Rapport Soutenance Finale
Solver de Sudoku

Victor Tang Olivier Bensemhoun Marc Moutarde
Jayson Vanmarcke

11 Décembre 2023

Résumé

Ce rapport présente l'état final de notre grand projet *Sudoku OCR* à la date du 9 décembre 2023. Les différents élements obligatoires demandés sont tous fonctionnels et le bonus qui consistait à réaliser un site internet est également opérationnel.

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

A ce jour, le projet s'est déroulé en respectant les délais demandés et les choses se présentent de façon à ce que le projet soit terminé pour l'échéance requise. Chaque membre du groupe a fait preuve de rigueur afin de proposer un travail de qualité.

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 intéressé. 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 fonction des hypothèses émises et que ces hypothèses en engendrent 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 neurals networks m'ont toujours intéressé. 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 détails. 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			
Rotation automatique				Responsable
Détection de la grille et de la position des cases	Responsable			
Découpage de l'image				Responsable
Reconstruction des grilles	Responsable	Responsable		
Affichage de la grille		Responsable		
Résolution du sudoku/hexadoku		Responsable		
Sauvegarde du résultat	Responsable			
Réseau de neurones			Responsable	
Interface Graphique				Responsable
Site Internet		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
Rotation automatique	Fonctionnel
Détection de la grille et de la position des cases	Fonctionnel
Découpage de l'image	Fonctionnel
Reconstruction de la grille	Fonctionnel
Affichage de la grille	Fonctionnel
Résolution du sudoku	Fonctionnel
Sauvegarde du résultat	Fonctionnel
Réseau de neurones	Fonctionnel
Interface Graphique	Fonctionnel
Site Internet	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'informer surtout que la documentation officielle n'est pas très compréhensible et ne donne pas énormément 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 implémentation 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 outil spécialisé dans la manipulation d'image. J'ai donc dû 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 : dans un premier temps, nous appliquons un filtre permettant de passer la grille en niveau de gris standard, en utilisant les coefficients classiques. Ensuite, nous cherchons la valeur optimale pour le filtre passe haut en utilisant le filtre de Hotsu. Cet outil consiste a ne plus considérer l'image comme une image, mais un ensemble de raies de niveaux de gris. Ceci permet de trouver la valeur de niveaux de gris propre a chaque image à partir de laquelle on garde le maximum d'information utiles, tout en supprimant le plus de bruit possible.

Ceci permet d'obtenir une image plus nette même sur des images peu lisibles. Une fois ceci fait, le filtre de Sobel est applique, il s'agit d'un masque discret passant sur l'image permettant de mettre en valeur les contours : deux masques passent, l'un pour les abscisses, l'un pour les ordonnés. Ces deux masques sont réunis dans un seul, dans le but de réduire le nombre de calculs et donc la complexité de l'algorithme.

Nous calculons ensuite un gradient indiquant une probabilité qu'il y ait un bord a l'endroit observé, puis nous appliquons un filtre de contraste afin de mettre en valeur ce résultat.

Une fois ce filtre appliqué, nous utilisons une transformée de Hough pour détecter les différentes droites qui composent l'image. L'implémentation de cette dernière n'as pas changer depuis le dernier rapport, aucune amélioration potentielle n'ayant été vue.

2.4 Rotation automatique

La transformé de Hough retourne une liste de ligne potentielle dans l'image. Nous allons donc utiliser la première ligne de cette liste et on suppose qu'il s'agit de la verticale. Puis, on applique une rotation de $2\pi - \alpha$ avec α l'angle étant détecté par la transformé de Hough. Ceci permet de mettre l'image d'aplomb afin de passer par dessus la détection des cases faites à l'aide d'une réapplication de la transformé de Hough et ensuite l'application de l'IA après récupération de l'image et suppression des restes de bordures.

2.5 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 détecter 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 enregistrer comme étant les différentes cases du Sudoku recherché. Pour ce faire, nous avons trié 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 l'abscisse et des droites horizontales en fonction de leur ordonnée. Une fois ce tri fait, les deux listes sont parcourues afin de trouver les cases.

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

Le quadrilatère connexe définit par ces 4 points est ensuite extrait de l'image pour être placé sur une image de taille minimale au fond blanc puis sauvegardé. 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.

2.6 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 renvoie vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

check_ line(int i, int digit, int** sudoku) ;

Par la suite, j'ai réalisé 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 renvoie vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette colonne.

check_ colum(int j, int digit, int** sudoku) ;

Puis, j'ai refait l'étape à l'identique que précédemment mais dans un minicarré (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 renvoie 0 (faux) sinon 1 (vrai). Si le programme renvoie « vrai », alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

check_ square(int j, 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.

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

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.

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.

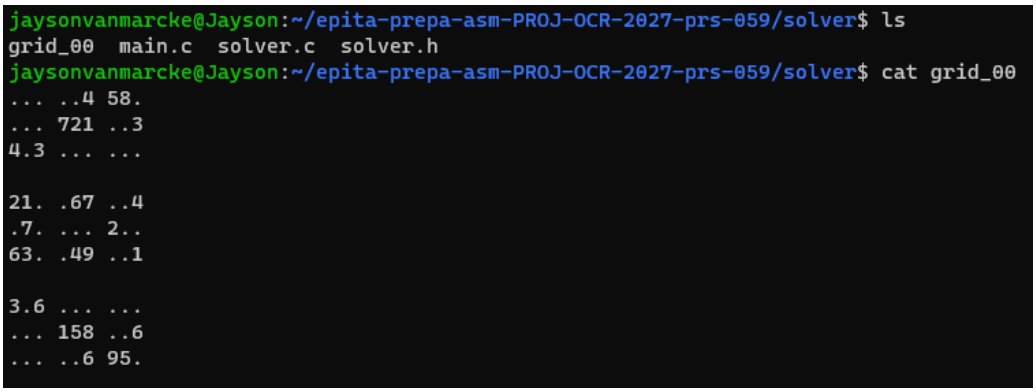


FIGURE 1 – Sudoku initialisé et sauvegardé

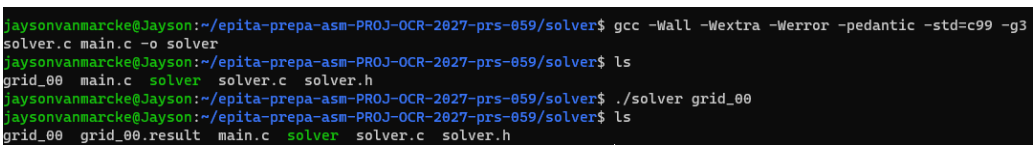


FIGURE 2 – Exécution du programme et du solver permettant d'obtenir un nouveau fichier

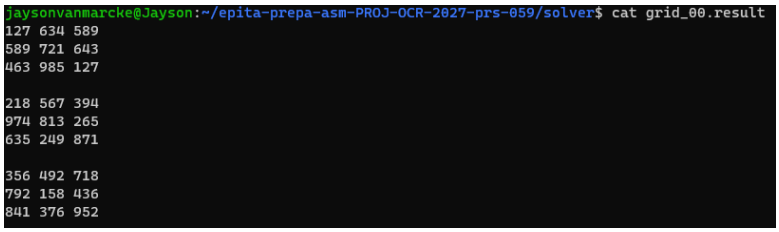


FIGURE 3 – Affichage du nouveau fichier contenant le sudoku rempli et du même format que la première image.

2.7 Reconstruction de la grille, affichage et sauvegarde

Cette partie consiste à reconstruire sous la forme d'image, une grille de sudoku qui a été résolue grâce à l'algorithme solver.

Tout d'abord, nous avons 2 fonctions principales pour réaliser cette tâche : la fonction *add_ number* qui ajoute l'image d'un nombre dans une case et détermine sa couleur. Puis, nous avons la fonction *generate* qui permet de générer une nouvelle image d'une grille résolue, de l'afficher et de la sauvegarder.

Dans un premier temps, commençons par détailler la fonction suivante : *add_ number*.

void add_ number(guchar* final_ img, int number, int x, int y, char mask[ ], int color) ;

Comme vous pouvez le constater, cette fonction est constituée de 6 paramètres :
— *final_ img* : une variable qui va contenir l'image finale avec les chiffres ajoutés
— *number* : il s'agit du chiffre à ajouter dans l'image finale
— x et y : permettant de savoir sur quelles lignes et colonnes on se trouve
— *mask* : nous permet de savoir si on est déjà passé sur une case
— *color* : nous permet de savoir si on change la couleur ou non du chiffre que l'on ajoute

Au début de notre fonction, on vérifie si le nombre est différent de zéro ou que l'on est déjà passé sur la case en question. Si on n'a pas parcourue la case et que le nombre est différent de 0, il faut mettre à jour le masque afin d'éviter de retravailler sur la même case. Ensuite, on calcule des coordonnées permettant de savoir où coller notre image comportant un nombre.

Puis, nous regardons la couleur : elle sera noir si le paramètre *color* est à 0, sinon, elle sera rouge. Ensuite, on assigne la variable number à une image précise en fonction du chiffre assigné. Par ailleurs, nous faisons 2 boucles for avec la taille de l'image contenant le nombre (la taille de notre image est de 100 par 100 pixels).

Lors du parcours, si la valeur rencontrée est 255, cela signifie que nous sommes sur le fond(partie blanche de l'image) donc nous mettons les 3 canaux à 255 pour représenter la couleur blanche et dans le cas contraire, si elle est inférieur a 255, cela signifie que nous sommes sur le chiffre donc il faut mettre la couleur à (*pixelcolor*,0,0)(rouge ou noir). *Pixelcolor* est une variable qui prend la valeur 255 lorsque le paramètre *color* est 1 ou 0 si le paramètre *color* est à 0.

Détaillons maintenant la fonction suivante : *generate*

GdkPixbuf* generate(grid_path) ;

Cette fonction n'est constituée que d'un seul paramètre. En effet, ce paramètre intitulé *grid_path* est le chemin d'une grille de sudoku. Grâce à ce chemin, on peut obtenir dans un "char grid[ ][ ]" la matrice correspondante au sudoku avec la fonction *readsudo* qui est expliquée dans la partie du solver.

Ensuite, nous ajoutons tous les nombres de départ avec la fonction *add_number* en donnant zéro comme paramètre de la couleur pour dessiner en noir. Avant les appels, nous créons un masque qui permettra de savoir si cette case a déjà été dessinée.

Puis, on appelle la fonction *solver_sudoku* (expliquée dans la partie solver aussi). La fonction *solver_sudoku* renvoie "vrai" si le sudoku s'est bien résolue et met à jour le paramètre sudoku donné en le remplissant avec les bons chiffres, placés aux bons endroits en fonction de la grille de base.

En vérifiant la valeur rendue par *solver_sudoku*, on peut afficher une image prédéfinie indiquant que le sudoku est insoluble sinon, on rappelle la fonction *add_number* en donnant 1 comme paramètre pour la couleur signifiant qu'on souhaite mettre la solution du sudoku en rouge, plus précisément, le chiffre qui n'était pas présent dans la grille initiale. Enfin, nous créons un *GdkPixbuf** à partir de *img_final* qui contient les pixels de l'image finale. On renvoie le *GdkPixbuf** afin de l'afficher sur notre interface graphique et, par la même occasion, se sauvegardera dans les dossiers de l'application.

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

Après quelques complications, nous arrivons au résultat suivant :

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

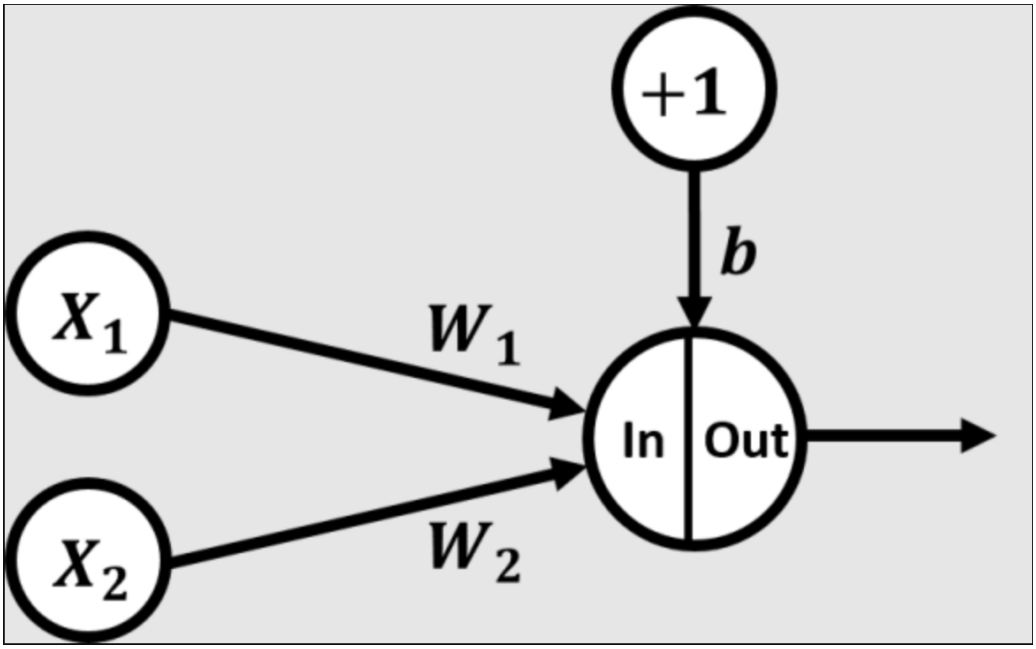
Par ailleurs, nous vous informons que les soucis rencontrés lors de l'exécution de cette tâche, vous serons détaillés dans la partie : Problèmes rencontrés.

2.8 Réseau de neurones

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

La première partie porte 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 précédente et le résultat du neurone de la layer précédente. 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.

Nous allons maintenant détailler le fonctionnement des différentes fonctions créés dans la bibliothèque que j'ai créé : "network.h".

Architecture du Réseau de Neurones : La fonction "struct Neural** initnetwork()" est responsable de l'initialisation du réseau de neurones. Cela implique la création des couches du réseau et l'allocation de mémoire nécessaire. Le réseau semble être construit avec une structure de pointeurs vers des couches neuronales. Une fois le réseau de neurone testé, nous pouvons utiliser free network() pour le libérer de la mémoire. De plus, nous avons set_network qui permet d'initialiser les valeurs de la première layer du réseau de neurones.

```c
struct Neural{
        double biases;
        void *previouslayer;
        double *weights;
        double value;
};
```

```c
struct Neural** initnetwork( );
```

```c
void free_network(struct Neural **network);
```

```c
void set_network( struct Neural **network , double *array ,size_t size);
```

Fonctions d'Activation : Les fonctions "double sigmoid()" et "double active func()" sont les fonctions qui utilisent la formule d'activation. La fonction active_func calcule la valeur de z dans la formule puis retourne le résultat de l'appel de la fonction sigmoïd.

```
double sigmoid(double z);
```

```
double active_func(struct Neural neural,size_t size);
```

Fonction Softmax : La fonction "size_t softmax()" est utilisée pour appliquer la fonction softmax à la sortie du réseau. La fonction softmax est utilisée pour obtenir des probabilités normalisées à partir des valeurs de la dernière layer du réseau.

```
size_t softmax(struct Neural *neural,size_t size, double *res);
```

Fonction use_network : La fonction "size_t use_network" est appliquée pour calculer le résultat du réseau de neurones en utilisant la fonction active_func pour chaque neurone sauf la première layer (qui stock juste les valeurs).

```
size_t use_network(struct Neural **network);
```

Entraînement du Modèle : L'entraînement du modèle est réalisé à l'aide de la fonction 'void train()'. Cette fonction utilise l'algorithme de descente de gradient pour ajuster les poids du réseau et minimiser la fonction de coût. Pour optimiser l'entraînement, la fonction train applique des minibatch de taille prédéfinie.

```
void train(struct Neural **network,double **vectorin,double *vectorout,
»·····»·····size_t nbtest,double tvs );
```

Fonctions liées à l'Entraînement : - 'void active_func_train()' : cette fonction est une version de la fonction d'activation spécifiquement adaptée à l'entraînement. Elle est optimisée pour réduire le nombre de calculs qui donne des informations inutiles pour l'entraînement. - 'double cost' : la fonction calcule le coût du modèle qui mesure la différence entre la valeur de sortie attendue et son résultat.

```
double cost(struct Neural **network,size_t  exepted);
```

- 'void backpropagation()' : la backpropagation est un élément clé de l'entraînement des réseaux de neurones. Cette fonction pourrait être responsable de la propagation de l'erreur à travers le réseau.

```
void backpropagation(struct Neural** network,size_t exepted,
»·····»·····double *set, double **w, double **b, double tvs);
```

- 'void gradient_descent()' : cette fonction utilise l'algorithme de descente de gradient, ajustant ainsi les poids du réseau pour minimiser la fonction de coût.

```
void gradient_descent (struct Neural **network,
»·····»·····double **gradientw, double **gradientb);
```

Sauvegarde et Chargement du Modèle : Les fonctions 'void save_network()' et 'struct Neural** initfromfile' permettent la sauvegarde et le chargement du modèle depuis un fichier texte prédéfini . Cela permet de stocker les poids du modèle après l'entraînement et de les charger ultérieurement pour une utilisation ou une poursuite de l'entraînement.

```
struct Neural** initfromfile();
```

```
void save(struct Neural **network);
```

Creation d'un dataset custom : La fonction 'void add dataset dir()' permet de rajouter un set d'image trié dans des sous-dossiers ayant pour nom leurs valeurs attendues. Cela aide à de rajouter un dataset préexistant pour en réaliser une version custom.

Sauvegarde d'un dataset : La fonction 'void save dataset()' permet d'enregistrer le dataset dans le fichier 'dataset.txt'.

```
void save_dataset(
    double** dataset, int* value, size_t nb, size_t len, int type);
```

Accès d'un dataset custom : La fonction 'void get dataset()' permet de récupérer un dataset sauvegarder depuis le fichier dataset.txt. Le principal intérêt de cette fonction est la possibilité de récupérer un dataset Custom.

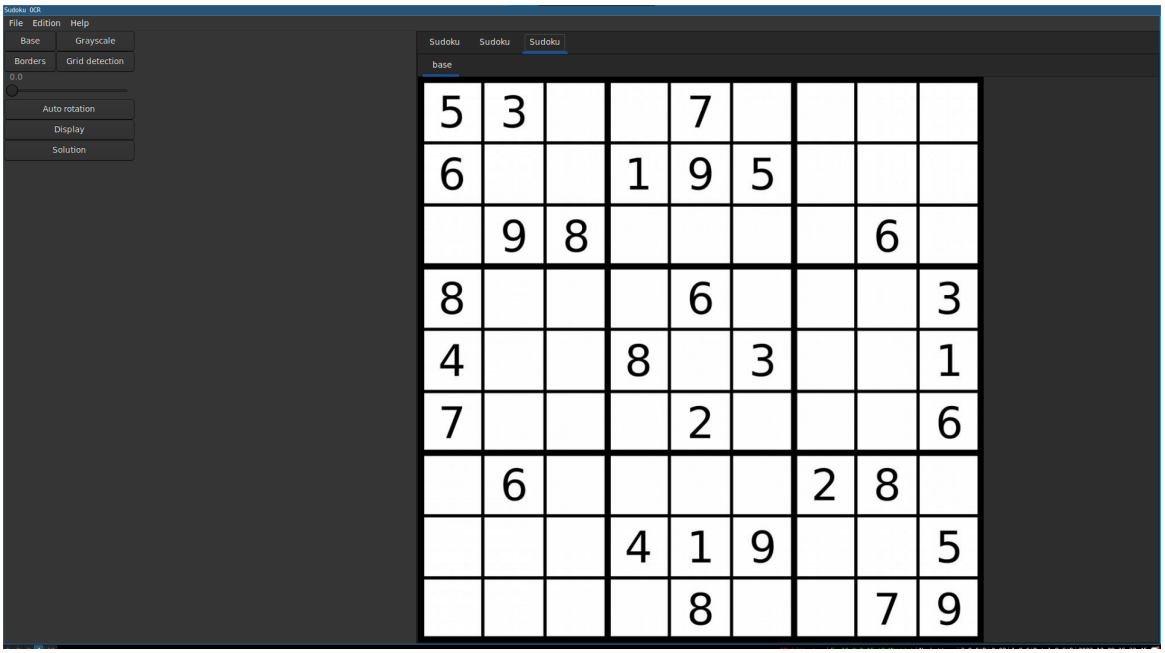
```
void get_dataset(size_t nb, size_t len, double*** dataset_p, int** output_p);
```

Entraînement sur la Base de Données MNIST : L'entraînement du modèle a été effectué par mini-batch, une approche courante pour améliorer l'efficacité de l'entraînement. Des images de la base de données MNIST ont été utilisées par lots pour ajuster les poids du réseau. Cependant, MNIST est uniquement basé sur des écritures manuscrites. Pour corriger le problème, nous avons rajouté des images de chiffres provenant de la base de données : https ://www.kaggle.com/datasets/shreyasshrawage/digits. Ce qui nous permet d'obtenir de meilleurs résultats sur des chiffres écrits sur un ordinateur.

Conclusion : Le développement d'un modèle de Deep Learning pour la reconnaissance optique de caractères est finalement fonctionnelle. Les fonctions que nous avons implémentées telles que l'initialisation du réseau, les fonctions d'activation, la backpropagation, la descente de gradient, l'entraînement sur des mini-batchs, et la sauvegarde/chargement du modèle, permettent d'avoir une IA facilement compressible avec de multiples informations et de pouvoir l'utiliser.

2.9 Interface graphique

Pour ce qui est de l'implémentation de l'interface graphique, le plus long était de créer une fenêtre GTK avec plusieurs propriétés. Ensuite, nous avons ajouté petit à petit des fonctionnalités supplémentaires.

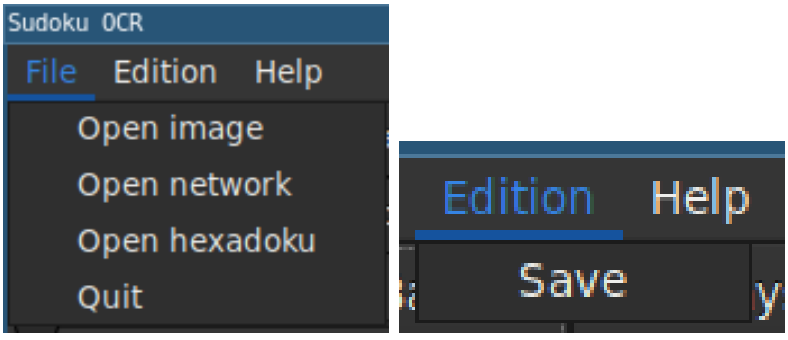


Le plus important fût d'ouvrir une image afin d'appliquer les modifications souhaitées. Pour cela, nous avons dérivé la classe GtkDrawingArea pour l'adapter à nos besoins. En effet, la classe GtkImage de base ne fournissait pas la capacité de tourner ou de redimentionner l'image comme nous le souhaitions. Nous avons donc créé une classe Image qui permet de traiter les images comme nous le voulions.

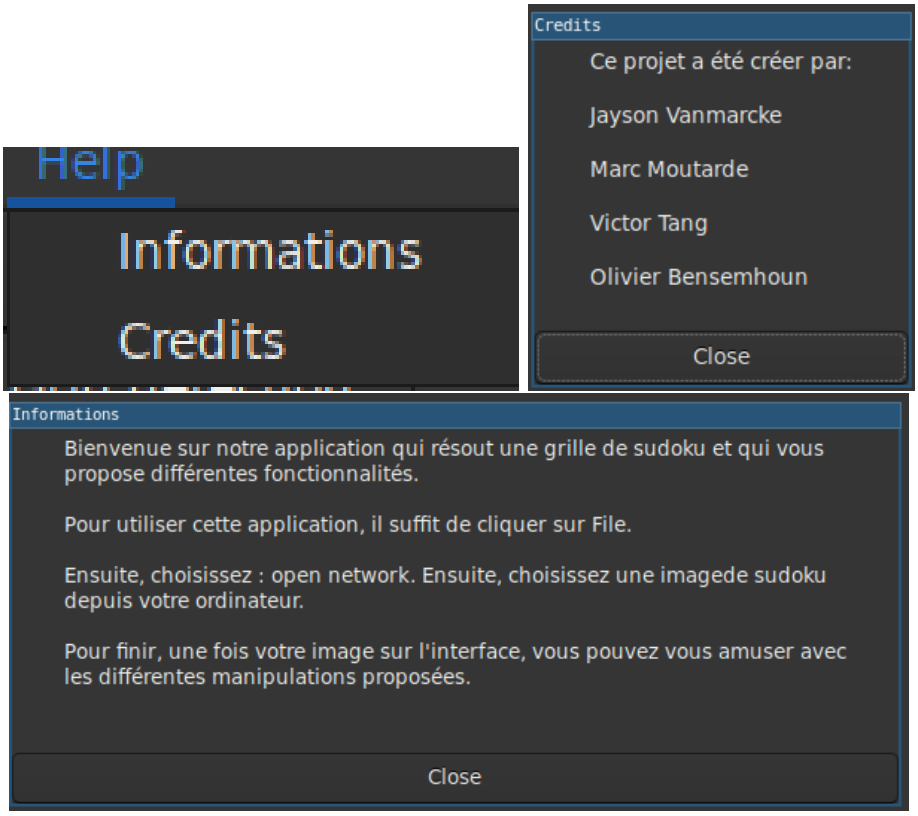
Ensuite, nous avons ajouté un menu qui a 3 boutons : File, Edition et Help.

En cliquant sur le bouton File, plusieurs options apparaissent :
— Open Image : permet d'ouvrir une image depuis les dossiers de l'ordinateur
— Open Network : permet d'ouvrir une liste de poids pour le réseau de neurones
— Open Hexa : permet d'ouvrir une grille d'un hexadoku et d'afficher son résultat
— Quit : permet de quitter l'application

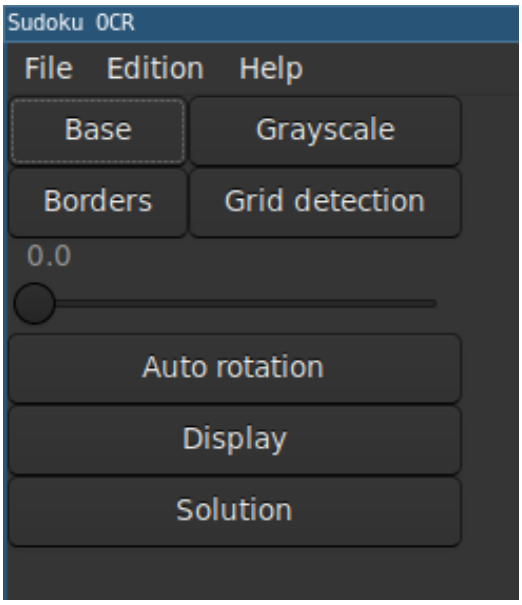
Ensuite, en cliquant sur Edition, un bouton save apparaît et permet de sauvegarder l'image affichée sur l'application.



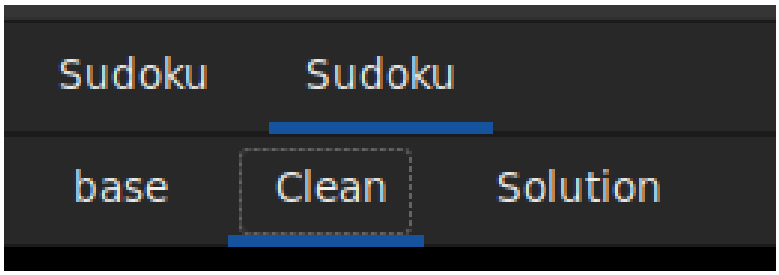
Le dernier bouton du menu est le bouton Help. Il permet d'avoir les informations nécessaires sur l'utilisation de l'application ainsi que les crédits des concepteurs de celle-ci.



Après avoir ouvert une image, l'utilisateur peut appliquer plusieurs filtres comme l'échelle de gris ou l'affichage des bordures mais aussi réafficher l'image de base. Dans le cas contraire, l'utilisateur peut appuyer sur le bouton *Auto Rotation* pour tourner automatiquement l'image ouverte afin d'avoir une meilleure détection de la grille du sudoku. Dans le cas où cela ne conviendrait pas à l'utilisateur, il pourra lui même appliquer une rotation entre 0 et 360 degrés. Ensuite, viennent le bouton*Display* qui permet d'afficher une reconstruction de la grille detectée par le réseau de neurones et enfin, le bouton Solution qui permet de résoudre la grille de sudoku et d'obtenir le résultat.



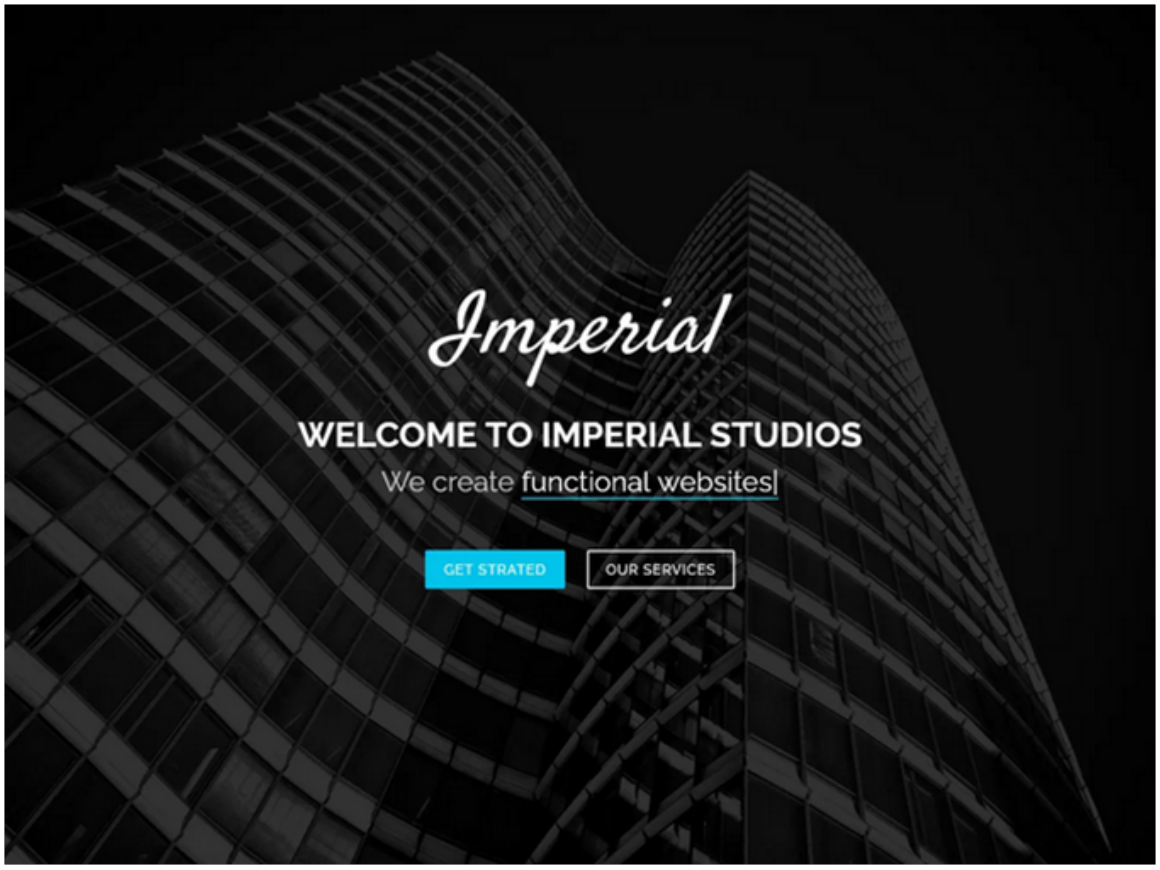
Les images sont contenues dans un *notebook* qui lui même est contenu dans un *notebook* contenant à son tour les images. Cela permet de se retrouver plus facilement lorsque l'utilisateur ouvre plusieurs sudokus.



2.10 Site Internet

Ce site internet sera la vitrine de notre projet. Il doit donc posséder un beau design et être simple de navigation. J'ai apprécié faire un nouveau site Internet pour le projet, car j'ai aussi effectué cette tâche pour le projet du S2.

Pour réaliser le site de notre projet, nous avons utilisé un Template : Bootstrap afin d'avoir une base sur laquelle travailler. Ce Template Imperial est un modèle d'une page Bootstrap moderne et créatif, idéal pour les agences de création, les studios, les agences de design numérique ou d'autres entreprises similaires. L'en-tête est livré avec une partie héros en plein écran. Imperial est également livré avec un menu mobile moderne hors toile pour une meilleure navigation et une meilleure expérience utilisateur. Bien évidemment, j'ai changé plusieurs lignes de codes afin d'obtenir un excellent résultat.

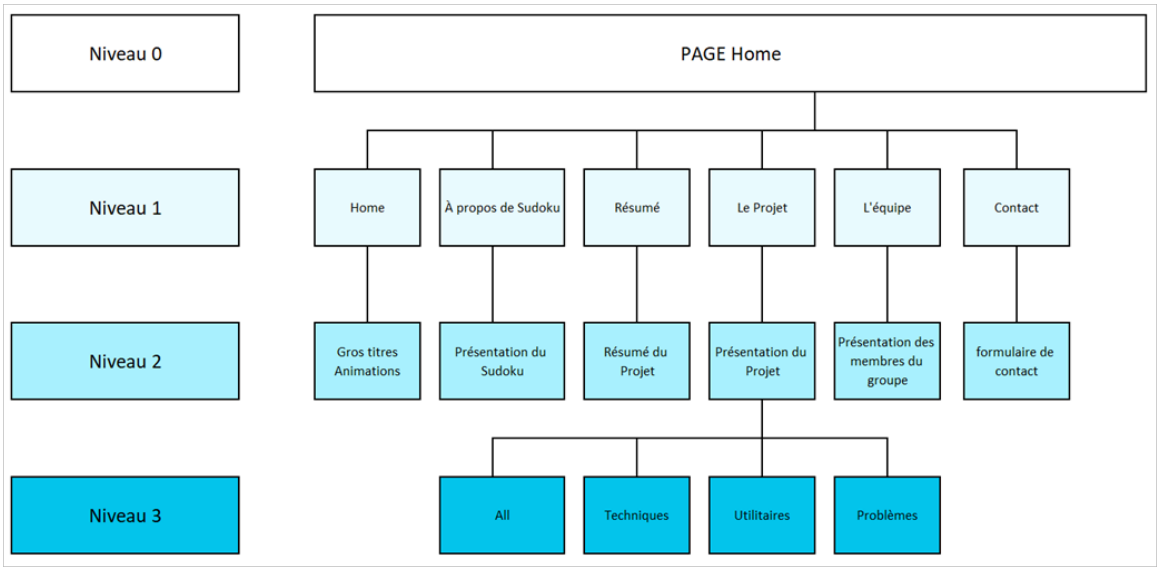


Commençons par parler de la conception graphique du site. La charte graphique doit être entièrement définie. Pour ce faire, nous avons défini les logos, couleurs et typographie que nous poserons sur un style graphique simple mais moderne.

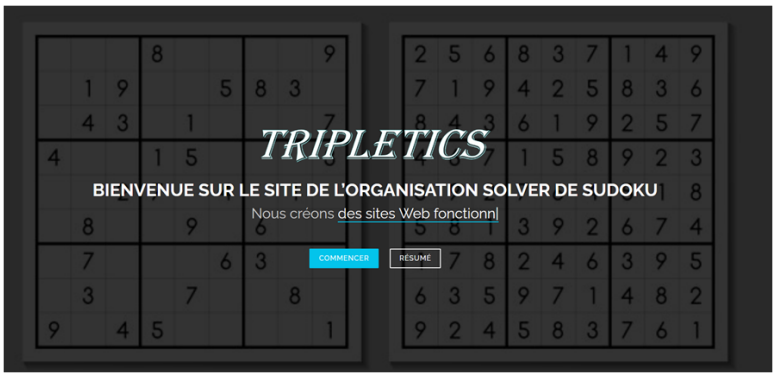
Le Logo est représenté par le nom de notre groupe, il est blanc sans background.

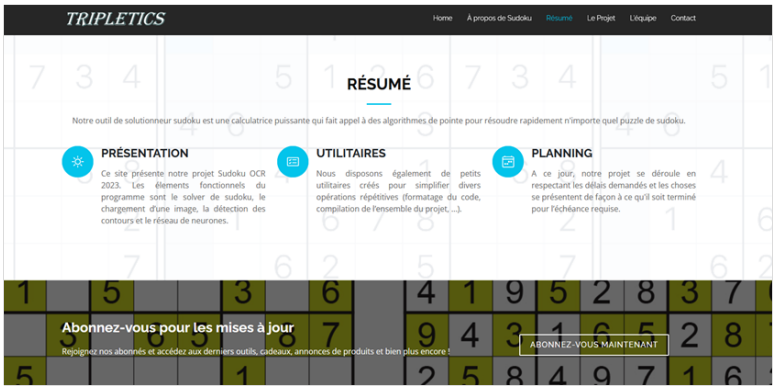
Ensuite, parlons des spécifications techniques. Le site est développé en partie sur une seule page appelée « one page », seule la partie présentation du projet renvoie pour chaque thème vers une autre page. Ce type de site est adapté car il y a peu de contenu et il présente l'avantage de mieux contrôler la navigation des internautes en proposant une expérience utilisateur adapté. Cela permet d'améliorer la conversion du site web, ce qui favorise le référencement.

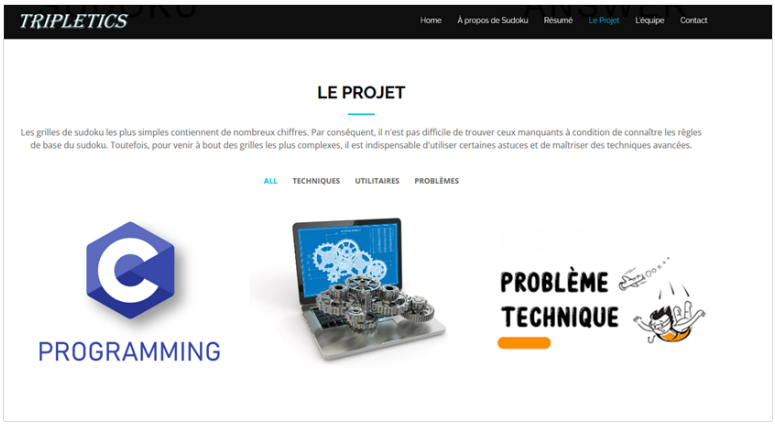
Voici l'arborescence de notre site Internet :

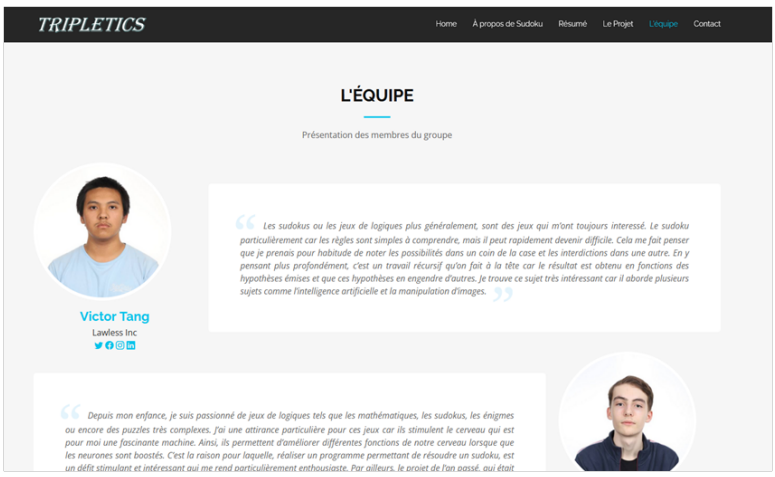


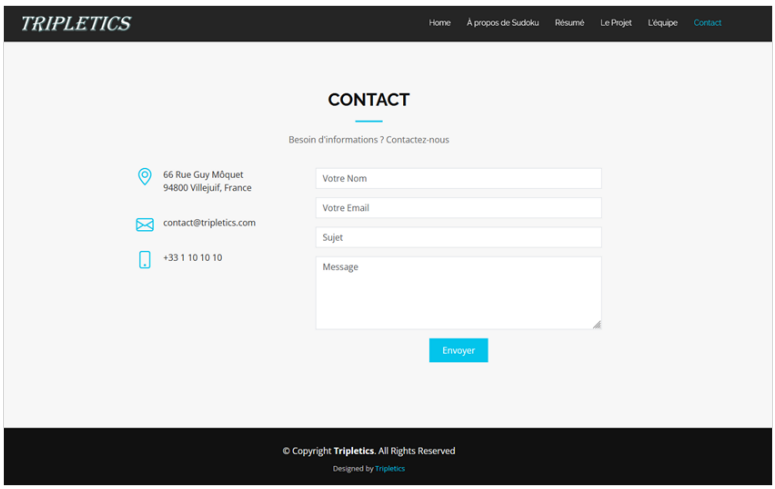
Laissez-moi vous montrer quelques capture de notre sites internet :




TRIPLETICS
Home
À propos de Sudoku
Résumé
Le Projet
L'équipe
Contact
RÉSUMÉ
Notre outil de solutionneur sudoku est une calculatrice puissante qui fait appel à des algorithmes de pointe pour résoudre rapidement n'importe quel puzzle de sudoku.
PRÉSENTATION
Ce site présente notre projet Sudoku OCR 2023. Les éléments fonctionnels du programme sont le solver de sudoku, le chargement d'une image, la détection des contours et le réseau de neurones.
UTILITAIRES
Nous disposons également de petits utilitaires créés pour simplifier divers opérations répétitives (formatage du code, compilation de l'ensemble du projet,...).
PLANNING
A ce jour, notre projet se déroule en respectant les délais demandés et les choses se présentent de façon à ce qu'il soit terminé pour l'échéance requise.
Abonnez-vous pour les mises à jour
Rejoignez nos abonnés et accédez aux derniers outils, gadeaux, annonces de produits et bien plus encore !
ABONNEZ-VOUS MAINTENANT


TRIPLETICS
Home
À propos de Sudoku
Résumé
Le Projet
L'équipe
Contact
LE PROJET
Les grilles de sudoku les plus simples contiennent de nombreux chiffres. Par conséquent, il n'est pas difficile de trouver ceux manquants à condition de connaître les règles de base du sudoku. Toutefois, pour venir à bout des grilles les plus complexes, il est indispensable d'utiliser certaines astuces et de maîtriser des techniques avancées.
ALL
TECHNIQUES
UTILITAIRES
PROBLÈMES
PROGRAMMING
PROBLÈME TECHNIQUE


TRIPLETICS
Home
À propos de Sudoku
Résumé
Le Projet
L'équipe
Contact
L'ÉQUIPE
Présentation des membres du groupe
Victor Tang
Lawless Inc
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.
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


TRIPLETICS
Home
À propos de Sudoku
Résumé
Le Projet
L'équipe
Contact
CONTACT
Besoin d'informations ? Contactez-nous
66 Rue Guy Môquet
94800 Villejuif, France
contact@tripletics.com
+33 1 10 10 10
Votre Nom
Votre Email
Sujet
Message
Envoyer
© Copyright Tripletics. All Rights Reserved
Designed by Tripletics

Pour finir, notre site sera compatible avec les navigateurs suivants : Microsoft Edge, Mozilla Firefox, Google Chrome, Safari ou encore Opéra. Le site est conçu de manière dite “responsive” pour qu'il assure une navigation optimale sur tous types d'appareils :Téléphones mobiles, tablettes, ordinateurs portables ou encore ordinateurs de bureau.

Voici le lien de notre site (nous l'avons hébergé sur un service en ligne gratuit) : Site sudoku

2.11 Solver Hexadoku

L'hexadoku est un puzzle plus stimulant et intéressant que le sudoku classique. En effet, le joueur n'a plus une grille de 9*9 mais une grille de 16*16, comprenant les lettres hexadécimales.

Le principe d'un hexadoku est identique au sudoku. 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 16*16 a des cases qui sont au préalable déjà remplies. Ainsi, la consigne est d'assigner des chiffres compris entre 0 et 9 puis des lettres comprises de A à F dans les cases de cette grille qui sont vides. Cependant, il est strictement interdit selon les règles de l'hexadoku d'avoir un doublon sur une ligne, une colonne ou dans un mini carré de taille 4*4. 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 solverhexa en différentes étapes prédéfinies :

Dans un premier temps, j'ai créé un programme permettant de vérifier si une case est vide dans l'hexadoku afin d'y inscrire un chiffre compris entre 0 et 9 et une lettre comprise entre A et F.

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 de l'hexadoku, 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 renvoie vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

check_ line(int i, char 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 renvoie vrai, alors on sait que l'on peut mettre le chiffre 2 sur cette colonne.

check_ colum(int j, char digit, int** sudoku) ;

Puis, j'ai refait l'étape à l'identique que précédemment mais dans un mini carré (4*4) 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 renvoie 0 (faux) sinon 1 (vrai). Si le programme renvoie « vrai », alors on sait que l'on peut mettre le chiffre 2 sur cette ligne.

check_ square(int j, int j, char digit, int** sudoku) ;

Pour finir, j'ai conçu le programme IsValidSudoku qui renvoie vrai si les 3 programmes précédents sont vérifiés. L'hexadoku sera alors valide et nous pourrons donc réaliser le solver de l'hexadoku qui résout l'hexadoku en entier.

IsValidSudoku(int i, int j, char digit, int** sudoku) ;

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 hexadoku incomplet est sauvegardé afin de le lire et le stocker.

readhexadoku(char* file, int** sudoku) ;

A partir du moment où la lecture est effectuée, je vérifie si l'hexadoku 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 l'hexadoku résolu en respectant les normes du fichier d'origine. Nous pouvons ainsi trouver un nouveau fichier créé contenant la grille de l'hexadoku remplie. Si l'hexadoku 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 lettres et qu'il n'y ait aucun chiffre/lettres en double sur une ligne, une colonne ou un carré de 4*4. Le format de sortie est le même que le format d'origine.

```
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ ls
Makefile  grid_00  grid_01  grid_02  hexa.c  hexa.h  main.c  mainhexa.c  solver.c  solver.h
jaysonvanmarcke@Jayson:~/epita-prepa-asm-PROJ-OCR-2027-prs-059/solver$ cat grid_02
9... .6.F C.5D ....
.1DF ..C. 8B.9 5...
CE.. 4... 12F0 B.A9
B... .97. ..6. C..1

...B ..E3 .... D.6.
0698 5.DB F..3 14E2
..CD 8... .604 .5.B
...1 9... 5... .8.7

D... C... 9..6 3...
.... F..D ...E .9B.
.964 3.27 A... ..D0
.... B16. .4.. ...5

6... 7D8. E... 0.C.
2F5. ..9C D... A.1.
A... ..5. BC.. 2.9E
78EC 2..0 6..F 4...
```

FIGURE 4 – Hexadoku initialisé et sauvegardé

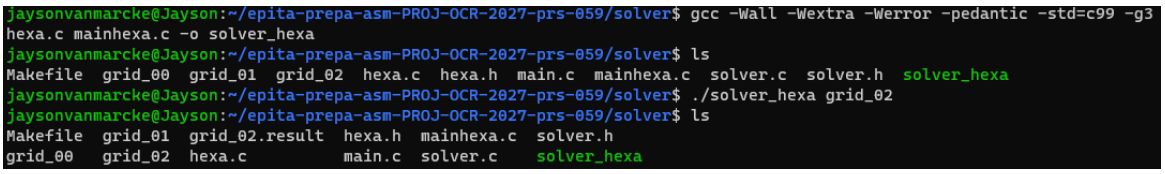


FIGURE 5 – Exécution du programme et du solver permettant d'obtenir un nouveau fichier

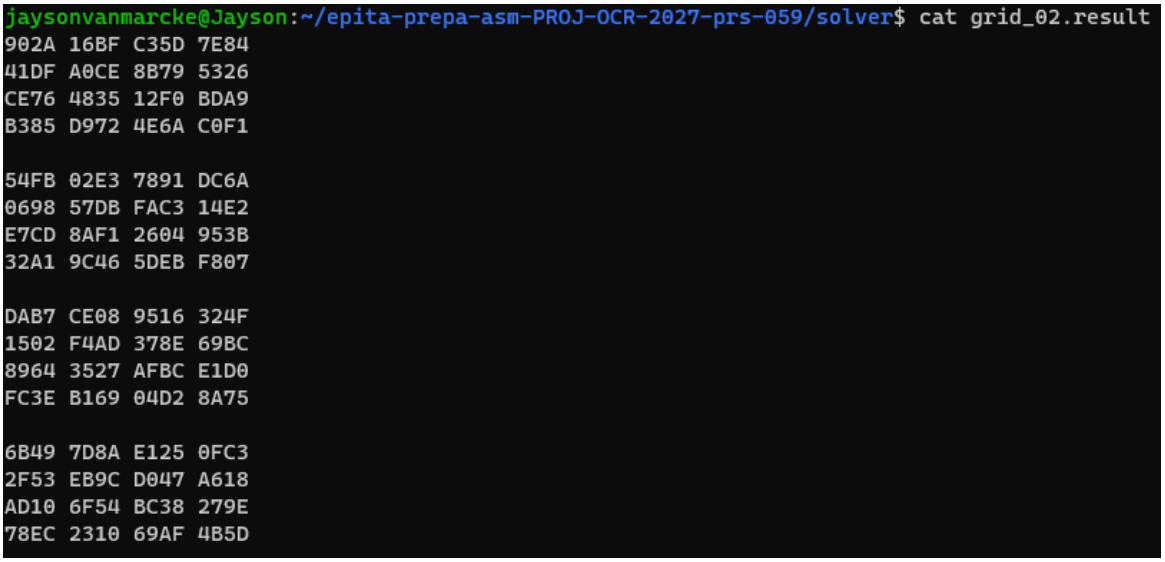


FIGURE 6 – Affichage du nouveau fichier contenant l'hexadoku rempli et du même format que la première image.

2.12 Reconstruction de la grille (hexadoku), affichage et sauvegarde

Cette partie consiste à reconstruire sous la forme d'image, une grille d'hexadoku qui a été résolue grâce à l'algorithme solver_hexa.

Tout d'abord, nous avons 2 fonctions principales pour réaliser cette tâche : la fonction *add_number_hexa* qui ajoute l'image d'un nombre dans une case et spécifie sa couleur. Puis, nous avons la fonction *generate_hexa* qui permet de générer une nouvelle image d'une grille résolue, de l'afficher et de la sauvegarder.

Commençons par détailler la fonction suivante : *add_number_hexa*.

void add_number_hexa(guchar* final_img, int number, int x, int y, char mask[ ], int color) ;

Comme vous pouvez le constater, cette fonction a 6 paramètres :
— *final_img* : une variable qui va contenir l'image finale avec les chiffres ajoutés
— *number* : il s'agit du chiffre à ajouter dans l'image finale
— x et y : permettant de savoir sur quelles lignes et colonnes on se trouve
— *mask* : nous permet de savoir si on est déjà passé sur une case
— *color* : nous permet de savoir si on change la couleur ou non du chiffre que l'on ajoute

Au début de notre fonction, on vérifie si le nombre est différent de -1 ou que l'on est déjà passé sur la case en question. Si on n'a pas parcourue la case et que le nombre est différent de -1, on met à jour le masque afin d'éviter de retravailler sur la même case. Ensuite, on calcule des coordonnées permettant de savoir où coller notre image comportant un nombre.

Par la suite, on regarde la couleur : elle sera noir si le paramètre *color* est à 0, sinon, elle sera rouge. Ensuite, on assigne la variable number à une image précise en fonction du chiffre/lettre assigné. Par ailleurs, nous faisons 2 boucles for avec la taille de l'image contenant le nombre (la taille de notre image est de 100 par 100 pixels).

Lors du parcours, si la valeur rencontrée est 255, cela signifie qu'on est sur le fond(partie blanche de l'image) donc on met les 3 canaux à 255 pour représenter la couleur blanche et sinon si elle est inferieur a 255, cela signifie qu'on est sur le chiffre/lettre donc on met la couleur à (*pixelcolor*,0,0)(rouge ou noir). *Pixelcolor* est une variable qui prend la valeur 255 quand le paramètre *color* est 1 ou 0 si le paramètre *color* est à 0.

Détaillons maintenant la fonction suivante : *generate_hexa*

GdkPixbuf* generate_hexa(grid_path) ;

Cette fonction n'a qu'un seul paramètre. En effet, ce paramètre *grid_path* est le chemin d'une grille d'un hexadoku. Grâce à ce chemin, on peut obtenir dans un "char grid[ ][ ]" la matrice correspondante à l'hexadoku avec la fonction *readhexadoku* qui est expliquée dans la partie du solver_hexadoku.

Ensuite, on ajoute tous les nombres de départ avec la fonction *add_number_hexa* en donnant zéro comme paramètre de la couleur pour dessiner en noir. Avant les appels, on créer un masque qui permettra de savoir si cette case a déjà été dessinée.

Puis, on appelle la fonction *solver_hexadoku* (expliqué dans la partie solver aussi). La fonction *solver_hexadoku* renvoie "vrai" si l'hexadoku s'est bien résolue et met à jour le paramètre sudoku donné en le remplissant avec les bons chiffres/lettres, placés aux bons endroits en fonction de la grille de base.

En vérifiant la valeur rendue par *solver_hexadoku*, on peut afficher une image prédéfinie disant que l'hexadoku est insoluble sinon, on rappelle la fonction *add_number_hexa* en donnant 1 comme paramètre pour la couleur signifiant qu'on souhaite mettre la solution de l'hexadoku en rouge, plus précisément, le chiffre/la lettre qui n'était pas présent dans la grille initiale. Enfin, on créer un *GdkPixbuf** à partir de *img_final* qui contient les pixels de l'image finale. On renvoie le *GdkPixbuf** afin de l'afficher sur notre interface graphique et, par la même occasion, se sauvegardera dans les dossiers de l'application.

9					6		F	C		5	D				
	1	D	F			C		8	B		9	5			
C	E			4				1	2	F	0	B		A	9
B					9	7				6		C			1
			B			E	3					D		6	
0	6	9	8	5		D	B	F			3	1	4	E	2
		C	D	8					6	0	4		5		B
			1	9				5					8		7
D				C				9			6	3			
				F			D				E		9	B	
	9	6	4	3		2	7	A						D	0
				B	1	6			4						5
6				7	D	8		E				0		C	
2	F	5				9	C	D				A		1	
A						5		B	C			2		9	E
7	8	E	C	2			0	6			F	4			

Nous arrivons ainsi au résultat suivant :

9	0	2	A	1	6	B	F	C	3	5	D	7	E	8	4
4	1	D	F	A	0	C	E	8	B	7	9	5	3	2	6
C	E	7	6	4	8	3	5	1	2	F	0	B	D	A	9
B	3	8	5	D	9	7	2	4	E	6	A	C	0	F	1
5	4	F	B	0	2	E	3	7	8	9	1	D	C	6	A
0	6	9	8	5	7	D	B	F	A	C	3	1	4	E	2
E	7	C	D	8	A	F	1	2	6	0	4	9	5	3	B
3	2	A	1	9	C	4	6	5	D	E	B	F	8	0	7
D	A	B	7	C	E	0	8	9	5	1	6	3	2	4	F
1	5	0	2	F	4	A	D	3	7	8	E	6	9	B	C
8	9	6	4	3	5	2	7	A	F	B	C	E	1	D	0
F	C	3	E	B	1	6	9	0	4	D	2	8	A	7	5
6	B	4	9	7	D	8	A	E	1	2	5	0	F	C	3
2	F	5	3	E	B	9	C	D	0	4	7	A	6	1	8
A	D	1	0	6	F	5	4	B	C	3	8	2	7	9	E
7	8	E	C	2	3	1	0	6	9	A	F	4	B	5	D

3 Utilitaires mis en place

3.1 Nix-shell

Un des gros avantages de Nix est de permettre de répliquer entièrement un environnement de développement à l'aide d'un programme appelé "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.

3.2 Formatage du code

Le formatage du code est plus que pratique et extrêmement confortable 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 et taille des indentations).

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

3.3 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 macros 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.

3.4 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.

4 Problèmes rencontrés lors du code

4.1 Dataset

La recherche et la sélection d'un ensemble de données pour la reconnaissance optique de caractères (OCR) axée sur des chiffres imprimés peuvent présenter plusieurs défis. Tout d'abord, il peut être difficile de trouver un ensemble de données spécifiquement dédié aux chiffres imprimé, car de nombreuses bases de données OCR incluent une variété de caractères. De plus, la qualité des images et la diversité des polices peuvent considérablement varier, impactant la performance du modèle. Certains ensembles de données peuvent au même titre présenter des problèmes tels que des variations d'échelles, des distorsions ou des bruits, ce qui peut éventuellement la tâche de reconnaissance plus complexe. En outre, la disponibilité d'un ensemble de données étiqueté de manière adéquate, avec des annotations précises pour chaque chiffre, peut être limitée, ce qui complique la formation et l'évaluation du modèle OCR.

4.2 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).

4.3 Usage de fonctions dépréciées

Dans certains morceaux du code, des fonctions dépréciées sont utilisées, faute d'avoir trouvé 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.

4.4 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 également été désactivés. 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.

4.5 Reconstruction de la grille, affichage et sauvegarde

Manipuler une image venant d'un *GdkPixbuf** n'est pas une tâche aisée qui demande une certaine concentration.. En effet, sur GDK, les données d'une image sont un tableau de valeur de taille : largeur*hauteur*(nombre de canaux) et contenant que des valeurs comprises entre 0 et 255.

Par exemple, le premier pixel (situé en haut à gauche) correspond aux 3 ou 4 premières valeurs du tableau : le canal rouge, vert, bleu et si il y a un quatrième canal, alpha qui est la transparence.

Dès lors, nous avons donc été confrontés à de multiples et diverses erreurs dans notre code, donnant une image non conforme à ce qui est attendu. Nous avons ainsi constaté que ces erreurs étaient dues à de mauvaises valeurs et que celles-ci se comptaient par dizaines. Voici quelques images tests obtenues :

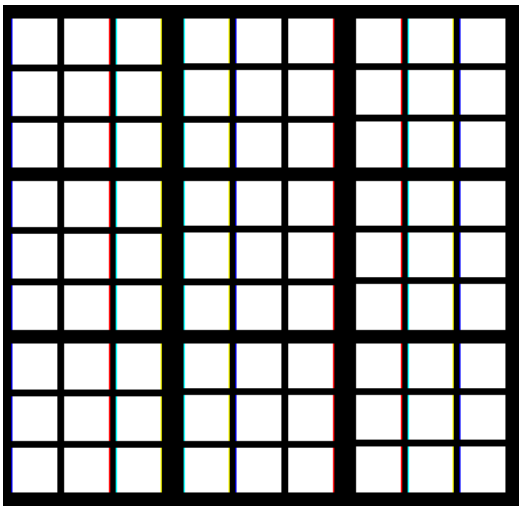


FIGURE 7 – Manque les chiffres

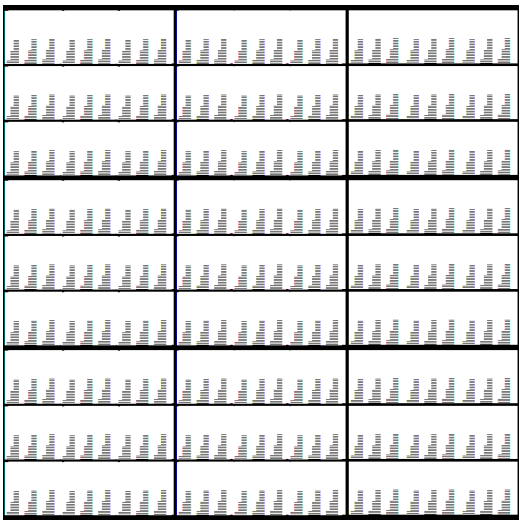


FIGURE 8 – Chiffres mal copiés et manque du quadrillage

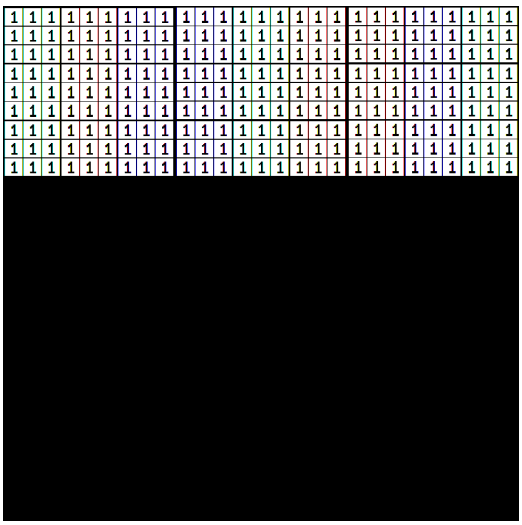


FIGURE 9 – Mauvaise taille de la grille

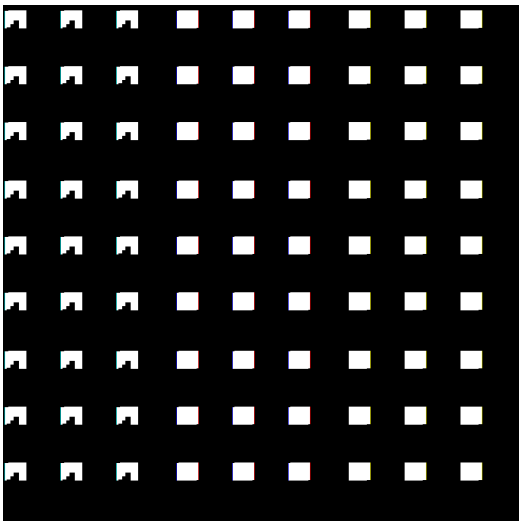


FIGURE 10 – Mauvaise taille des chiffres

5 Techniques d'optimisations utilisées

5.1 Précalculs

Dans la transformé de Hough, un usage important de sinus et cosinus est effectué. Chacune de ces fonctions a un coût en temps de calcul non négligeable, au debut de la transformé, nous commençons donc par réaliser une table des valeurs de sinus et cosinus entre 0 et 2π. Cette technique permet de remplacer ces opérations coûteuses par des opérations simples sur les tableaux, qui sont des accès plus ou moins instantanés. Cette méthode présente en revanche le défaut de nécessiter des allocations de mémoire afin de pouvoir générer cette liste de pixels.Dès lors, nous gagnons en temps de calculs avec comme contrepartie une augmentation de la consommation de mémoire.

5.2 Utilisation des instructions AVX2

La compilation du projet s'effectue en utilisant les instructions AVX2, permettant d'utiliser une seule instruction pour traiter de multiples données. Nous utilisons l'AVX2 et non l'AVX-512 en raison du faible déploiement de ce jeu d'instruction sur les processeurs Intel comme les processeurs AMD.

Les tests effectués avec ce jeu d'instruction présentait des gains de temps considérables sur les opérations nécessitant l'usage de grande quantité de nombres flottants.

6 Améliorations potentielles

6.1 Utilisation des threads

Il est possible de mettre les traitements de l'image dans des threads en utilisant un double buffering sur la partie concernée afin de réduire les temps durant lesquels l'interface est paralysée.

Dans le code actuel, l'interface est bloquée durant une fraction de secondes lors de chaque traitement, ce qui est potentiellement dommage.

6.2 Possibilité de résolution

Il est navrant que l'on se contente de montrer les grilles detectées et finales. Il est confortable de rendre possible la résolution de la grille par l'utilisateur en lui proposant également des fonctionnalités comme par exemple, un chronomètre.

6.3 Possibilité de génération

Actuellement, nous ne pouvons que détecter les grilles puis les afficher et les résoudres automatiquement. Il serait possible de générer une nouvelle grille, de façon procéduralle, permettant ainsi la création d'une quantité importante de nouvelles grilles pour les utilisateurs qui le souhaitent.

7 Conclusion

En conclusion, nous constatons que le groupe fonctionne car nous avons su produire un produit finit simple et fonctionnel, ne demandent que des fonctionnalités supplémentaires. La synergie du groupe aura permis une implication, une bienveillance, une entraide, une rigueur et une détermination à toutes les épreuves rencontrées lors de ce projet. De plus, nous sommes particulièrement fiers de notre production commune car le résultat attendu pour cette ultime soutenance est atteint. La finalité est donc complète avec en prime la réalisation de trois bonus venant s'ajouter et enrichir le projet de base. Certes, nous avons rencontré quelques désagréments concernant la détection de la grille et des cases qui ne fonctionnaient pas sur tous les éléments de l'ensemble des tests, les fuites de mémoires mais aussi sur la reconstruction de la grille. Cependant, nous avons réussi à palier ces failles grâce à un travail acharné ou la non faisabilité n'était pas envisageable afin de vous proposez un projet complet et sérieux qui fut au final amusant à faire. Désormais, c'est avec grand plaisir que nous souhaitons vous dire « défi relevé ». Pour terminer, nous tenions à vous remercier pour les conseils fournis lors de la première soutenance. Merci à vous !