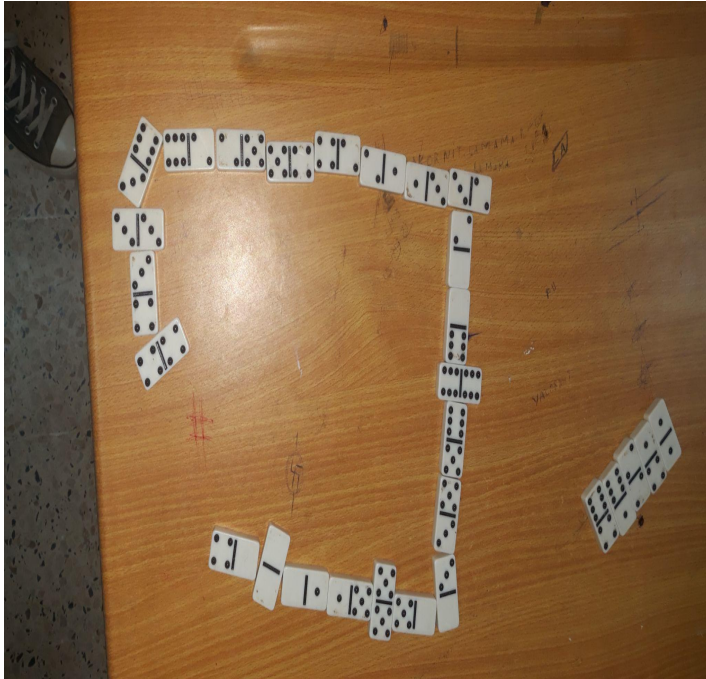


# rapport projet traitement d'image



IGE 45.  
ÉLABORÉS PAR :  
Mestar Nour El houda  
Hamnouché Houcine  
Figuigui Miloud

# Introduction:

Le traitement d'image est une discipline passionnante qui vise à analyser, améliorer et interpréter les images numériques. Avec l'avancée des technologies numériques, le traitement d'image est devenu une partie intégrante de nombreux domaines, tels que la vision par ordinateur, la robotique, la médecine, la surveillance, la réalité augmentée, et bien d'autres encore. Dans le cadre de notre parcours académique, nous avons été confrontés à un challenge de traitement d'image qui nous a permis d'explorer divers aspects de cette discipline et de relever des défis spécifiques.

Ce challenge s'est articulé autour de deux axes principaux : la décomposition de Fourier et le traitement d'une image spécifique liée aux dominos. L'objectif était de trouver des solutions optimales et de développer des algorithmes génériques capables de résoudre les problématiques rencontrées. Cependant, nous avons rapidement réalisé que la réalité était plus complexe que ce à quoi nous nous attendions.

Dans cette introduction, nous allons présenter les difficultés que nous avons rencontrées lors de ce challenge, les obstacles qui se sont dressés sur notre chemin, ainsi que les approches que nous avons adoptées pour surmonter ces difficultés. Nous soulignerons également l'importance de la combinaison de différentes techniques de traitement d'image pour obtenir des résultats satisfaisants.

Ce challenge a été une opportunité pour nous d'approfondir nos connaissances en matière de traitement d'image, d'explorer de nouvelles méthodes et d'acquérir une expérience pratique dans la résolution de problèmes complexes. Grâce à ce challenge, nous avons pu développer une vision plus nuancée du domaine du traitement d'image et comprendre l'importance de l'adaptation des méthodes existantes à des situations spécifiques.

Dans la suite de ce rapport, nous décrirons en détail les difficultés auxquelles nous avons été confrontés, les méthodes que nous avons explorées et les résultats que nous avons obtenus. Nous conclurons par une réflexion sur les apprentissages que nous avons tirés de ce challenge et les perspectives d'amélioration futures.

# Challenge Dominos :

Séparer les dominos de l'arrière-plan et éliminer le bruit s'est révélé être une tâche complexe, nécessitant plusieurs essais. J'ai finalement obtenu deux résultats finaux, le deuxième étant une combinaison entre le premier résultat et une nouvelle idée. **Initialement:**

## 1- Algorithme de Séparation des Dominos de l'Arrière-plan et Élimination du Bruit : Une Approche basée sur la Détection du Flash et la Morphologie Mathématique

1-Lire l'image : J'ai commencé par charger l'image d'origine dans l'algorithme.

2-Transformation en niveaux de gris : J'ai converti l'image en niveaux de gris afin de simplifier le traitement ultérieur.

3-Binarisation de l'image : J'ai appliqué une binarisation à l'image pour la segmenter en pixels noirs et blancs.

4-Érosion pour éliminer le bruit : J'ai construit un élément structurant de forme diamant et j'ai appliqué une opération d'érosion pour éliminer le bruit blanc causé par le flash du téléphone. Cependant, bien que l'érosion soit suffisante pour éliminer les autres bruits, elle ne suffisait pas à elle seule, car le flash était intense. Si j'appliquais l'érosion plusieurs fois, les trous des dominos devenaient trop grands, ce qui entraînait une déformation importante.



voici le des 4 étapes précédentes:

```
clear all;  
close all;
```

```

clc;
img = imread('img.jpg');

gimg=rgb2gray(img);% Convertir l'image en niveaux de gris

binary_img = imbinarize(gimg);% Binarisation de l'image

sel=strel('diamond',4);% Définir l'élément structurant

er=imerode(gimg,sel);% Application de l'érosion

```

5-Détection du flash : Pour résoudre ce problème, j'ai développé une méthode de détection du flash. J'ai remarqué que le flash se traduit par une variation importante dans l'image. J'ai utilisé le gradient et le filtre de Sobel pour détecter cette variation. En appliquant un seuil de 35 sur le gradient, j'ai pu éliminer les valeurs plus petites et ne conserver que les valeurs importantes correspondant au flash.



6-Complément de l'image gradient : J'ai obtenu le complément de l'image gradient pour obtenir la zone du flash en noir. J'ai également effectué une érosion pour augmenter la taille de cette zone noire.

7-Élimination du flash : En utilisant une opération logique entre l'image originale en niveaux de gris et l'image binarisée du flash (étape 6), j'ai réussi à éliminer le flash de l'image originale. En effet, le flash étant noir, il est exclu de cette opération, ne laissant que les parties du domino dans l'image. Voici l'explication en code et en image :

Le code des étapes 5,6 et 7 :

```

[Gmag, Gdir] = imgradient(gimg, 'Sobel');% Détection du contour sur l'image

Gmag(Gmag<35)=0; % Seuil du module de gradient

imshow(Gmag,[]);
title('Module de gradient après seuillage');

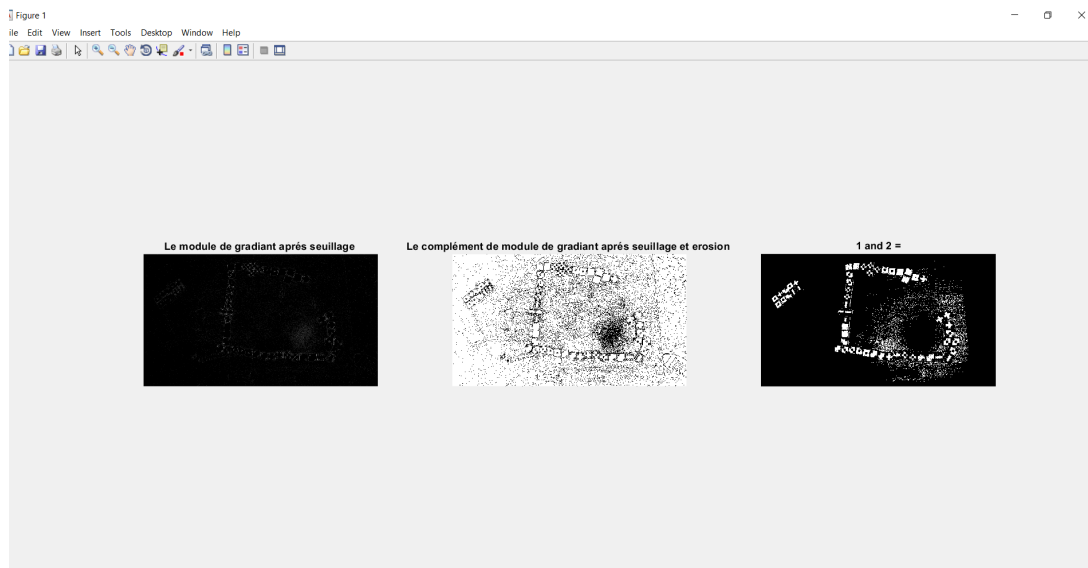
imcomp=imcomplement(Gmag); % Complément du module de gradient pour
éliminer les variations de flash

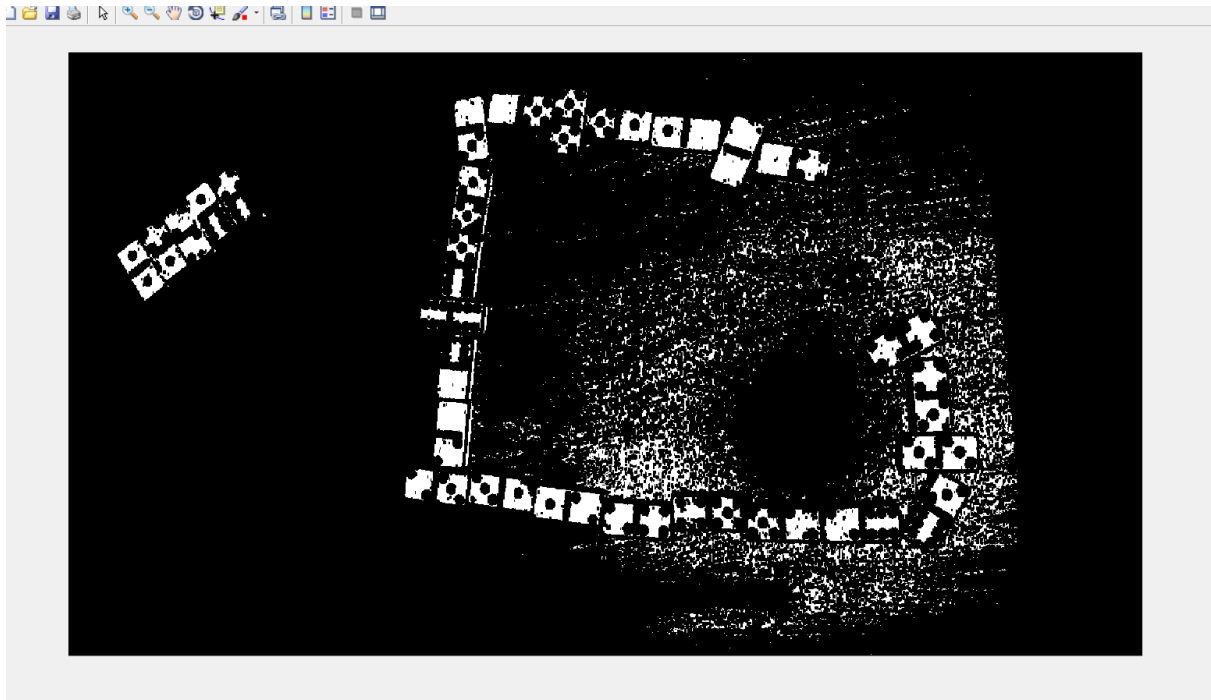
imb=imbinarize(imcomp);% Binarisation du complément du module de gradient

sel2=strel('rectangle',[10 5]);% Définition de l'élément structurant pour l'érosion

erc=imerode(imb,sel2);% Application de l'érosion
an=bitand(binary_img,erc);% Opération logique AND entre l'image binaire et le
complément de module de gradient

```



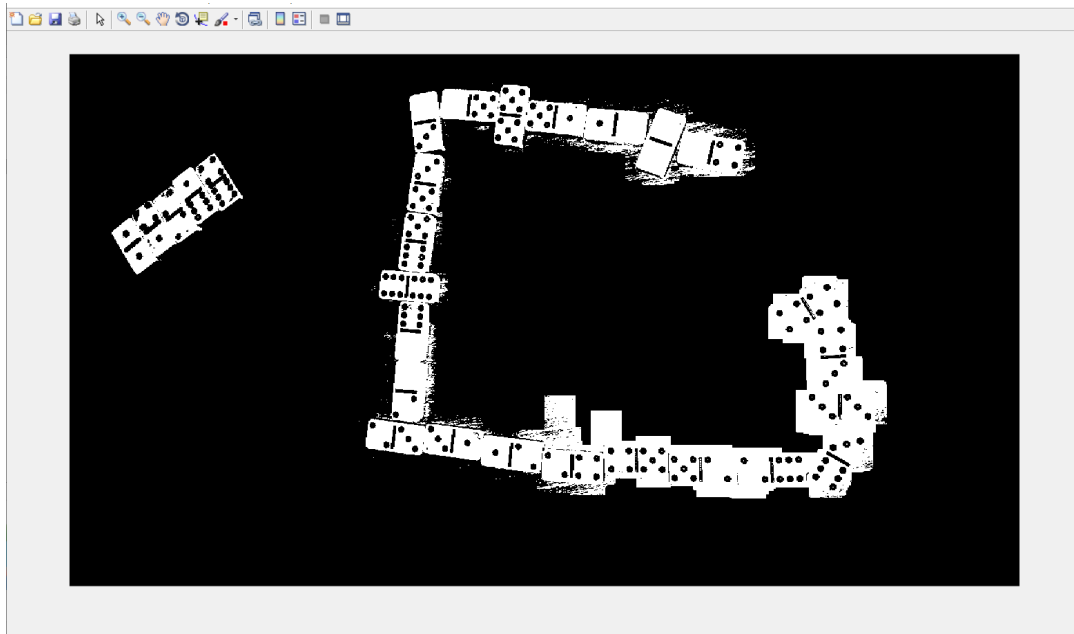


8-Malheureusement, le bruit n'était pas complètement éliminé, et il restait encore des artefacts indésirables. Pour remédier à cela, j'ai pris l'image résultante de l'étape précédente et appliqué un filtre rectangulaire de taille  $[20, 20]$ . J'ai d'abord effectué une ouverture pour éliminer les éléments connexes plus petits que l'élément structurant. Ensuite, j'ai réalisé six dilations pour éliminer les petits points noirs des dominos et ne conserver que le squelette principal des dominos. Enfin, j'ai effectué une fermeture pour fusionner les formes proches et combler les petits trous.

Donc j'ai trouvé ça :



9-Opération logique finale : Dans la dernière étape, j'ai réalisé une opération logique "ET" entre l'image résultante de l'étape précédente (étape 7) et l'image résultante de l'étape de réduction du bruit (étape 8). Donc Cela:



Le code de l'étape 8 et 9:

```
an=bitand(binary_img,erc);% Opération logique AND entre l'image binaire et  
le complément de module de gradient  
  
imshow(an);  
  
sel=strel('rectangle',[20 20]);% Définition de l'élément structurant pour les  
opérations morphologiques  
  
er=imopen(an,sel);% Ouverture  
  
for i=1:6  
    er=imdilate(er,sel);% Dilatation multiple  
end  
  
er=imclose(er,sel);% Fermeture  
  
imshow(er);
```

```
new=bitand(binary_img,er);% Opération logique AND entre l'image binaire  
et le résultat de l'érosion et la dilatation  
  
imshow(new);
```

### **L'algorithme 1 complet :**

```
clear all;  
close all;  
clc;  
img = imread('img.jpg');  
  
gimg=rgb2gray(img);% Convertir l'image en niveaux de gris  
  
binary_img = imbinarize(gimg);% Binarisation de l'image  
  
sel=strel('diamond',4);% Définir l'élément structurant  
  
er=imerode(gimg,sel);% Application de l'érosion  
  
[Gmag, Gdir] = imgradient(gimg, 'Sobel');% Détection du contour sur l'image  
  
Gmag(Gmag<35)=0; % Seuil du module de gradient  
  
imcomp=imcomplement(Gmag); % Complément du module de gradient  
pour éliminer les variations de flash  
  
imb=imbinarize(imcomp);% Binarisation du complément du module de  
gradient  
  
sel2=strel('rectangle',[10 5]);% Définition de l'élément structurant pour  
l'érosion  
  
erc=imerode(imb,sel2);% Application de l'érosion  
  
an=bitand(binary_img,erc);% Opération logique AND entre l'image binaire et  
le complément de module de gradient  
  
sel=strel('rectangle',[20 20]);% Définition de l'élément structurant pour les  
opérations morphologiques  
  
er=imopen(an,sel);% Ouverture
```



```
for i=1:6
    er=imdilate(er,sel);% Dilatation multiple
end

er=imclose(er,sel);% Fermeture

new=bitand(binary_img,er);% Opération logique AND entre l'image binaire
et le résultat de l'érosion et la dilatation

imshow(new);
```

**Cependant, malgré mes tentatives pour éliminer le bruit en utilisant l'érosion et en ajustant les paramètres de seuil, il reste toujours un bruit considérable. Ce problème persistant m'a conduit à la conclusion que cet algorithme n'était pas suffisant pour une élimination complète du bruit. J'ai donc ressenti le besoin de trouver une nouvelle idée pour résoudre ce problème.**

## **2-Algorithme 2 (final):**

Après avoir réalisé l'algorithme initial et obtenu les résultats partiels, j'ai réalisé qu'il y avait un potentiel d'amélioration en intégrant une nouvelle idée dès le début du processus. Voici comment j'ai intégré cette idée supplémentaire :

J'ai commencé par charger l'image d'origine dans l'algorithme et l'ai affichée pour une meilleure visualisation.

J'ai défini des valeurs RGB spécifiques pour détecter les dominos dans l'image. Ces valeurs ont été choisies en fonction des caractéristiques de couleur des dominos que je souhaitais détecter.

Pour tenir compte des variations de couleur dues à l'éclairage ou à d'autres facteurs, j'ai défini une plage de tolérance pour chaque canal RGB. Cela permet d'accepter des valeurs légèrement différentes tout en considérant les pixels comme étant similaires à la couleur spécifiée.

J'ai créé deux masques binaires en utilisant les plages de tolérance et les valeurs RGB définies. Chaque masque sélectionne les pixels ayant des couleurs similaires à celles des dominos que je souhaite détecter.

J'ai combiné les deux masques à l'aide de l'opération logique "OU". Cela a permis de créer un masque global qui sélectionne les pixels correspondant à l'une ou l'autre des couleurs de domino spécifiées.

En appliquant ce masque à l'image d'origine à l'aide de l'opération logique "ET", j'ai obtenu une nouvelle image dans laquelle seuls les pixels correspondant aux dominos étaient conservés. Les autres pixels étaient mis à zéro.

Voici le code de l'idée :

```
clear all;
clc;
% Charger l'image
img = imread('img.jpg');
% Figure 1
figure(1);

% Subplot 1
subplot(1, 3, 1);
imshow(img);
title('Image originale');
imshow(img);

% Les valeurs RGB pour détecté le domino
r1 = 188;
g1 = 189;
b1 = 181;

r2 = 141;
g2 = 142;
b2 = 134;

% Définir une plage de tolérance pour chaque canal RGB
tolerance = 30;

% Créer un masque binaire qui sélectionne les pixels ayant des couleurs
similaires à r1, g1, b1
mask1 = (img(:,:,1) >= r1-tolerance) & (img(:,:,1) <= r1+tolerance) & ...
        (img(:,:,2) >= g1-tolerance) & (img(:,:,2) <= g1+tolerance) & ...
        (img(:,:,3) >= b1-tolerance) & (img(:,:,3) <= b1+tolerance);

% Créer un masque binaire qui sélectionne les pixels ayant des couleurs
% similaires à r2, g2, b2
mask2 = (img(:,:,1) >= r2-tolerance) & (img(:,:,1) <= r2+tolerance) & ...
        (img(:,:,2) >= g2-tolerance) & (img(:,:,2) <= g2+tolerance) & ...
        (img(:,:,3) >= b2-tolerance) & (img(:,:,3) <= b2+tolerance);

% combinaison des deux masque a l'aide de l'opération logique ou
mask = mask1 | mask2;

% application du masque à l'image
```

```

result = bsxfun(@times, img, cast(mask, 'like', img));

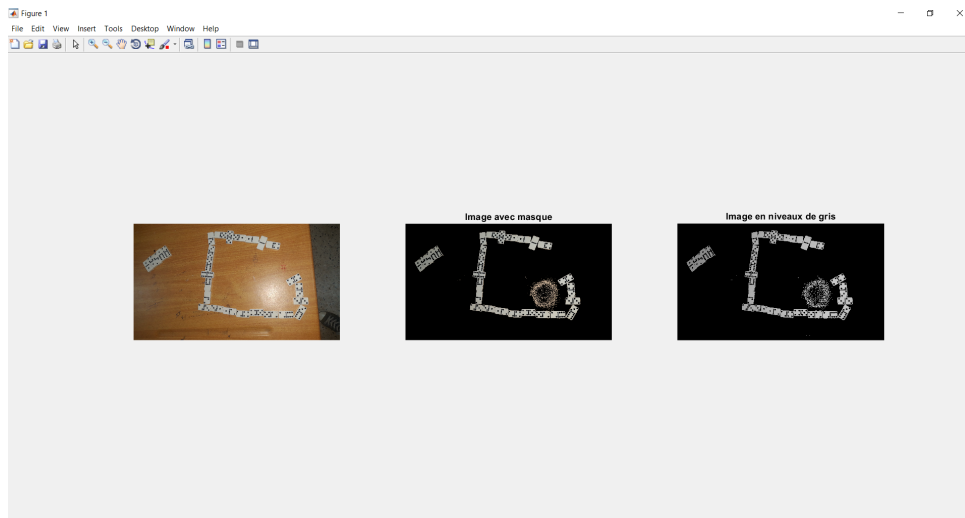
% Conversion de l'image résultante en niveaux de gris
gimg=rgb2gray(result);

% Afficher l'image résultante et l'image en niveaux de gris
subplot(1, 3, 2);
imshow(result);
title('Image avec masque');
subplot(1, 3, 3);
imshow(gimg);
title('Image en niveaux de gris');

% Binarisation de l'image en niveaux de gris
binary_img1 = imbinarize(gimg);

```

Après l'application de la nouvelle idée sur l'image, j'ai réussi à obtenir des contours précis pour les dominos, mais le bruit causé par le flash était toujours présent. Vous pouvez voir l'image résultante des dominos bien contournés ici :



Note : Prenons en compte que les valeurs R1, G1, B1, R2, G2 et B2 ont été fixées au début de l'algorithme, où l'utilisateur sélectionne deux positions dans l'image pour prendre les valeurs RGB correspondantes et les utiliser pour éliminer les autres couleurs. Avec cette méthode, j'ai effectué plusieurs essais en ajustant ces valeurs pour obtenir les meilleurs résultats possibles.

On continue :

Pour résoudre ce problème de bruit, j'ai pris cette image, l'ai convertie en niveaux de gris, et l'ai binarisée. Ensuite, j'ai combiné le résultat avec les résultats de l'algorithme 1.

La combinaison se fait en réalisant une opération logique "ET" entre le premier résultat obtenu (issu de l'algorithme 1) et le deuxième résultat de la nouvelle idée utilisant les canaux RGB. Cette opération m'a permis d'effectuer une séparation efficace entre les dominos et l'arrière-plan.

Ces étapes supplémentaires ont contribué à améliorer la précision de la détection des dominos et à réduire le bruit persistant causé par le flash.

### **Le code 2 (code final):**

```
clear all;  
clc;  
% Charger l'image  
img = imread('img.jpg');  
% Figure 1  
figure(1);  
  
% Subplot 1  
subplot(1, 3, 1);  
imshow(img);  
title('Image originale');  
imshow(img);  
  
% Les valeurs RGB pour détecté le domino  
r1 = 188;  
g1 = 189;  
b1 = 181;  
  
r2 = 141;  
g2 = 142;  
b2 = 134;  
  
% Définir une plage de tolérance pour chaque canal RGB  
tolerance = 30;  
  
% Créer un masque binaire qui sélectionne les pixels ayant des  
couleurs similaires à r1, g1, b1
```

```

mask1 = (img(:,:,1) >= r1-tolerance) & (img(:,:,1) <= r1+tolerance) & ...
        (img(:,:,2) >= g1-tolerance) & (img(:,:,2) <= g1+tolerance) & ...
        (img(:,:,3) >= b1-tolerance) & (img(:,:,3) <= b1+tolerance);

% Créer un masque binaire qui sélectionne les pixels ayant des
couleurs
% similaires à r2, g2, b2
mask2 = (img(:,:,1) >= r2-tolerance) & (img(:,:,1) <= r2+tolerance) & ...
        (img(:,:,2) >= g2-tolerance) & (img(:,:,2) <= g2+tolerance) & ...
        (img(:,:,3) >= b2-tolerance) & (img(:,:,3) <= b2+tolerance);

% combinaison des deux masque a l'aide de l'opération logique ou
mask = mask1 | mask2;

% application du masque à l'image
result = bsxfun(@times, img, cast(mask, 'like', img));

% Conversion de l'image résultante en niveaux de gris
gimg=rgb2gray(result);

% Afficher l'image résultante et l'image en niveaux de gris
subplot(1, 3, 2);
imshow(result);
title('Image avec masque');
subplot(1, 3, 3);
imshow(gimg);
title('Image en niveaux de gris');

% Binarisation de l'image en niveaux de gris
binary_img1 = imbinarize(gimg);

% Conversion de l'image d'origine en niveaux de gris
gimg1=rgb2gray(img);
% Binarisation de l'image d'origine pour detecter le contour avec cette
% image
binary_img = imbinarize(gimg1);

% Définition de l'élément structurant pour l'érosion et la dilatation
sel=strel('diamond',4);

% Application de l'érosion
er=imerode(gimg1,sel);

% Afficher l'image après l'érosion et la dilatation
figure(2)

```

```

subplot(1, 3, 1);
imshow(er);
title('Image après érosion et dilatation');

% Figure 2
figure(2);

% Détection du contour de l'image
[Gmag, Gdir] = imgradient(gimg1, 'Sobel');%detection de contour

% Suppression des contours faibles
Gmag(Gmag<35)=0; %pour binarisé l'image

% Complément de l'image gradient pour éliminer les variations dues
au flash
imcomp=imcomplement(Gmag); %le complément de Gmag pour faire
éliminé la grande variation de flash

% Binarisation de l'image complémentaire
imc=imbinarize(imcomp);%binarisation pour faire les operation logique

% Définition de l'élément structurant pour l'érosion
sel2=strel('rectangle',[10 5]);%element struct

% Application de l'érosion
erc=imerode(imc,sel2);

% Combinaison de l'image binaire originale et de l'image résultante de
l'érosion
an=bitand(binary_img,erc);

% Afficher l'image résultante après la combinaison
subplot(1, 3, 2);
imshow(an);
title('Combinaison binaire');

% Définition de l'élément structurant pour des les opérations
% morphologiques

%element struct
sel=strel('rectangle',[20 20]);

%Ouverture
er=imopen(an,sel);

```

```
%Multiple dilatation
for i = 1:6
    er = imdilate(er, sel);
end

%Fermeture
er=imclose(er,sel);

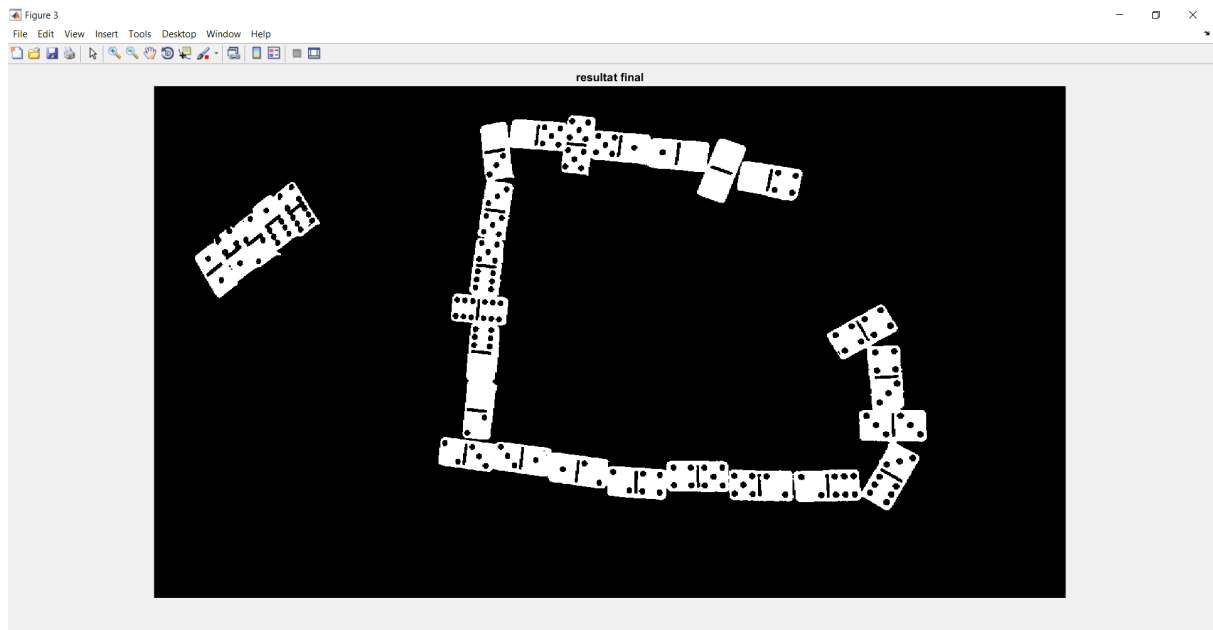
% Afficher l'image résultante après les operations
subplot(1, 3, 3);
imshow(er);
title('Le support pour le AND logique');

% Combinaison du résultat de l'opération logique et de l'image binaire
résultante
resultat=bitand(er,binary_img1);

% Filtrage médian pour réduire plus le bruit
resultat = medfilt2(resultat, [10, 10]);

% Afficher l'image résultante après l'operation and et réduction de
bruit
figure(3)
imshow(resultat);
title('resultat final');
```

**Le résultat final :**

**Note 2 :**

j'ai ajouté une étape supplémentaire consistant à appliquer un filtre moyenneur.

Ce filtre moyenneur a pour objectif de réduire les imperfections et le bruit restant dans l'image. Il permet de lisser les contours des dominos et d'améliorer la séparation entre les dominos et l'arrière-plan. En utilisant ce filtre moyenneur avant d'obtenir le résultat final, j'ai réussi à améliorer significativement la qualité et la précision de la détection des dominos.



## **squelettisation:**

Le code:

```
% Créer un élément structurant pour la dilatation
se1 = strel("disk", 4);

% Dilater l'image résultante pour renforcer le squelette
skel = imdilate(resultat, se1);

% Eleminer le trous noir
skel = medfilt2(skel, [50, 50]);
skel = medfilt2(skel, [20, 20]);
skel = medfilt2(skel, [15, 15]);

% Effectuer une opération logique OR entre le squelette et l'image résultante
skel2 = bitor(skel, resultat);

% Effectuer plusieurs érosions pour affiner le squelette
for i = 1:7
    skel2 = imerode(skel2, se1);
end

% Effectuer une squelettisation sur l'image binaire
skeleton = bwmorph(skel2, 'skel', inf);

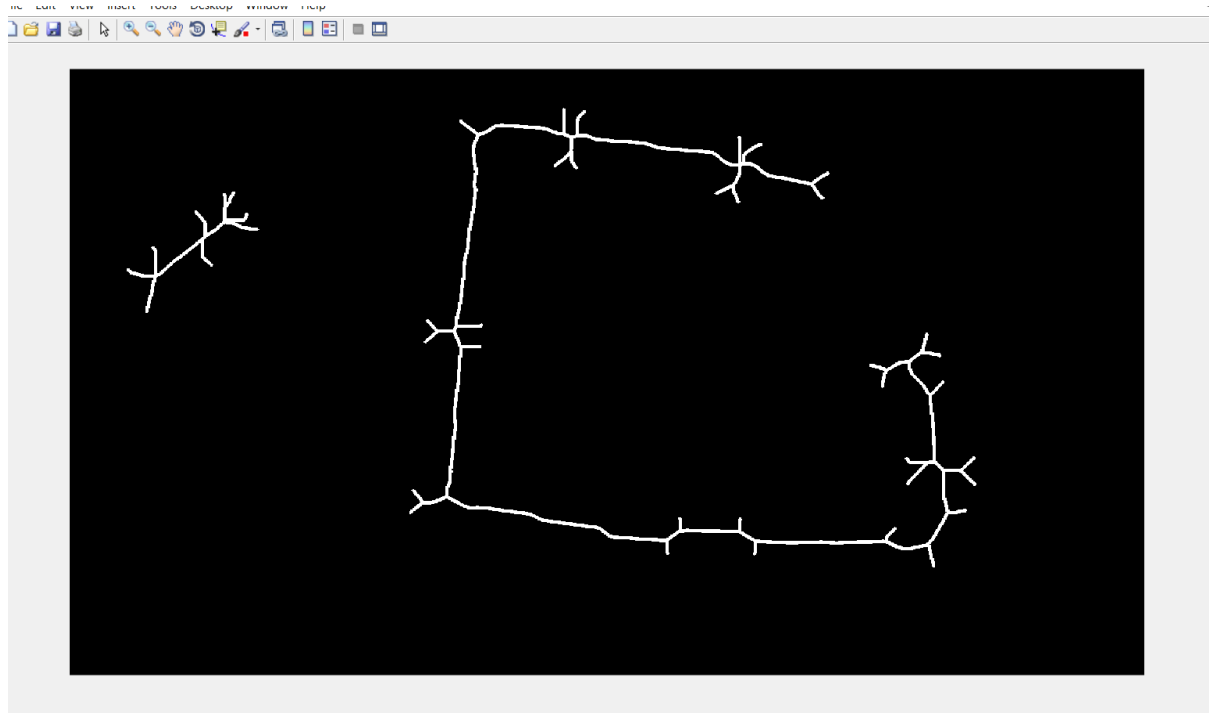
% Effectuer un amincissement sur la ligne centrale squelettisée
thinned = bwmorph(skeleton, 'thin', Inf);

% Dilater la ligne amincie pour améliorer sa visibilité
se2 = strel("disk", 4);
thinned = imdilate(thinned, se2);
thinned = imdilate(thinned, se2);

% Afficher l'image amincie
imshow(thinned);
```

Dans la première partie du code de squelettisation, j'ai essayé d'éliminer les trous noirs du domino et les pixels noirs qui touchent le contour du domino (sans faire de dilatation). Pour ce faire, j'ai appliqué plusieurs filtres moyenneurs et j'ai éliminé tous les trous noirs. Ensuite, j'ai effectué une opération logique avec l'image initiale afin de récupérer les contours déformés. Ensuite, j'ai effectué une squelettisation suivie d'un amincissement à l'aide de la fonction prédéfinie bwmorph. Pour rendre le squelette bien visible, j'ai réalisé des dilatations à la fin.

voici le résultat final :



### -Pouvoir compter le nbr de pièces de Domino :

L'algorithme que j'ai utilisé est un algorithme simple , l'idée est de choisir une pièce moyenne cad une pièce dont le nbr de points noir est de 6 et la sélectionner en utilisant un masque , j'ai utilisé "impixelinfo" pour extraire les coordonnées de la pièce moyenne 4/2 pour les utiliser pour former le masque , ensuite j'ai compter le nbr de pixels blancs de la pièce 4/2 en utilisant une boucle for , après j'ai calculé le nbr de pixels blancs dans toute l'image originale en utilisant une boucle for toujours , j'ai divisé :

nbr de px blanc dans toute l'image/nbr de pixels blancs de la pièce 4/2  
pour avoir un résultat qui égale presque à 28 .  
avec un float to int c'était le bon résultat 28.  
voici le bout de code :

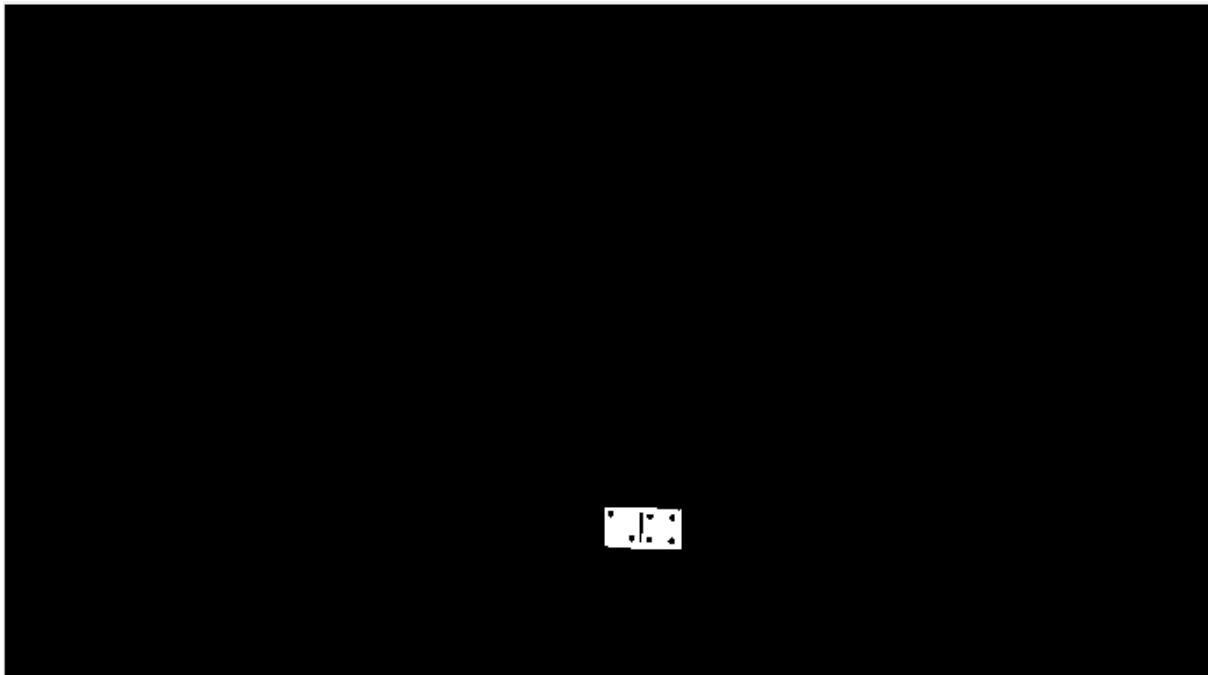
```
impixelinfo;%pour choisir les coordonnees d'une piece moyenne.
masque=zeros(2322,4128); % un masque noir
figure()
imshow(masque);
for i=1728:1870
    for j=2057:2322
        masque(i,j)=1;
    end
end %la boucle pour selectionner la place de la piece , j'ai choisi 4/2
puisque c'est une piece moyenne (contient 6 points noir).
imshow(masque);
ress=bitand(masque,skel);%l'intersection pour selectionner la piece 4/2.
imshow(ress);
```

```

ss=0;
for i= 1:2322
    for j=1:4128
        if ress(i,j)==1;
            ss=ss+1;
        end
    end
end %la boucle pour compter le nbr de pixel blancs dans la piece moyenne
4/2
ss
rs=0;
for i= 1:2322
    for j=1:4128
        if skel(i,j)==1;
            rs=rs+1;
        end
    end
end %la boucle pour compter le nbr de pixel blancs dans toutes le pieces.
rs
ans= int32(rs/ss);%on divise: nbr px blancs dans toutes le pieces / nbr px
blancs dans la piece moyenne 4/2
fprintf('le nombre de pieces de domino est :');%on obtient le nbr de pieces
de domino.
ans

```

**-le resultat :**



```
le nombre de pieces de domino est :  
ans =  
  
int32  
  
28
```

### **Pouvoir sélectionner une pièce, et de compter le nombre de points qu'elle représente.**

**-la première idée** qui m'a arrivé est d'utiliser la croissance de région cad :

je clique , je mets une graine , elle fait de l'expansion jusqu'à ce qu'elle remplit tout l'espace blanc de la pièce , les trous sont en noir , la pièce est sélectionnée en blanc , le masque de croissance est circulaire et la condition d'appartenance est stricte (pas de tolérance) ,puisque c'est une image binaire..

#### **les challenges :**

- Premièrement, il faut cliqué sur l'espace blanc de la pièce , pas le noir, sinon ça selectionne le point noir pas la pièce. .
- Deuxièmement , les traits qui séparent les pièces et limitent la croissance de la région d'une pièce à une autre sont presque gommés à cause des prétraitements .

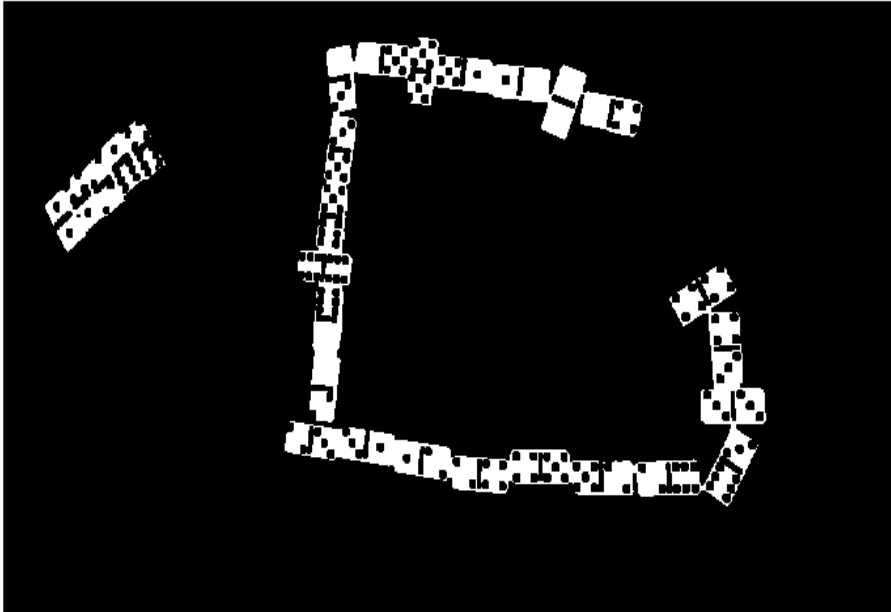
#### **les essais de solution :**

**1-** J'ai essayé une érosion pour montrer le traits de séparation , avec ce bout de code :

```
sel2=strel('line',15,0);  
  
imres=imerode(skel,sel2);  
  
sel2=strel('line',15,90);  
  
imres2=imerode(imres,sel2);  
  
figure(10)  
  
imshow(imres2);
```

le résultat :

- plus de séparation mais toujours pas satisfaisante ( il ya des pièces qui sont collées)
- + déformation des pièces.



2- J'ai essayé d'utiliser la fonction "hough" qui sélectionne les lignes pour faire apparaître ces traits mais j'ai pas eu de résultat , puisqu'il ya des pièces qui ne sont pas séparées du tout même pas un trait mince.

j'ai abandonné l'idée .

#### autres essais :

je sélectionne mes résultats avec les fonctions matlab de segmentation comme "impoly".

Pour un début j'ai essayé de faire un algorithme qui prend un élément structurant au début sous forme d' un disque et compter les occurrences et puis l'algorithme tout ou rien ca n'a pas marché l' algorithme a donné des résultats erronés même non significatifs.

Ensuite j ai implementer un algorithme ou j ai fais une detection de contour avec une dilatation erosion j ai reussi a detecter les contours meme des points noir qui se presentais comme des cercles j ai essayer plusieurs fonction matlab de reconnaisencxe de cercle mais tous ont donnees de mauvais resultats



```
gimg=rgb2gray(img);%img en gris
rect = imrect;
wait(rect);
pos = getPosition(rect);
piece_domino = imcrop(gimg , pos);
```

```

imshow(piece_domino);
se=strel('disk',3);
newim=imerode(piece_domino,se);
figure(2);
imshow(newim);
newim= newim < 40 ;
imshow(newim);
%binary_img = imbinarize(newim);
binary_img=imopen(newim,se);
imshow(binary_img);
cont1=imerode(newim,se);
cont2=imdilate(newim,se);
cont=cont2-cont1;
imshow(cont);

```

les fonctions que j'ai utilisé :

```

%[centersDark, radiiDark] = imfindcircles(newim, [13 19], 'ObjectPolarity', 'bright');
% stats = regionprops('table', binary_img, 'Centroid', 'Eccentricity', 'EquivDiameter');
ces fonctions ne sont pas assez robustes .

```

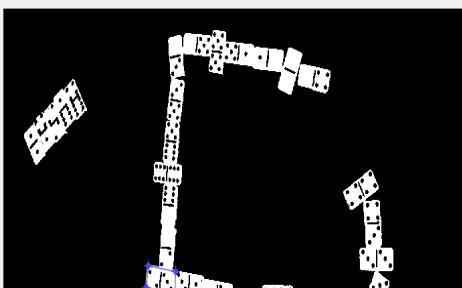
Puis j'ai essayé de faire un algo dans le principe c'est de compter le nombre de pixel noir dans une piece de domino selectionnee de l'image et j'ai estimer la valeur de pixel de chaque point noir dans le domino pour diviser sur ce nombre mais le probleme c'etait que ce dernier n'est pas le meme dans l'image l'image n est pas parfaite et comprends plusieurs defaut malgre le pretraitement qu'on a fait donc c'est assez difficile de trouver un algo générique qui couvre toutes les formes d'une façon satisfaisante .

voici le code que j'ai programme:

```

h = impoly;
wait(h);
pos = h.getPosition;
%création du mask qui va nous sélectionner le domino choisie
mask = h.createMask;
figure(2);
imshow(mask);
%compter le nombre de pixel blanc de cette zone du mask
nb_pixels_blanc1 = sum(sum(mask== 1));
l_partie_selectionnee = skel .* mask;%skel image de domino prétraitée
imshow(l_partie_selectionnee);
%compter le nombre de pixel blanc de cette zone de l'image selectionnee
nb_pixels_blanc2 = sum(sum(l_partie_selectionnee== 1));
%deduire le nombre de pixels noir
nb=nb_pixels_blanc1-nb_pixels_blanc2;
nb
noir=nb/466; % valeur estime de pixels noir de chaque zone connexe(chaque point)
noir=noir-0.50;%0.5 pour les pixels du rectangle au milieu du domino
noir

```

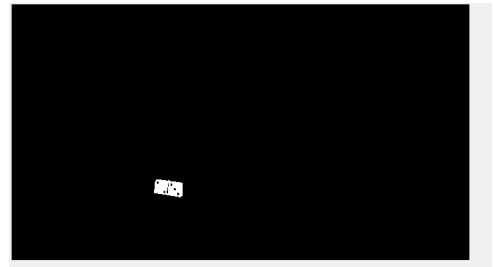


je sélectionne mon domino : double clique puis :

le résultat :

```
nb =  
  
    2349
```

```
noir =  
  
    4.0408
```




-la deuxième idée du calcul: (l'idée prise comme solution) :

- J'ai remarqué que les algorithmes basés sur le nbr de pixels ne sont pas efficaces dans ce cas puisque les pièces ne sont pas tous identiques et contiennent des différences importantes dans le nbr de px blanc/noir qui engendrent l'erreur .

j'ai abandonné cette méthode de nbr de pixels.

J'ai cherché à montrer les points noirs de la pièce sélectionnée en blanc et l'arrière plan en noir pour détecter les formes connexes (les points) et les compter .

j'avais besoin d'un outil de détection et de compte de formes connexes que j'ai trouvé sur le web ( vdo youtube :  Hole Counting Algorithm for Binary Images | Digital Image Processing | MATLAB )

la fonction `bwlabel()` qui détecte les formes connexes et les compte , pour plus d'informations :

[https://www.mathworks.com/help/images/ref/bwlabel.html?searchHighlight=bwlabel&s\\_tid=srchtitle\\_bwlabel\\_1](https://www.mathworks.com/help/images/ref/bwlabel.html?searchHighlight=bwlabel&s_tid=srchtitle_bwlabel_1)

+un peu de prétraitement pour éliminer le trait qui divise la pièce en deux et les autres formes non-voulues

**NB :** il faut bien sélectionner la pièce du domino de façon à ne pas ajouter des pixels qui n'appartiennent pas à la pièce sinon c'est une forme connexe en plus, et le résultat sera erroné.

voici le bout de code :

```
h = impoly;  
  
wait(h);  
  
pos = h.getPosition;
```

```
mask = h.createMask;

maskinv=imcomplement(mask); %l'inverse du masque

mask=bitor(maskinv,skel);% le OR logique entre l'inverse du masque et l'image d'origine nous
donne les pts en noir /le reste en blanc

maam=imcomplement(mask);% l'inverse nous donne les pts en blanc / le reste en noir

s1=strel('diamond',5);

maam2=imerode(maam,s1);% une erosion pour eliminer le trait qui divise la piece

figure(9);

imshow(maam2);

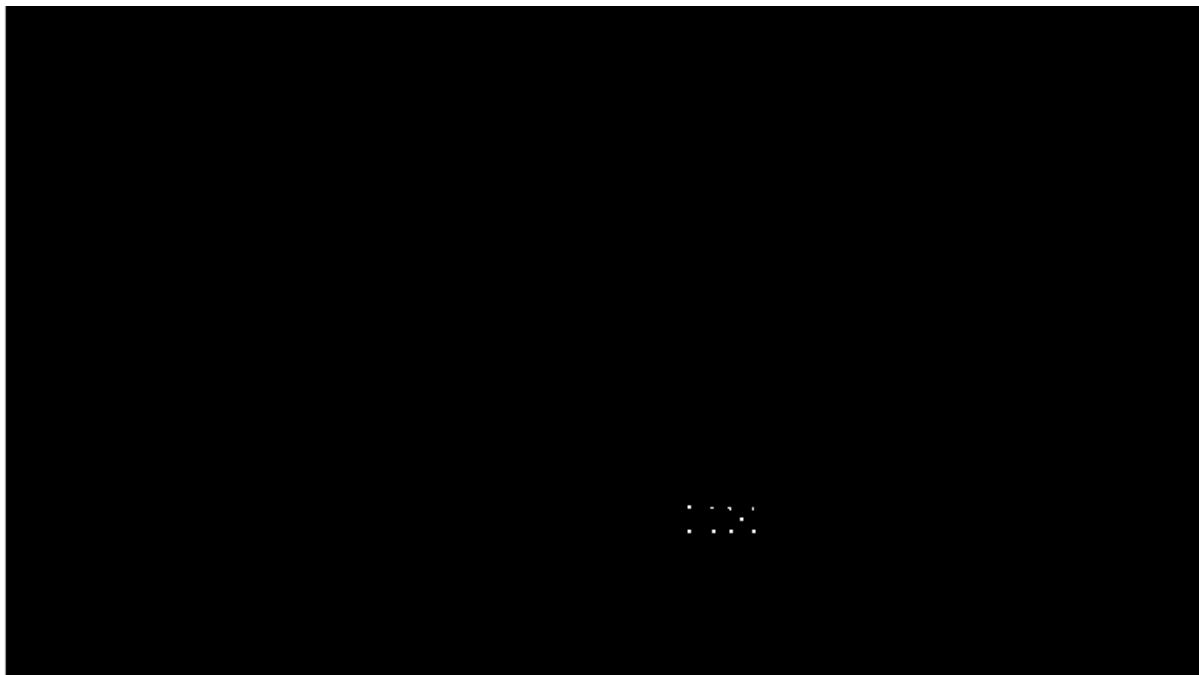
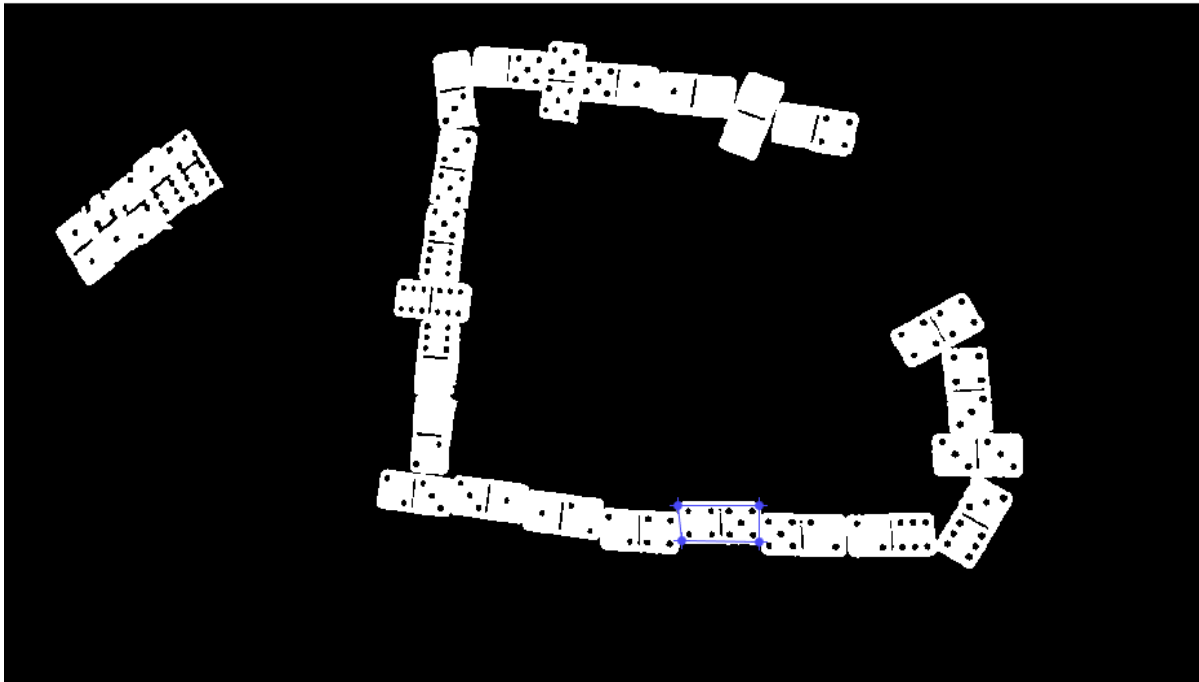
[l m]=bwlabel(maam2,8); % la fonction bwlabel pour compter le nbr de formes
connexes(8-connexe) dans une image.

fprintf('le nombre de pts noirs dans cette piece est :');%on obtient le nbr pts noirs dans cette piece.

m
```

**-Le résultat :**





le nombre de pts noirs dans cette piece est :

m =



# **Rapport sur la décomposition et reconstitution d' une image par la transformée de fourier**

1ere solution :

Pour commencer j'ai essayé d'implémenter un algorithme qui fait la reconstitution de l'image à partir de l'équation mathématique brute de la transformée de Fourier inverse que j'ai essayé de comprendre depuis le site [Découvrez les fréquences spatiales et la représentation spectrale - Initiez-vous aux traitements de base des images numériques - OpenClassrooms](#) qui donne une explication simple et très intéressante sur ce phénomène .

j'ai rencontré plusieurs obstacles :

- un algorithme très long surtout pour les grandes images .
- Le résultat était déphasé de l'image originale .

l'équation que j'ai utilisé :

$$im(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} a(u, v) \cos \left( 2\pi \left( \frac{ux}{N} + \frac{vy}{M} \right) \right) + b(u, v) \sin \left( 2\pi \left( \frac{ux}{N} + \frac{vy}{M} \right) \right)$$

```
clear all;
clc;
im = imread('img3.jpg');
im = rgb2gray(im);
imshow(im);
im_fft = fft2(im);
figure(1);
imshow(abs(im_fft), []);
F_shifted = fftshift(im_fft);
figure(2);
imshow(abs(F_shifted), []);
s = size(im_fft);
N = s(1); %ligne
M = s(2); %colonne
[x, y] = meshgrid(1:M, 1:N);
som = zeros(N, M); %Initialisation de som à une matrice de zéros de taille N x M
for v = 1: M %colonne
    for u = 1: N %ligne
        a = real(F_shifted(u,v));
        b = imag(F_shifted(u,v));
        z = a*cos(2*pi*((u-1)*x/N+(v-1)*y/M));
        t = b*sin(2*pi*((u-1)*x/N+(v-1)*y/M));
        %     imshow(z,[]);
        %     imshow(t,[]);
        som = som + z + t;
        figure(3);
        imshow(abs(som), []);
    end
end
figure(3); imshow(abs(som), []);
```

les résultats obtenus :

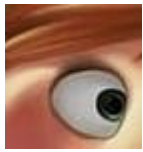
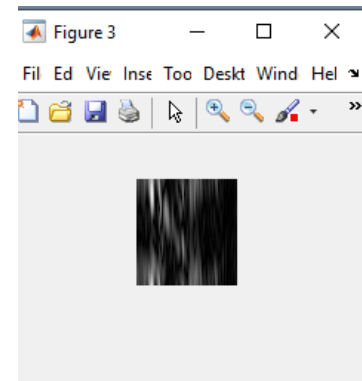
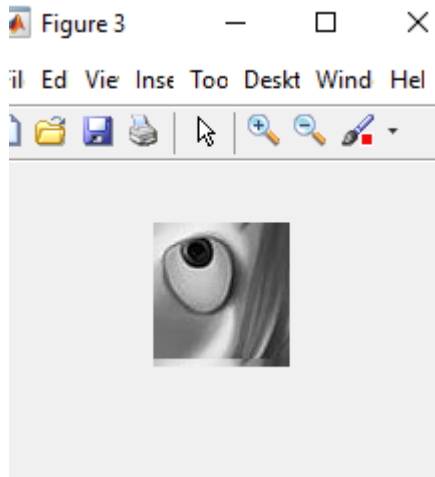


image d'origine

la reconstitution



le résultat final :



comme le montre les figures le résultat n'était pas satisfaisant .

l'idée c'était de tirer les composantes complexe du spectre et calculer les coeff de fourier a et b le code avez besoin d'optimisation et peut être la reformulation des équations et même réduire les bandes de fréquences.

ensuite j'ai cherché quelque code sur le net pour comprendre mieux l'implémentation de cette algorithme j'ai trouver quelques une mais qui était plus en moins complexe à comprendre donc j'ai pensé à une autre idée qui était de faire un filtrage du spectre et afficher le résultat de chaque partie de ce spectre .

2eme idée(prise comme solution final):

**le code utiliser :**

```
clear all ;
clc;
im = imread('img.jpg');
im = rgb2gray(im);
im_fft = fftshift(fft2(im));
copy=im_fft;
[l, c] = size(im_fft);
t = ones(l, c);
c1=l/2;
c2=c/2;
mask1 = false(l, c);
mask1(c1-2:c1+2,c2-2:c2+2) = true;
se = strel('rectangle', [8 8]);

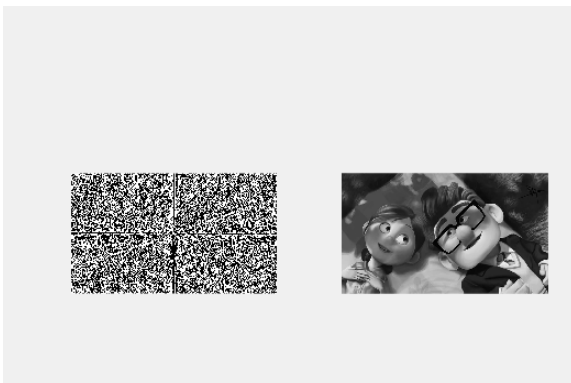
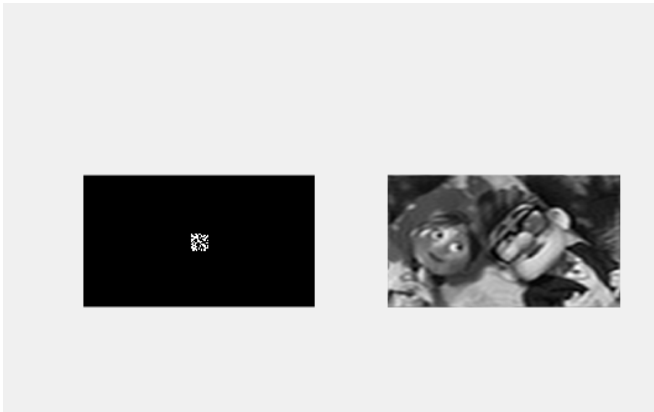
while true
    %faire une dilatation du masque jusqu'à terminer le spectre
```

```

mask1 = imdilate(mask1, se);
res = t - mask1;
if isequal(res(:), zeros(numel(res), 1))
    break;
end
copy=im_fft;
%selectionner une bande de frequences par le masque
copy=copy.*mask1;
im_reconstructed = ifft2(fftshift(copy));
%onstructe; Afficher l'image reconstruite
subplot(1,2,1);
imshow(copy);
subplot(1,2,2);
imshow(uint8(im_reconstructed),[]);
pause(1);
end

```

quelques résultats :



On peut bien remarquer que comme on a vu au cours les basses fréquences comportent plus de 50% de l'information les hautes fréquences représentent les détails de l'image.

# Conclusion :

Donc on a choisit 2 challenges celui de la décomposition de fourier et l'autre de l'image domino et ses traitements on avait pour objectifs de trouver les solutions les plus optimales et faire des algo les plus génériques possible .C'était pas toujours le cas c'était pas toujours possibles on s'est rencontré par plusieurs obstacles :

## 1er challenge :

- l'image n'est pas homogène et il est difficile de la segmenter .
- une distribution de niveau de gris mélangé sur tous les objets.
- présence importante du bruit .
- difficulté d'implémenter seulement les algo vus dans le cours qui donne pas un bon résultat

## 2eme challenge :

- on a pas pu extraire les rayures de fréquences.
- la lentes des algos .

la meilleure solution c'était de mélanger toutes les solutions et travailler avec plusieurs méthodes et les combiner pour avoir un bon résultat (dilatation , érosion , le filtrage , détection de contour avec le gradient ....)

Nos solution ne peuvent pas être implémentées sur une autre image souvent spécialement dans le premier challenge pour celui de fourier il marche avec n'importe quelle photo .

Le challenge était très intéressant, on a appris à manipuler les différentes méthodes de traitement d' image et avoir une logique de combinaison entre ces différents outils .

En conclusion, les challenges de décomposition de Fourier et de traitement d'image liés aux dominos ont été des opportunités enrichissantes pour explorer et appliquer diverses méthodes de traitement d'image. Malgré les obstacles rencontrés, tels que la complexité de la segmentation, la présence de bruit et la difficulté d'extraire les rayures de fréquences, nous avons pu développer des approches combinant plusieurs techniques pour obtenir des résultats plus satisfaisants.

Il est important de noter que les solutions que nous avons développées sont spécifiques aux images sur lesquelles nous avons travaillé, en particulier dans le premier challenge de dominos .Cependant, les concepts et les méthodes que nous avons explorés peuvent servir de base pour aborder d'autres problèmes de traitement d'image et être adaptés à des situations spécifiques.

Ce challenge nous a également permis de comprendre l'importance de la combinaison de différentes méthodes de traitement d'image, telles que la dilatation, l'érosion, le filtrage et la détection de contour, afin d'obtenir des résultats plus robustes et précis. Nous avons développé une approche générique en utilisant ces outils de manière synergique pour résoudre les problèmes spécifiques auxquels nous étions confrontés.



En conclusion, ce challenge a été une expérience précieuse qui nous a permis d'approfondir notre compréhension des techniques de traitement d'image et de renforcer nos compétences en matière de résolution de problèmes complexes. Nous sommes conscients qu'il reste encore des opportunités de recherche et de développement pour améliorer davantage les performances des algorithmes et les rendre plus adaptés à une variété de situations.

Nous sommes reconnaissants d'avoir participé à ce challenge et nous avons hâte de continuer à explorer et à contribuer au domaine du traitement d'image à l'avenir.