



Résolution de Sudoku Android

Mickael CAMPOY – Frédéric CANO

Sommaire

1. Outils utilisé.....	2
2. Répartition des tâches.....	3
3. Déroulement des processus.....	4
4. Détection de la grille.....	5
5. Calibrage de la grille.....	6
6. Reconnaissance de caractère.....	7
7. Résolution de sudoku.....	8
8. Projection du résultat.....	9
9. Auto-évaluation.....	10

Outils utilisé

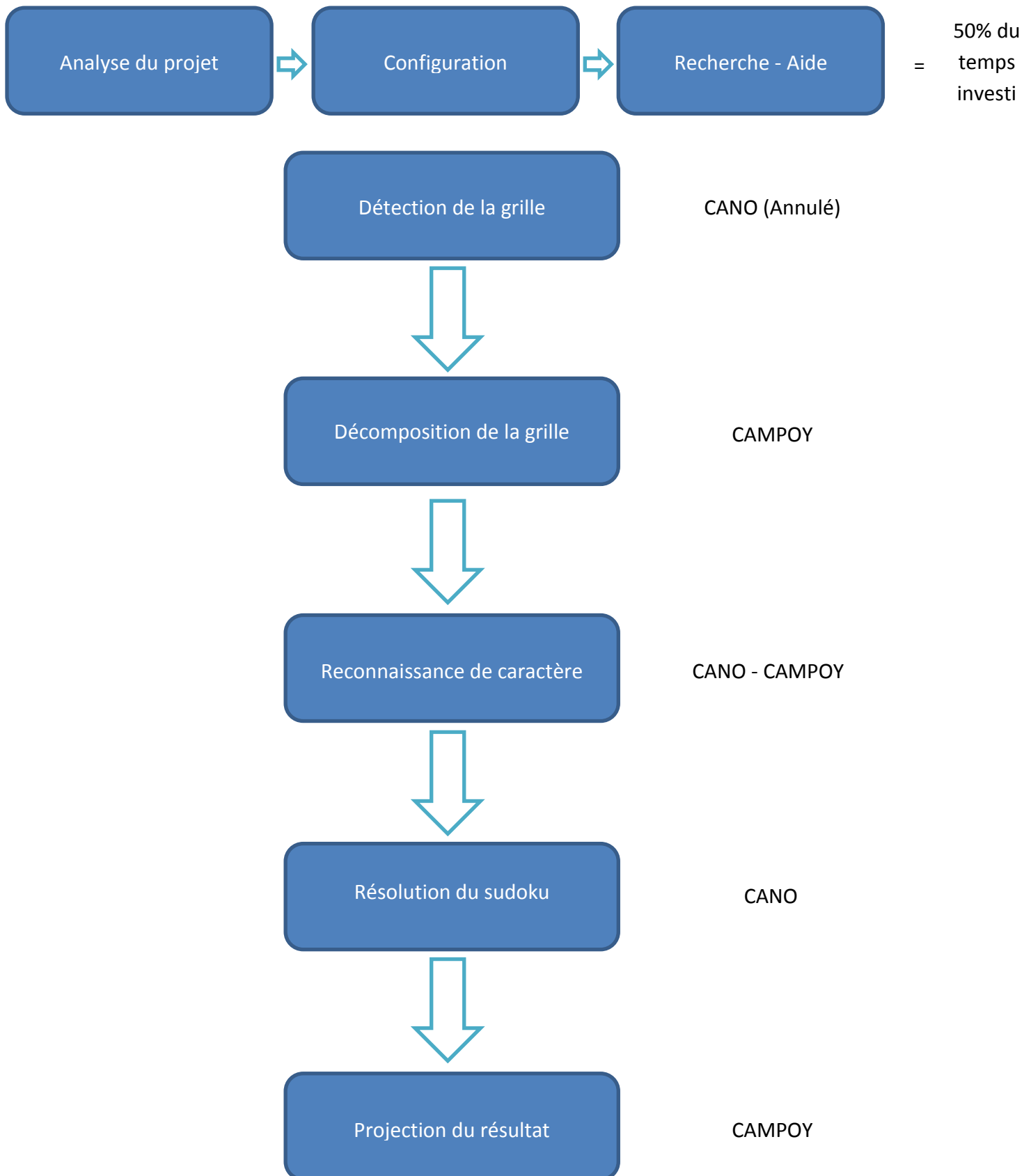
Logiciel – Librairie :

- IDE Éclipse + environnement Android.
- Librairie Open CV
- Librairie Tesseract (OCR)

Source principal :

- Projet Android 2013 « David Defour »
- Open CV – Samples
 - ❖ <http://opencv.org/>
 - ❖ <http://opencvpython.blogspot.in/2012/06/sudoku-solver-part-1.html>
- Tesseract (OCR)
 - ❖ <https://code.google.com/p/tesseract-ocr/>
 - ❖ <https://github.com/GautamGupta/Simple-Android-OCR>

Répartition des tâches



Déroulement des processus

Une fois l'application ouverte, pour lancer le mode sudoku, appuyé sur Option du téléphone puis « Sudoku ». La grille de calibrage s'affiche.

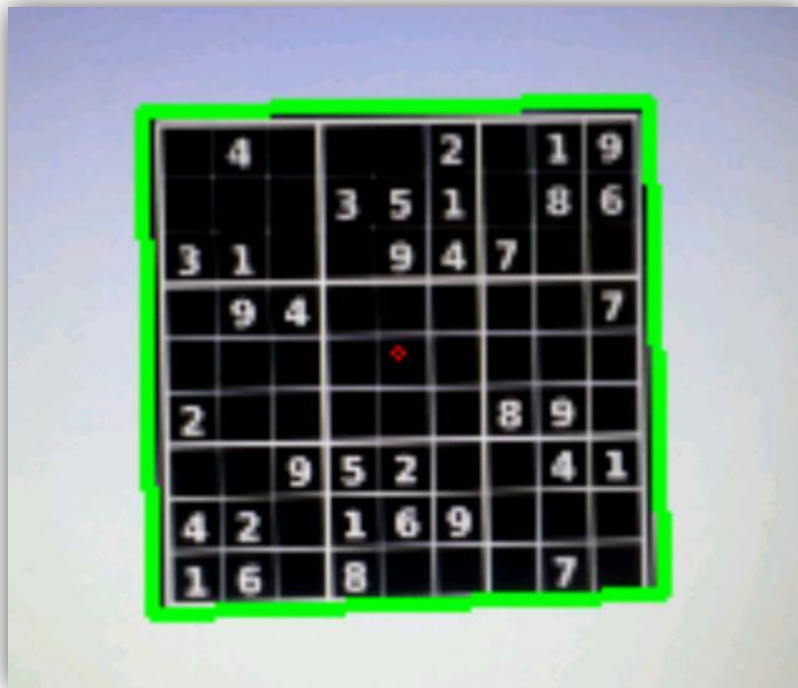
Une fois que le téléphone est bien positionné, appuyer sur l'écran afin de lancer la résolution.

À ce moment la matrice de la grille est décomposé en sous matrice qui correspond à chaque carré de la grille, le traitement de reconnaissance de caractère est exécuté pour chacune des cases. Le résultat retourné est enregistré dans une liste avec ses coordonnées.

Une fois la matrice de la grille remplie, on fait appel à une fonction qui la valide et si elle est correcte on lance la résolution du sudoku.

Quand le traitement est terminé, à partir de la liste ont écrit les chiffres sur l'écran.

Détection de la grille



La détection de la grille se fait à partir d'un script compilé en C++ accompagné de sa librairie (so).

Après détection de la grille je n'ai pas réussi à manipuler la matrice image afin de la transformer en grille parfaitement carré ou même ajuster le contraste.

J'ai tenté de lancer une reconnaissance de caractère sur la matrice mais le programme planté sans explication.

Par la suite, j'ai effectué plusieurs recherche, afin d'essayer une nouvelle approche mais je n'ai pas trouvé d'aide ou exemple pour détecter un carré à l'aide d'Open CV avec du Java ou Android.

Ce projet est présent sur le Git Hub :

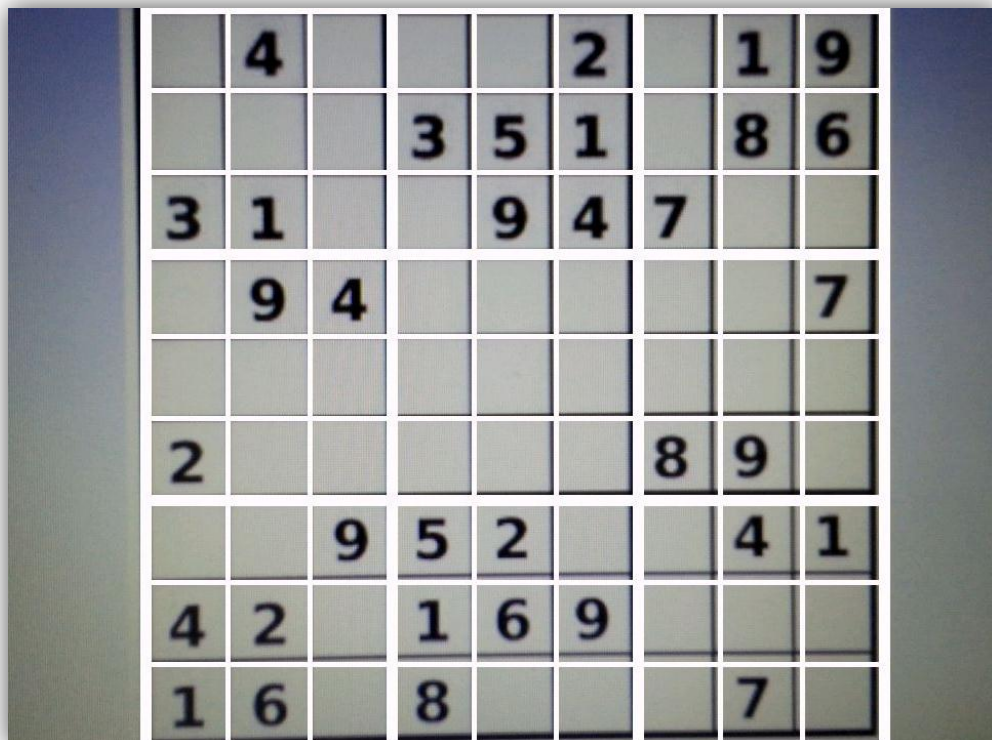
https://github.com/PancakeInvaders/ANDROID_UPVD_L3/tree/master/DetectionGrille

Le fichier C++ se trouve dans le dossier « jni » et sa librairie dans le dossier « libs/armeabi-v7a »

Calibrage de la grille

Notre application est structurée autour de la vidéo fournie par l'appareil photo. Suite à notre échec à détecter la grille directement de la photo, nous avons décidé de dessiner la grille de sudoku sur l'image, et de laisser l'utilisateur viser.

C'est la méthode « onCameraFrame » qui s'occupe de traiter chaque image envoyée par l'appareil photo. C'est donc dans cette fonction que j'ai développé, le dessin de la grille. Pour l'affichage de cette grille, j'ai utilisé les fonctions de dessin de la librairie open CV comme Core.line (), que j'ai paramétré en fonction de la largeur et de la longueur de l'écran.



Reconnaissance de caractère

<https://github.com/rmtheis/tess-two/>

Cette librairie une fois téléchargé nécessite d'être compilé à l'aide de la commande « ndk-build ». Ensuite il suffit de la référencer au projet.

La reconnaissance des caractères consiste en une fonction qui prend en paramètres une image, et renvoie une chaine de caractères représentant soit le chiffre lu, soit un message d'erreur, on passe en paramètre un objet de type Mat, classe de la librairie Open CV représentant une image. Un petit désagrément est que la méthode de lecture de Tesseract prend un paramètre un objet Bitmap, classe de la librairie de base Android représentant une image. Pour convertir notre objet Mat en Bitmap, nous devons créer un fichier temporaire dans lequel nous écrivons notre image Mat, puis lire ce fichier temporaire en tant que Bitmap.

Nous devons créer et configurer un objet TessBaseApi, qui nous servira à utiliser la méthode getUTF8Text sur notre objet Bitmap. Si cela ne nous renvoie pas un nombre adéquat, nous renvoyons la chaine «échec lecture», ce qui sera interprété comme une absence de caractère.

Le problème que nous avons rencontré est que, lorsque la case est vide, getUTF8Text nous renvoie quand même un chiffre, et lorsque la case est remplie, la fonction ne renvoie pas forcément le bon chiffre. De ce fait, la grille à résoudre n'est la plus part du temps pas valide.

```
File file = new File(pathtess, filename);
//
filename = file.toString();
Highgui.imwrite(filename, img);

// Crée un objet BitmapFactory
BitmapFactory.Options options = new BitmapFactory.Options();
options.inSampleSize = 4;
// Decode le fichier en bitmap
Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath(), options);
// Copie le bitmap avec manipulation
bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);

// Si pathtess n'existe pas
if(pathtess.exists() == false){
    // Crée le dossier
    boolean result = pathtess.mkdir();
    // Si le dossier est créé
    if(result == true){
        // Message
        System.out.println("dir created");
    }
}

/*
 * Reconnaissance de caractère
 * Utilisation de Tesseract
 */
// Crée un objet de type TessBaseAPI et l'initialise
TessBaseAPI baseApi = new TessBaseAPI();
baseApi.setDebug(true);
baseApi.init(path.getAbsolutePath(), lang);
baseApi.setImage(bitmap);
baseApi.setVariable("tessedit_char_whitelist", "123456789"); // Définie le type de variable (caractère autorisé)

// Recupere dans une chaine le caractère retourné
String recognizedText = baseApi.getUTF8Text();
```


Résolution de sudoku

Solver
+ int grille[][]
+ Solver(int grille[][])
+ affiche() : void
+ absentSurLigne(int k, int i) : bool
+ absentSurColonne (int k, int j) : bool
+ absentSurBloc (int k, int i, int j) : bool
+ solve (int position) : bool
+ isValid(int[][] board) : bool

La technique de résolution se base sur la récursivité.

Je me suis basé sur un algorithme développé en Python. Que j'ai retranscrit en Java avec quelques retouches personnelles.

Après avoir initialisée la grille dans le constructeur surchargé de la classe, on appelle la fonction solve en passant en paramètre la position de départ (0).

```
/**
 * Fonction qui résout une grille de sudoku, de manière récursive
 * Utilisation du backtracking
 * @param int : Position
 * @return boolean
 */
public boolean solve (int position) {
    // Si on a fini de parcourir la grille, retourne vrai
    if (position == this.grille.length*this.grille.length) return true;

    // Recupere les coordonnées de la case i et j
    int i = position/this.grille.length;
    int j = position%this.grille.length;

    // Si la case n'est pas vide, on passe à la suivante
    if (this.grille[i][j] != 0) return solve(position+1);

    /*
     * BACKTRACKING
     */
    // Enumeration des valeurs possible
    for (int k=1; k <= this.grille.length; k++) {
        // Si la valeur est absente (autorisée)
        if (absentSurLigne(k,i) && absentSurColonne(k,j) && absentSurBloc(k,i,j)) {
            // Enregistrement de k dans la grille
            this.grille[i][j] = k;

            // Appel récursif de la fonction, afin de vérifier si par la suite ce choix est correct.
            if (solve(position+1)) return true;
        }
    }

    // Tous les chiffres ont été testés, aucun n'est bon, on réinitialise la case
    this.grille[i][j] = 0;
    return false;
}
```

Projection du résultat

Une fois la grille de sudoku résolu, je parcours la matrice et pour chaque case, je calcule les coordonnées. A chaque fois, j'ajoute les coordonnées et le chiffre dans une liste.

Pour l'affichage du résultat sur le téléphone, avec la librairie d'Open CV, j'utilise en particulier la méthode « putText » de « Core ».

Dans la méthode « onCameraFrame », je parcours la liste des chiffres à écrire, calcule les coordonnées en fonction de la taille de l'écran, et les inscris à l'endroit voulu.

```
// MODE GRID
case ImageManipulationsActivity.VIEW_MODE_GRID:
    // Autorise l'execution du thread
    threadStopRequest = false;

    // Fonction qui dessine la grille du sudoku
    drawGridVoid();

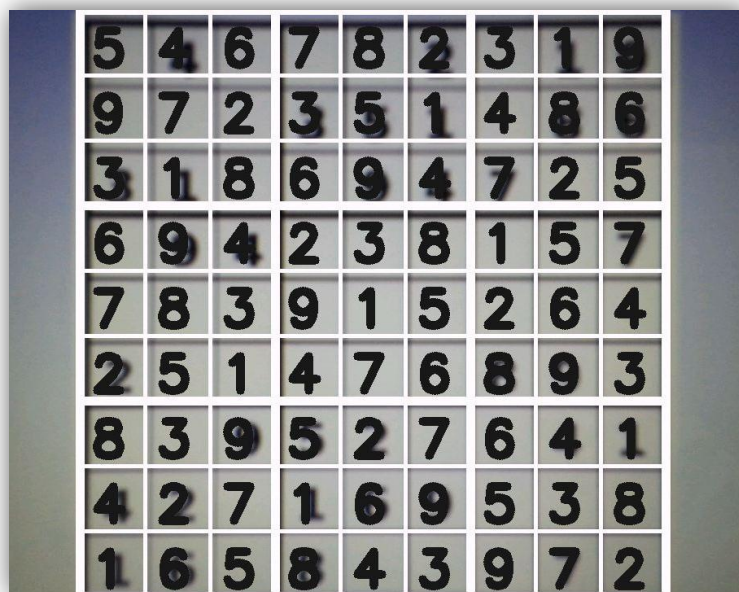
    // On dessine les chiffres de la grille résolue
    // Initialise les points x et y
    double x;
    double y;

    for(PointValue pv : listADessiner){
        // Recupere les coordonnées pour ecrire
        x = pv.getPoint().x;
        y = pv.getPoint().y;

        // Ecrit le chiffre sur la grille
        Core.putText(rgba, "" + pv.getValue(), new Point (x + (sc2/2) - 5, y + sizeCell - (sc2/2) + 7), Core.FONT_HERSHEY_SIMPLEX, 2, new
    }

    // Si l'image doit etre calculée alors
    if(computingImage == true){
        // Affiche la chaine d'attente
        String s = "Calcul de l'image... Cela peut prendre plusieurs minutes.";
        Core.putText(rgba, s, new Point (w/2 - (8.2)*s.length(), h - 50), Core.FONT_HERSHEY_SIMPLEX, 1, new Scalar(255,117,5), 3);
    }

    break;
```



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

Auto-  valuation

Pour l'auto-  valuation nous sommes partie sur ces diff  rents crit  res :

- *Difficult   rencontr  e*
- *Temps pass   sur le projet*
- *Qualit   et clart   du code*
- *Satisfaction personnelle*
- *R  sultat*

<i>Fonctionnalit��s</i>	<i>Notes /20</i>
<i>D��tection de la grille</i>	<i>5</i>
<i>Calibrage de la grille</i>	<i>17</i>
<i>Reconnaissance de caract��re</i>	<i>13</i>
<i>R��solution du sudoku</i>	<i>18</i>
<i>Projection du r��sultat</i>	<i>17</i>
<i>TOTAL</i>	<i>14</i>