OCR

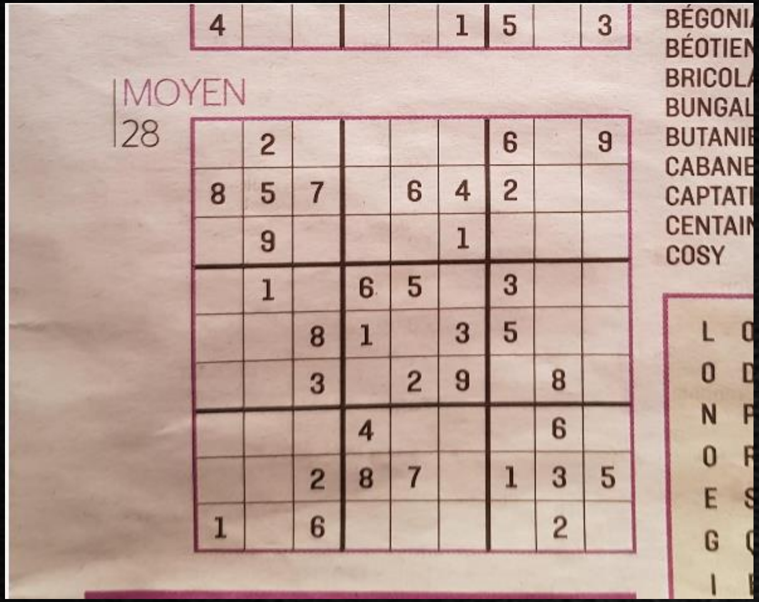
Lucas Duport *Mattéo Baussart* *Matthieu Correia* *Julie Blassiau*

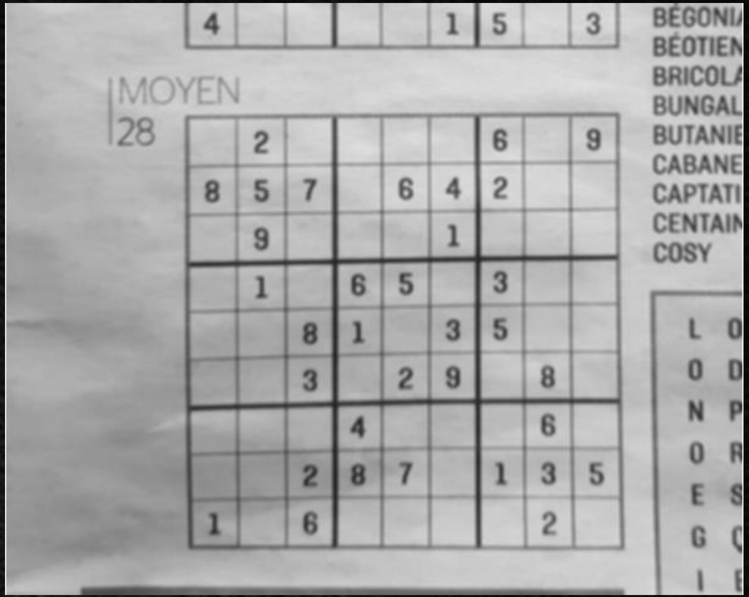
Sommaire

- Présentation du Projet
  - Prétraitement de l’image
  - Traitement de l’image
  - Détection des chiffres
  - Résolution du Sudoku
  - Interface
- Démonstration
- Explication d’extraits de code
  - Construction de la grille
  - Solver
  - Activation des couches
  - Redimensionnement manuel

Prétraitement de l’image

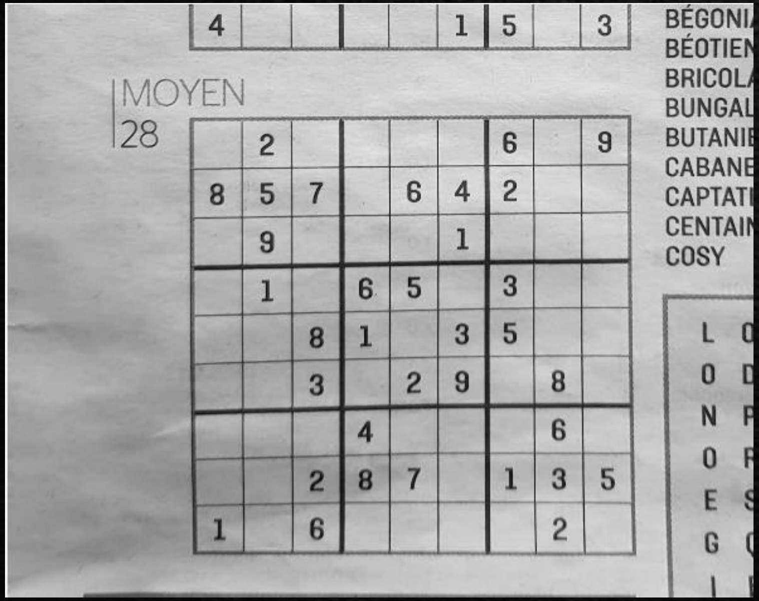
Originale

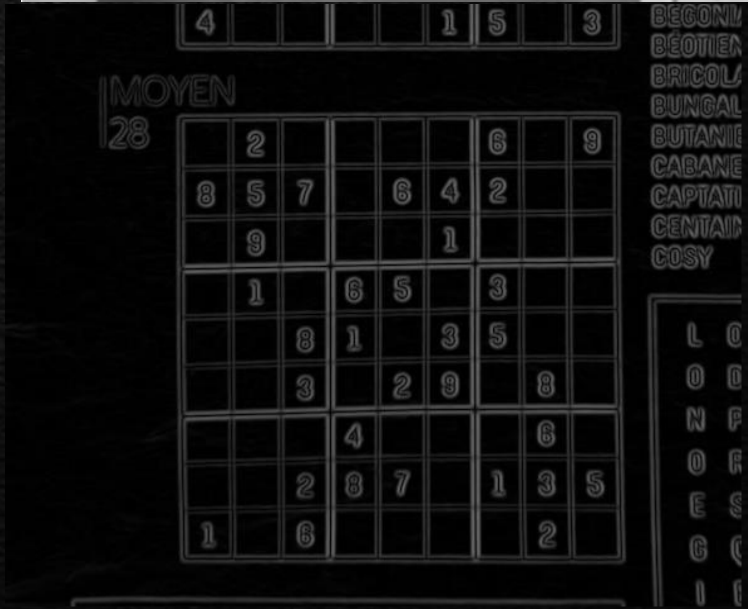




Filtre Gaussien

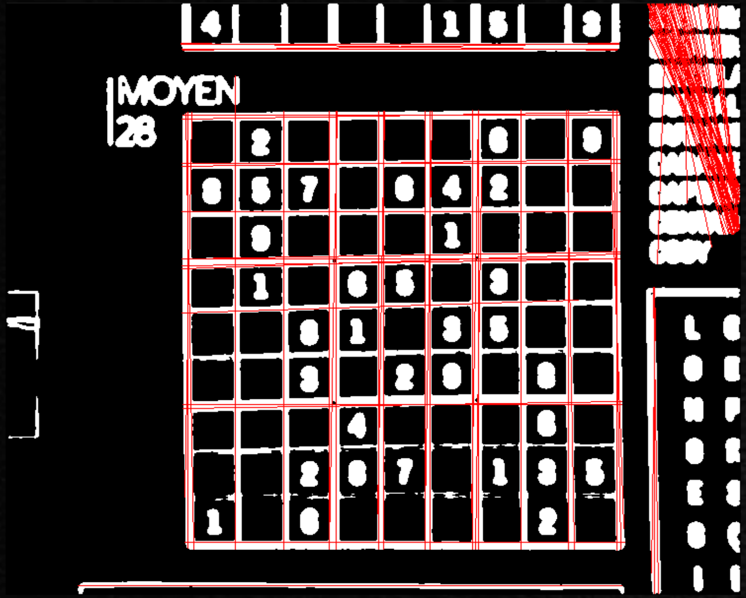
En Gris

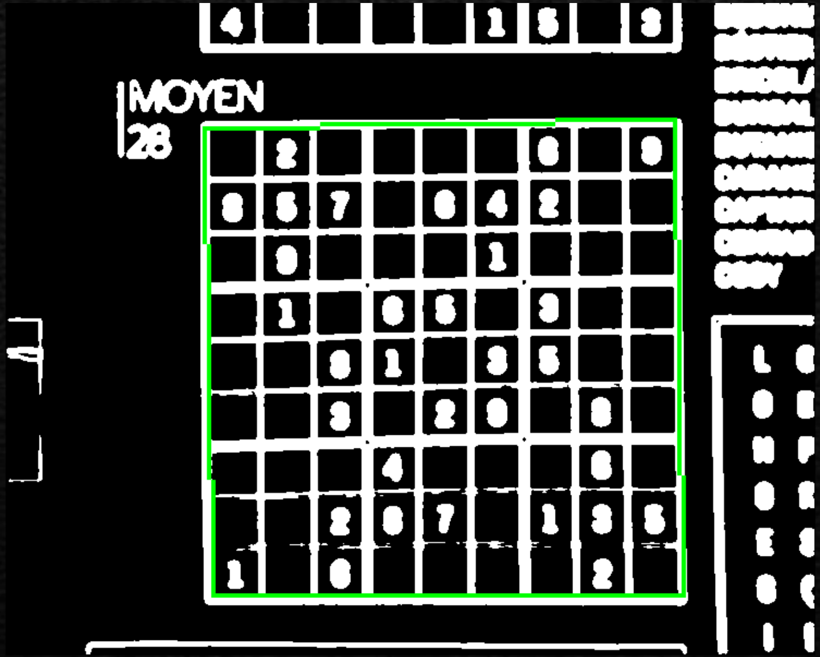


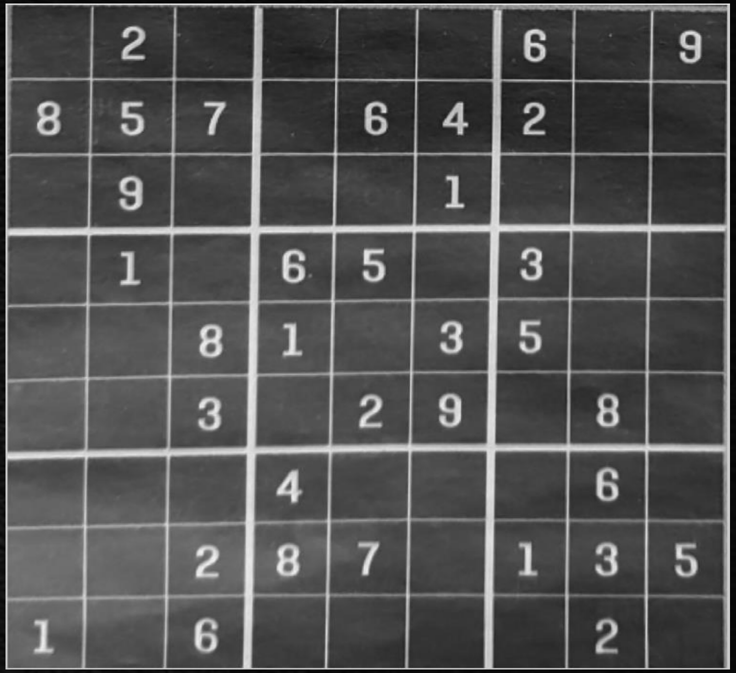


Filtre Sobel

Traitement de l’image

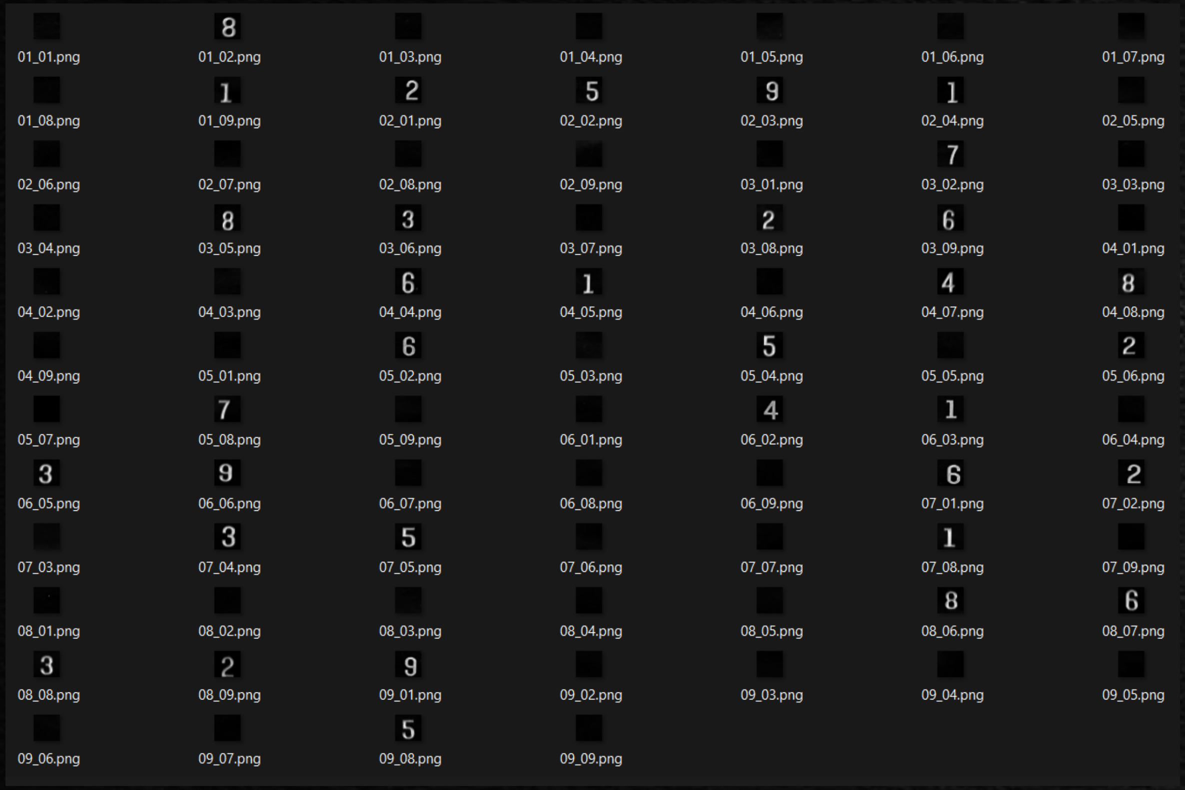




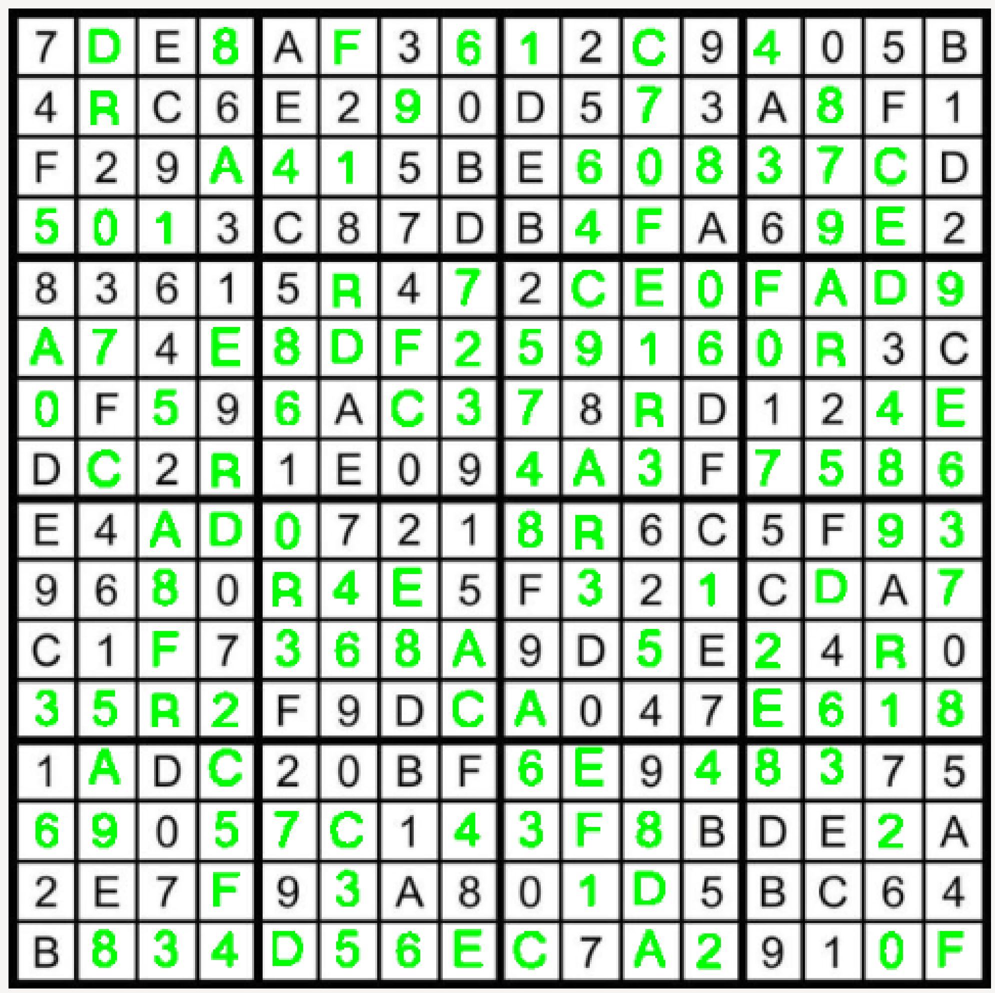


	2					6		9
8	5	7		6	4	2		
	9				1			
	1		6	5		3		
		8	1		3	5		
		3		2	9		8	
			4				6	
		2	8	7		1	3	5
1		6					2	

Traitement de l'image

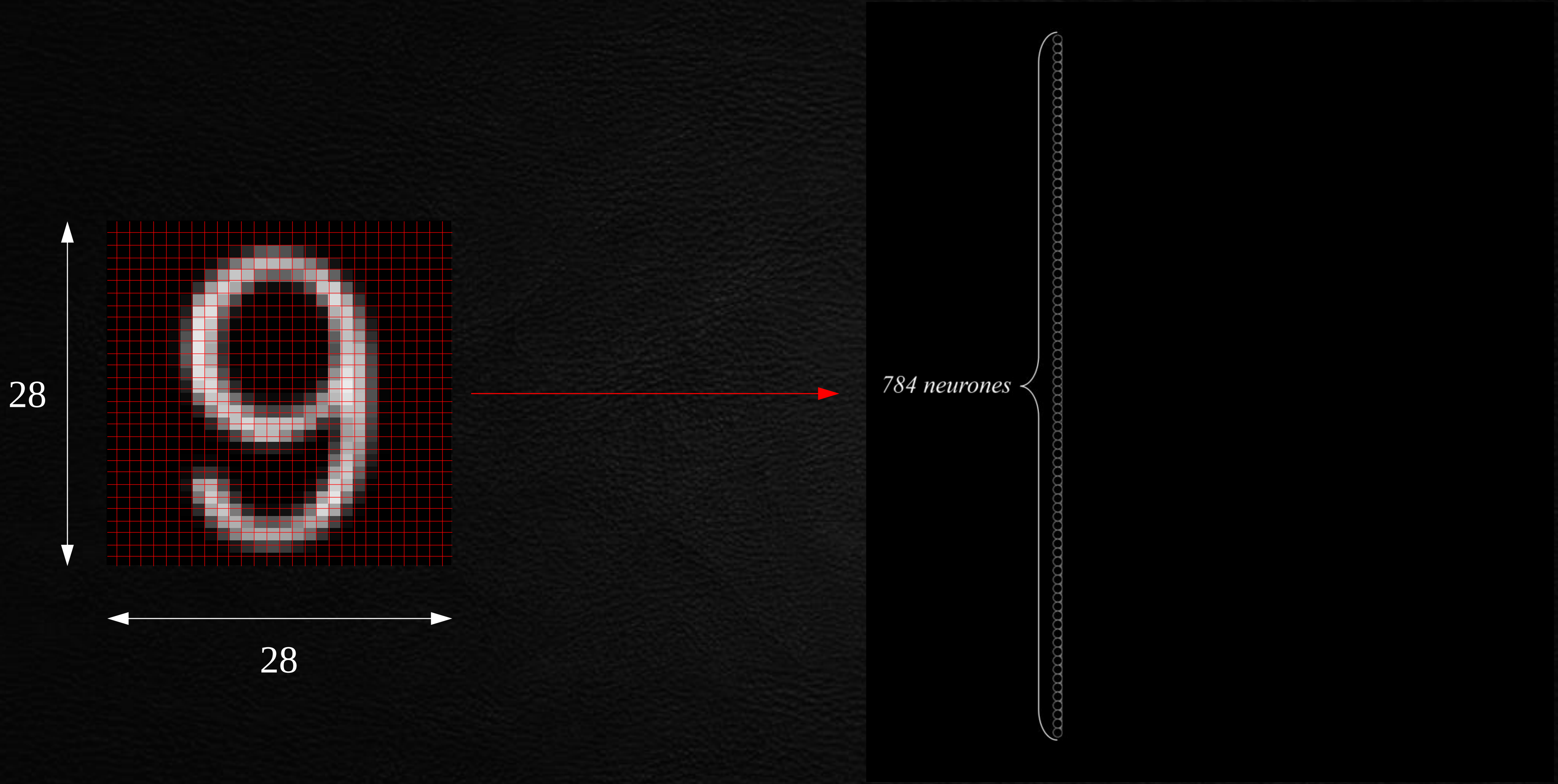


01_01.png	8 01_02.png	01_03.png	01_04.png	01_05.png	01_06.png	01_07.png
01_08.png	1 01_09.png	2 02_01.png	5 02_02.png	9 02_03.png	1 02_04.png	02_05.png
02_06.png	02_07.png	02_08.png	02_09.png	03_01.png	7 03_02.png	03_03.png
03_04.png	8 03_05.png	3 03_06.png	03_07.png	2 03_08.png	6 03_09.png	04_01.png
04_02.png	04_03.png	6 04_04.png	1 04_05.png	04_06.png	4 04_07.png	8 04_08.png
04_09.png	05_01.png	6 05_02.png	05_03.png	5 05_04.png	05_05.png	2 05_06.png
05_07.png	7 05_08.png	05_09.png	06_01.png	4 06_02.png	1 06_03.png	06_04.png
3 06_05.png	9 06_06.png	06_07.png	06_08.png	06_09.png	6 07_01.png	2 07_02.png
07_03.png	3 07_04.png	5 07_05.png	07_06.png	07_07.png	1 07_08.png	07_09.png
08_01.png	08_02.png	08_03.png	08_04.png	08_05.png	8 08_06.png	6 08_07.png
3 08_08.png	2 08_09.png	9 09_01.png	09_02.png	09_03.png	09_04.png	09_05.png
09_06.png	09_07.png	5 09_08.png	09_09.png			



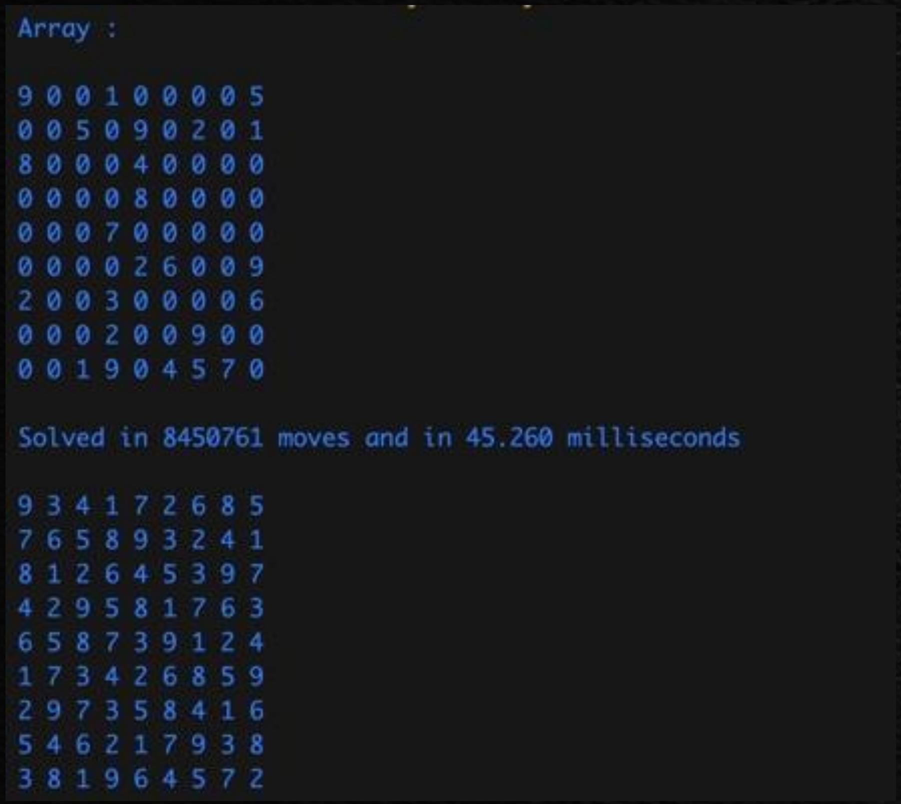
7	D	E	8	A	F	3	6	1	2	C	9	4	0	5	B
4	R	C	6	E	2	9	0	D	5	7	3	A	8	F	1
F	2	9	A	4	1	5	B	E	6	0	8	3	7	C	D
5	0	1	3	C	8	7	D	B	4	F	A	6	9	E	2
8	3	6	1	5	R	4	7	2	C	E	0	F	A	D	9
A	7	4	E	8	D	F	2	5	9	1	6	0	R	3	C
0	F	5	9	6	A	C	3	7	8	R	D	1	2	4	E
D	C	2	R	1	E	0	9	4	A	3	F	7	5	8	6
E	4	A	D	0	7	2	1	8	R	6	C	5	F	9	3
9	6	8	0	R	4	E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

Détection des chiffres

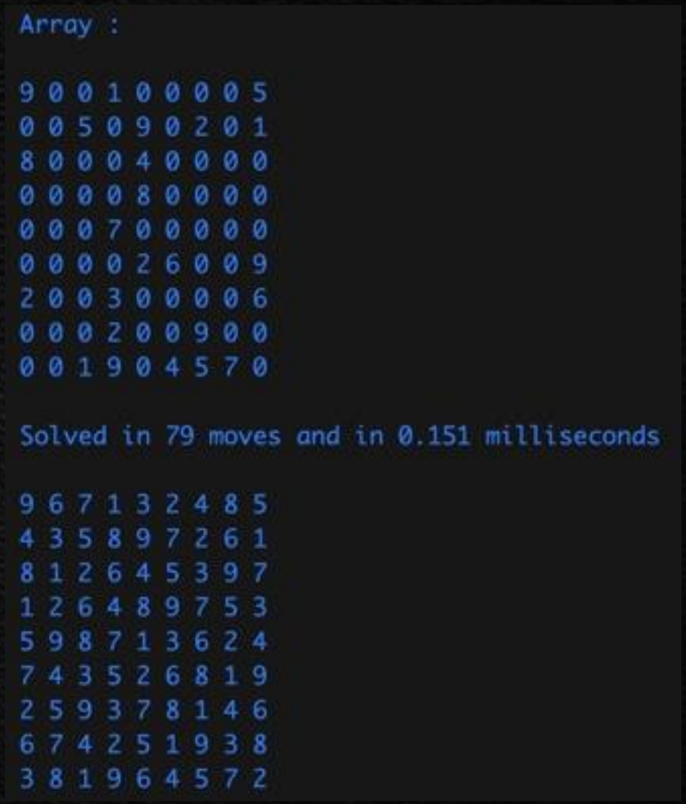


0 : 0.000000
1 : 0.000000
2 : 0.000000
3 : 0.000000
4 : 0.000000
5 : 0.000000
6 : 0.000000
7 : 0.000000
8 : 0.000006
9 : 0.999994

Résolution du Sudoku



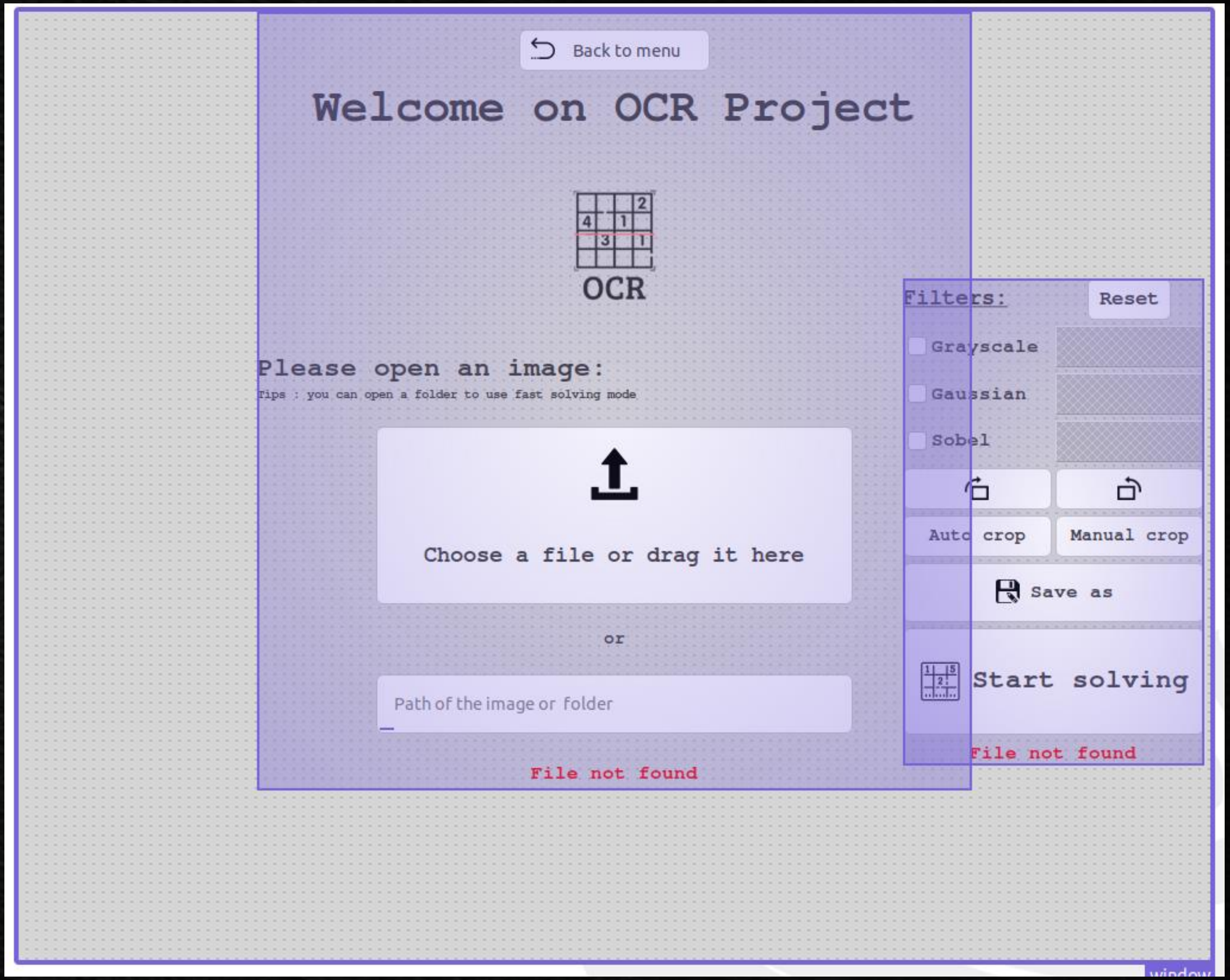
```
Array :

9 0 0 1 0 0 0 0 5
0 0 5 0 9 0 2 0 1
8 0 0 0 4 0 0 0 0
0 0 0 0 8 0 0 0 0
0 0 0 7 0 0 0 0 0
0 0 0 0 2 6 0 0 9
2 0 0 3 0 0 0 0 6
0 0 0 2 0 0 9 0 0
0 0 1 9 0 4 5 7 0

Solved in 8450761 moves and in 45.260 milliseconds

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



```
Array :

9 0 0 1 0 0 0 0 5
0 0 5 0 9 0 2 0 1
8 0 0 0 4 0 0 0 0
0 0 0 0 8 0 0 0 0
0 0 0 7 0 0 0 0 0
0 0 0 0 2 6 0 0 9
2 0 0 3 0 0 0 0 6
0 0 0 2 0 0 9 0 0
0 0 1 9 0 4 5 7 0

Solved in 79 moves and in 0.151 milliseconds

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

Interface

- Un objectif clair et facile à comprendre
- Un design intuitif
- Des éléments visuels
- Des feedbacks et messages d'erreur utiles.



Welcome on OCR Project

Please open an image:

Tips : you can open a folder to use fast solving mode

Choose a file or drag it here

or

Path of the image or folder

File not found

Filters:

□ Grayscale

□ Gaussian

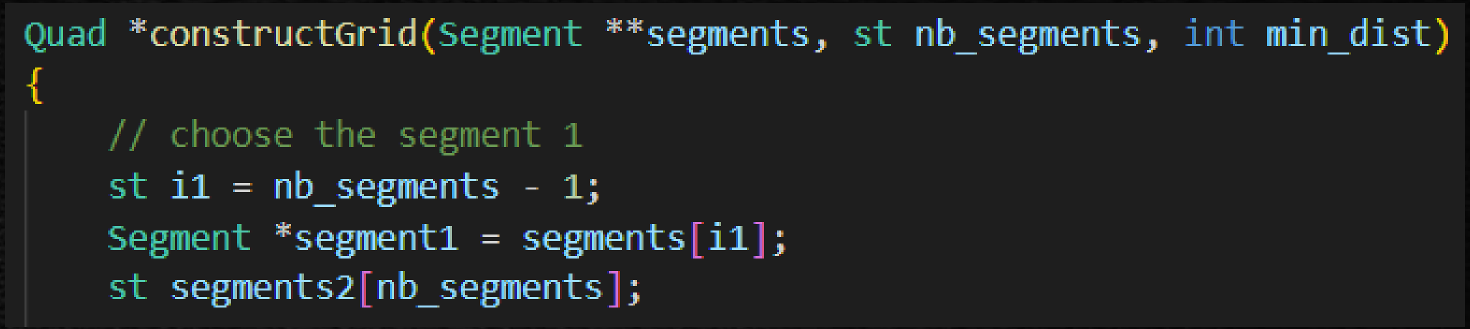
□ Sobel

File not found

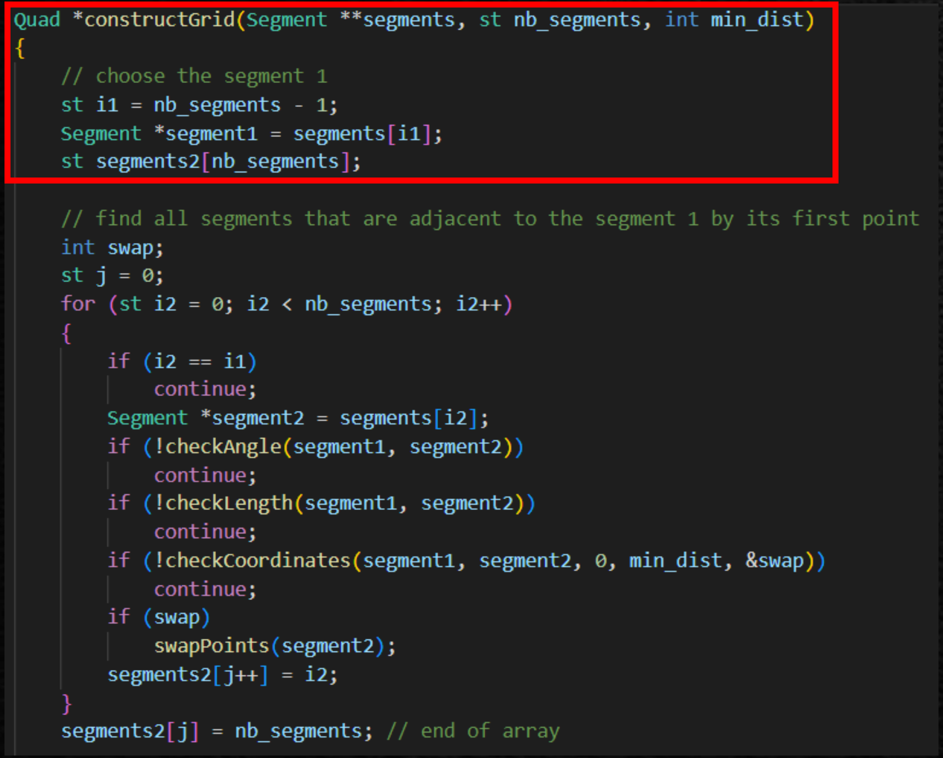
Extraits de Code :

- Construction de la grille
- Solver
- Activation des couches
- Interface utilisateur

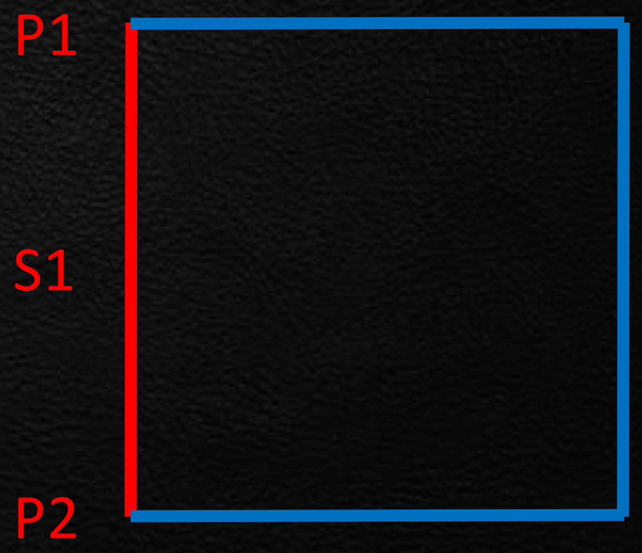
Construction de la grille



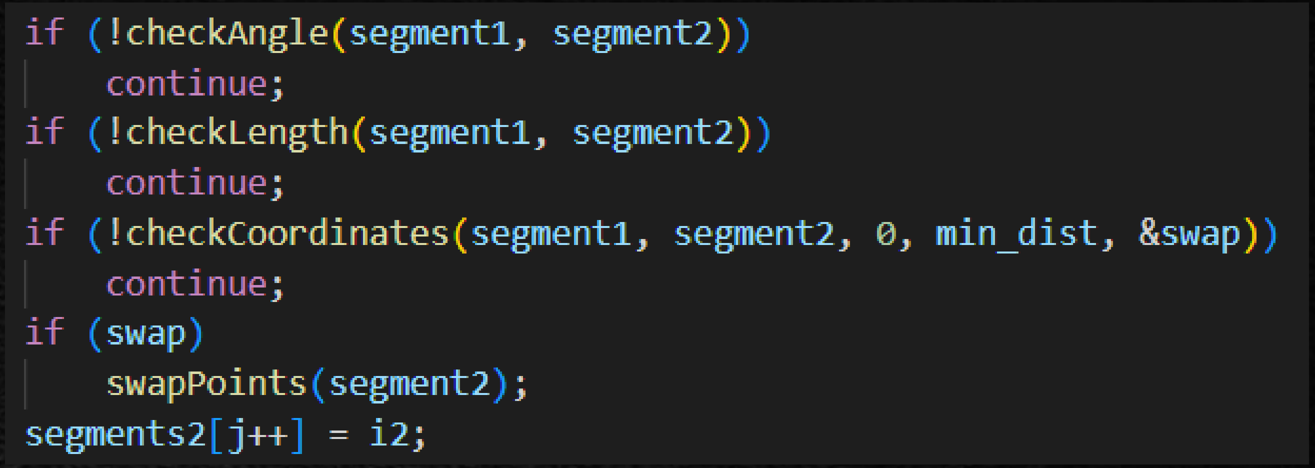
```
Quad *constructGrid(Segment **segments, st nb_segments, int min_dist)
{
    // choose the segment 1
    st i1 = nb_segments - 1;
    Segment *segment1 = segments[i1];
    st segments2[nb_segments];
```



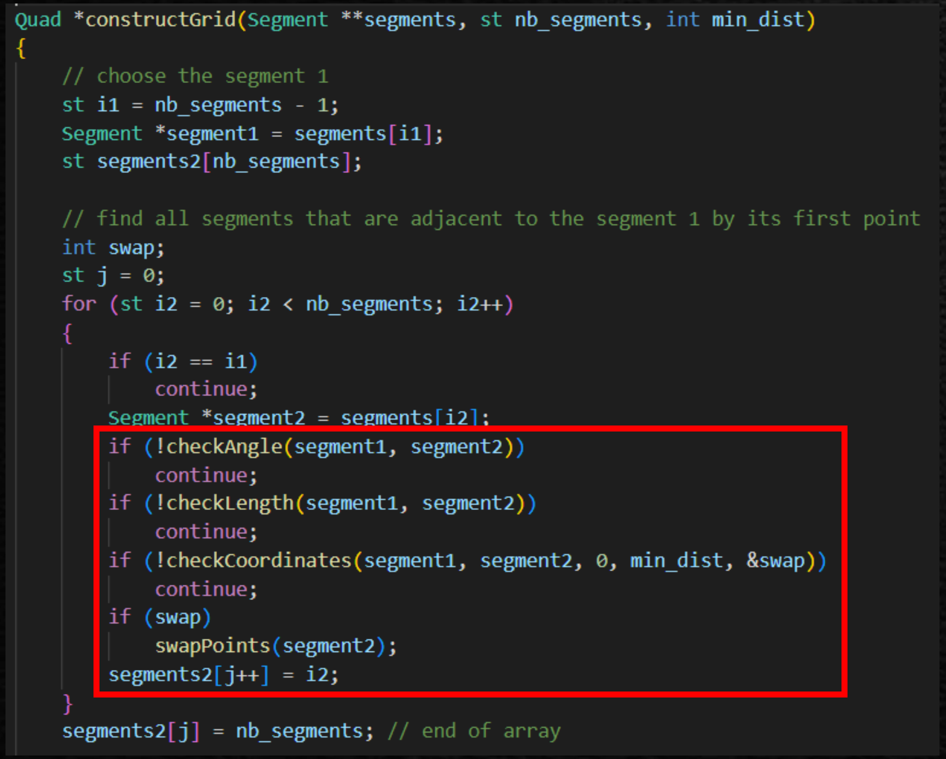
```
Quad *constructGrid(Segment **segments, st nb_segments, int min_dist)
{
    // choose the segment 1
    st i1 = nb_segments - 1;
    Segment *segment1 = segments[i1];
    st segments2[nb_segments];

    // find all segments that are adjacent to the segment 1 by its first point
    int swap;
    st j = 0;
    for (st i2 = 0; i2 < nb_segments; i2++)
    {
        if (i2 == i1)
            continue;
        Segment *segment2 = segments[i2];
        if (!checkAngle(segment1, segment2))
            continue;
        if (!checkLength(segment1, segment2))
            continue;
        if (!checkCoordinates(segment1, segment2, 0, min_dist, &swap))
            continue;
        if (swap)
            swapPoints(segment2);
        segments2[j++] = i2;
    }
    segments2[j] = nb_segments; // end of array
```



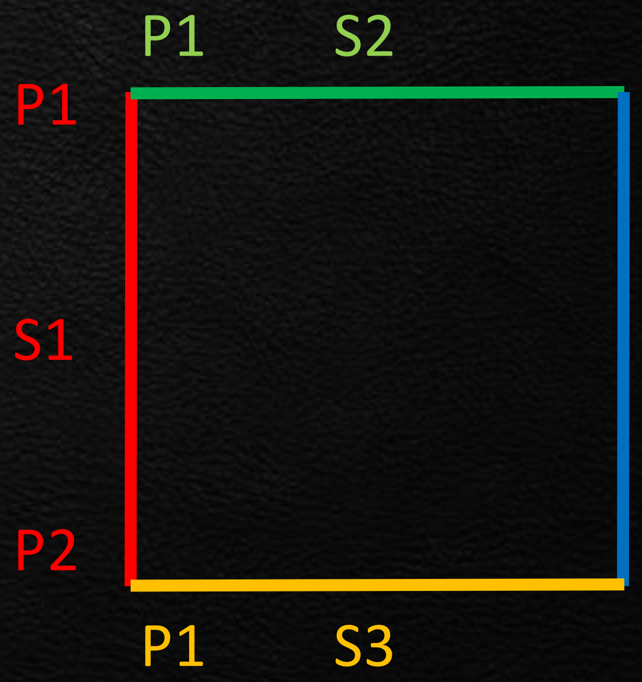
Construction de la grille



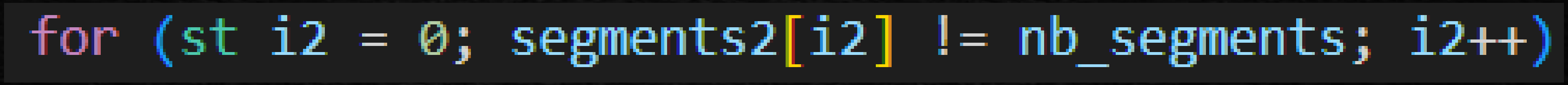
```
if (!checkAngle(segment1, segment2))
    continue;
if (!checkLength(segment1, segment2))
    continue;
if (!checkCoordinates(segment1, segment2, 0, min_dist, &swap))
    continue;
if (swap)
    swapPoints(segment2);
segments2[j++] = i2;
```



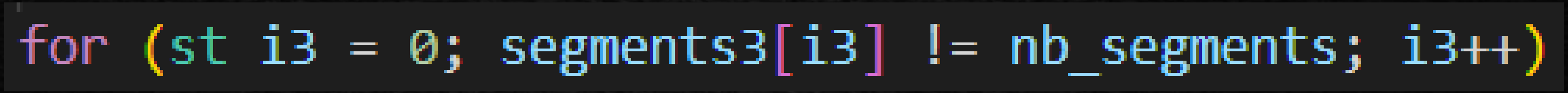
```
Quad *constructGrid(Segment **segments, st nb_segments, int min_dist)
{
    // choose the segment 1
    st i1 = nb_segments - 1;
    Segment *segment1 = segments[i1];
    st segments2[nb_segments];

    // find all segments that are adjacent to the segment 1 by its first point
    int swap;
    st j = 0;
    for (st i2 = 0; i2 < nb_segments; i2++)
    {
        if (i2 == i1)
            continue;
        Segment *segment2 = segments[i2];
        if (!checkAngle(segment1, segment2))
            continue;
        if (!checkLength(segment1, segment2))
            continue;
        if (!checkCoordinates(segment1, segment2, 0, min_dist, &swap))
            continue;
        if (swap)
            swapPoints(segment2);
        segments2[j++] = i2;
    }
    segments2[j] = nb_segments; // end of array
```



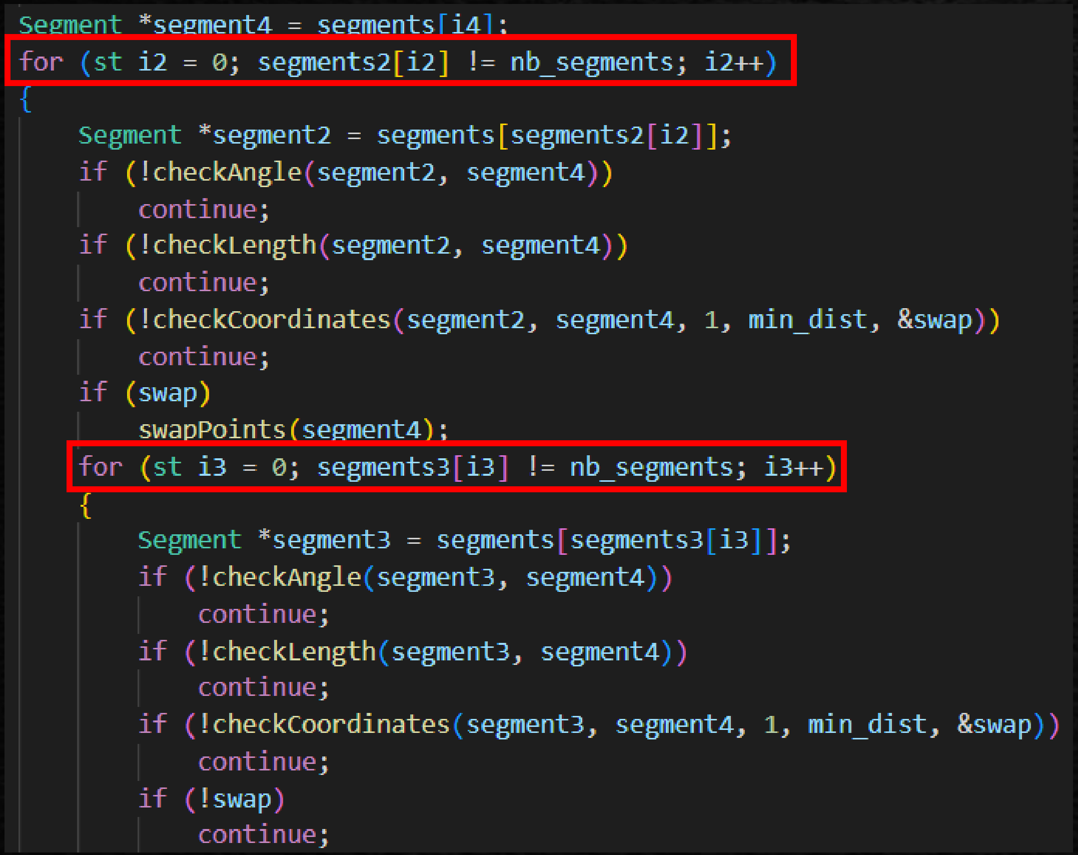
Construction de la grille



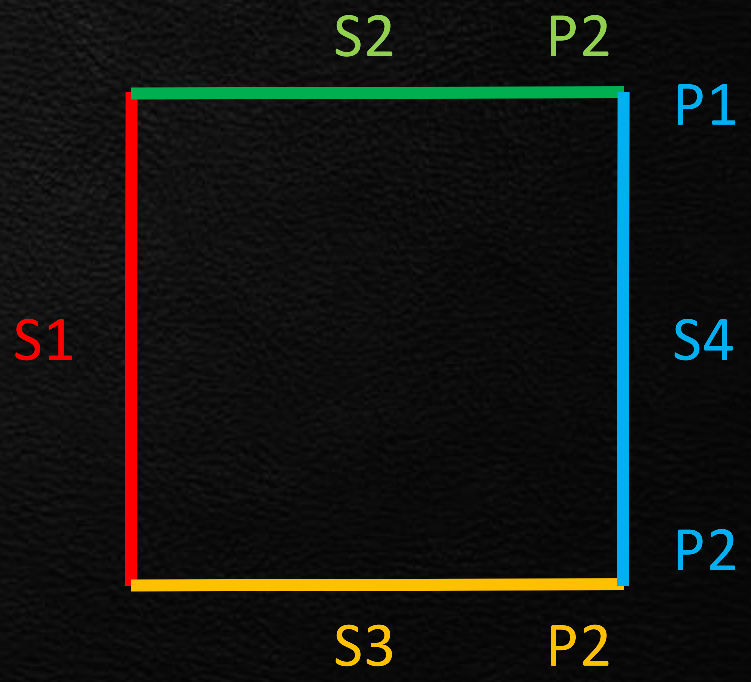
```
for (st i2 = 0; segments2[i2] != nb_segments; i2++)
```



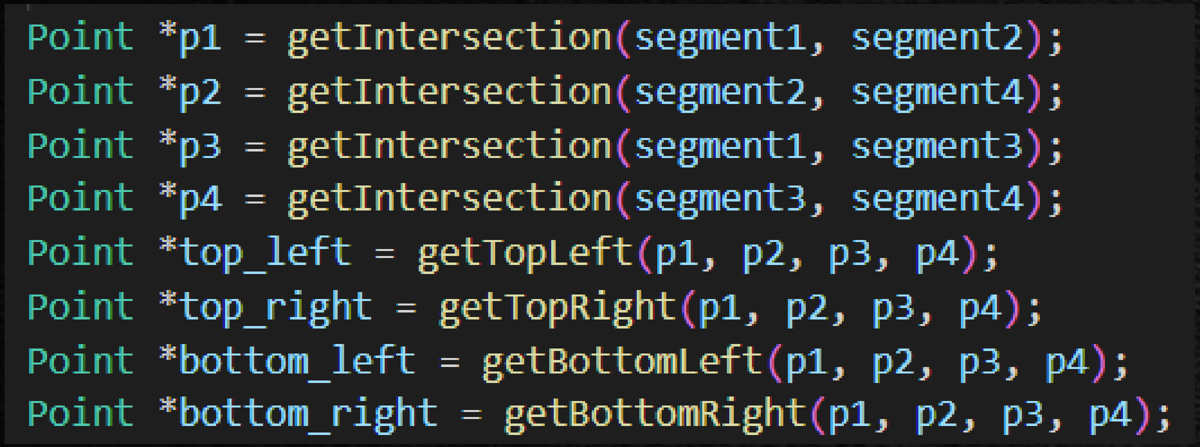
```
for (st i3 = 0; segments3[i3] != nb_segments; i3++)
```



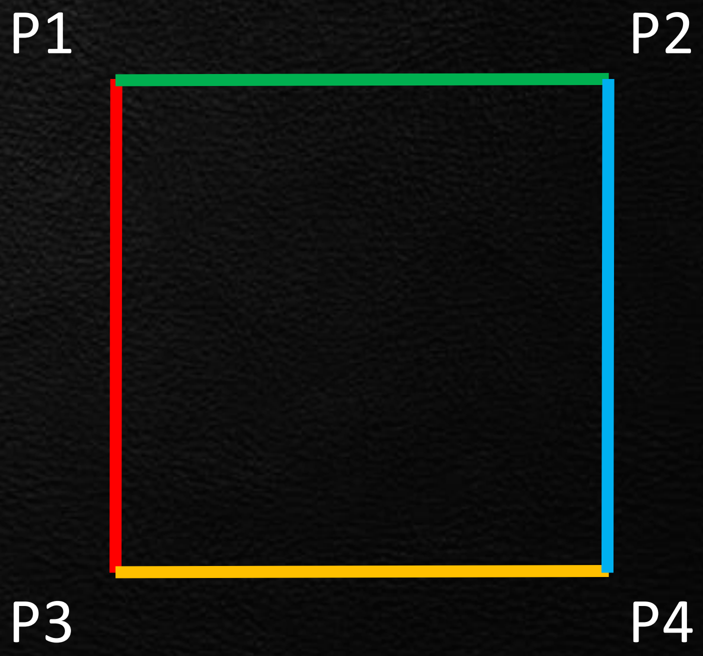
```
Segment *segment4 = segments[i4];
for (st i2 = 0; segments2[i2] != nb_segments; i2++)
{
    Segment *segment2 = segments[segments2[i2]];
    if (!checkAngle(segment2, segment4))
        continue;
    if (!checkLength(segment2, segment4))
        continue;
    if (!checkCoordinates(segment2, segment4, 1, min_dist, &swap))
        continue;
    if (swap)
        swapPoints(segment4);
    for (st i3 = 0; segments3[i3] != nb_segments; i3++)
    {
        Segment *segment3 = segments[segments3[i3]];
        if (!checkAngle(segment3, segment4))
            continue;
        if (!checkLength(segment3, segment4))
            continue;
        if (!checkCoordinates(segment3, segment4, 1, min_dist, &swap))
            continue;
        if (!swap)
            continue;
```



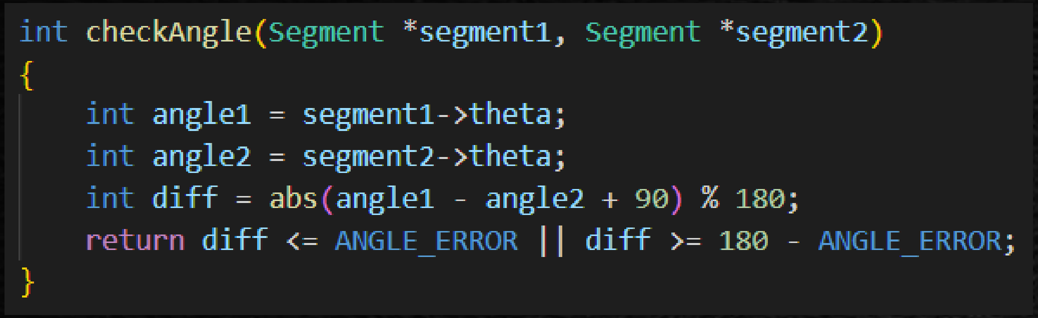
Construction de la grille



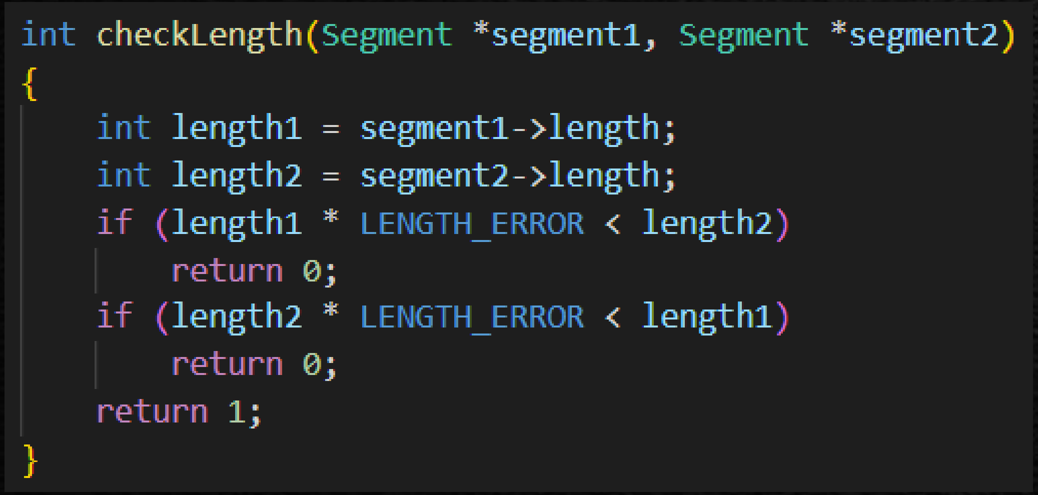
```
Point *p1 = getIntersection(segment1, segment2);
Point *p2 = getIntersection(segment2, segment4);
Point *p3 = getIntersection(segment1, segment3);
Point *p4 = getIntersection(segment3, segment4);
Point *top_left = getTopLeft(p1, p2, p3, p4);
Point *top_right = getTopRight(p1, p2, p3, p4);
Point *bottom_left = getBottomLeft(p1, p2, p3, p4);
Point *bottom_right = getBottomRight(p1, p2, p3, p4);
```



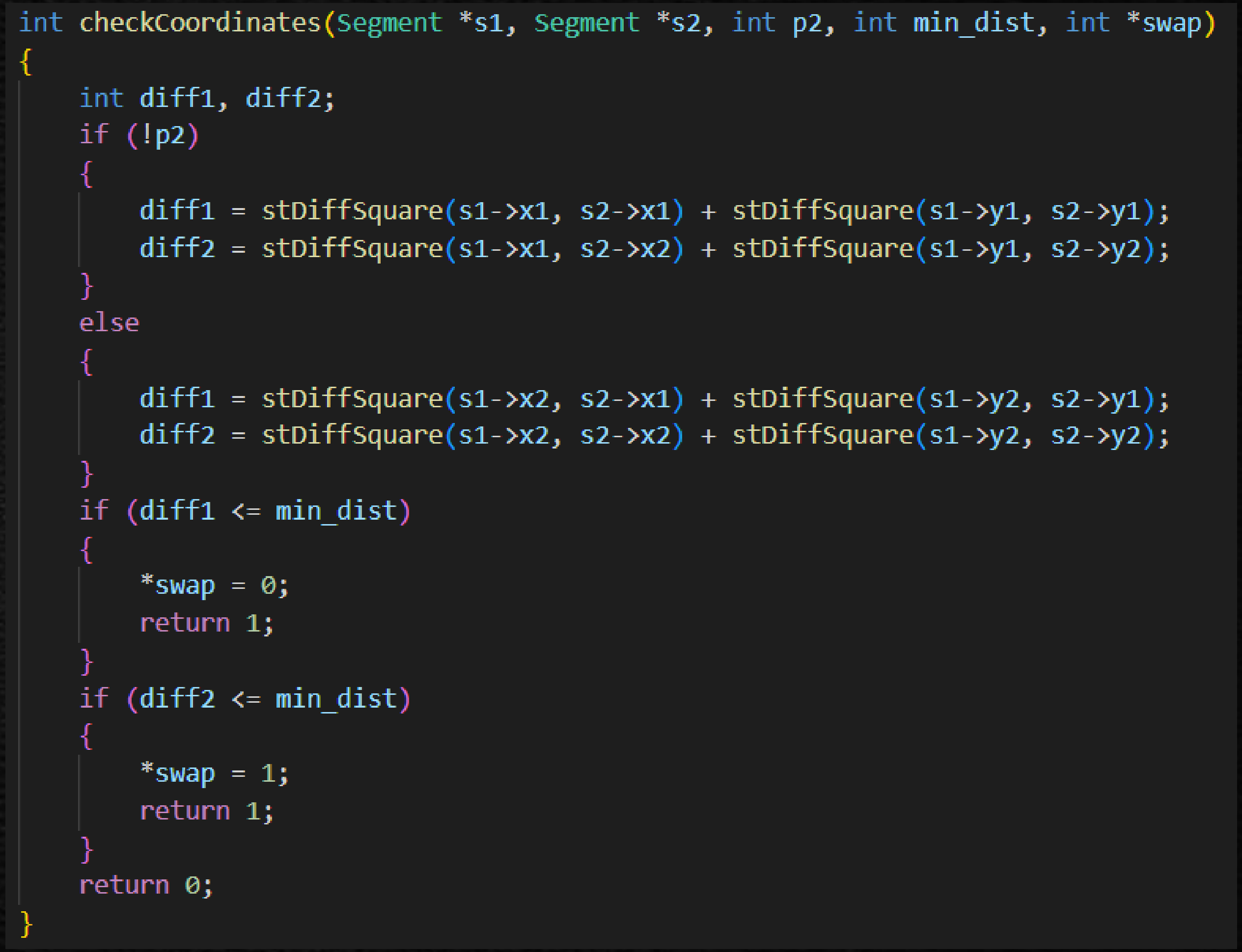
Construction de la grille



```c
int checkAngle(Segment *segment1, Segment *segment2)
{
    int angle1 = segment1->theta;
    int angle2 = segment2->theta;
    int diff = abs(angle1 - angle2 + 90) % 180;
    return diff <= ANGLE_ERROR || diff >= 180 - ANGLE_ERROR;
}
```

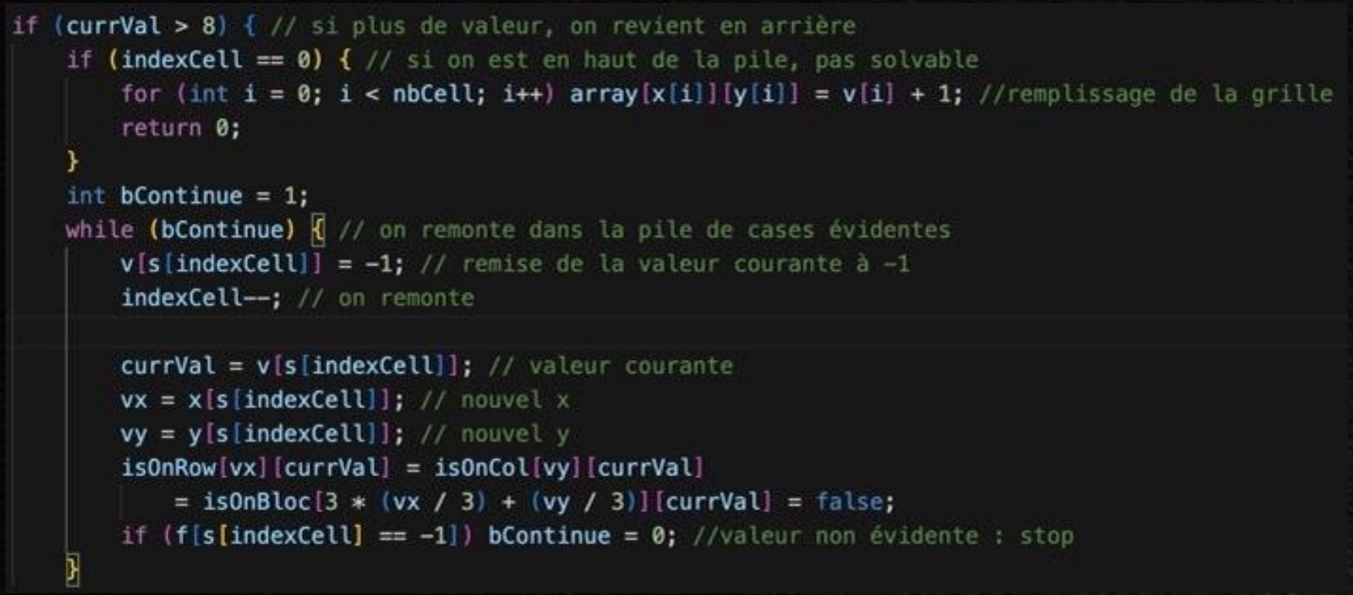


```c
int checkLength(Segment *segment1, Segment *segment2)
{
    int length1 = segment1->length;
    int length2 = segment2->length;
    if (length1 * LENGTH_ERROR < length2)
        return 0;
    if (length2 * LENGTH_ERROR < length1)
        return 0;
    return 1;
}
```

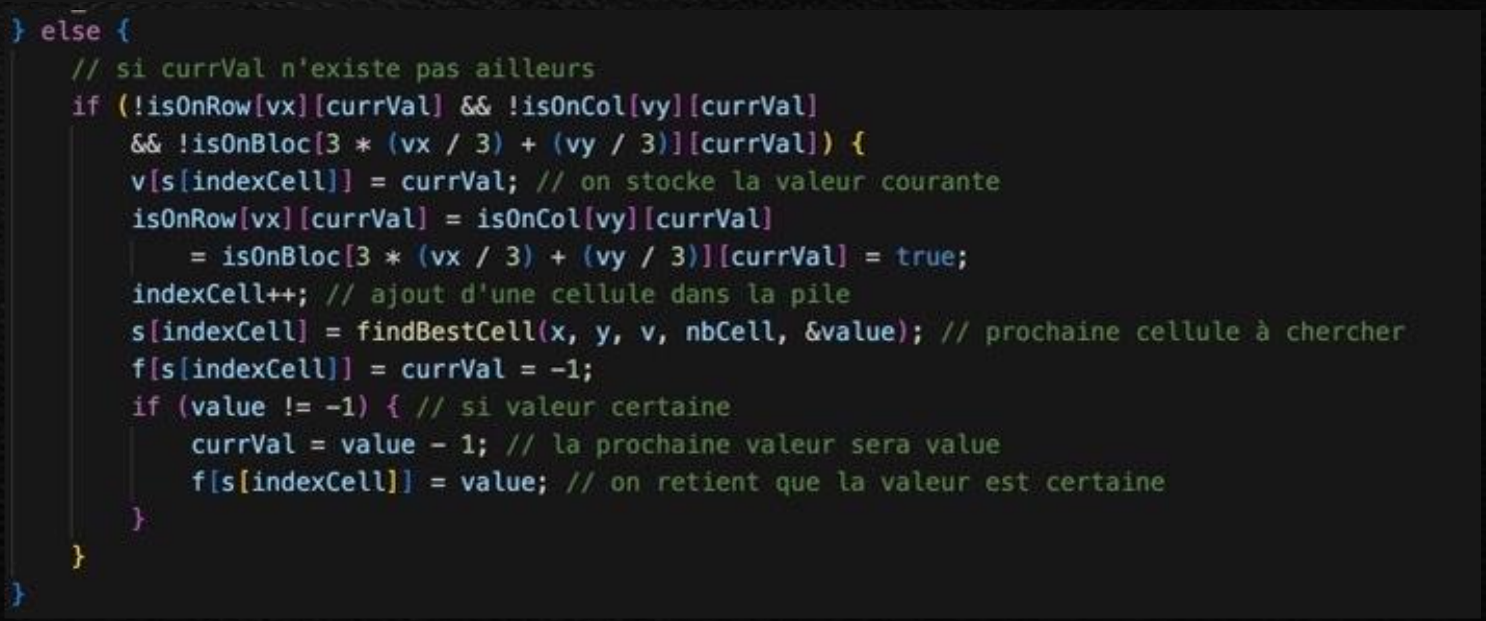


```c
int checkCoordinates(Segment *s1, Segment *s2, int p2, int min_dist, int *swap)
{
    int diff1, diff2;
    if (!p2)
    {
        diff1 = stDiffSquare(s1->x1, s2->x1) + stDiffSquare(s1->y1, s2->y1);
        diff2 = stDiffSquare(s1->x1, s2->x2) + stDiffSquare(s1->y1, s2->y2);
    }
    else
    {
        diff1 = stDiffSquare(s1->x2, s2->x1) + stDiffSquare(s1->y2, s2->y1);
        diff2 = stDiffSquare(s1->x2, s2->x2) + stDiffSquare(s1->y2, s2->y2);
    }
    if (diff1 <= min_dist)
    {
        *swap = 0;
        return 1;
    }
    if (diff2 <= min_dist)
    {
        *swap = 1;
        return 1;
    }
    return 0;
}
```

Solver



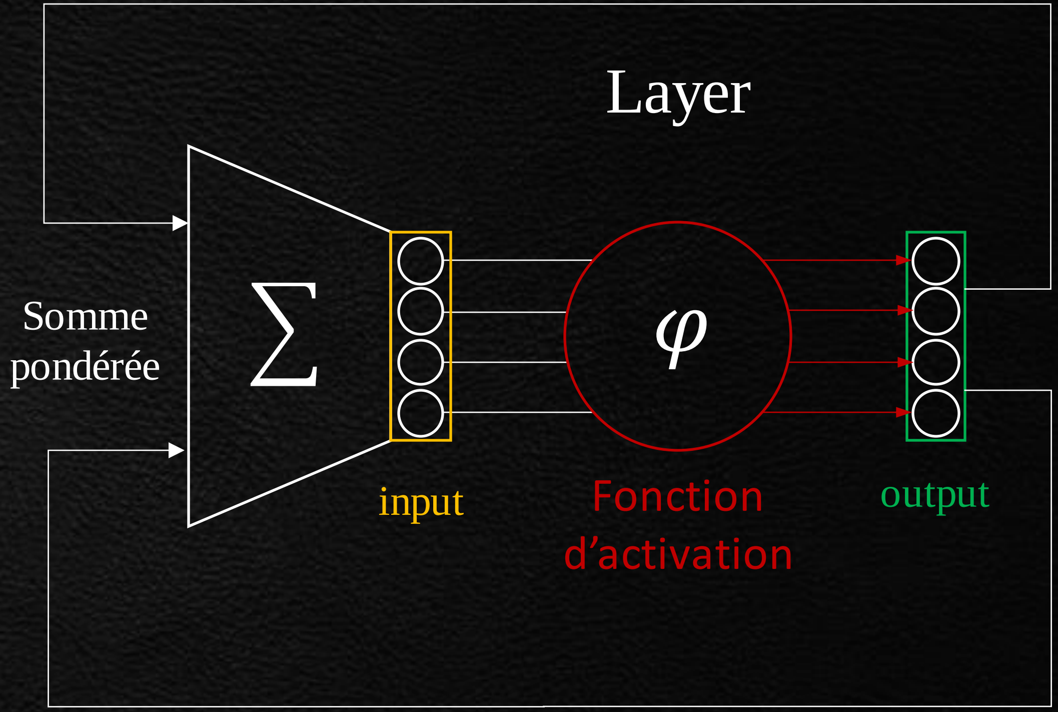
```
if (currVal > 8) { // si plus de valeur, on revient en arrière
    if (indexCell == 0) { // si on est en haut de la pile, pas solvable
        for (int i = 0; i < nbCell; i++) array[x[i]][y[i]] = v[i] + 1; //remplissage de la grille
        return 0;
    }
    int bContinue = 1;
    while (bContinue) { // on remonte dans la pile de cases évidentes
        v[s[indexCell]] = -1; // remise de la valeur courante à -1
        indexCell--; // on remonte

        currVal = v[s[indexCell]]; // valeur courante
        vx = x[s[indexCell]]; // nouvel x
        vy = y[s[indexCell]]; // nouvel y
        isOnRow[vx][currVal] = isOnCol[vy][currVal]
            = isOnBloc[3 * (vx / 3) + (vy / 3)][currVal] = false;
        if (f[s[indexCell] == -1]) bContinue = 0; //valeur non évidente : stop
    }
```



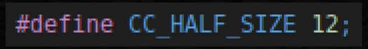
```
} else {
    // si currVal n'existe pas ailleurs
    if (!isOnRow[vx][currVal] && !isOnCol[vy][currVal]
        && !isOnBloc[3 * (vx / 3) + (vy / 3)][currVal]) {
        v[s[indexCell]] = currVal; // on stocke la valeur courante
        isOnRow[vx][currVal] = isOnCol[vy][currVal]
            = isOnBloc[3 * (vx / 3) + (vy / 3)][currVal] = true;
        indexCell++; // ajout d'une cellule dans la pile
        s[indexCell] = findBestCell(x, y, v, nbCell, &value); // prochaine cellule à chercher
        f[s[indexCell]] = currVal = -1;
        if (value != -1) { // si valeur certaine
            currVal = value - 1; // la prochaine valeur sera value
            f[s[indexCell]] = value; // on retient que la valeur est certaine
        }
    }
}
```

Activation des couches

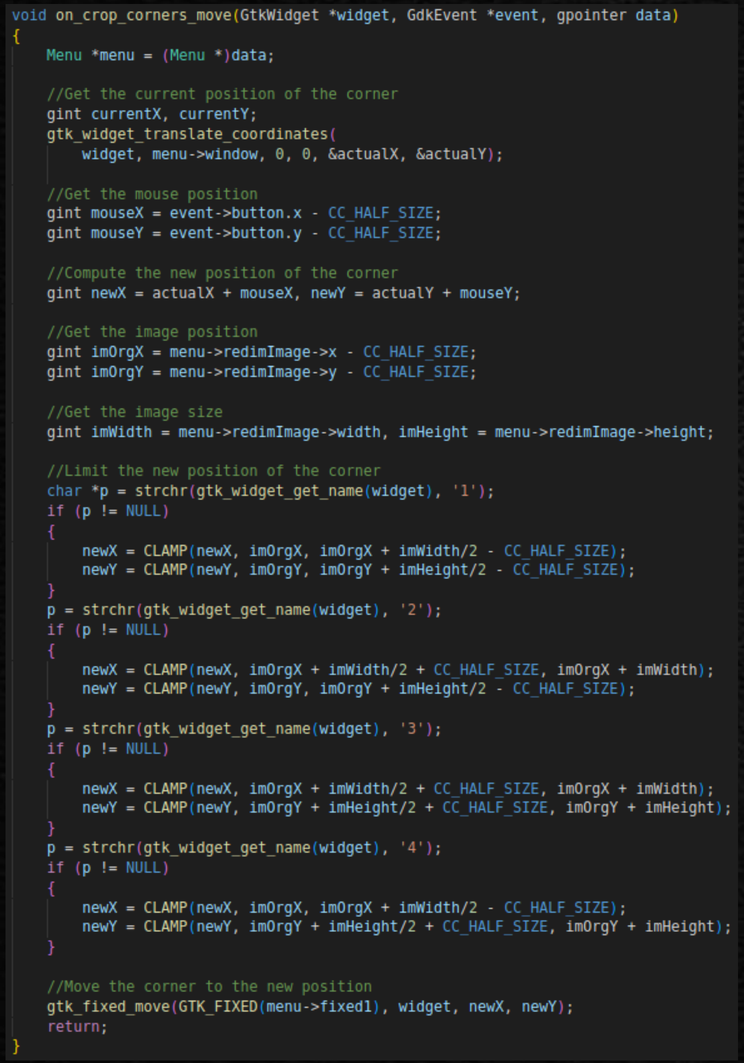
```
void Layer_Activate(Layer *layer) {









}
```



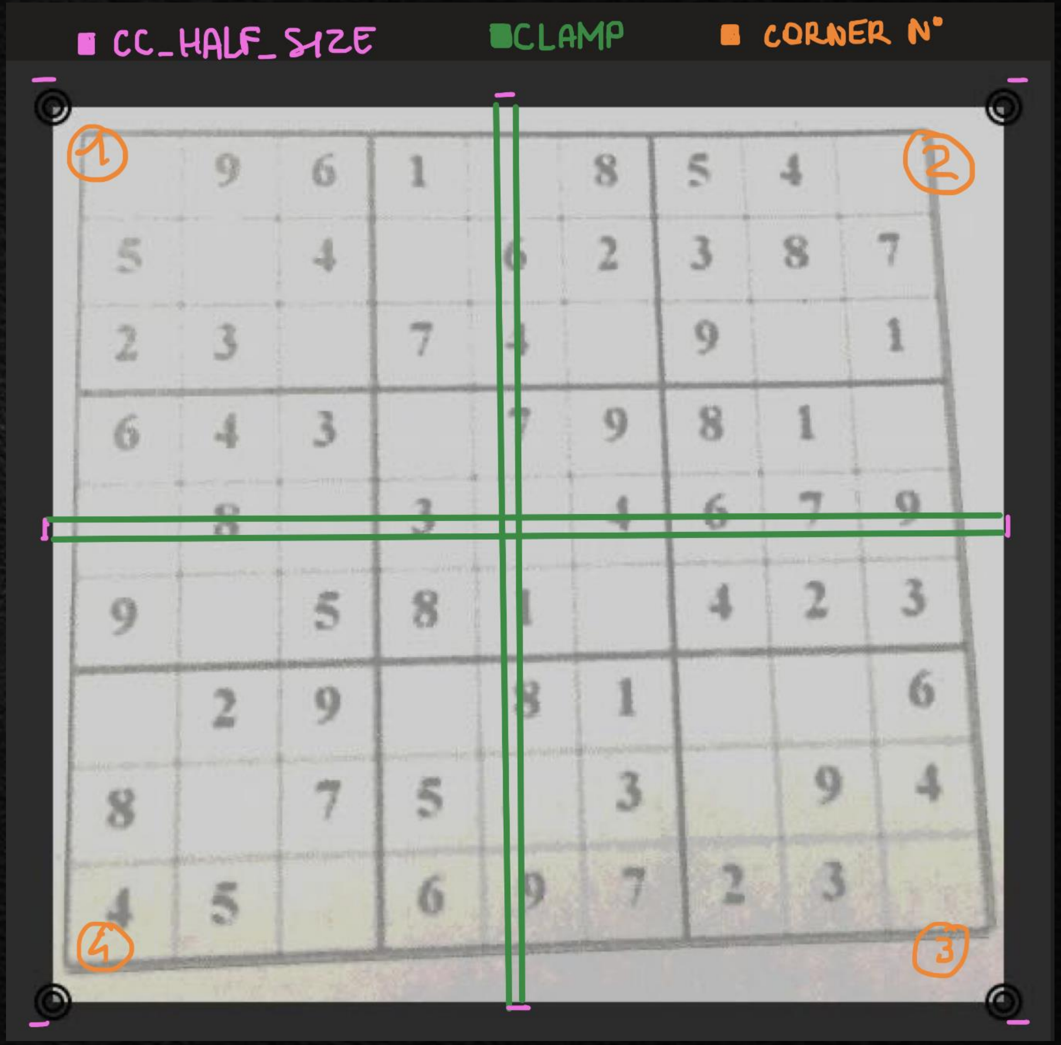
Redimensionnement manuel



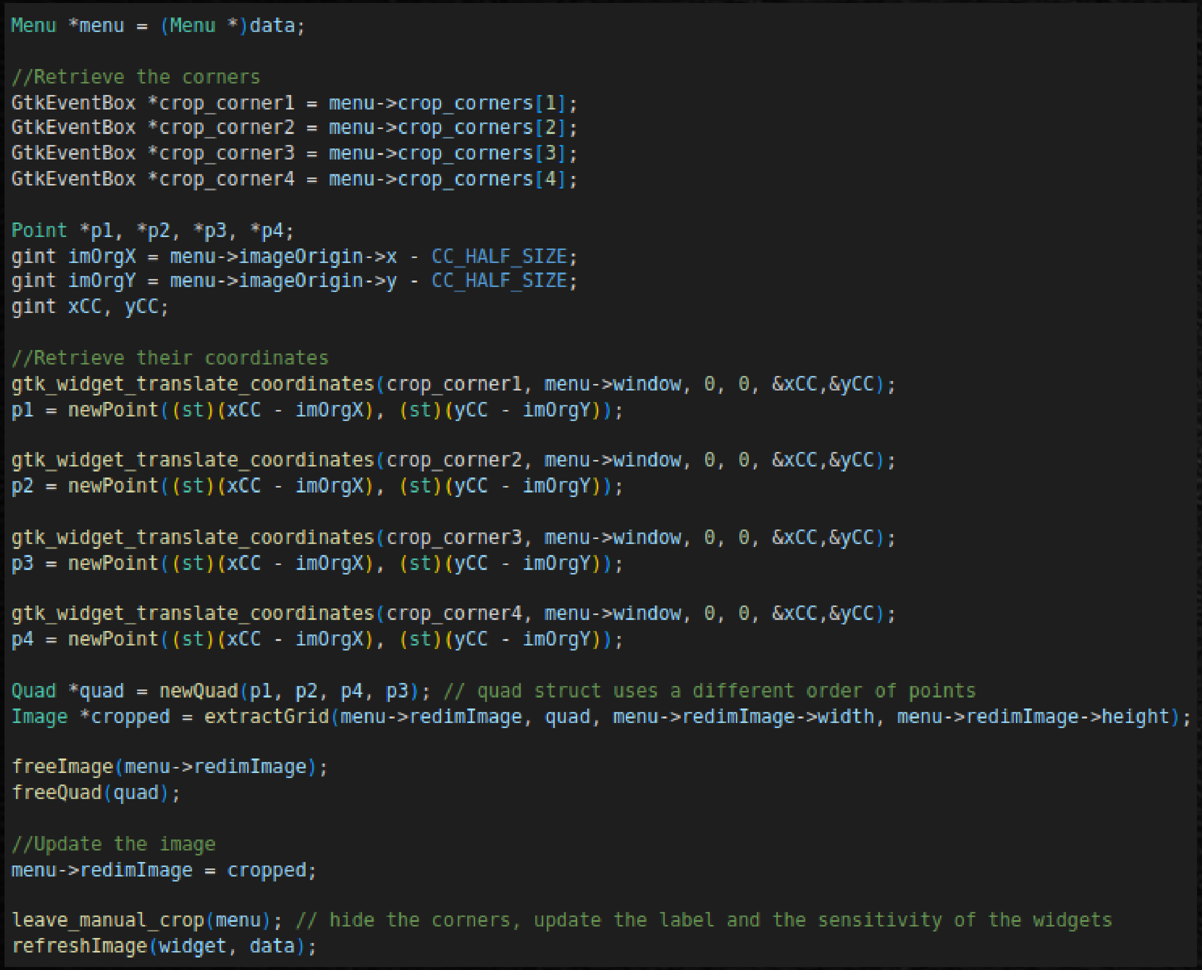
```c
#define CC_HALF_SIZE 12;
```



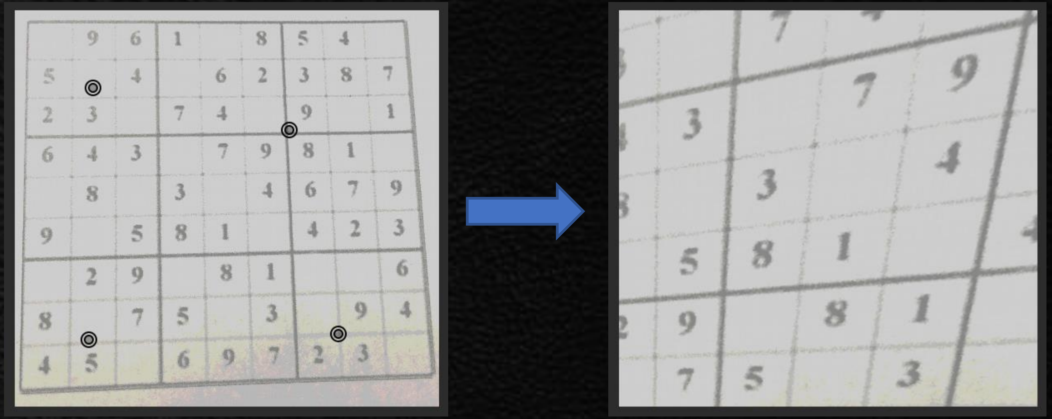
```c
void on_crop_corners_move(GtkWidget *widget, GdkEvent *event, gpointer data)
{
    Menu *menu = (Menu *)data;

    //Get the current position of the corner
    gint currentX, currentY;
    gtk_widget_translate_coordinates(
        widget, menu->window, 0, 0, &actualX, &actualY);

    //Get the mouse position
    gint mouseX = event->button.x - CC_HALF_SIZE;
    gint mouseY = event->button.y - CC_HALF_SIZE;

    //Compute the new position of the corner
    gint newX = actualX + mouseX, newY = actualY + mouseY;

    //Get the image position
    gint imOrgX = menu->redimImage->x - CC_HALF_SIZE;
    gint imOrgY = menu->redimImage->y - CC_HALF_SIZE;

    //Get the image size
    gint imWidth = menu->redimImage->width, imHeight = menu->redimImage->height;

    //Limit the new position of the corner
    char *p = strchr(gtk_widget_get_name(widget), '1');
    if (p != NULL)
    {
        newX = CLAMP(newX, imOrgX, imOrgX + imWidth/2 - CC_HALF_SIZE);
        newY = CLAMP(newY, imOrgY, imOrgY + imHeight/2 - CC_HALF_SIZE);
    }
    p = strchr(gtk_widget_get_name(widget), '2');
    if (p != NULL)
    {
        newX = CLAMP(newX, imOrgX + imWidth/2 + CC_HALF_SIZE, imOrgX + imWidth);
        newY = CLAMP(newY, imOrgY, imOrgY + imHeight/2 - CC_HALF_SIZE);
    }
    p = strchr(gtk_widget_get_name(widget), '3');
    if (p != NULL)
    {
        newX = CLAMP(newX, imOrgX + imWidth/2 + CC_HALF_SIZE, imOrgX + imWidth);
        newY = CLAMP(newY, imOrgY + imHeight/2 + CC_HALF_SIZE, imOrgY + imHeight);
    }
    p = strchr(gtk_widget_get_name(widget), '4');
    if (p != NULL)
    {
        newX = CLAMP(newX, imOrgX, imOrgX + imWidth/2 - CC_HALF_SIZE);
        newY = CLAMP(newY, imOrgY + imHeight/2 + CC_HALF_SIZE, imOrgY + imHeight);
    }

    //Move the corner to the new position
    gtk_fixed_move(GTK_FIXED(menu->fixed1), widget, newX, newY);
    return;
}
```



Redimensionnement manuel



```
Menu *menu = (Menu *)data;

//Retrieve the corners
GtkEventBox *crop_corner1 = menu->crop_corners[1];
GtkEventBox *crop_corner2 = menu->crop_corners[2];
GtkEventBox *crop_corner3 = menu->crop_corners[3];
GtkEventBox *crop_corner4 = menu->crop_corners[4];

Point *p1, *p2, *p3, *p4;
gint imOrgX = menu->imageOrigin->x - CC_HALF_SIZE;
gint imOrgY = menu->imageOrigin->y - CC_HALF_SIZE;
gint xCC, yCC;

//Retrieve their coordinates
gtk_widget_translate_coordinates(crop_corner1, menu->window, 0, 0, &xCC,&yCC);
p1 = newPoint((st)(xCC - imOrgX), (st)(yCC - imOrgY));

gtk_widget_translate_coordinates(crop_corner2, menu->window, 0, 0, &xCC,&yCC);
p2 = newPoint((st)(xCC - imOrgX), (st)(yCC - imOrgY));

gtk_widget_translate_coordinates(crop_corner3, menu->window, 0, 0, &xCC,&yCC);
p3 = newPoint((st)(xCC - imOrgX), (st)(yCC - imOrgY));

gtk_widget_translate_coordinates(crop_corner4, menu->window, 0, 0, &xCC,&yCC);
p4 = newPoint((st)(xCC - imOrgX), (st)(yCC - imOrgY));

Quad *quad = newQuad(p1, p2, p4, p3); // quad struct uses a different order of points
Image *cropped = extractGrid(menu->redimImage, quad, menu->redimImage->width, menu->redimImage->height);

freeImage(menu->redimImage);
freeQuad(quad);

//Update the image
menu->redimImage = cropped;

leave_manual_crop(menu); // hide the corners, update the label and the sensitivity of the widgets
refreshImage(widget, data);
```



Merci pour votre écoute