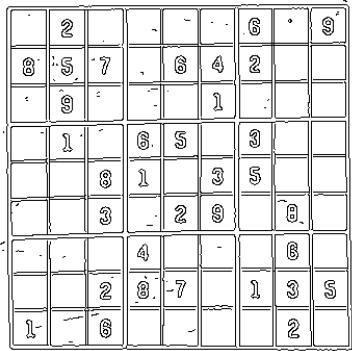
Figure 2: Grille testée après rotation

4.3. Filtres

a. Filtre Canny

La phase de détection de la grille repose sur l'implémentation soigneuse du filtre Canny, une technique avancée de détection de contours. Le filtre Canny, développé par John Canny en 1986, s'est établi comme une méthode robuste pour extraire les contours significatifs d'une image. Cette méthode opère en plusieurs étapes, débutant par l'application d'un lissage gaussien afin de réduire le bruit et d'atténuer les détails indésirables de l'image. La deuxième étape du filtre Canny consiste en le calcul des gradients de l'image. Cette opération met en évidence les variations d'intensité en calculant la magnitude du gradient, indiquant l'intensité du changement, et en déterminant la direction du gradient, révélant la direction de la variation. Ces informations sur les gradients sont essentielles pour la détection précise des contours. Cependant, le filtre Canny va au-delà d'une simple détection de contours. Il intègre un mécanisme de suppression des non-maxima, éliminant ainsi les pixels qui ne constituent pas des maxima locaux le long des directions des gradients. Cette étape contribue à affiner la précision de la détection des contours en ne conservant que les points les plus significatifs. Une composante cruciale de notre implémentation

résidé dans le réglage minutieux des paramètres du filtre Canny. Nous avons adapté les seuils hauts et bas pour optimiser la détection des contours de la grille, assurant une sensibilité suffisante pour repérer les lignes tout en minimisant les faux positifs. La détection précise des contours par le filtre Canny est essentielle pour la rotation automatique ultérieure, garantissant que l'angle de rotation est calculé avec une grande précision. En résumé, l'utilisation du filtre Canny dans notre processus de prétraitement ne se limite pas à une simple détection de contours, mais offre une approche exhaustive pour identifier les caractéristiques significatives de la grille Sudoku. Son intégration judicieuse constitue un élément clé pour la réussite du prétraitement, contribuant à l'exactitude globale du processus d'analyse d'image en vue de la résolution automatique du Sudoku.



b. Contraste Stretching

L'application du contraste stretching constitue une étape cruciale dans le processus de prétraitement, visant à améliorer la visibilité des détails au sein de l'image du Sudoku. Cette technique s'appuie sur la dynamique des niveaux de gris pour étendre la plage d'intensités présente dans l'image, garantissant ainsi une meilleure distinction entre les éléments de la grille et le fond. Mathématiquement, le Contraste Stretching peut être exprimé par la formule suivante, où r représente la valeur d'intensité actuelle du pixel, (r_{min}) est la valeur minimale d'intensité présente dans l'ensemble de l'image, et (r_{max}) est la valeur maximale d'intensité présente dans l'ensemble de l'image :

$$s = 255 * \frac{(r - r_{min})}{(r_{max} - r_{min})}.$$

5. TRAITEMENT DE L'IMAGE

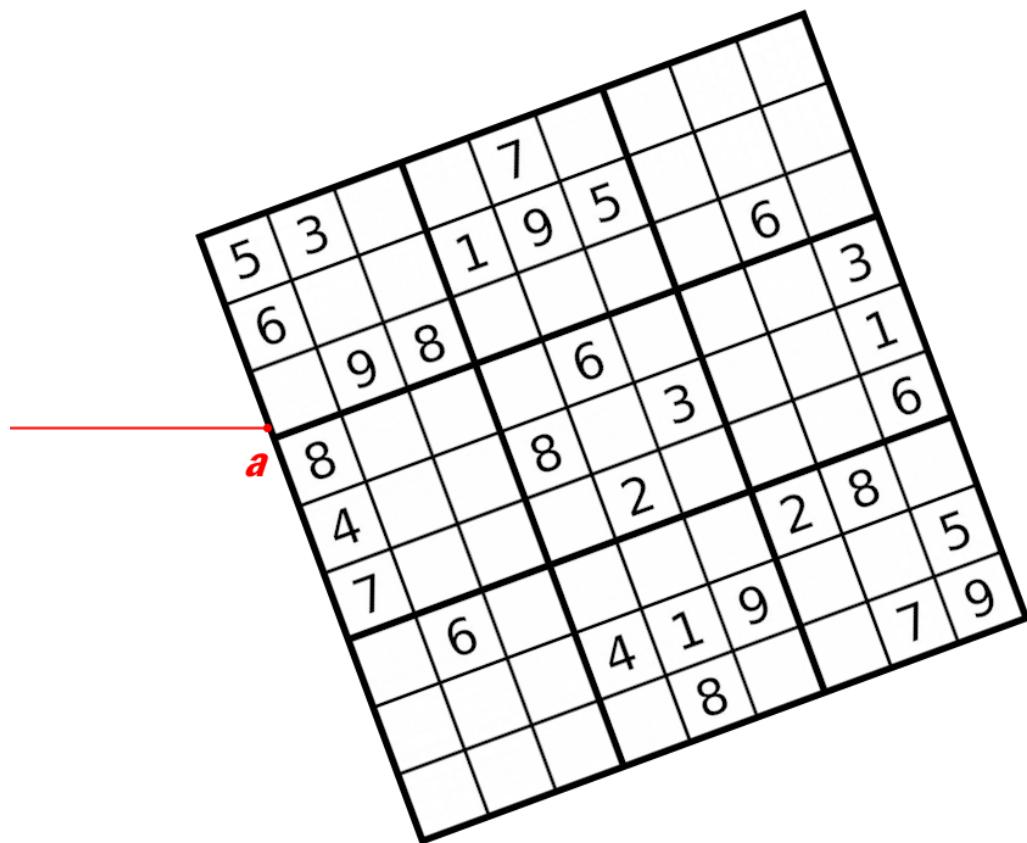
Tout l'algorithme est présent dans le fichier "carving.c". Ce dernier est expliqué dans l'ordre chronologique de sa construction. La première version a été implémentée pour la première soutenance et sera détaillée avec précision dans le premier chapitre, puis la version finale de l'algorithme avec toutes ses améliorations et ses changements sera détaillée dans le second chapitre.

Le traitement de l'image est une étape très importante. Dans notre cas, elle consiste en l'implémentation d'algorithmes de détection de formes sur une surface **SDL**. La philosophie de ce travail a été dès le départ de concevoir un algorithme sans se baser sur un préexistant. On peut alors d'ores et déjà affirmer que le résultat ne sera pas aussi satisfaisant et sophistiqué que celui d'un algorithme professionnel, mais la volonté de garder un algorithme fait main a été gardée jusqu'à la fin du projet. Cet algorithme a donc été conçu pour fonctionner uniquement avec des grilles de sudoku (et plus généralement des formes rectangulaires). Le découpage fait parti intégrante de cet algorithme, à l'aide de quelques formules mathématiques simples pour calculer la position des cases. Les deux parties suivantes s'engagent à décrire de la manière la plus claire possible le fonctionnement de l'algorithme *Carving* pensé pour le projet.

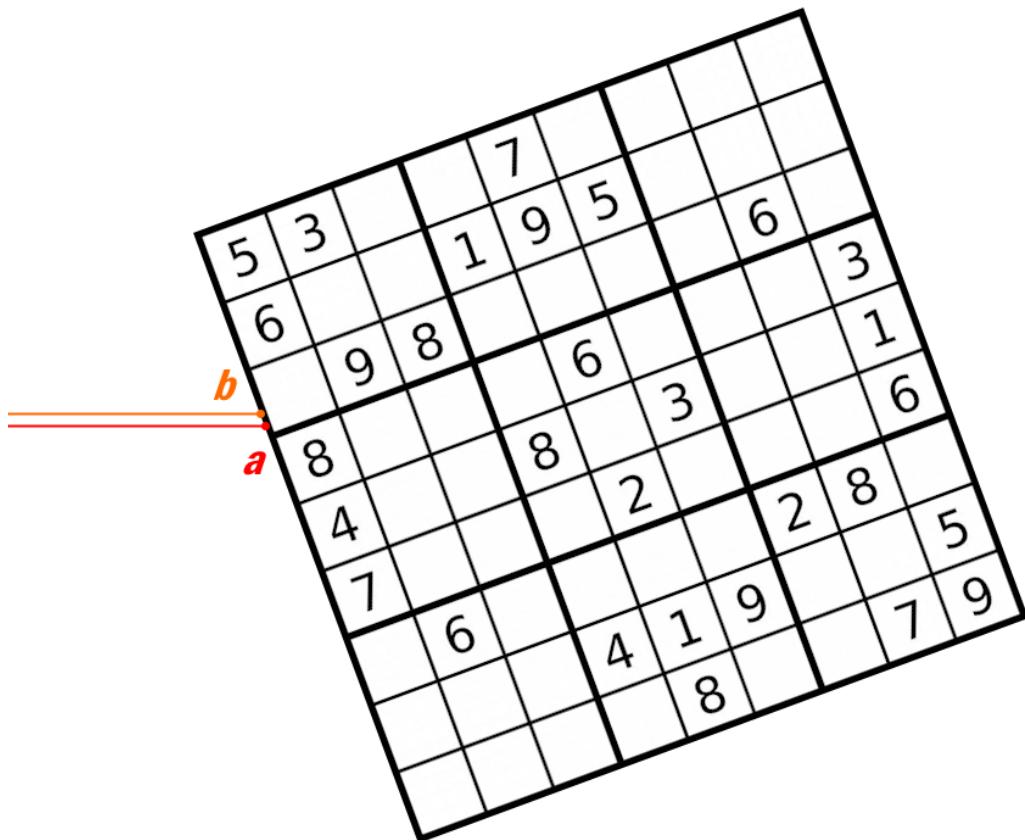
5.1. Algorithme Carving

a. Première version de l'algorithme Carving

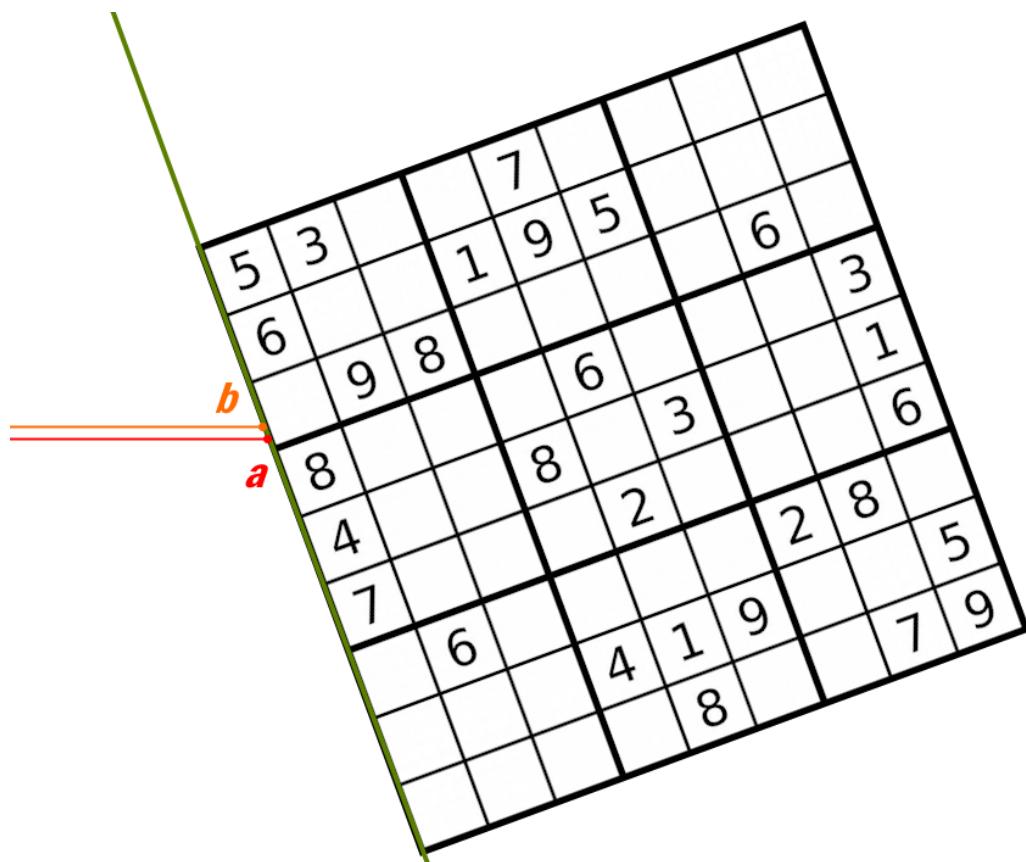
Pour cette première version, nous avons utilisé une méthode simple. Le problème de celle-ci est qu'elle nécessite un prétraitement parfait, car l'algorithme ne fonctionne pas si il existe une imperfection sur l'image ou si la grille est trop petite dans un coin de l'image. Cependant cet algorithme est capable de détecter n'importe quelle forme géométrique à 4 côtés, et d'en extraire un quadrillage de 9 cases par 9. Ce qui est donc très pratique pour les grilles de sudoku. Il fonctionne de la manière suivante. Nous partons du milieu gauche de l'image, et nous projetons le plus loin possible jusqu'à croiser un pixel noir (on note ce point *a*).



Une fois que ce dernier est trouvé, nous partons du même endroit 10 pixels plus haut et nous répétons la même fonction (on note ce deuxième point *b*).

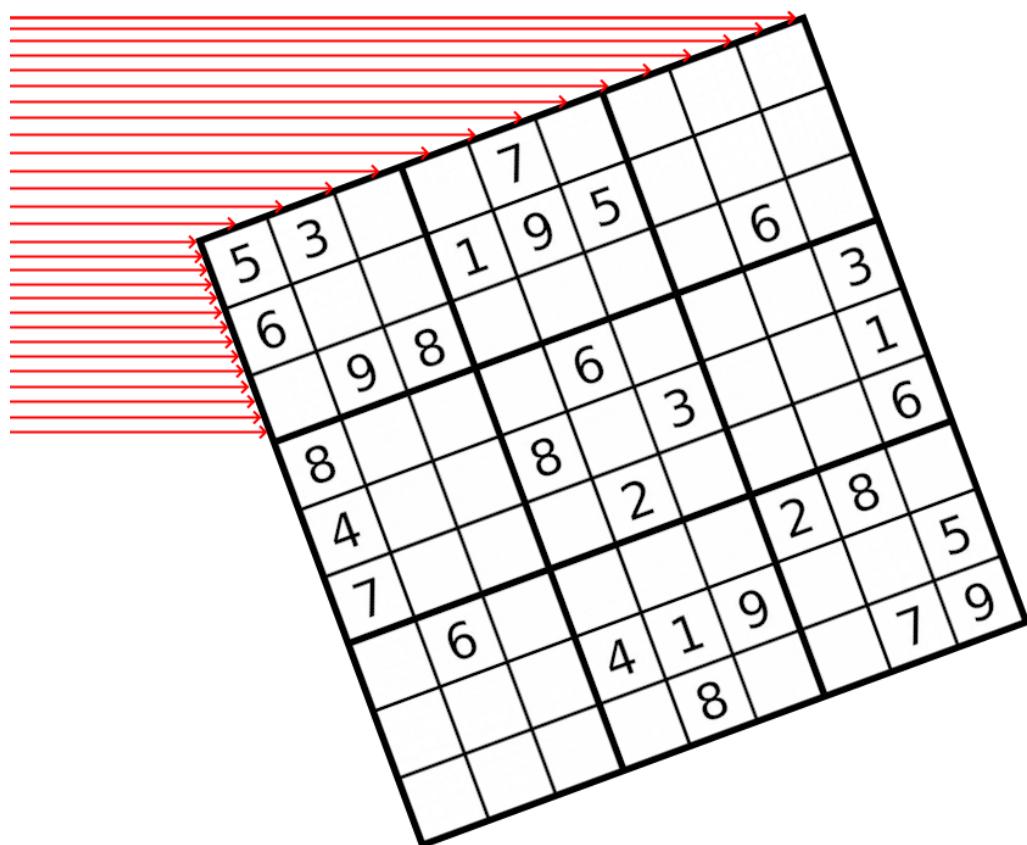


En théorie, si l'image est bien nettoyée, nous obtenons donc avec certitude 2 points distincts du côté gauche de la grille. Grâce à ces deux points on peut alors déterminer si on a affaire à une pente montante ou descendante, ou une ligne droite, mais ce cas est inclus dans un des deux autres.



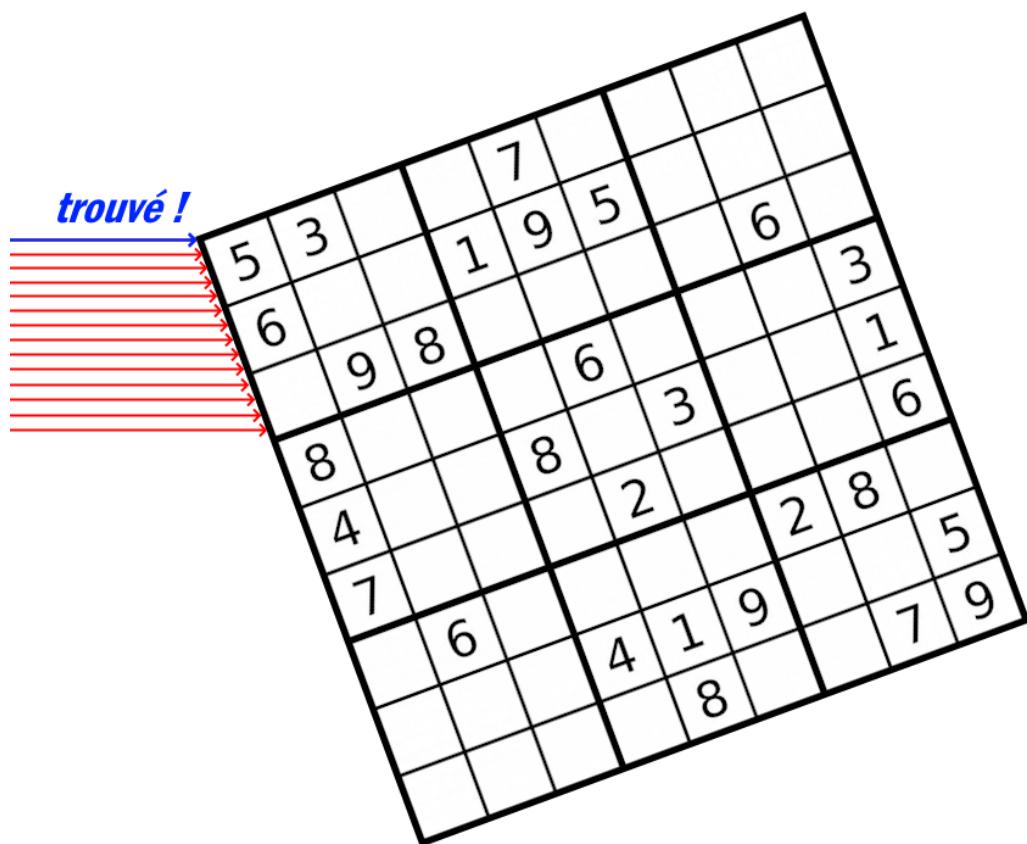
On voit ci-dessus que ce cas présente une pente descendante.

Le but, à partir de cette étape, est d'accéder aux 2 extrémités du côté gauche. Ici nous avons affaire à une pente descendante, alors, pour le bord supérieur gauche, nous parcourons la matrice de pixel à partir du milieu gauche (donc $y = \text{hauteur} / 2$ et $x = 0$). L'algorithme peut être schématisé comme présenté ci-dessous.



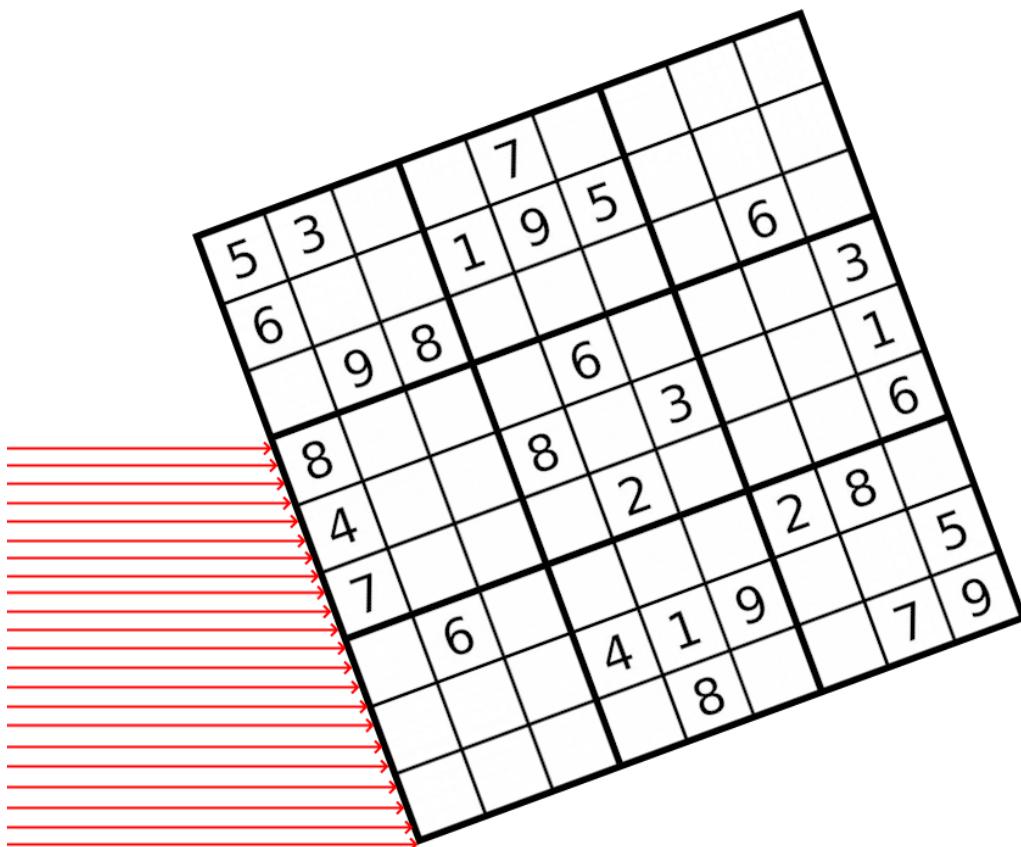
Ceci est une façon de schématiser l'algorithme, cependant, il est important de rappeler que chaque ligne de pixel est parcourue par le programme, jusqu'à croiser un pixel noir.

À la suite de cette étape, à chaque ligne de pixels, si nous croisons un pixel noir, si la coordonnée en x de ce pixel se trouve être moins élevée que le pixel avec la coordonnée en x la plus basse précédent, alors nous le remplaçons par celui là, et ainsi de suite jusqu'à arriver en haut de l'image. Cela donne le résultat suivant.



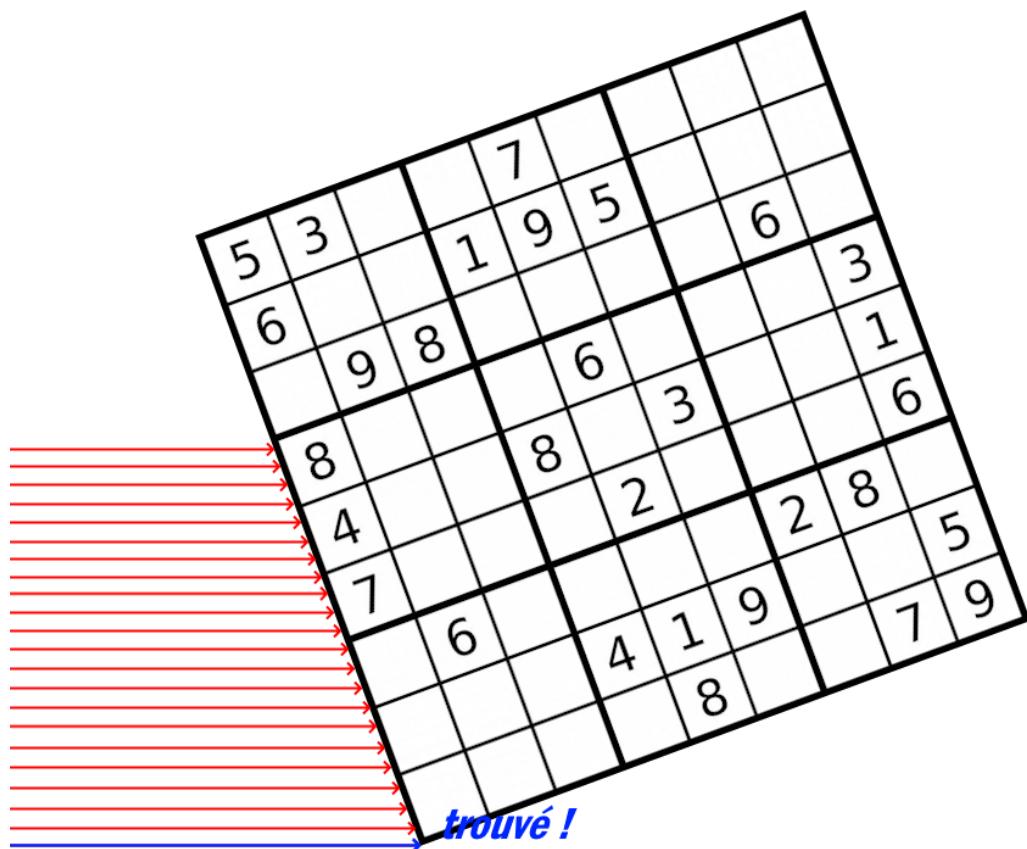
Le pixel noir le plus à gauche est récupéré, ici, il s'agit du bord haut gauche de la grille

En revanche pour le bord inférieur gauche, nous partons encore une fois du milieu gauche, sauf que cette fois ci l'objectif est juste de récupérer le premier pixel noir sur la dernière ligne où il y en a. On peut schématiser cette étape comme ceci.



Encore une fois, ceci est un schéma, toutes les lignes de pixels sont parcourues et pas seulement celles montrées ici en rouge.

Comme le côté bas de la grille est forcément en biais, et en quelque sorte "cachée" derrière le bord, le fait de récupérer le dernier pixel noir en descendant ne pose pas problème ici car il n'y a aucun risque que la détection soit faussée par le bord bas, contrairement au coin supérieur où il fallait s'arrêter sur le pixel le plus à gauche. On obtient alors le résultat suivant.

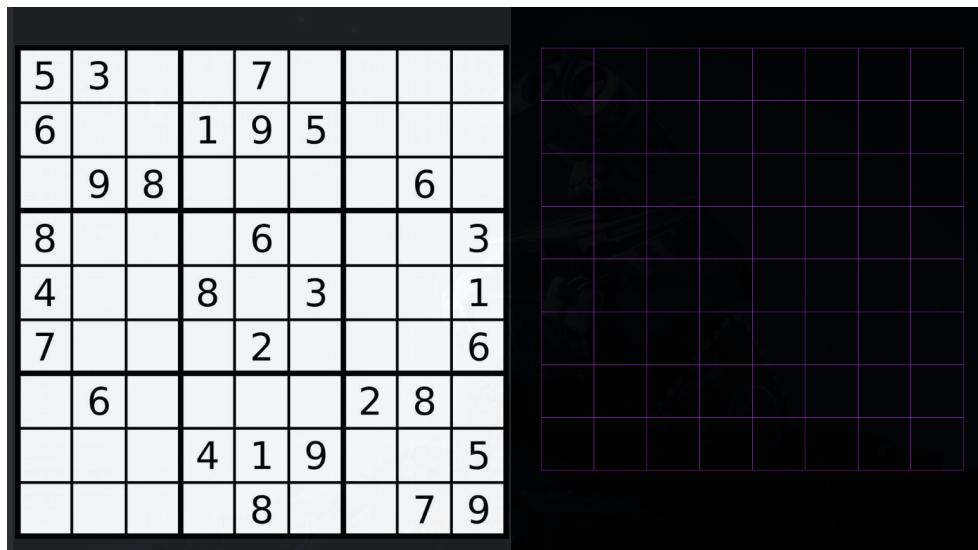


On garde en mémoire le dernier pixel noir croisé en descendant vers le bas.

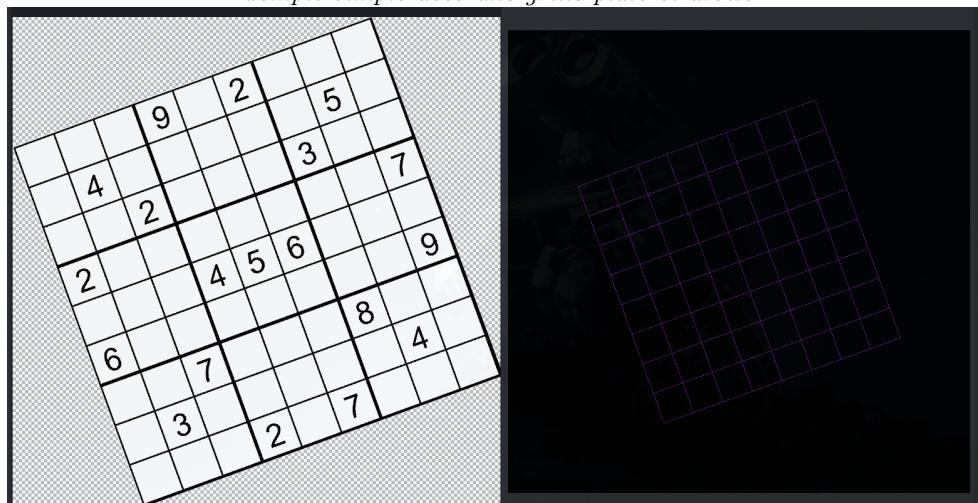
Grâce à ces 2 bords que nous récupérons, nous pouvons tracer le côté gauche de la grille.

Le côté droit fonctionne exactement de la même façon, il est inutile de le détailler à nouveau, à ceci près qu'il faut inverser les méthodes de détection des bords selon l'inclinaison de la pente. Ainsi, tous les côtés de la grille sont reliés. Ensuite chaque côté est divisé en 9 sous-segments, et les points séparant les sous-segments sont reliés à leurs opposés, qui se trouvent, vous l'aurez deviné, sur le côté opposé. Un segment ab se divise en 9 avec la formule suivante : pour la coordonnée x , $a + (i / 9) * (b - a)$, pour i allant de 0 à 9. La coordonnée y est calculée identiquement. Ensuite l'algorithme fonctionne toutes les intersections du quadrillage créé. Ces dernières sont calculées sur le même principe. Elles seront très utiles pour le découpage.

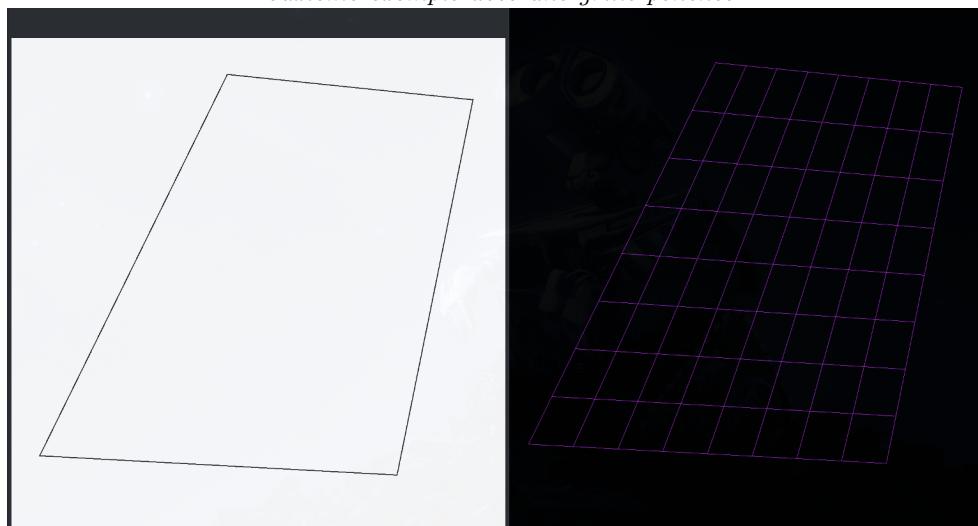
Quelques exemples de cette première version page suivante :



Exemple simple avec une grille plate et droite.



Deuxième exemple avec une grille penchée.



En effet, l'algorithme est également capable de découper un quadrillage en 9x9 d'un quadrilatère relativement déformé, si la photo de la grille a été prise de côté par exemple.

b. Deuxième version de l'algorithme Carving

La première version de l'algorithme permettait de détecter, avec une bonne efficacité, un quadrilatère et d'en extraire un quadrillage de 9 par 9. Cependant, il apparaissait alors un problème évident, si il y a la moindre petite imperfection sur l'image, ce qui est, pour ainsi dire, inévitable, l'algorithme n'est plus capable de détecter les bordures de la grille et donc ne fonctionne plus. Il était donc indispensable de lui apporter quelques améliorations.

La première idée a été de sélectionner plus précisément les pixels noirs lors du parcours de l'image. Auparavant, même s'ils étaient isolés, ils pouvaient être considérés comme des pixels faisant partie d'une des bordures de la grille, alors qu'évidemment, ça ne peut pas être le cas.

C'est donc le moment d'introduire la fonction `finder()`. Elle a pour but d'éliminer les pixels trop isolés, ou des pixels qui ne font pas parti d'une bordure. Pour cela, la fonction récupère autour du pixel sélectionné une fenêtre de 5 de largeur et de 11 de hauteur. Ceci est possible grâce à l'algorithme Canny qui est appliqué en prétraitement de la fonction et qui filtre les bordures de l'image de façon à les faire mieux ressortir, ici, les bordures sont d'un pixel de large dans la grande majorité des cas. Étant donné que les bordures de la grille détectées, celles de côté, sont verticales, les dimensions de la fenêtre qui sont plus hautes que larges sont relativement adaptées. La fonction consiste simplement à vérifier un certain nombre de propriétés concernant cette fenêtre de 5 par 11 pixels. Elles sont les suivantes.

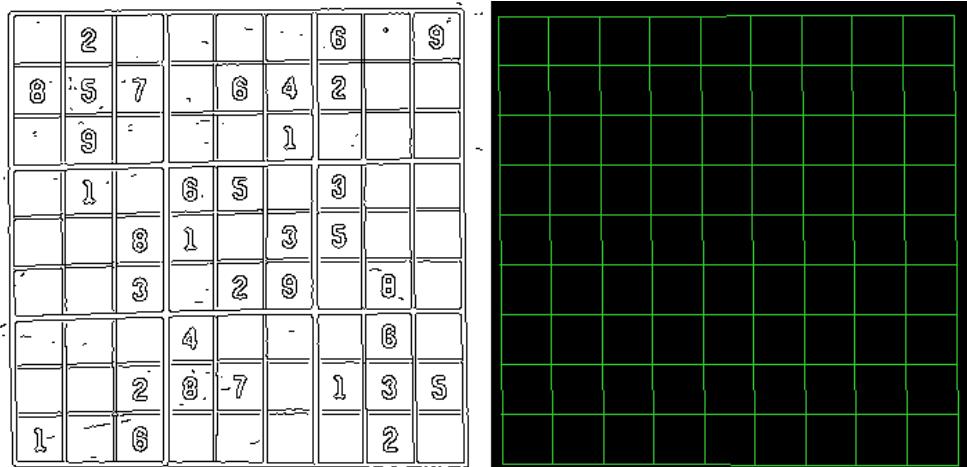
- Pas plus de 3 colonnes différentes de la moitié supérieure ou inférieure de la fenêtre peuvent posséder un pixel noir. (*Le cas des 2 moitiés est traité séparément*)
- Si il y a 2 ou 3 colonnes différentes de la moitié supérieure ou inférieure qui possèdent un pixel noir, alors il faut qu'elles soient côte à côte.
- Le nombre total de pixels dans la moitié haute ou basse doit être au minimum de 4, mais ne doit pas excéder les 6.
- Au moins 5 lignes contenant un pixel noir doivent être comptées.

Grâce à ces conditions, la fonction de recherche des bordures ne s'arrête plus sur n'importe quel pixel noir, mais seulement sur certains d'entre eux avec un paterne précis.

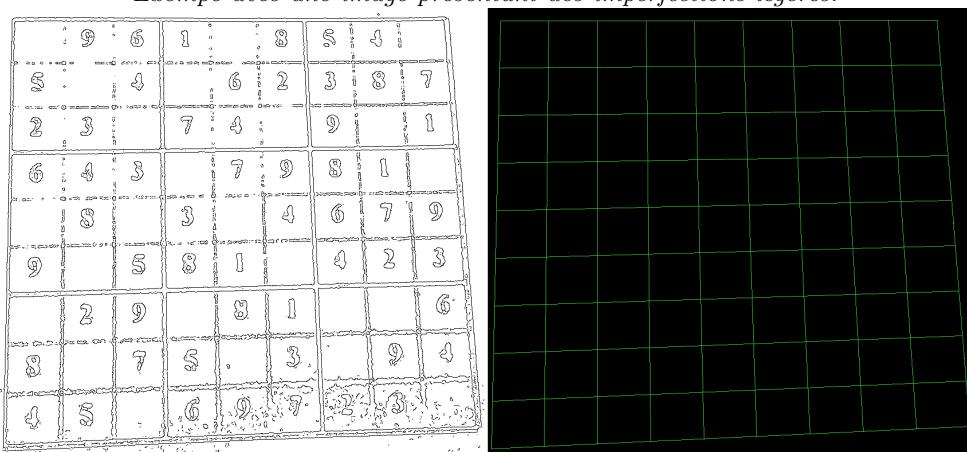
De plus, les conditions de déplacement ont été modifiées. En effet, si on se replace dans le cas de la première version de l'algorithme, la première étape qui consistait à déterminer la pente de la bordure était souvent source de problème car elle faisait varier les conditions de récupération des bordures. Désormais dans cette version 2, lorsque l'on veut récupérer le bord bas gauche par exemple, on se déplace juste qu'en bas jusqu'à trouver un pixel noir qui vérifie la fonction `finder()` et que la différence de la coordonnée en x de l'ancienne valeur et la nouvelle ne dépasse pas 2. On effectue ce calcul grâce au calcul de la différence des deux passée ensuite en valeur absolue. En appliquant ce principe à chaque bord de la grille sans faire de distinctions, on arrive à des résultats satisfaisants. Le principe est que, on ne changera jamais de pixel de bordure si la différence avec celui que nous avions en mémoire est supérieure à 2, autrement dit, nous sommes presque sûrs de retomber sur un pixel de la bordure, et non pas celui d'une imperfection de l'image.

Grâce à toutes ces améliorations, l'algorithme est un petit peu plus efficace et permet de détecter une grille même quand l'image présente quelques saletés.

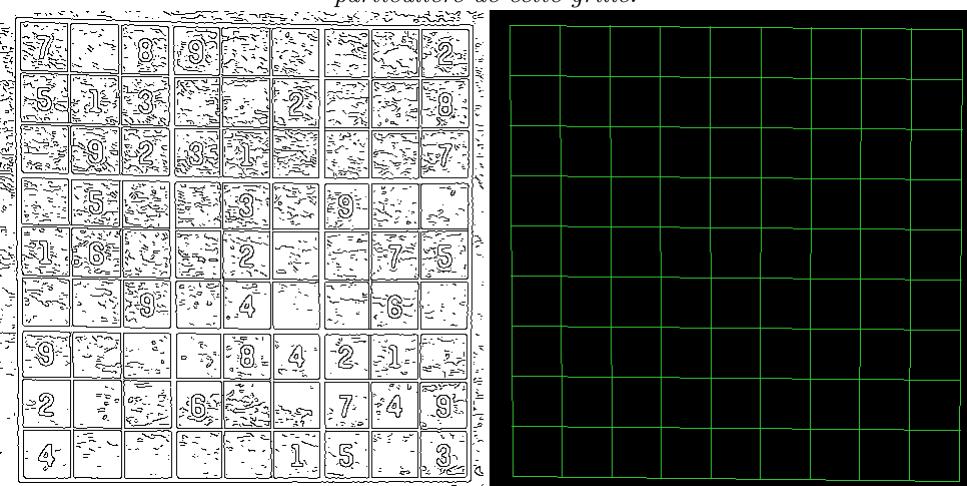
Quelques exemples sont donnés page suivante. La grille de gauche n'est pas la grille originale, c'est une grille rognée et passée par le filtre Canny, implémenté dans le projet pour l'occasion.



Exemple avec une image présentant des imperfections légères.



Deuxième exemple avec une grille légèrement inclinée. La zone de détection est bien identique à la forme particulière de cette grille.



Dernier exemple et sans aucun doute le plus intéressant, sur une image présentant de très nombreux défauts, l'algorithme est tout de même capable de détecter les bordures.

5.2. Découpage de l'image

Comme nous possédons chaque intersection du quadrillage obtenu à la fin de la détection, le découpage se fait en parcourant le tableau à 2 dimensions des intersections. Ensuite nous récupérons les coordonnées x et y maximums de ces 4 points pour ne pas couper la case si celle-ci n'est pas droite, on crée une nouvelle surface grâce à ces nouvelles coordonnées, puis nous copions le contenu de l'image source entre les x et y maximums dans la nouvelle surface. Cette nouvelle surface correspond alors à une case, et elle est sauvegardée avec ses coordonnées x et y comme suit : "*tile_xy.png*", en coordonnées matricielles. Cette étape est répétée pour toutes les cases et elles sont rangées dans le dossier *grid/*. Cette méthode permet d'avoir des résultats satisfaisants même si la grille n'est pas droite du tout, cependant les résidus de bordure de grille sont inévitables.

6. RÉSEAU DE NEURONES

6.1. Structure globale

Pour le réseau de neurones, nous avons globalement gardé la même implémentation que XOR avec un réseau à 3 couches et une fonction sigmoid et la dérivée de la fonction sigmoid. Il est découpé en différentes parties, une partie propagation avant, qui va avoir pour but de prédire une valeur de sortie, ensuite une propagation arrière qui va servir à corriger les éventuelles erreurs du réseau en ajustant les poids et biais de chaque couche du réseau.

6.2. Reconnaissance de chiffre

À la différence du XOR, pour la reconnaissance de chiffre, notre réseau prend un image en paramètre de taille 28*28. Cela oblige donc notre première couche du réseau à avoir 784 neurones. Notre troisième couche aussi augmente à 10 neurones. Pour la couche intermédiaire, nous avons choisi de mettre 450 neurones à l'intérieur car lorsque notre réseau était fini nous avons testé avec de nombreuses valeurs différentes (64,100,200,350,600,...) et c'est sur 450 neurones dans la couche intermédiaire que nous avons eu les meilleurs résultats lors de l'apprentissage. Nous avons aussi fait varier le taux d'apprentissage pendant l'entraînement et nous sommes arrivé sur un taux d'apprentissage de 0.01. Nous avons aussi ajouté le fait que lorsque nous entraînons notre réseau, nous pouvons sauvegarder les poids et biais du réseau dans un fichier texte ce qui permet de ne pas à avoir à entraîner notre réseau à chaque utilisation.

7. RÉSOLVEUR DE SODOKU

Le solveur a été implémenté via l'utilisation du "backtracking", c'est une méthode dite de "brute force" qui va tester toutes les possibilités par récursion et ainsi trouver une solution.

Notre solver reconstruit maintenant une image avec les nouveaux chiffres en rouge et les anciens en noir.

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

Figure 3: Grille testée

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

Figure 4: Grille résolue

8. INTERFACE GRAPHIQUE

L'interface graphique est un autre élément clef de notre projet. Celui-ci aura pour but de lier toutes nos différentes parties évoquées précédemment de façon simple et efficace.

8.1. Composition de l'interface

a. Fenêtre d'accueil

Nous avons décidé de réaliser une interface à deux fenêtres.

La première fenêtre est une fenêtre d'accueil, elle est assez simple avec seulement le titre du projet, le nom de notre équipe, un bouton pour continuer et un bouton pour fermer notre fenêtre.



Figure 5: Fenêtre d'accueil

b. Fenêtre de traitement

La seconde fenêtre est beaucoup plus complète car c'est dans celle-ci que nous réaliserons tout notre traitement de résolution de grille de sudoku. Cette fenêtre est composée d'un espace d'affichage d'image, d'un bouton de recherche de fichier pour récupérer notre grille à

traiter, une option de rognage de l'image, les options de pré-traitement de l'image, l'option de la détection de grille, les options pour résoudre notre grille, une description de comment utiliser notre résolveur et puis un bouton pour fermer notre interface et un bouton pour revenir à la fenêtre d'accueil.

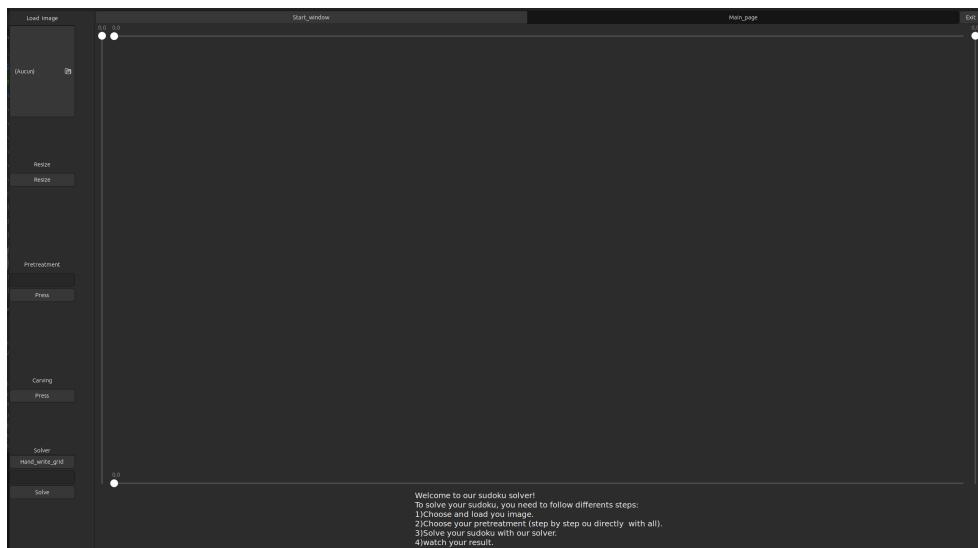


Figure 6: Fenêtre de traitement

Pour réaliser cette interface, il nous était demandé d'utiliser la librairie GTK et le logiciel Glade.

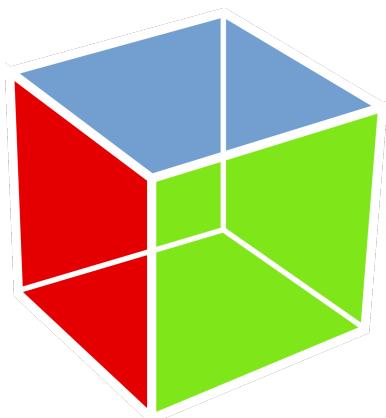


Figure 7:Gtk



Figure 8: Glade

8.2. Fonctionnement et construction

a. Le plan

Pour se faire, nous avons d'abord commencé par Glade. Ce logiciel nous a tout d'abord permis de réaliser un plan de l'architecture de nos fenêtres en posant nos différentes fonctionnalités à notre guise. Après la réalisation de notre plan, il nous restera plus qu'à construire notre GTK en fonction de ce plan et d'appeler les différents éléments présents dans les fenêtres pour les initialiser.

b. Changement de fenêtre et sortie de l'interface

Nous avons donc tout d'abord commencé par préparer la fenêtre d'accueil. Pour faire cela, nous avons initialisé nos deux boutons qui sont le bouton "Exit" et le bouton de changement de fenêtre comme leur nom l'indique. Le premier bouton nous fera sortir de l'interface graphique s'il est pressé et l'autre passera à la fenêtre suivante. Ces deux boutons sont trouvables dans la seconde fenêtre également et fonctionnent de façon identique.

c. Récupération de l'image

Après avoir réalisé notre fenêtre de traitement, nous avons commencé la fenêtre de traitement du sudoku.

Nous avons commencé par traiter de la cas de la récupération de l'image de notre sudoku puis de l'affichage dans notre interface. Pour faire ça nous avons utilisé le bouton de recherche de fichier auquel nous avons attribué une fonction qui renvoie dans l'emplacement de notre image le fichier qu'il a récupéré.

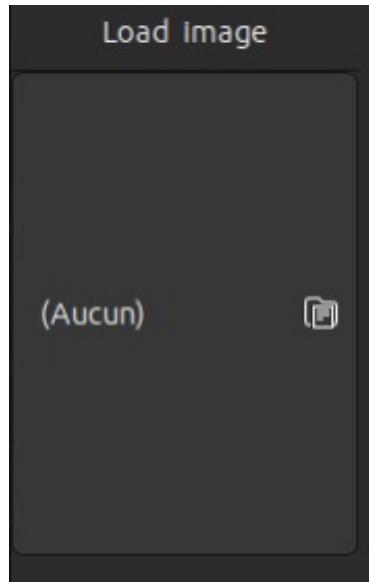


Figure 9: Sélecteur de fichier bouton

d. Rognage de l'image

Pour continuer, pour traiter les cas où notre image n'est simplement pas un grille de sudoku, nous avons réalisé un option de rognage de notre image. Pour réaliser cela nous avons utilisé quatre ajusteurs nous permettant de récupérer la partie rogner et le pourcentage de rognage et ensuite l'appui sur le bouton "resize" lance la fonction de rognage.

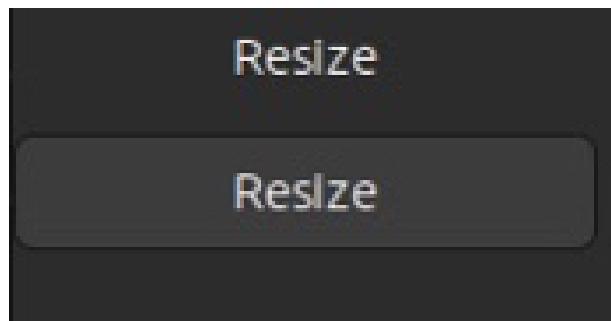


Figure 10: Bouton de rognage

e. Pré-traitement

Ensuite, nous commençons le pré-traitement de notre image. Pour ce faire nous avons utilisé un buffer si vous voulez réaliser la rotation du pré-traitement de façon manuel

sinon si vous appuyez sur un bouton cela exécutera le pré-traitement complet de notre image et ré-affichera le résultat.

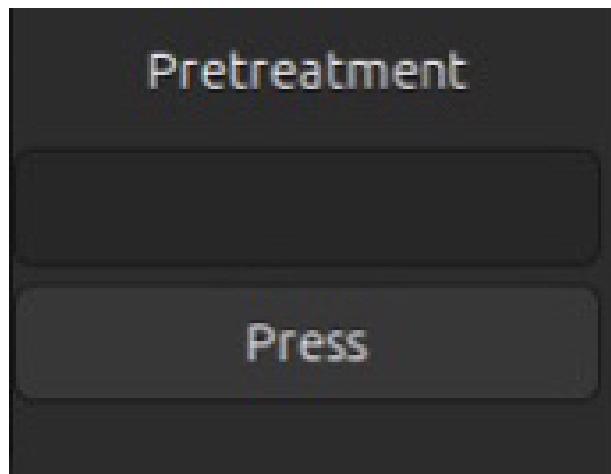


Figure 11: Bouton de pré-traitement

f. Détection et découpage de la grille

Pour continuer sur le cas de notre détection de grille et découpage, le simple appui sur le bouton en-dessous du label "carving" exécutera toute la détection et le découpage.

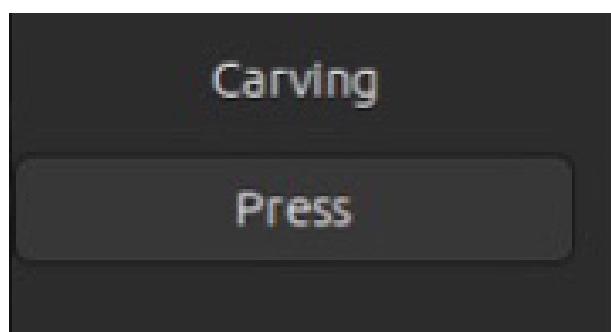


Figure 12: Bouton de la détection de la grille

g. Compréhension d'image et résolution du sudoku

Pour finir notre résolution de sudoku, il nous reste la détection des caractères avec le réseau de neurones et la résolution de la grille. Pour faire ceci l'appui sur le bouton "solve" suffit

à exécuter la compréhension de caractères et la résolution de la grille. Mais dans le cas où notre compréhension de caractères échoue il est toujours possible d'obtenir votre grille résolu. Pour faire cela il faut appuyer sur le bouton "Hand write grid", taper la grille de sudoku à la main (de la façon suivante : "5. .4596. ... 7.." les changements de ligne sont représentés par l'absence d'espace) et ré-appuyer sur le bouton "solve".

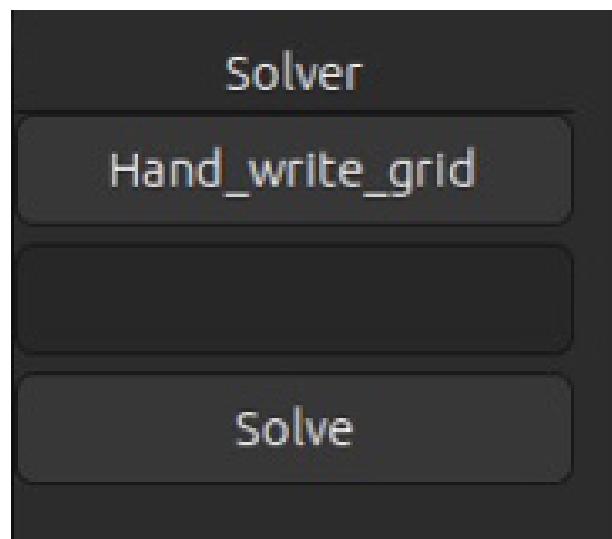


Figure 13: Bouton résolveur

9. NOS CONTRAINTES ET NOS MOYENS

9.1. Nos contraintes

a. Langage C

Il est impératif que tout notre projet soit exclusivement écrit en langage C.

b. Machine de l'école

Les machines de l'école sont utilisées comme machine de référence pour évaluer notre travail. Il est donc nécessaire que notre projet compile et s'exécute sur l'environnement de travail fournit par l'école (ici Linux). L'utilisation de tous les logiciels installés sur les machines est donc autorisée.

c. Contraintes liées au code

Les lignes de codes ne devront pas dépasser 80 colonnes et ni contenir d'espaces inutiles en fin de ligne. Les identifiants utilisés et les commentaires devront être impérativement en anglais.

d. Contraintes de compilation

Le code doit pouvoir compiler sans erreur avec les options : -Wall -Wextra

e. Dépôt GIT

Pour notre rendu il est obligatoire d'utiliser le dépôt GIT qui nous a été donné en début de projet. Notre dépôt GIT doit uniquement contenir nos rapports en format PDF, tout le code source de notre projet, un fichier Makefile aidant à compiler notre code, un fichier README expliquant comment utiliser notre logiciel, un fichier AUTHORS contenant les informations des membres et des images qui serviront d'exemple durant nos soutenances.

9.2. Outils & Logiciels

a. GIT

Nous utilisons GIT pour structurer notre projet, pour partager nos travaux entre les membres et pour éviter les conflits lors de notre rendu de projet.

b. Emacs / VIM / CLion / Visual Studio Code

Ces logiciels vont nous permettre de rédiger nos programmes en langages C et de nous assister en cas d'erreur de syntaxe.

c. GNU Make

Cet outil nous permet de compiler nos programmes en C plus efficacement et plus rapidement et nous averti des différentes erreurs créées par nos programmes.

d. Discord

Nous utilisons l'application Discord pour communiquer entre nous et organiser nos sessions de travail grâce à des salons structurés sur un serveur créé à cet effet.

e. Gtk/Glade

Comme dit dans la partie sur l'interface graphique,Gtk et Glade nous ont permis de construire notre propre interface graphique.

10. CONCLUSION

Pour conclure ce rapport, nous sommes fiers du travail que nous avons apporté lors de ce projet. Notre travail est plus que satisfaisant et nous sommes heureux de pouvoir dire que nous l'avons terminé.

Ce fut pour chacun d'entre nous une expérience nouvelle très enrichissante, et l'occasion de perfectionner notre esprit de groupe, indispensable à tout ingénieur, que nous avions déjà affûté lors du projet de S2.