Projet Sudoku par

Chef de projet : Luke Goboyan
Evan Le-Clech
Kélyan Bouras
Charlotte Farinas

Sommaire :

I-Introduction

Je m'appelle Charlotte et j'ai 19 ans. Contrairement au projet de l'année dernière, je n'ai pas vraiment de précédentes expériences de création d'application ou de programmation (sauf cours). Ce projet m'a tout l'air d'être plus technique en termes de code et de réalisation. En terminale, j'avais réalisé un article sur le Deep Learning qui utilise justement les réseaux de neurones sur WIX (création de site internet). Je suis donc très curieuse de pouvoir travailler sur un projet de Sudoku qui permet de créer un réseau de neurones.

Je m'appelle Evan, j'ai 18 ans. Je n'ai jamais réalisé de projets similaires mais j'ai déjà pu m'entraîner à la réalisation d'interface graphique notamment dans le cas du projet de première année ce qui sera utile pour la réalisation de ce programme.

Je m'appelle Kelyan, j'ai 18 ans. Je n'ai jamais réalisé de projets de ce genre,qui s'apparente clairement à du développement de logiciel, et cela m'a l'air plutôt complexe, surtout que je n'ai pas beaucoup d'expérience dans le traitement d'image et que les algorithmes de résolution de problèmes ne sont pas mon fort.

Dans la création de ce projet, je me suis occupé du traitement de l'image qui sera passé en paramètre du programme, et qui sera ensuite envoyé aux autres programmes

Mes fonctions s'occupent principalement de la rotation de l'image , de la détection de la grille de sudoku, afin que celle-ci puisse être envoyée au réseau de neurones

Je m'appelle Luke Goboyan, j'ai 19 ans. Bien que j'ai réalisé des tâches comme celles impliquées pour la construction de ce projet, je n'ai jamais travaillé sur un projet aussi complet en termes de contenu et fonctionnalités. Cela m'a donné l'occasion d'exercer mes connaissances acquises, ainsi que d'apprendre de nouvelles choses.

Pour la première soutenance, je me suis occupé de concevoir le réseau de neurones qui reconnaît une porte XOR. Pour la seconde, j'ai fait le réseau de neurone complet.

II- Aspects techniques

II.1- Solver

Dans ce projet, j'ai réalisé la partie de résolution du sudoku pour débuter la création du Solver. Pour le réaliser, il y a plusieurs fonctions que j'ai écrites. La première est la création du programme de résolution, puis la prise du fichier et la création du fichier final. Pour cette deuxième soutenance, j'ai finalisé le prétraitement c'est-à-dire réaliser la binarisation et la détection d'angle. Enfin, j'ai affiché et sauvegardé l'image finale du sudoku résolue.

Pour commencer, j'ai regardé beaucoup de sites concernant la réalisation de programmes pouvant résoudre la grille de sudoku. La méthode qui ressort le plus pour réaliser ce programme est ce que l'on appelle le « BackTracking » également appelé en français « Retour sur trace ». Dans notre cas, il consiste à retourner en arrière si aucun chiffre n'est valide à cette case. Pour ce faire, le programme est récursif ce qui permet ce retour en arrière. On peut le comparer au système des arbres que nous apprenons où nous parcourons (généralement) vers le fils gauche puis on remonte au père pour atteindre le fils droit. Ce principe est presque le même que le « BackTracking » si le chiffre n'est pas valide on retourne à la case précédente (vide initialement) pour essayer un autre chiffre etc…
Une fois l'idée claire et précise de la fonction mère que je dois créer, j'ai réalisé 3 fonctions secondaires.

La première fonction permet de vérifier si le chiffre passé en paramètre est déjà sur la ligne. S'il est déjà présent, la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

La seconde fonction permet de vérifier si le chiffre passé en paramètre est déjà sur la colonne. De même, s'il est déjà présent la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

La troisième fonction permet de vérifier si le chiffre passé en paramètre est déjà sur le carré. De la même manière, s'il est déjà présent la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

A présent, la création de la fonction récursive principale « IsValid ». Elle a plusieurs tests à faire. Le premier est de vérifier si toute la grille a été parcourue, si c'est le cas la fonction renvoie le tableau résolu. Ensuite, si la case n'est pas vide cela veut dire qu'aucun chiffre doit être ajouté alors on incrémente la position. Enfin, une boucle permet d'ajouter un chiffre. On appelle ensuite récursivement la fonction IsValid dans un « if « qui vérifie que la prochaine position est valide également sinon la case retourne à zéro car aucun chiffre est possible. C'est ici que le retour en arrière se passe.

a- Ouverture du fichier : open_my_file

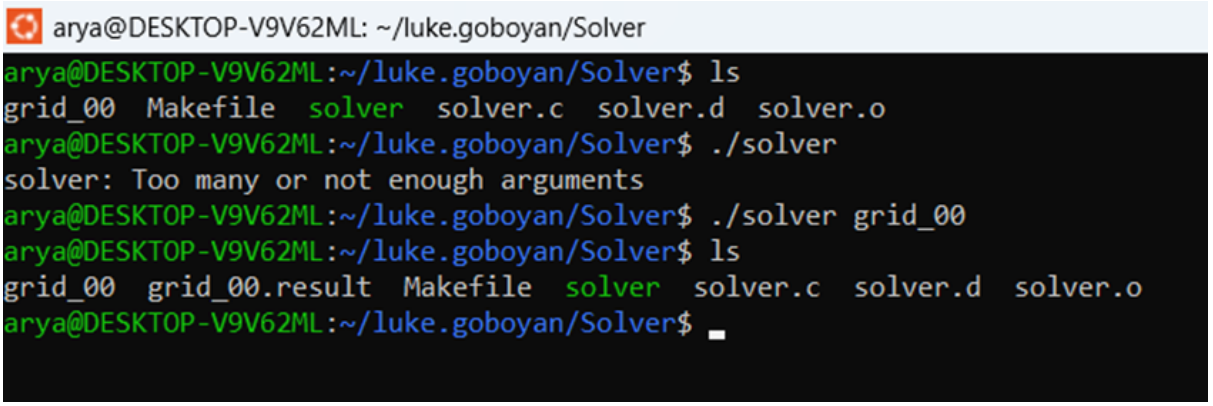
Dans cette deuxième partie, j'ai appris à utiliser la lecture d'un fichier. On peut y trouver des ressemblances avec le python même si ce n'est pas exactement pareil. Tout d'abord pour la simple et bonne raison que le C utilise un pointeur de type FILE.

Pour cette fonction, la grande partie du travail est de comprendre comment les fonctions pour ouvrir un fichier fonctionnent. Il a donc fallu encore de nombreuses recherches pour pouvoir l'écrire. De plus, j'ai ajouté une erreur qui permet de savoir si le fichier a eu un problème pour s'ouvrir. Finalement, j'ai utilisé la fonction « fgetc » qui prend notre variable du pointeur comme paramètre et renvoie un entier correspondant au caractère du code ASCII. J'ajoute par la suite le caractère à mon tableau d'entier qui représente la grille de sudoku.

b- -Création du fichier final : my_result_file

Pour cette dernière fonction, j'ai également dû apprendre à utiliser les fonctions afin de créer et écrire dans un fichier. Une nouvelle fois, la partie la plus longue a été la recherche et la compréhension des fonctions et de choisir la meilleure solution.
Dans premier temps, j'ai créé mon fichier avec le «.result » pour identifier la grille de sudoku résolu. Ensuite, je parcours mon tableau d'entier résolu, après avoir appelé IsValid, puis j'écris dans mon fichier le caractère correspondant grâce à la fonction « fputc ». Elle permet d'écrire le caractère souhaité tant qu'il est placé en tant que paramètre de cette fonction.

Voici un aperçu de ce que le solver renvoi :



Nous pouvons observer la grille résolu enregistrer sous le nom grid_00.result qui permet d'enregistrer le résultat autrement que l'affichage de la grille que nous aborderons plus tard.

```
arya@DESKTOP-V9V62ML:~/luke.goboyan/Solver$ ls
grid_00  grid_00.result  Makefile  solver  solver.c  solver.d  solver.o
arya@DESKTOP-V9V62ML:~/luke.goboyan/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
arya@DESKTOP-V9V62ML:~/luke.goboyan/Solver$
```

Ici nous pouvons voir ce que contient le fichier « .result ».

II - Prétraitement

Lors de la première soutenance, ils nous manquaient des fonctionnalités. Je me suis occupée de corriger le prétraitement en ce qui concerne l'image plus particulièrement la binarisation. Notre groupe a décidé de faire la binarisation de sorte que le fond soit noir et que la grille et les chiffres ressortent en blanc.

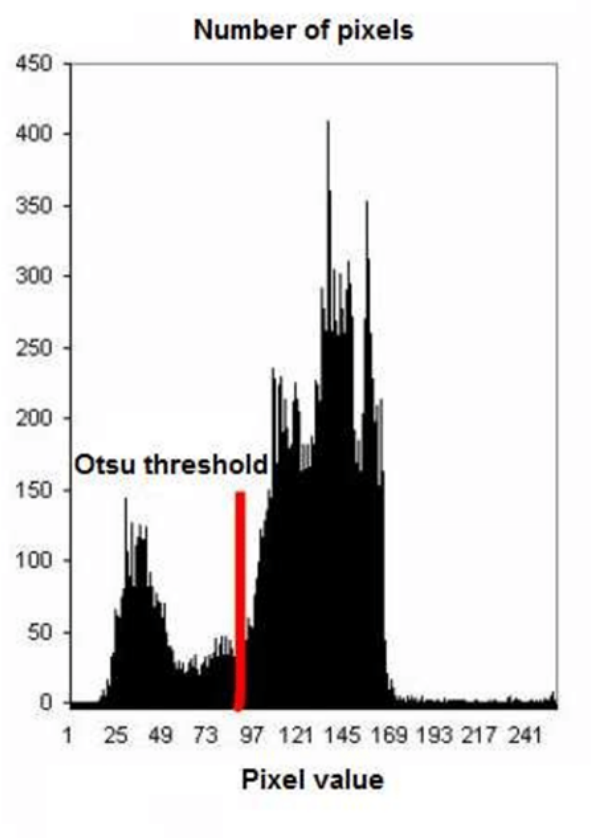
Tout d'abord, il faut appliquer sur l'image un filtre gris. Ce filtre permet de transformer les couleurs en gris. Les couleurs vivent tels que le bleu, le vert, le jaune ou encore le rouge. Ce filtre va nous simplifier la vie pour la suite. En effet, après l'application de cette fonction, l'étape de la binarisation commence. Pour ce faire l'utilisation de la méthode communément appelé « Otsu » est nécessaire.

a - Explication d'OTSU

Le principe d'Otsu est assez simple à comprendre. Son principe est de repérer et de faire des différences entre les couleurs de l'arrière-plan et du premier. Pour les différencier, la méthode d'Otsu génère un seuil qui d'un côté colore les pixels en blanc et de l'autre les pixels de l'image en noir.

Pour détailler, il existe différentes façons d'utiliser et de réaliser cette fonction. D'une part, il faut forcément créer un histogramme de façon à calculer le seuil que nous avons évoqué précédemment. Un histogramme est une représentation graphique qui permet de représenter la répartition d'une

variable aléatoire en la représentant graphiquement avec des colonnes correspondantes chacune à une classe de cette manière :



Pour calculer le seuil, il faut remplir l'histogramme. Il est initialisé à 255 car c'est la couleur maximale que peut prendre un pixel comme couleur. La moyenne des couleurs est calculée et représente l'index à laquelle on ajoute 1 à chaque fois que le pixel à la même couleur.

Prenons un exemple simple et imaginons que nous obtenions comme couleur : r = 255, b = 50 et g = 10. Nous effectuons la moyenne 255+50+10 = 315/3 = 105. L'histogramme à l'index 105 augmente alors de 1. En réalisant ce procédé sur toute l'image nous pouvons obtenir l'histogramme de ci-dessus par exemple. Ici, l'application du filtre en gris est d'autant plus importante car il permet grandement de simplifier l'histogramme.

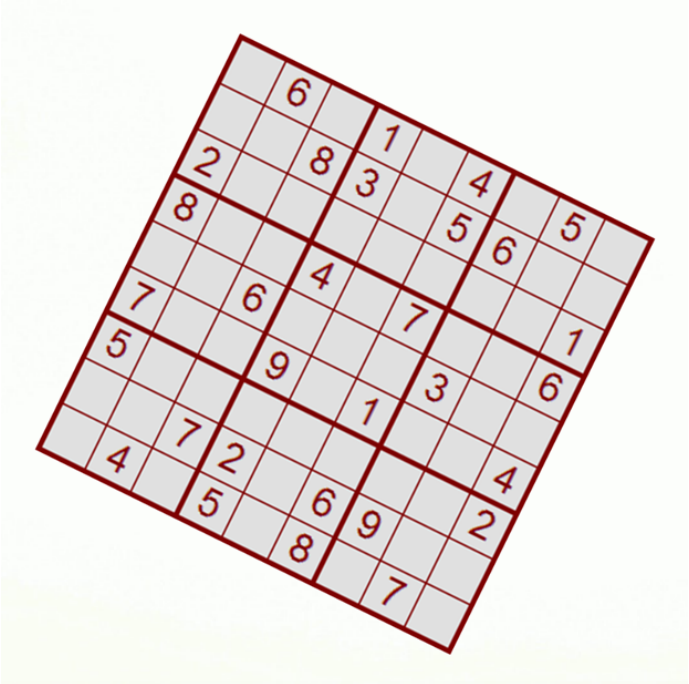
Ensuite la partie la plus compliquée entre en jeu. L'étape des calculs est la suite de la réalisation de la fonction d'Otsu pour trouver le seuil.

Nous faisons varier le t entre 0 et 255 et calculons la probabilité d'intensité et c'est ce que nous appelons le poids « weight » en anglais de l'arrière-plan puis nous déduisons celui du premier plan. Nous déterminons

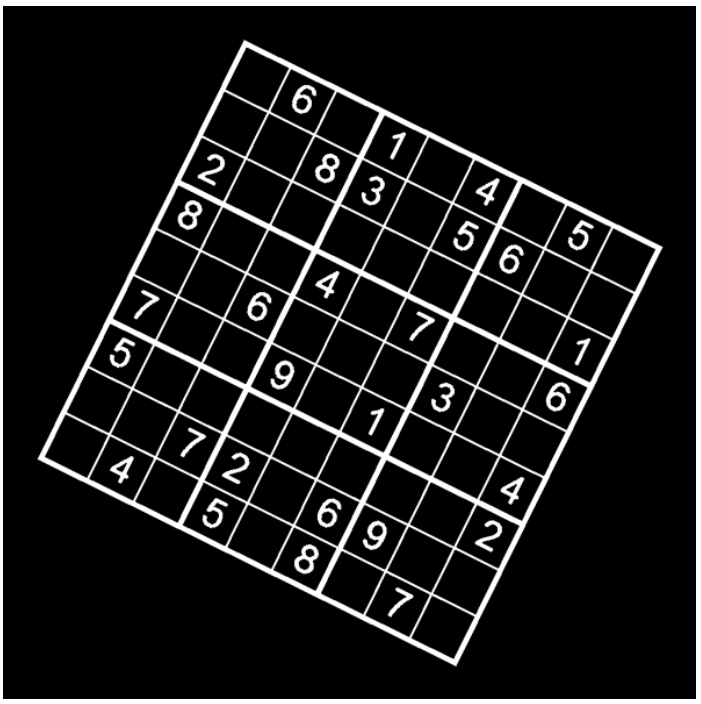
ensuite ce que l'on appelle la variance de chaque classe. Lorsque que nous parlons de classe nous entendons par là que les calculs sont réalisés deux fois l'un pour l'arrière-plan puis l'autre pour le premier plan. Il suffit de calculer les écarts par rapport à la moyenne dans chaque classe. Dans notre exemple nous n'avons que deux classes pour symboliser les deux plans. C'est ce que l'on appelle calculer la variance intra-classe. Une fois que tous ses écarts sont calculés il ne reste plus qu'à prendre le maximum pour trouver le seuil de l'image recherché.

Nous arrivons à la fin de la méthode d'Otsu puisque une fois le seuil trouvé il suffit de comparer les pixels de couleurs et le seuil. Si la couleur est supérieure au seuil alors c'est un élément du fond tandis que si la couleur est inférieure au seuil alors le pixel est un élément du premier plan tel que la grille ou les chiffres.

Le résultat est celui-ci-dessous :



Application de la méthode Otsu :



La principale difficulté que j'ai rencontré concernant Otsu est de comprendre comment les calculs doivent s'effectuer. En effet, de nombreux sites internet décrivent la méthode et l'expliquent mais dans chacun d'entre eux les calculs ne sont pas complets. Il a donc fallu que je cherche la clé qui me permettrait de résoudre ce problème.

b - Détection d'angle

Pour la détection d'angle étant donné que l'algorithme de Hough n'était pas encore opérationnel, j'ai créé un code afin de le calculer pour dépanner. J'ai donc décidé d'utiliser mes connaissances en physique pour le déterminer.

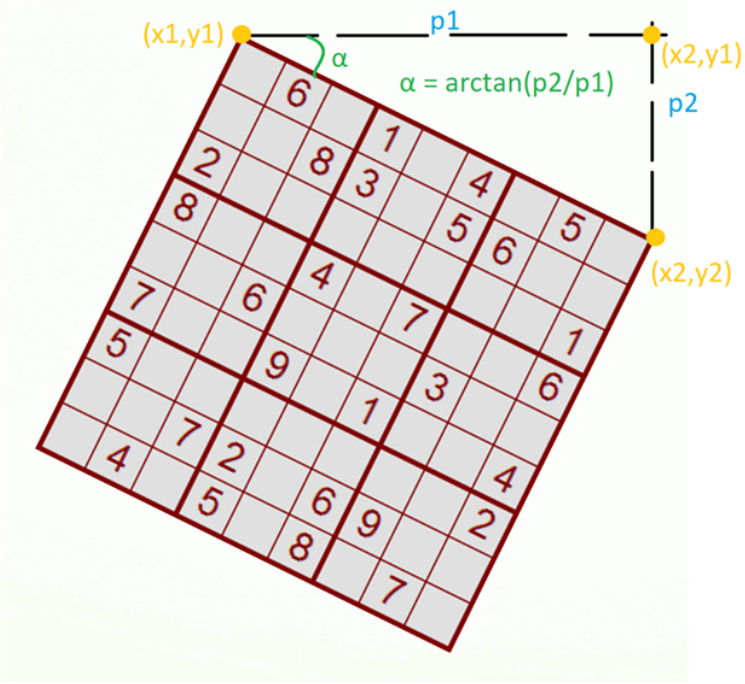
Dans un premier temps, j'ai décidé de parcourir l'image de droite à gauche. Dès que je tombais sur un pixel blanc, j'en déduisais que c'était la grille. Une fois le premier pixel je repartait d'un x différent et je descendais sur l'axe y afin de trouver un autre pixel blanc. Une fois trouvé, je pouvais à l'aide des deux coordonnées appliqués la tangente.

Néanmoins l'angle que je trouvais, il avait 0.8 de différence de celui recherché. J'ai décidé d'appliquer une seconde méthode qui s'avéra plus précise. Je parcours une première fois ma grille de droite à gauche. J'utilise le

même principe que celui énoncé précédemment : lorsque je tombe sur un pixel blanc, je mémorise c'est coordonné. Puis je parcours de haut en bas en partant de la largeur maximum pour appliquer le même principe. Par rapport à la première méthode, celle-ci permet d'avoir des longueurs plus précises et par conséquent obtenir un angle qui varie à peine de 0.1 degré.

Dans cette partie, la difficulté a été de remettre en cause mon code lorsque l'angle avait 0.8 degré de différence et que nous le remarquons à l'œil nu. Quand je dis le remettre en cause, je me posais la question : Pourquoi il n'est pas aussi précis que je le voudrais ? Une fois que j'ai compris que les pixels ne forment pas une ligne droite parfaite, j'ai compris qu'il fallait deux longueurs plus grandes pour apporter de la précision.

Voici un schéma pour symboliser :



Je m'appelle Kelyan, j'ai 18 ans. Je n'ai jamais réalisé de projets de ce genre,qui s'apparente clairement à du développement de logiciel, et cela m'a l'air plutôt complexe, surtout que je n'ai pas beaucoup d'expérience dans le traitement d'image et que les algorithmes de résolution de problèmes ne sont pas mon fort.

Dans la création de ce projet, je me suis occupé du traitement de l'image qui sera passé en paramètre du programme, et qui sera ensuite envoyé aux autres programmes

Mes fonctions s'occupent principalement de la rotation de l'image , de la détection de la grille de sudoku, afin que celle-ci puisse être envoyée au réseau de neurones

1-) Rotation de l'image

Une fois que l'angle de rotation a été trouvé, il faut effectuer une rotation de l'image avec cet angle, pour cela,il faut utiliser une matrice de rotation calculé pour un angle θ par:

cos θ	-sin θ
sin θ	cos θ

et plus précisément utiliser l'inverse de cette matrice de rotation

Il faut pour chaque pixel de l'image destination, il faut calculer les coordonnées de son pixel source puis récupérer la couleur de celui-ci pour pouvoir l'appliquer au pixel de destination, la raison pour laquelle on procède comme cela est pour éviter les pertes de donnés.

2-) Détection de la grille

I)-Détection des lignes :

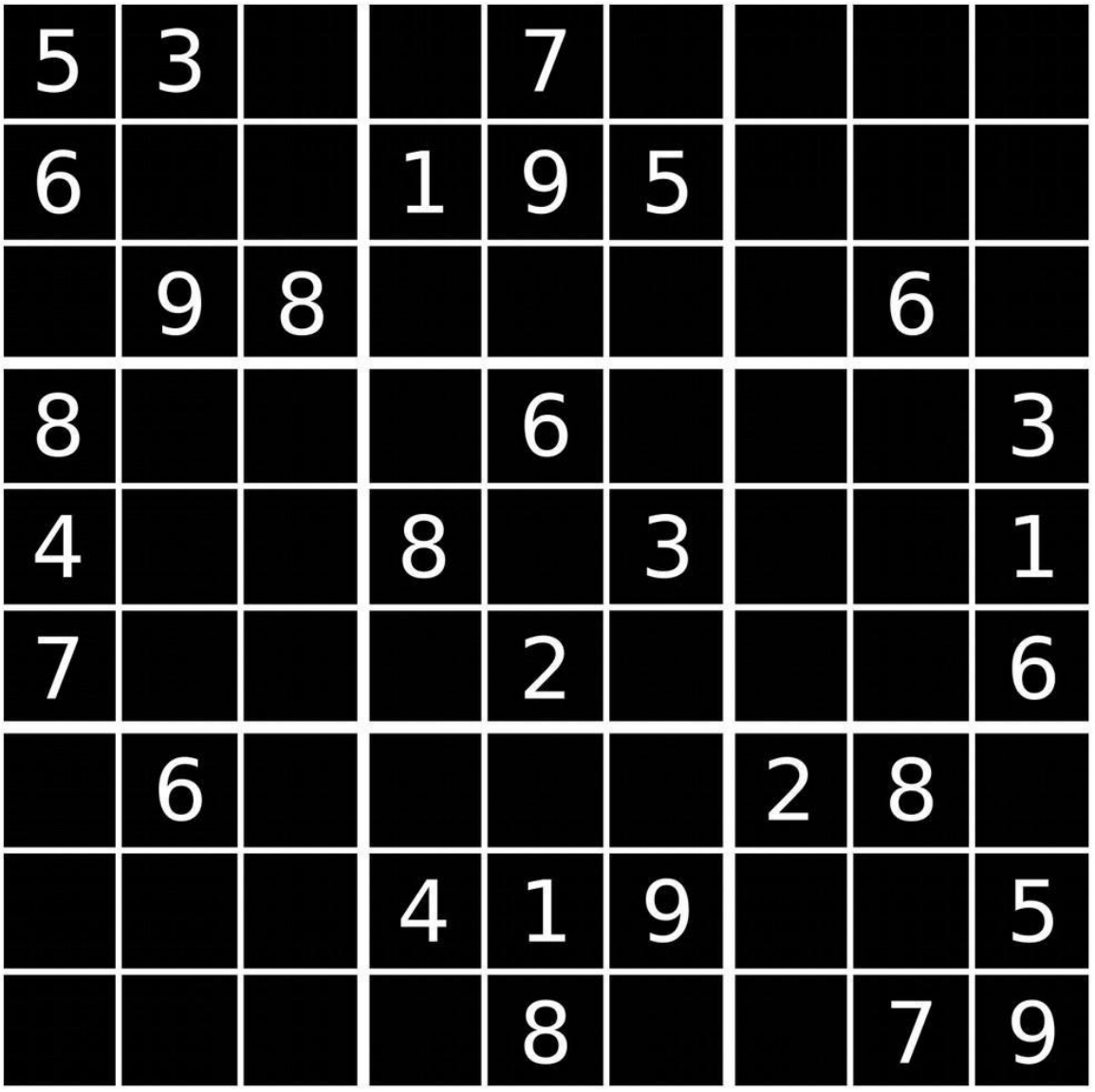
Pour détecter les lignes dans une image, j'utilise la transformée de Hough (plus connue sous son nom anglais : Hough Transform)

La transformée de Hough consiste à détecter pour chaque point de l'image l'ensemble des droites passant par ce point puis à détecter les droites qui reviennent le plus souvent.

Explication :

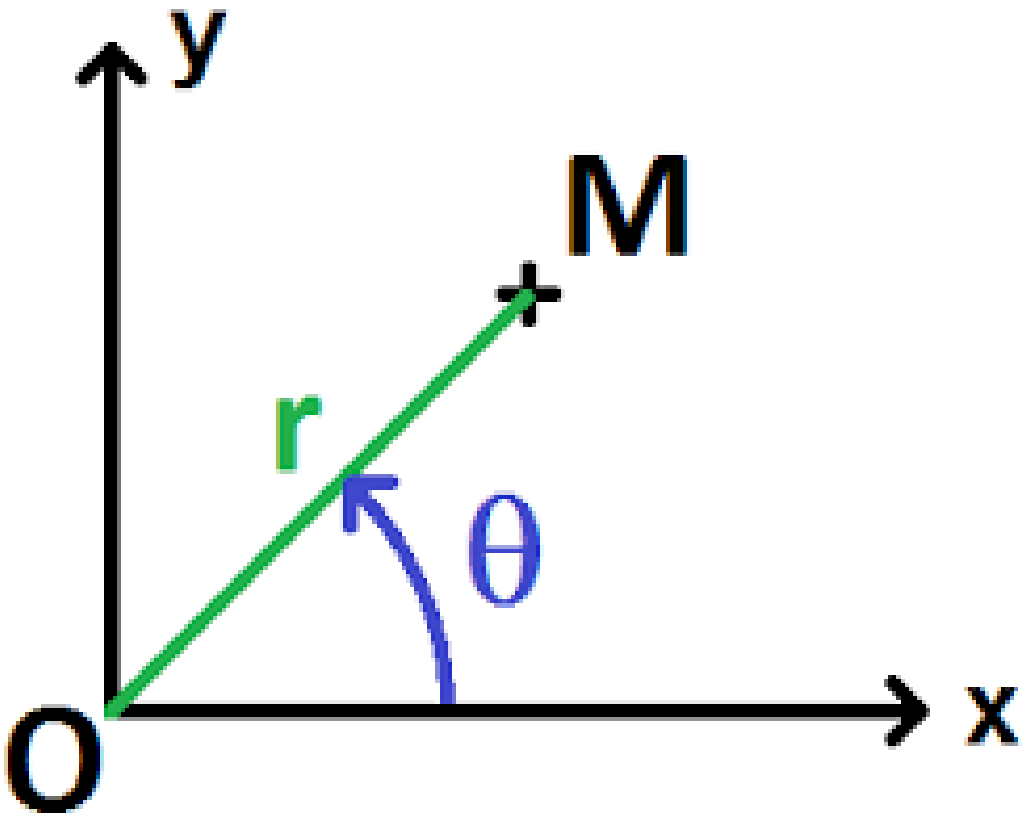
Mon algorithme de Hough prendra en paramètre une image en noir et blanc avec les lignes en blancs :

Exemple :



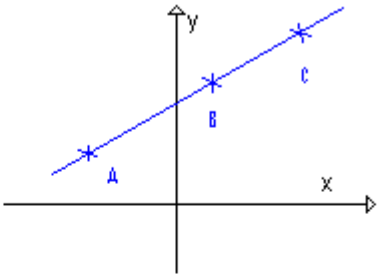
Un droite est caractérisée par une équation y = a*x + b, on pourrait théoriquement utiliser cette représentation pour notre programme mais il ya un problème : dans cette équation, “a” a tendance à tendre vers l’infini si la droite est complètement verticale. C’est pour cela que l’on utilise les coordonnées polaires, une coordonnée en polaire est caractérisée par deux paramètres (ρ et θ). L’avantage d’utiliser ces coordonnées est que les bornes

sont connues : ρ va varier de –d a +d où d est la diagonale de l'image et θ est un angle qui va varier entre 0 et π rad (0 et 180°).

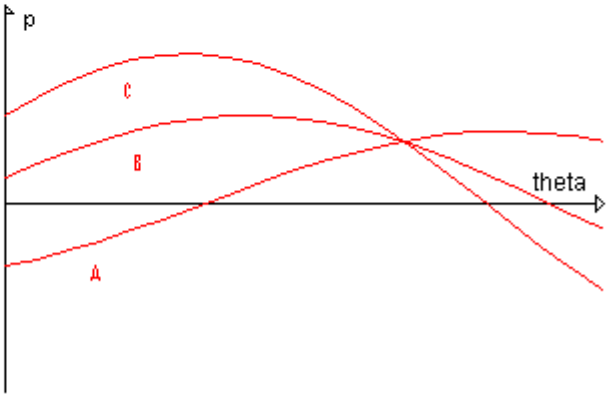


L'algorithme consiste a créer une matrice de taille ρ * θ initialisé uniquement avec des 0. Cette matrice représente notre espace de Hough. Puis pour chaque point blanc de l'image (car les bords sont en blanc sur notre image) il faut calculer les coordonnées polaires de chaque droite passant par ce point et incrémenter les coordonnées correspondantes dans notre espace de Hough.

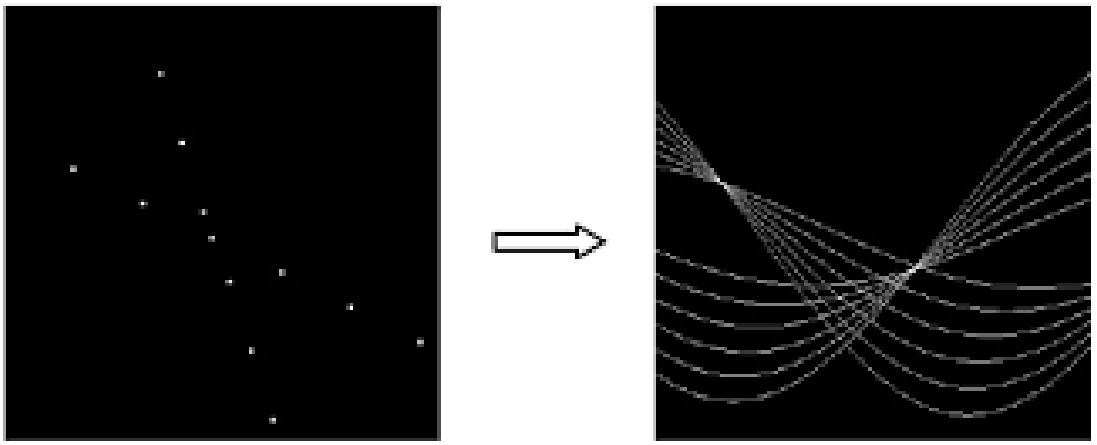
Image originale



Espace de Hough



Il ne reste plus qu'à détecter les pics de valeurs dans notre espace de Hough et on aura les coordonnées polaires de l'ensemble des droites



(a) $p_i = (x_i, y_i)$ (b) $\rho = x_i \cos(\theta) + y_i \sin(\theta)$

Sur cette représentation de l'espace de Hough, on peut voir deux gros pics d'intersection, ces deux pics correspondent aux deux droites.

Dans notre cas, on sait qu'une grille de sudoku contient 20 lignes (10 lignes horizontales et 10 lignes verticales).

Il faut donc détecter les 20 premiers pics

II-) Détection des intersections :

Une fois que les lignes ont été déterminées, il faut trouver les intersections entre les droites, pour cela, on peut utiliser la formule permettant de trouver les coordonnées X et Y du point d'intersection entre deux droites chacune caractérisées par deux points :

$$P_x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \qquad P_y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}$$

III-) Détection des rectangles :

Un rectangle est caractérisé par deux points (son coin supérieur gauche et son coin inférieur droit).

Pour pouvoir par la suite détecter les cases, il faut stocker tous les rectangles possibles constructibles avec notre liste de points d'intersections, (si les deux points ont la même abscisse ou ordonnée, on ne construit pas le rectangle)

Puis on stocke tous ces rectangles dans une liste

IV-) Détection des cases :

Une fois que l'on a la liste de tous les rectangles possible, il faut extraire ceux qui représentent nos cases, pour cela il faut ne garder dans la liste finale uniquement les rectangles qui sont des carrés puis les trier en ordre croissant selon leurs aires.

Ensuite, comme on sait qu'une grille de sudoku est en 9*9, il faut garder les 81 premiers carrés, ceux-ci représentés l'ensemble de nos cases et pourront être envoyé au réseaux de neurones

II.3 - Réseau de neurone

Le réseau de neurone est nécessaire pour reconnaître les nombres à partir d'une image de sudoku.

a-Structure d'un réseau de neurone:

En développement, un neurone est considéré comme un objet algorithmique qui prend un certain nombre de valeurs, par example n nombres numérotés x_1, x_2,... à x_n , et renvoie un résultat unique. Il existe deux types de neurones: les perceptrons et les sigmoides.

Soit un neurone qui prend une liste de valeurs [x_1, x_2,...,x_n], et z résultat d'une équation:

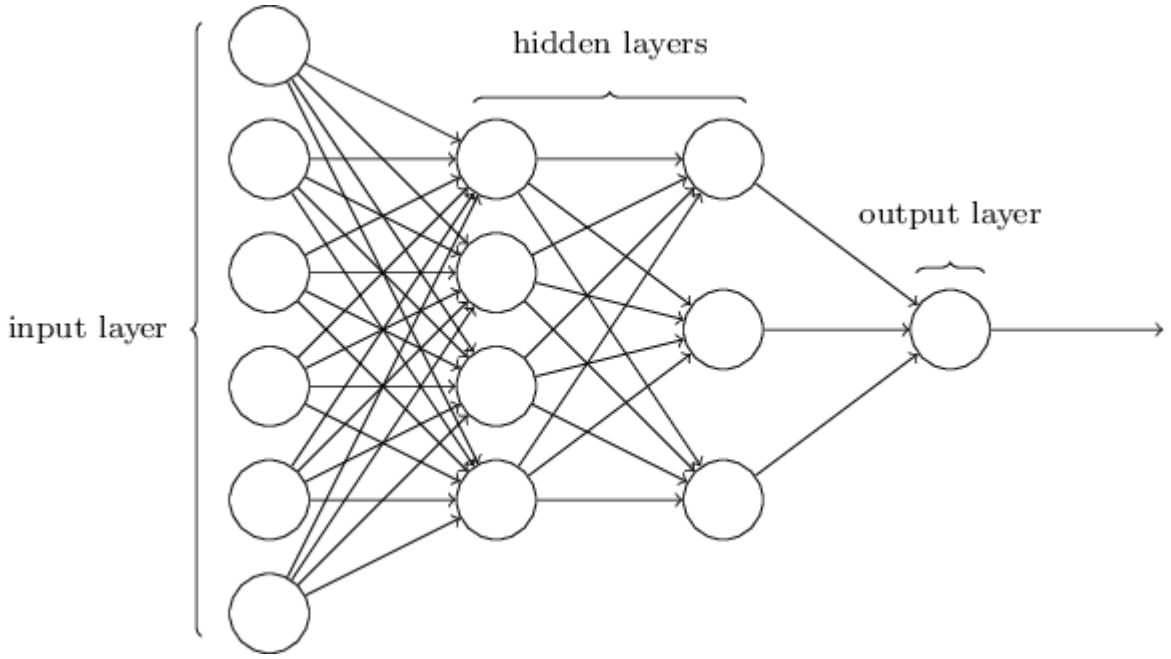
$$z = (\sum_{k=1}^{n} x_k * w_k) + b$$

w est le poids et une valeur unique à chaque x qui représente son importance dans l'équation. b représente le biais du neurone, et un seuil que z-b doit dépasser pour que z soit supérieur ou égal à 0, dans le cas contraire il est inférieur. Le perceptron ne renvoie que deux résultats: 1 si z >= 0, 0 si z < 0. Il est le plus simple des deux modèles dans son fonctionnement et renvoie des résultats clairs, mais il pose un problème: si l'on change une des valeurs de poids ou de biais dans une tentative d'obtenir un résultat final différent de peu, la nature du changement brusque des résultats de perceptrons rend la manière dont le changement final est affecté de manière imprévisible.Voici le résultat d'un neurone sigmoïde:

$$s = 1/(1+e^{-z})$$

Sachant que z représente le même calcul, on remarque que le résultat n'est plus un 1 ou un 0, mais une fraction qui représente un nombre entre 0 et 1; de plus, si z = 0 alors s = ½ le milieu de l'intervalle, tandis que plus z s'éloigne de 0, plus il se rapproche de 1 si positif ou 0 si négatif. Si on regarde les limites:
$z \to +\infty \Rightarrow e^{-z} \to 0 \Rightarrow s \to 1$
$z \to -\infty \Rightarrow e^{-z} \to +\infty \Rightarrow s \to 0$
Ce fonctionnement permet à ce qu'un changement de valeur dans un réseau de neurones sigmoïdes entraîne un changement de même ampleur dans le résultat final.

Finalement, voici l'organisation d'un réseau de neurones complet:



Les neurones sont répartis en plusieurs colonnes, dites "couches" se suivant de gauche à droite. La première couche, dite couche d'entrée, est composée de neurones trous prenant en argument les valeurs données manuellement au réseau. Toutes les autres couches sauf la dernière sont dites "cachées", et chacun des neurones prend en argument les résultats de tous les neurones de la couche précédente. Enfin, la couche finale renvoie le résultat attendu du réseau de neurones. Il faut noter que bien que les tailles et chaque couches et types de neurones utilisés sont décidés "manuellement", les valeurs de poids et biais sont décidées par apprentissage automatique.

b-implémentation et apprentissage du réseau de neurone

L'implémentation du réseau de neurone en c se fait à l'aide de struct, spécifiquement trois types utilisés: Network, Layer, et matrices. Le Network est composé principalement de pointeurs vers les couches, ainsi que du nombre de couches et une "cost function" qui sert à déterminer à quel point il faut adapter les valeurs de biais et de poids. Le Layer possède deux attributs, une matrice pour les poids et une pour les biais de chaque neurone. Les matrices sont des manières de stocker les décimaux, il possèdent deux attributs de dimension ainsi qu'un tableau de décimaux. Ainsi, une matrice de poids dans une couche a pour dimensions respectives [nombre de neurones dans la couche]*[nombre de neurones de couche précédente]. L'implémentation en matrice est avantageuse car plusieurs opérations

matricielles peuvent jouer un rôle important dans l'apprentissage du réseau de neurone.
L'algorithme d'apprentissage s'explique comme tel:
-prendre une fraction des données d'apprentissage, nommée *mini batch* , entre lesquels on prévoit de changer les valeurs de poids et biais;
-initier deux tableaux de matrices, nabla_w et nabla_b, avec seulement des 0;
-pour chaque valeur du mini bach, calculer son résultat avec les poids et biais actuels et les conserver dans une liste de matrices à part;
-multiplier les poids de chaque neurone avec leur valeur d'entrée respective, et additionner les valeurs d'entrée avec les biais;
-

c-Utilisation du réseau de neurone:

Lors du projet de sudoku, le rôle que va jouer le réseau de neurone sera de processeur chaque image individuelle prise du découpage du sudoku; le réseau est à ce moment là initié avec le stockage des valeurs, et en conservant l'ordre des images, va renvoyer les valeurs auxquelles ces images correspondent afin de renvoyer un double tableau d'entiers qui sera résolu par l'algorithme de résolution de sudoku.

II.4 - Affichage de la grille

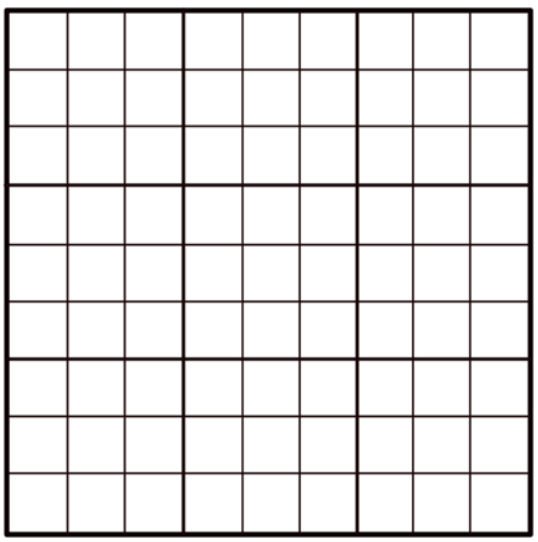
La dernière tâche que j'ai réalisée est l'affichage de la grille et donc la création d'une image pour afficher la grille résolue.

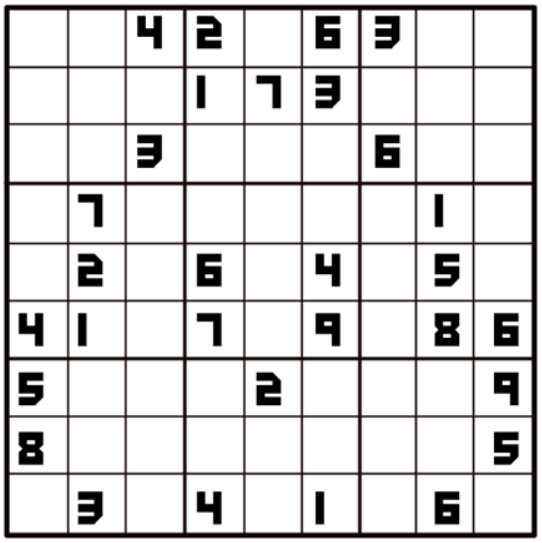
Tout d'abord, j'ai cherché une méthode pour afficher les chiffres sur l'images. La solution que j'ai finalement adoptée est de créer une surface à chaque fois que j'écris un chiffre et que je la colle sur la surface correspondant à une grille de sudoku vide. Pour appliquer cette méthode, il a fallu utiliser la fonction SDL_BlitSurface. En premier lieu, je crée la surface sur laquelle j'écris le chiffre. En second, j'utilise la fonction SDL_BlitSUrface.

Cette fonction est très pratique parce qu'elle permet de copier la surface sur laquelle je viens d'écrire un chiffre sur une autre surface contenant la grille de sudoku vide en étant délimité par un rectangle. En effet, cette technique permet de construire des rectangles équivalents pour écrire les chiffres un par un de mon tableau afin qu'ils soient à équidistances des uns des autres. L'autre partie de la fonction était de trouver par rapport à la grille quelle taille devrait faire les rectangles pour permettre d'avoir un visuel dans lequel les chiffres sont aux milieux des cases et centrés.

```
SDL_Surface *text = TTF_RenderText_Solid(font,digit,textColor);
SDL_Rect rect = {20+x,8+y,68,68};
SDL_BlitSurface(text,NULL,final,&rect);
SDL_FreeSurface(text);
```

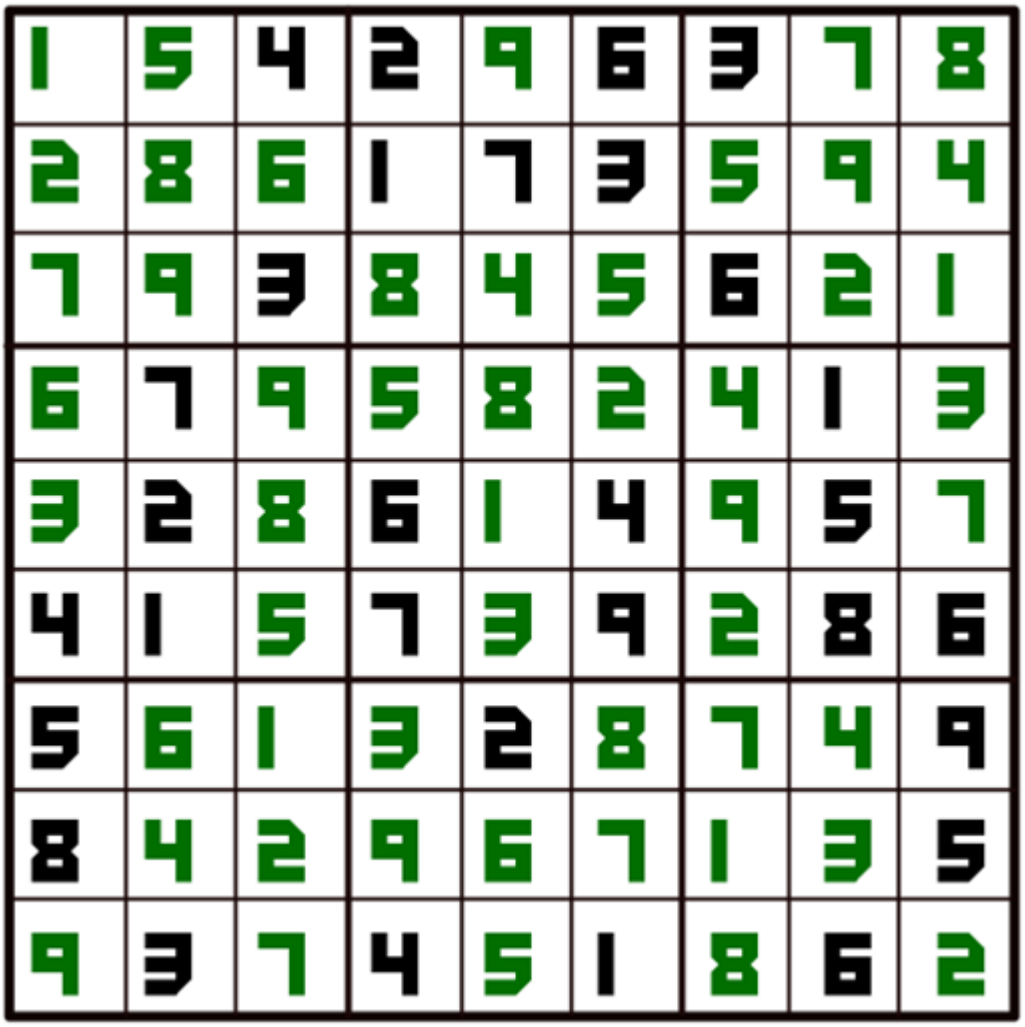
De plus, nous avons décidé de créer une image où le sudoku n'est pas résolu mais se remplit en fonction des chiffres à l'intérieur de la grille donnée par l'utilisateur initialement. Nous pouvons voir le résultat ci- dessous.
Résultat de l'affichage :





L'image vide puis remplit sans avoir appelé le solver.

Ensuite, l'utilisateur appuie sur le bouton solve pour afficher la grille finie.



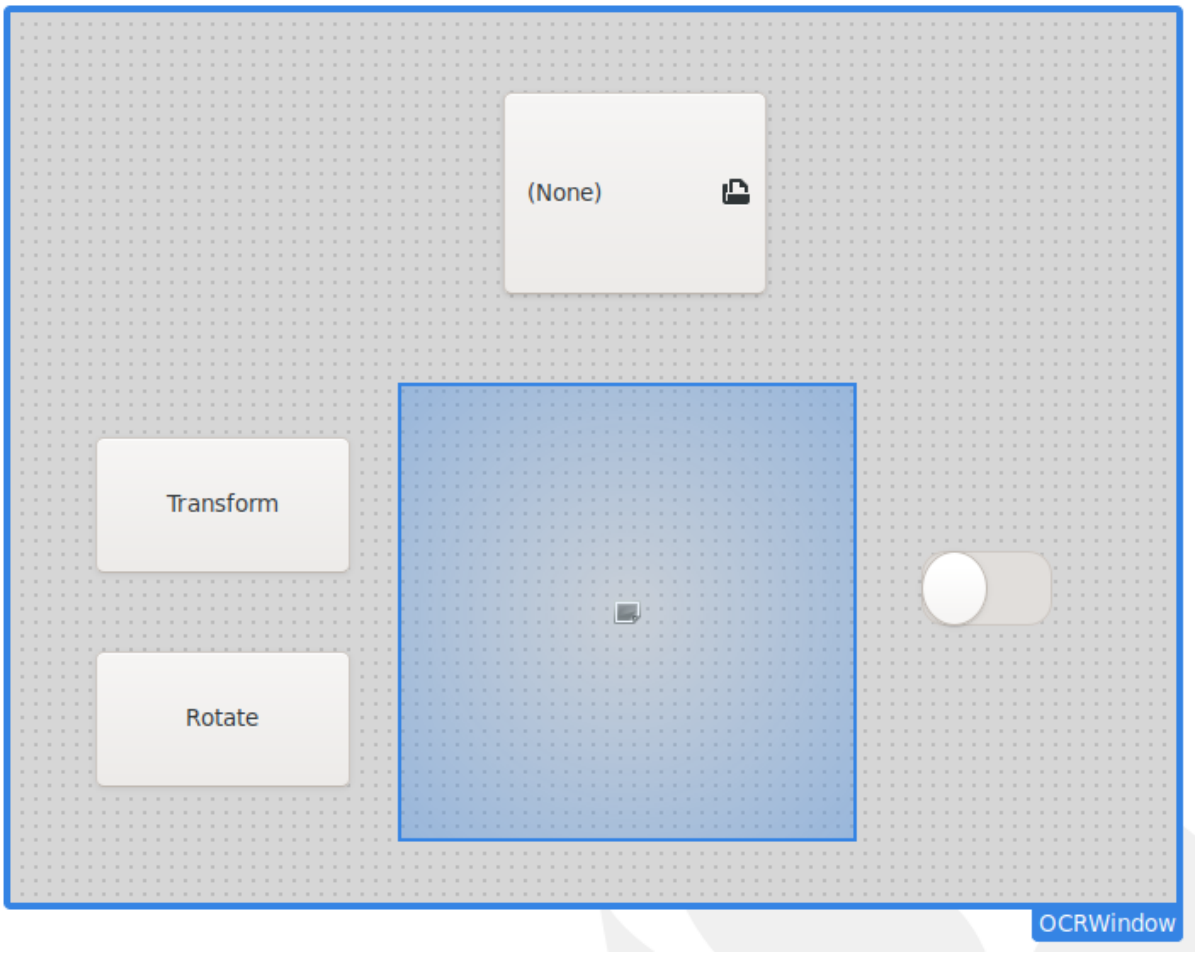
Grille résolue et affichée avec les chiffres trouvés grâce au solveur.

Pour la plupart des fonctions, la partie la plus compliqué est la recherche afin de trouver quel procédé nous allons faire pour parvenir à la fonctionnalité voulue. Il a été difficile de trouver la manière dans laquelle je voulais avancer. Néanmoins, une fois la fonction BlitSurface découverte, la fonction est devenue claire dans mon esprit et je savais à peu près comment obtenir le résultat attendu.

II.5 - Interface graphique

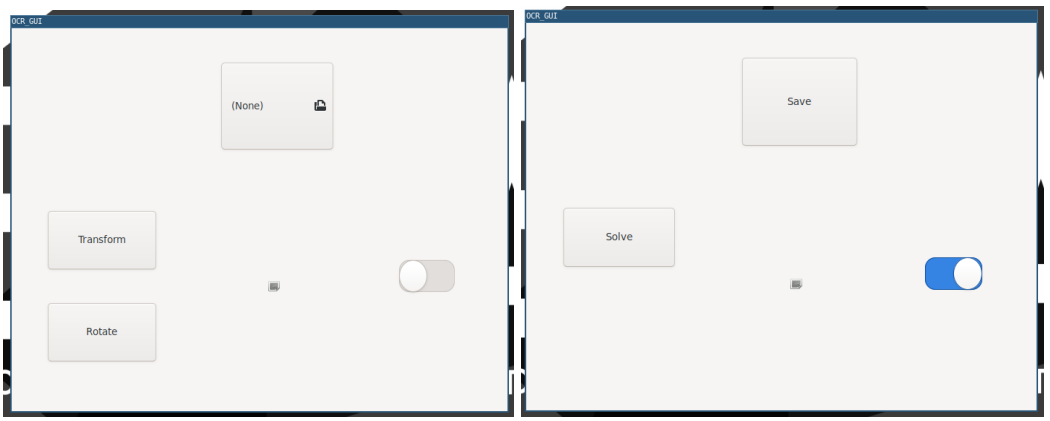
Pour cette soutenance, j'ai réalisé l'interface graphique. J'ai donc également lié les fichiers déjà créés pour les faire fonctionner entre eux.

Dans un premier temps, il a été nécessaire de créer un modèle sur Glade.Tout d'abord, on place une grille fixe sur la fenêtre qui accueillera les éléments. Sur celle-ci, on dispose les boutons et différents éléments qui forment l'interface graphique.

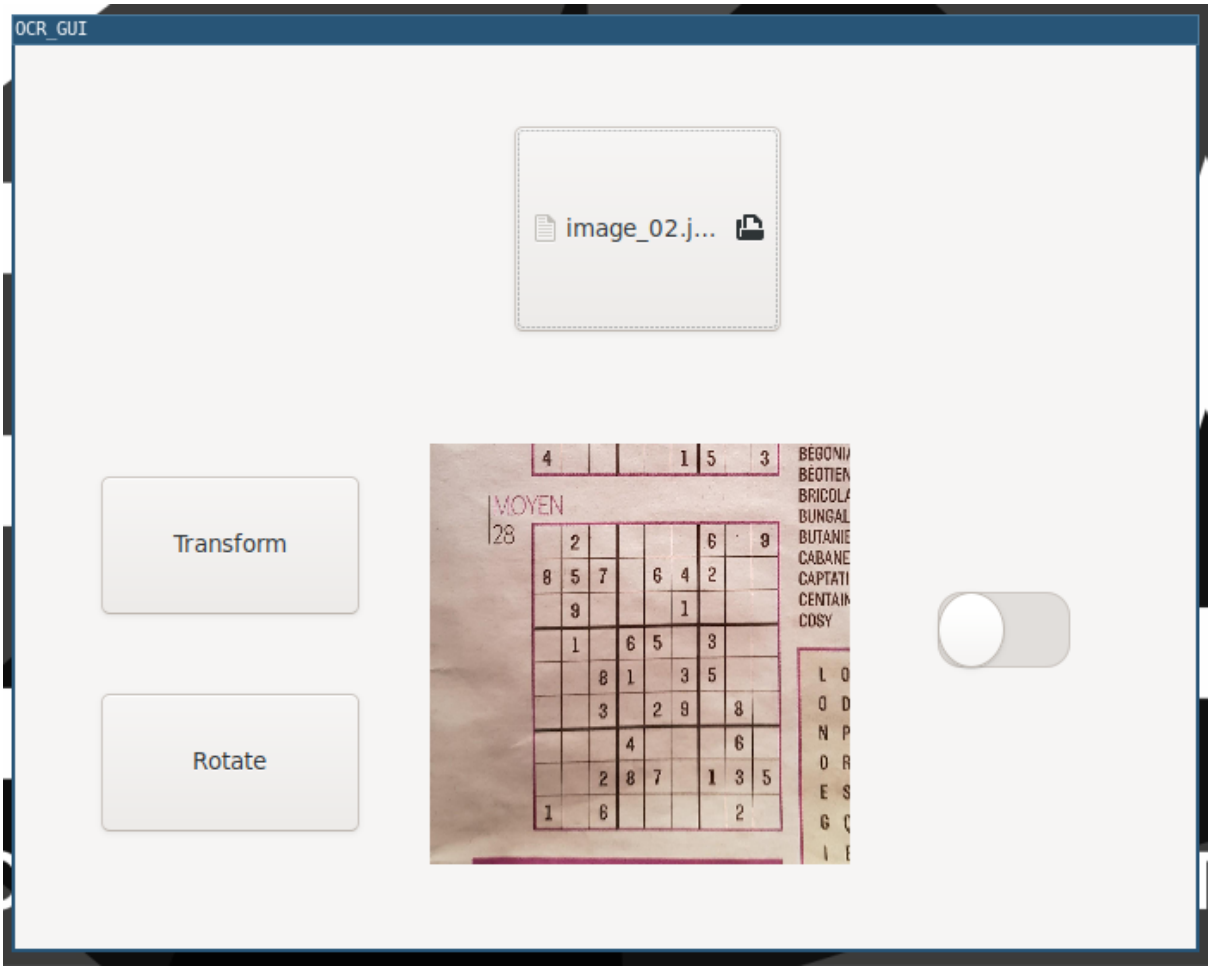


Au centre de l'application se trouve un affichage d'image. A gauche, des boutons qui permettent d'utiliser les fonctionnalités de l'application. En haut, un bouton spécial pour importer des fichiers et à droite, un bouton switch qui permet d'échanger entre le mode grille et le mode édition d'image.

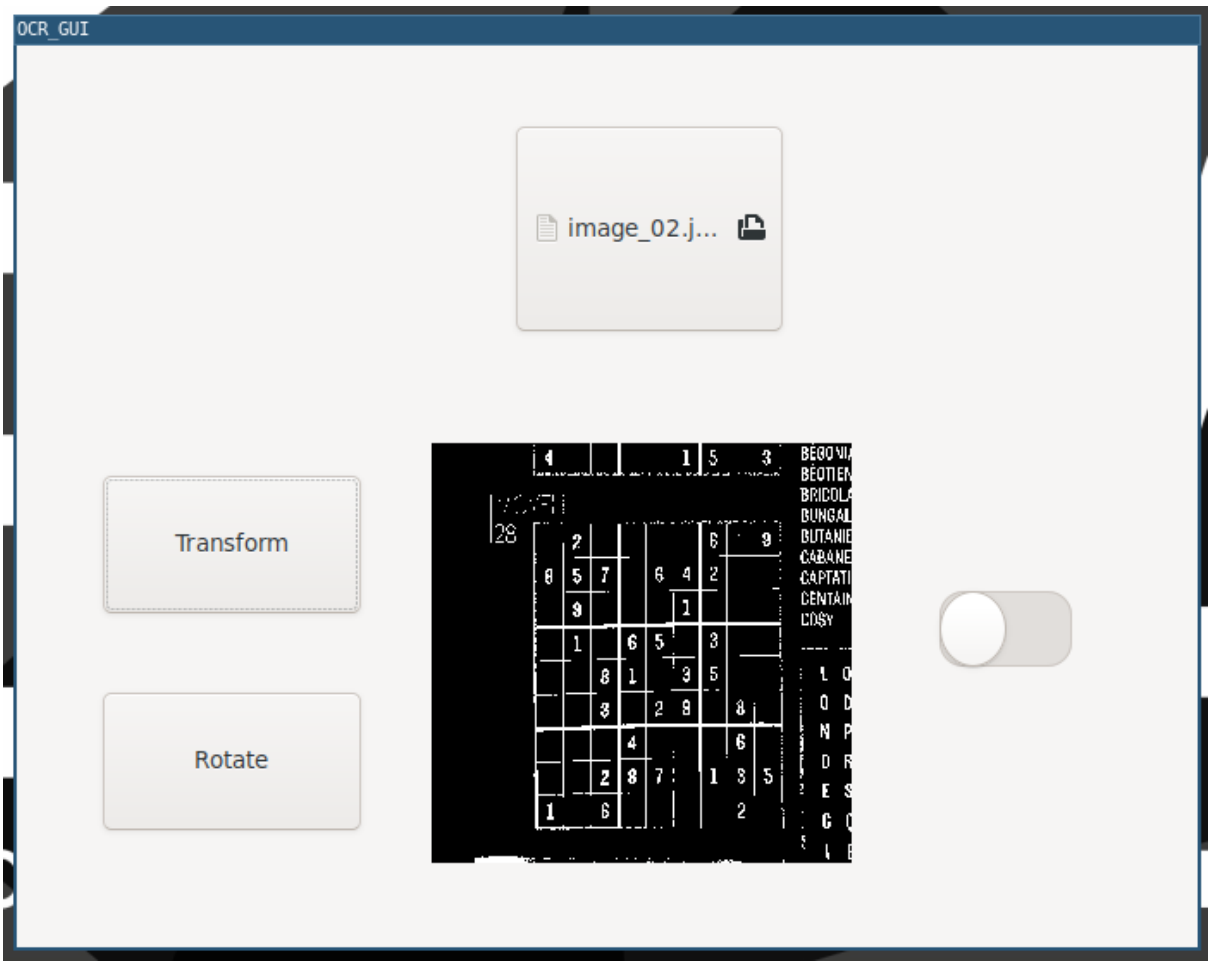
Maintenant, lorsque le bouton switch est cliqué, il fait apparaître et disparaître certains boutons à l'écran.



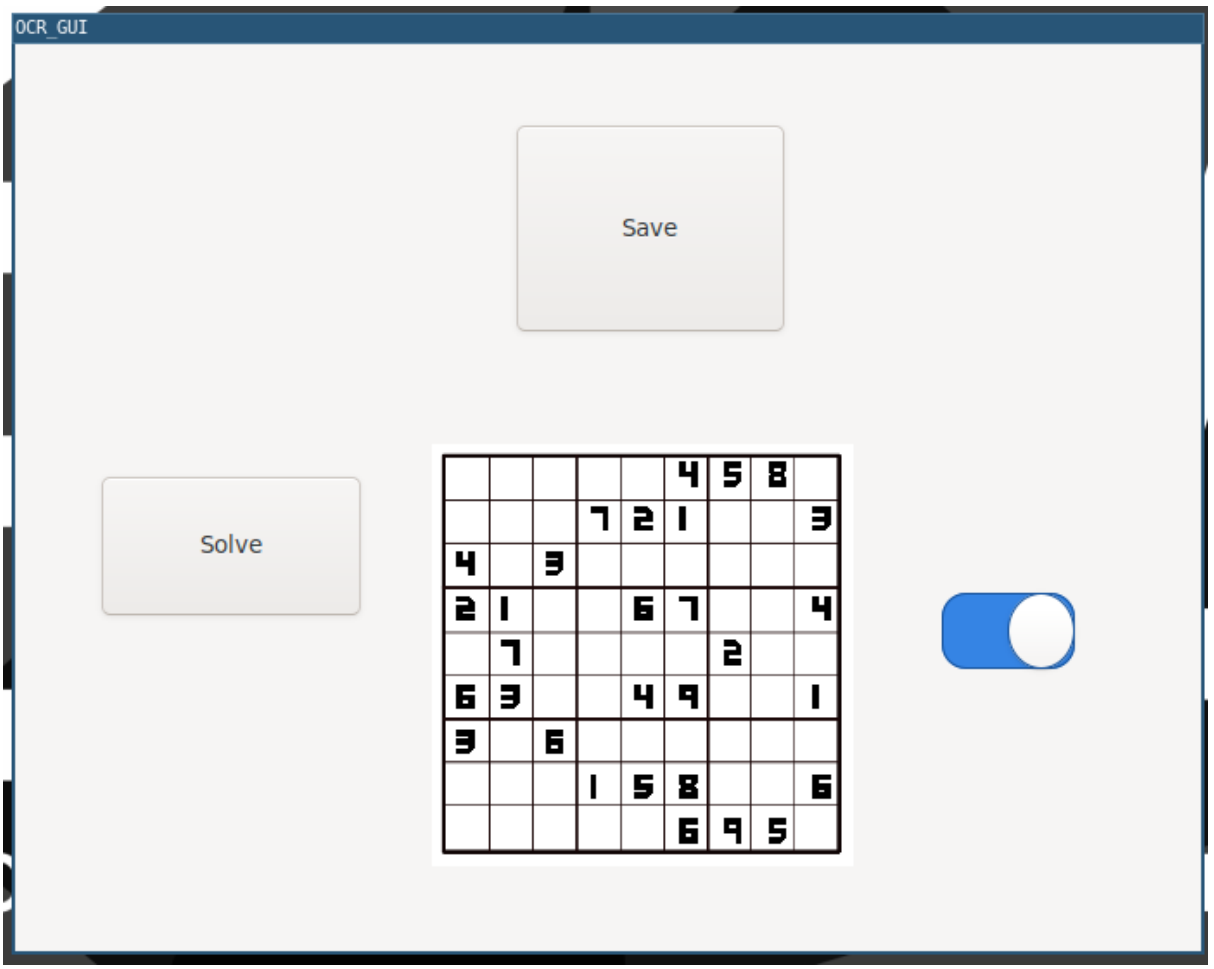
Dans le premier mode, le bouton d'import lance une fenêtre de dialogue. Ainsi, on obtient l'emplacement du fichier. On réalise ensuite un pixbuf pour redimensionner l'image et on actualise l'image du visualiseur avec ce pixbuf.



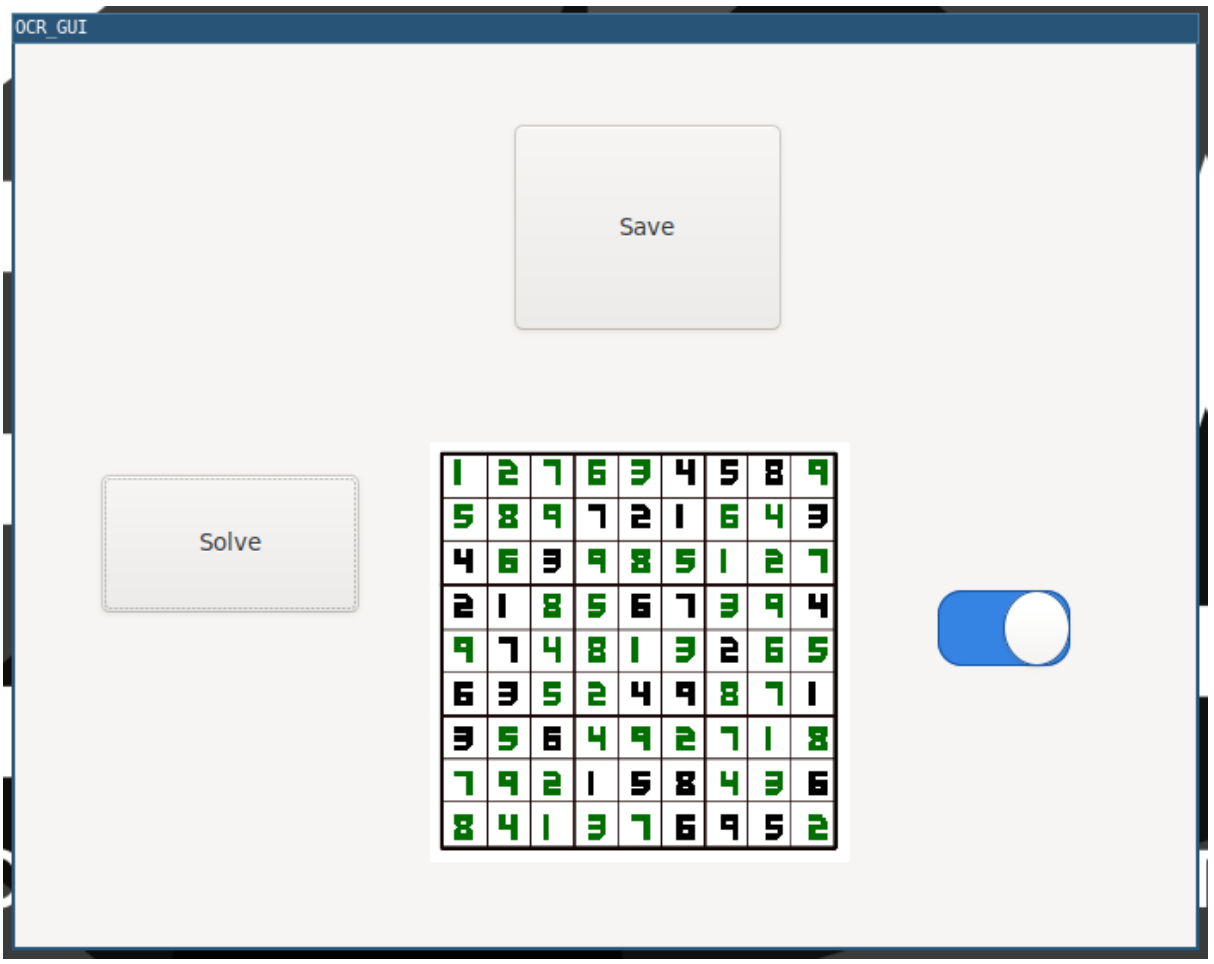
On peut ensuite lui appliquer les différentes modifications. Le premier bouton "Transform", change les couleurs de l'image pour faciliter le traitement. Le bouton "Rotate" tourne l'image afin que la grille soit droite.



Dans le mode grille, on obtient un affichage artificiel de la grille de Sudoku lorsque l'on change d'affichage. On affiche cette grille de la même façon que l'image importée en créant un fichier de l'image.



Enfin le bouton solve change l'image pour la grille résolue. Les chiffres ajoutés à la grille initiale après résolution apparaissent en vert.



Pour faire le liens avec les autres fichiers, il n'était pas possible de simplement les utiliser en les incluants puisque ces derniers ont déjà un main. J'ai d'abord essayé de créer une copie de ses fichiers sans leur main et d'ajouter le code enlevé dans les fonctions de l'interface graphique. Cette méthode étant approximative et posant des problèmes de compatibilité, elle a été abandonnée. Pour la remplacer, le programme émule une ligne de commande exécutant les autres programmes et l'image générée par ces derniers est utilisée comme stockage du résultat.

III - Conclusion

Le groupe est satisfait du projet réalisé même si les fonctionnalités tel que la détection de la grille, la segmentation des cases et le réseau de neurones ne sont pas disponibles. Pour ce dernier, il marche partiellement mais étant donné qu'ils manquaient des fonctionnalités et il n'est pas implémenté directement avec l'interface graphique.

Créer une application pour réaliser un sudoku résolu nous a permis de comprendre ce que représente véritablement un projet, en effet contrairement à ce que l'on a pu faire pendant les tps ou encore pendant l'AFIT, ce type de projet était réellement du développement de logiciel. Nous en ressortons tous enrichis avec de nouvelles compétences et une autre expérience à notre actif.