

# Rapport première soutenance de Projet

CHANE-YOCK-NAM David  
FERMELI FURIC Thomas  
HARKOUS Ahmad  
HUANG Félix

Octobre 2021

## OCR Sudoku Solver

# Sommaire

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Image_processing</b>	<b>4</b>
2.1	Displaying the image . . . . .	4
2.1.1	Libraries used : SDL & Pixel_operations . . . . .	4
2.1.2	Display . . . . .	5
2.2	Grayscale & binarisation . . . . .	5
2.3	Deskew manually (Rotation) . . . . .	6
2.4	Quit the interface . . . . .	8
2.5	Missing Parts . . . . .	8
<b>3</b>	<b>Reconnaissance de caractères</b>	<b>10</b>
3.1	le Run Length Smoothing Algorithm . . . . .	10
3.1.1	Principe de l'Algorithme . . . . .	10
3.2	Le XY-cut . . . . .	12
3.2.1	Principe de l'Algorithme . . . . .	12
3.3	Détection avec les composantes connexes . . . . .	12
3.4	Missing part . . . . .	13
<b>4</b>	<b>Character Recognition-Neural network</b>	<b>14</b>
<b>5</b>	<b>Sudoku grid resolution</b>	<b>16</b>
5.1	Backtracking algorithm approach to solve sudoku . . . . .	16
5.2	Display of the result . . . . .	17
<b>6</b>	<b>Conclusion</b>	<b>18</b>
6.1	Objectifs pour la dernière soutenance . . . . .	18

# 1 Introduction

Bienvenue dans ce premier rapport sur notre projet S3 EPITA. Nous sommes ravis de vous expliquer ici l'avancement de notre travail quant à la création de ce résolveur de sudoku.

C'est un projet qui nous tient à coeur pour plusieurs raisons ci-après. Tout d'abord, c'est probablement le seul projet que nous aurons à faire entre amis cette année, puisque nous nous séparons ensuite dans les quatre coins du monde.

C'est donc l'occasion pour nous de travailler et de passer du bon temps ensemble, pour être à la fin fiers de nous et de notre rendu, du moins on l'espère.

C'est aussi l'occasion de faire nos premiers pas dans plusieurs aspects très importants de l'informatique. On va devoir notamment s'intéresser à l'intelligence artificielle, dans laquelle nous nous plongeons réellement pour la première fois et que nous utiliserons sûrement à de nombreuses reprises par la suite.

Nous pensons aussi au traitement d'image qui est une étape indispensable à la vision par ordinateur, discipline qui est désormais omniprésente dans la vie de tous les jours.

Plus largement, c'est la première fois que nous développons un logiciel et c'est avec joie que nous faisons toutes les erreurs qui ne se reproduiront pas lorsque nous aurons plus de responsabilités. C'est enfin l'occasion d'apprendre ou d'approfondir nos connaissances du langage C, qui est probablement le langage le plus célèbre et le plus important en informatique. ouuuuuhyeeaaaah

## 2 Image\_processing

Cette partie a été réalisée par David.

Pour ma part je me suis occupé du chargement de l'image, de la suppression des couleurs et du prétraitement .

Pour commencer je me suis dirigé vers cette partie car elle me paraissait la moins technique, la plus abordable et la plus concrète .

Or il s'avère aussi qu'au fil des recherches et de l'expérience que celle-ci se révèle bien plus complexe qu'il n'y paraît.

Bien que le principe de ma partie soit simple, l'exécution requiert des compétences en mathématiques assez précises. Cela me pousse donc à effectuer de nombreuses recherches, afin de me documenter sur la résolution des problèmes.

Ainsi, pour la partie filtre par exemple, il ne s'agit que d'appliquer les filtres. Mais cette partie reste difficile quand à son application car il faut convertir toute l'image en matrice de données car c'est avec celle-ci que les algorithmes de filtrage s'appliquent.

### 2.1 Displaying the image

#### 2.1.1 Libraries used : SDL & Pixel\_operations

Pour cette première soutenance je me suis occupé de réaliser la partie sur l'affichage d'une image.

Pour cela j'ai utilisé les fonctions incluses dans la librairie Simple DirectMedia Layer 'SDL'.

Parmi celles-ci on peut retrouver les fonction permettant d'initialiser des surfaces, de les dimensionner, et d'interagir sur leurs propriétés.

Pour charger l'image j'ai utilisé une fonction `load_image` qu'on peut retrouver dans le fichier auxiliaire `pixel_operations.c`. C'est un ensemble de fonctions qui sont utiles pour la manipulation à la fois des images, et comme son nom l'indique, des pixels.

### 2.1.2 Display

Donc pour afficher une image à partir d'un fichier je commence par initialiser une interface SDL grâce à la fonction `SDL_init`. Ensuite je peux créer une zone où sera transposée une image grâce à la fonction `SDL_set_vidéo_mode`.

Cette fonction associe plusieurs caractéristiques, comme la taille qu'on souhaite afficher de l'écran, le type de pixel(ici ce sera 32), et enfin le la composante de l'ordinateur qu'on va mobiliser pour afficher et s'occuper de cette tâche.

Ensuite il faut créer un rectangle vide qui va couvrir la fenêtre qu'on vient de créer. Il ne faut pas oublier d'actualiser cette zone pour qu'elle apparaisse sur l'écran. Cela est faisable, au moyen de la fonction `SDL_Flip`.

De plus, je dois m'occuper de l'image que je souhaite afficher. Pour cela je vais recréer une autre surface grâce à la fonction `SDL_surface` qui aura pour contenu donc le premier argument que je vais passer à ma fonction principale (`main`).

## 2.2 Grayscale & binarisation

Pour la suppression des couleurs j'ai intégré directement à la fonction principale l'ensemble de l'instruction qui peut s'apparenter à une fonction à part qu'on pourrait nommer `grayscale`.

Cette fonction est assez simple. Elle prend pour chaque pixel de l'image sa composante en rouge, puis bleue, et enfin verte, les somme et divise par 3. Ainsi on obtient un pixel gris. Ce procédé est itéré pour chaque pixel, et on aboutit à une image toute grise.

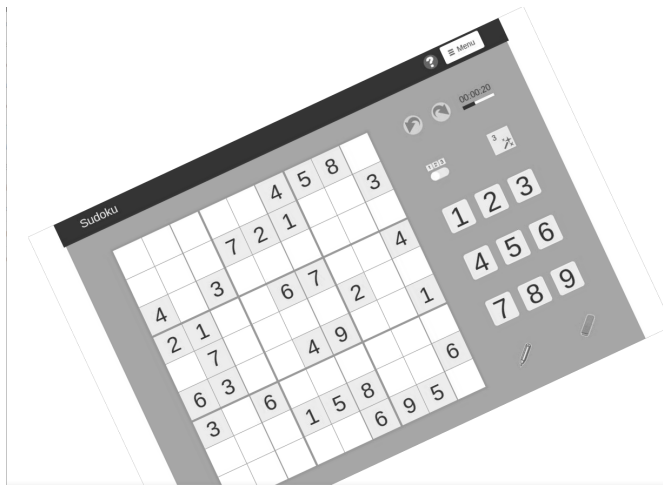
Concernant la binarisation de l'image c'est-à-dire la suppression des couleurs pour l'instant je travaille avec un modèle à seuil statique fixé à 12. C'est donc la valeur médiane entre 0 et 255. On obtient un rendu noir et blanc fonctionnel mais très brut, peu maniable, surtout lorsqu'on tombe sur des exemples/modèles imparfait.

### 2.3 Deskew manually (Rotation)

Enfin concernant la rotation je me suis servi de la fonction 'rotozoomSurface' qui est disponible dans la librairie SDL\_GFX.

Cette fonction prend en paramètre une surface créée par SDL, un double qui indique la valeur de l'angle à tourner, un coefficient d'échelle et un coefficient de lissage de l'image.

La fonction appliquée sur la surface d'origine est ensuite stockée sur une variable surface résultante qui est affichée.



Fonctions Grayscale et Rotation appliquées à une image d'entrée en couleurs.



Maintenant, une autre question se pose : Comment gérer les différentes surfaces ? SDL offre la possibilité de gérer les plans. En ‘blitant’, c’est-à-dire en affichant par-dessus les images précédentes par l’image actuelle, on peut recouvrir l’ancienne surface et afficher à l’utilisateur le rendu souhaité.

## 2.4 Quit the interface

Une boucle pseudo infini a été mise en place pour quitter le programme.

Cette boucle est située en fin de programme. Elle repère l’événement `SDL_Quit`, qui correspond au fait de cliquer sur la croix rouge par un utilisateur.

En testant sur une autre distribution (Archlinux), on a pu constater l’absence de la croix-rouge pour fermer la fenêtre. On a donc du mettre un moyen de secours, à savoir le fait d’appuyer sur la fleche du bas(“DownKey”).

Ainsi, lorsqu’ elle croise les événements de type `SDL_Quit` ou lorsque l’utilisateur appuie sur la fleche du bas, la pseudo-boucle s’arrête, quitte le programme et libère toutes les surfaces mémoire, grâce à `SDL_FreeSurface`.

## 2.5 Missing Parts

Pour la rotation automatique, celle-ci devra détecter les contours de la grille et la retourner pour avec un angle optimal.

Comme mentionné, plus haut, le but est aussi de réaliser une binarisation à seuil adaptatif (méthode d’Otsu/Sauvola).

Enfin il me reste toute la partie filtre à réaliser et c’est cette partie qui la plus technique car elle nécessite de bonnes connaissances en mathématiques et en manipulation des matrices

La plupart des filtres se ressemblent, leur principe est d’appliquer



une formule mathématique, sur chaque pixel et sa zone aux alentours.

De nombreux filtres existent et ils ont chacun leur utilité. Certains sont utiles pour la réduction de bruit, d'autres pour les contrastes etc.

### 3 Reconnaissance de caractères

Cette partie a été réalisée par Félix

Pour ma part, je me suis occupé du reconnaissance de caractères pour le projet.

Quand je l'avais choisit, je n'avais pas eu conscience du important travail que cela demander, puisseque je devais finir ma partie pour la première soutenance. Au final je n'ai pas pu finir cette partie.

Mais je vais toute même essayer d'expliquer ce que je voulais faire, la reconnaissance de caractères se base sur une multitude d'algorithme.

#### 3.1 le Run Length Smoothing Algorithm

La RLSA est utilisé pour la segmentation des blocs de textes et d'image, dans notre cas il va nous servir pour mettre en bloc le sudoku et de l'isolé

##### 3.1.1 Principe de l'Algorithme

Cet algorithme s'applique sur une séquence de nombre binaire dont les pixels blancs sont représentés par les 0 et les pixels noirs par des 1, comme l'agorithme va s'appliquer sur une image binarisé, il y aura donc que des pixels blancs et noirs. L'algorithme va changer une séquence de binaire Y en une séquence Y suivant les règles suivantes.

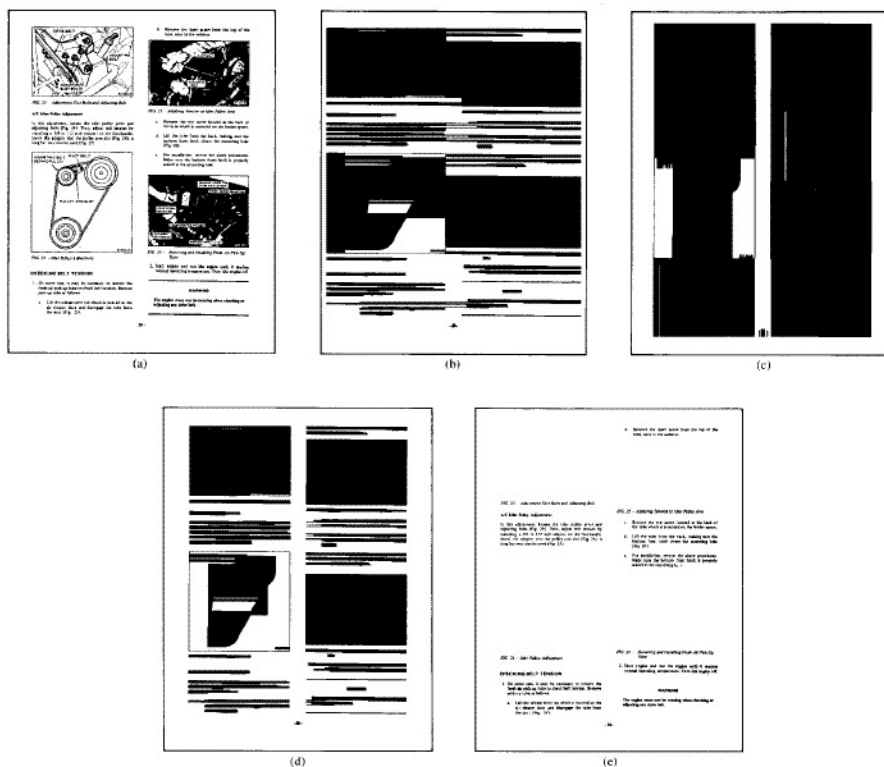
1. Les 0 dans X sont changés en 1 sauf si le nombre de 0 adjacent est inférieur ou égal à une limite prédéfinie C
2. Les 1 dans X resteront inchangés.

Par exemple, avec  $C = 4$ , la séquence X suivant avec la RLSA donnera:

```
x : 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0
y : 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1
```

Lorsqu'il est appliqué à des matrices, le RLSA a pour effet de relier entre elles des zones noires voisines séparées par moins de  $C$  pixels. Avec un choix approprié de  $C$ , les zones liées seront des régions d'un type de données commun, pour ce nombre j'ai directement opté pour un très gros montant pour bien tout mettre en bloc.

Le RLSA est appliqué ligne par ligne ainsi que colonne par colonne à un document, produisant deux bitmaps distincts. Les deux bitmaps sont ensuite combinés dans une opération ET logique.



(a) Document binarisé original. (b) RLSA appliqué verticalement (c) RLSA appliqué horizontalement (d) Résultat final (e) Suppression du bloc

## 3.2 Le XY-cut

le XY-cut est un algorithme de segmentation qui parcourt une image de gauche à droite et de haut en bas afin de renvoyer les différentes zones de texte dans l'ordre de lecture.

### 3.2.1 Principe de l'Algorithme

L'algorithme consiste à parcourir l'image de haut en bas afin de trouver la plus large bande blanche horizontale et de séparer l'image en deux à partir de celle-ci (découpage Y). On stocke ensuite les deux parties découpées et on applique l'algorithme récursivement sur les deux parties, cette fois-ci en parcourant l'image de gauche à droite afin de trouver la bande blanche verticale la plus large et de séparer l'image en deux à partir de celle-ci (découpage X). on réapplique alors le découpage Y sur chaque image alors générée. L'algorithme s'arrête lorsqu'il n'y a plus de bande blanche d'une largeur suffisante.

## 3.3 Détection avec les composantes connexes

La composante connexe en traitement d'image représente une agrégation de pixels connectés sur une image. La technique de marquage de composantes connexes est basée sur une détection des contours des composantes. Suite à cette détection des contours, les pixels d'une composante donnée sont marqués par un même label suivant leur relation de connexité (4 ou 8 connexes) . L'exemple ci-dessous donne un exemple de résultat d'un marquage de composantes en 4-connexité.

Lorsque le système détecte une composante connexe, il renvoie la zone rectangulaire la plus petite qui peut contenir cette composante connexe trouvée

0	0	0	0	0	0	0	0	0
0	1	1	0	0	3	3	3	0
0	0	1	0	0	0	3	3	0
0	1	0	0	0	3	0	3	0
0	0	0	0	0	0	0	0	0
0	0	0	2	2	0	0	0	0
0	0	0	0	2	0	0	0	0
0	0	0	2	0	0	0	0	0
0	0	0	0	2	0	0	0	0

Exemple de marquage de composantes en 8-connexité avec la zone retournée en rouge.

Cette algorithme permet de détecter efficacement le bloc noir créé par la RLSA, si on retourne le plus grand des bloc noir, la grille de sudoku sera retournée et si on applique l'algorithme XY-cut on devrait pouvoir avoir tout les chiffres.

### 3.4 Missing part

Bien que j'ai fait tout les algorithmes présenté ci-dessus, mes parties ne compile pas, il y a beaucoup de problème de mémoire qui sont un enfer à gérer, et mon Makefile me semble très fragile. Plus j'avance dans les algorithmes plus il y a de problème, au final je n'ai pas pu résoudre tout les problèmes présentes, donc Je n'ai pu rendre qu'un produit qui ne compile pas. Mais à coté j'ai aussi mis une version antérieur qui marchait pour ce qu'il fait (Veuillez regarder README) Donc les objectifs pour cette séances n'ont pas pu être atteint pour le moment.

## 4 Character Recognition-Neural network

Pour cette première soutenance, je me suis occupé de créer le réseau de neurones qui reconnaît un chiffre à partir d'une image.

Pour l'instant, l'architecture de mon réseau de neurones est très basique. Il ne possède qu'une couche d'entrée (input layer) et une couche de sortie (output layer). Il ne possède pas de couche cachée.

Pour l'activation des neurones de ma couche de sortie, j'utilise la fonction softmax. Cette fonction renvoie une probabilité, ce qui est parfaitement adapté aux problèmes de classification.

Pour la backpropagation, j'utilise la descente de gradient stochastique.

Pour entraîner et tester mon réseau de neurones, j'ai utilisé le jeu d'images MNIST qui contient 60 000 images d'entraînement (avec les labels associés) et 10 000 images de test (avec les labels associés).

Lors de mon entraînement, je crée des batches de 100 images et répète toutes les opérations pendant 1000 époques. A partir d'une certaine époque, la précision de mon réseau de neurones varie entre 90% et 100%.



Courbes représentant l'entraînement du réseau de neurones

A l'issue de mon test, j'obtiens une précision d'environ 91% sur la reconnaissance des 10 000 images fournies par MNIST.

D'ici la seconde soutenance, je souhaiterais rajouter une couche cachée de neurones (hidden layer) qui me permettrait d'atteindre les 98% de précision. Je n'ai pas encore décidé de la taille de cette couche cachée, mais je sais que j'utiliserai la fonction d'activation ReLU pour activer mes neurones cachés. Cette fonction ReLU a l'avantage d'être linéaire et, associée avec softmax, elle creuse un écart important entre la valeur la plus probable et les autres valeurs.

## 5 Sudoku grid resolution

Cette partie a été réalisée par Ahmad.

Quant à moi je me suis occupé de la résolution de la grille. Ainsi pour cette première soutenance j'ai réussi à créer un programme **solver** qui permet de résoudre une grille de sudoku.

Le main de mon programme prend comme paramètre un fichier sous format **txt** et affiche tout simplement la grille résolue. Le résultat est directement sauvegardé dans un autre fichier sous l'extension **.result**.

### 5.1 Backtracking algorithm approach to solve sudoku

Pour pouvoir résoudre la grille j'ai implémenté un algorithme qui utilise le principe du Backtracking .

Au début j'ai essayé de résoudre le problème avec la méthode naïve de Brute force. Le principe c'est de générer toutes les configurations possibles de nombres de 1 à 9 pour remplir les cases vides de la grille et ensuite essayez chaque configuration une par une jusqu'à ce que la configuration correcte soit trouvée.

Le principe est simple mais pas du tout optimisé, pour cela j'ai opté pour l'autre solution : **backtracking**

Pour pouvoir réduire le temps d'exécution du programme, l'algorithme essaie de construire une solution progressivement en éliminant (Back-track) les possibilités qui ne sont pas vérifiées par des critères prédéfinis et termine en gardant la seule solution au problème. Le backtracking est ainsi comparable à une forme de parcours en profondeur d'un arbre avec des contraintes sur les nœuds.

Pour implementer l'algorithme j'ai codé 2 principales fonctions :  
La première prends en paramètre une valeur compris entre 0 et 9 et une position (row,col) et renvoie si la position est valide ou non , c'est-à-dire si la colonne ou la ligne ou la matrice 3x3 contenant la valeur ne contient pas la même valeur.  
Cette fonction définit ainsi les critères que va utiliser l'algorithme du Backtracking .

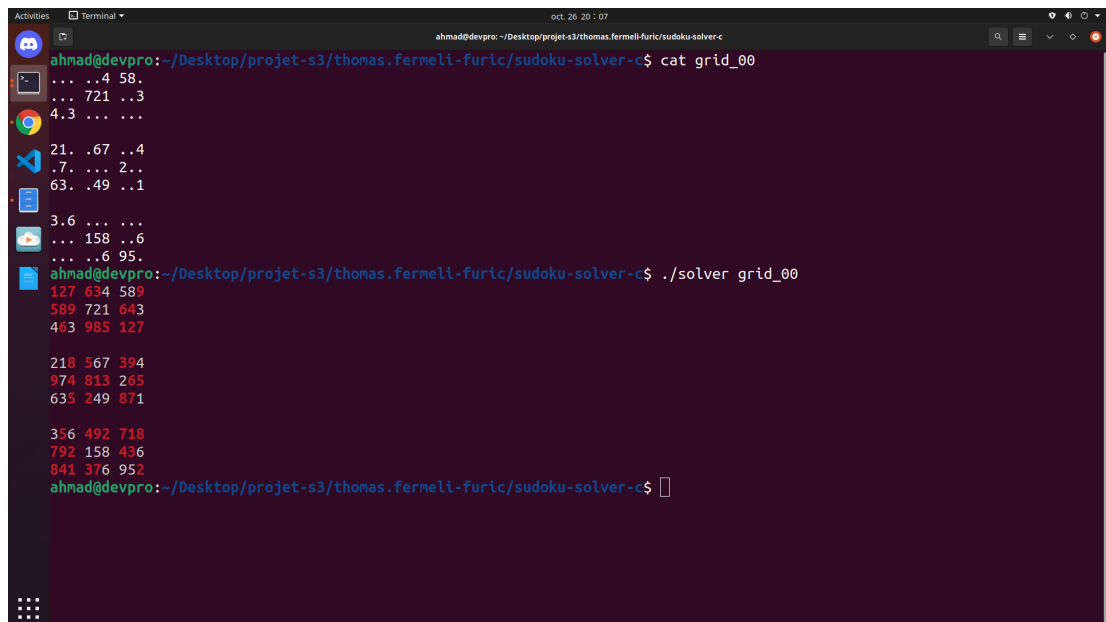


L'autre fonction est une fonction récursive qui déroule l'algorithme du Backtracking . Elle prend comme paramètres 2 matrices carrée (9x9). La première contient la grille à résoudre et l'autre le résultat ou la solution. La vérification des possibilités se fera à l'aide de la première fonction. (Le déroulement de la fonction est expliqué en détail dans le commentaire du code.)

## 5.2 Display of the result

Comme vous pouvez le voir, le programme est uniquement exécutable en ligne de commande.

La grille affichée possède 2 couleurs différents. Les valeurs représentées en rouge sont les valeurs remplis et modifier par le solver.



```

ahmad@devpro: ~/Desktop/projet-s3/thomas.fermeli-furic/sudoku-solver-c
ahmad@devpro:~/Desktop/projet-s3/thomas.fermeli-furic/sudoku-solver-c$ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ..
21. .67 ..4
.7. ... 2..
63. .49 ..1
3.6 ... ..
... 158 ..6
... ..6 95.
ahmad@devpro:~/Desktop/projet-s3/thomas.fermeli-furic/sudoku-solver-c$ ./solver grid_00
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
ahmad@devpro:~/Desktop/projet-s3/thomas.fermeli-furic/sudoku-solver-c$

```

display

## 6 Conclusion

### 6.1 Objectifs pour la dernière soutenance

Pour cette dernière soutenance, nous devons réussir à réaliser un programme unique et complet .

Il nous restera donc à faire :

- Réussir à implémenter les filtres
- Le prétraitement complet et amélioré (rotation et binarisation)
- Complexification du réseau de neurones
- Faire compilé tout l'algorithme détecteur des chiffres
- La reconstruction de la grille
- Affichage de la solution sous forme d'une image
- Une interface graphique permettant d'utiliser tous ces éléments

Pour conclure, réaliser ce projet nous plaît à tous. Hormis le fait que la reconnaissance d'image n'a pas pu être achever à temps, On arrive bien à se représenter le travail qu'il nous est demandé.

Les difficulté rencontrer par Félix sont d'une difficultés majeures mais tout cela sera résolue à coups sur pour la prochaine et dernière soutenance. Mais à part cela nous n'avons pas rencontré d'autre problème majeur, avec l'énorme quantité d'investissement que ce projet requiert, nous faisons de notre mieux pour aller de l'avant et performer le plus possible.

Mention spéciale à git, qui, pour l'instant( on croise les doigts), fonctionne parfaitement...

De plus, nous ressentons une grande gratification.

En effet, le retour de ce qu'on implémente est instantané et concret.

Aussi, c'est l'occasion d'approfondir nos connaissances dans plusieurs domaines spécifiques. Notamment, dans le domaine du traitement d'image, des réseaux de neurones, et bien sûr, des différents algorithmes qu'on peut utiliser afin de résoudre des problèmes simples

et complexes : résoudre un sudoku et reconnaître des formes dans une image.

Tout cela fait que, chacun d'entre nous apprend énormément, aussi d'une manière différente. En effet, la recherche en ligne sur des forums ou des sites de libre publication est très formateur. Ainsi, cela permet de développer par la même occasion nos compétences en autonomie. Maintenant, il nous faut relier tous nos code ensemble, et faire toute la partie de liaison et d'harmonisation !

Ce document arrive à sa fin, nous vous remercions pour toute votre attention.

