Projet Scientifique Informatique

Modifications entre la version 2 et 3 :

- J'ai recopié le paragraphe concernant les TD3-TD5 en reprenant ce qui était écrit dans le préambule.
    - Certains étudiants s'étonnaient qu'il n'y ait pas de contenu pour ces 3 TDs.
- Par ailleurs, suite à une remarque d'un étudiant avancé sur le QR Code, j'ai effectivement modifié le nombre d'octets pour coder les données pour la version 1 : 19 octets et 7 supplémentaires pour coder la correction d'erreurs un total de 26 octets
- Les valeurs retournées du correcteur d'erreur du QRCode (page 20) sont incorrectes !
  Il est indiqué : **172 17 92 44 246 245 191 or**
  Si vous utilisez l'instruction d'encodage avec 2 paramètres avec le type par défaut DataMatrix vous obtenez
  160 101 198 15 4 221 73
  Si vous utilisez l'instruction d'encodage avec 3 paramètres et que le type est QRCode vous obtenez
  209 239 196 207 78 195 109

Table des matières

Prérequis Etudiants

Tous les concepts vus pendant les 3 premiers semestres sont à connaître pour la résolution de ce projet informatique**. Votre solution devra faire l'objet d'une application en programmation objet.**

Préambule

Vous avez 8 sessions de 3 heures et autant de travail personnel minimum. Vous pouvez réaliser votre projet seul ou en binôme

Vous avez 3 CMOs :

14/01 : Présentation du projet -> TD5

03/03 : Présentation WPF & TD6-7-8

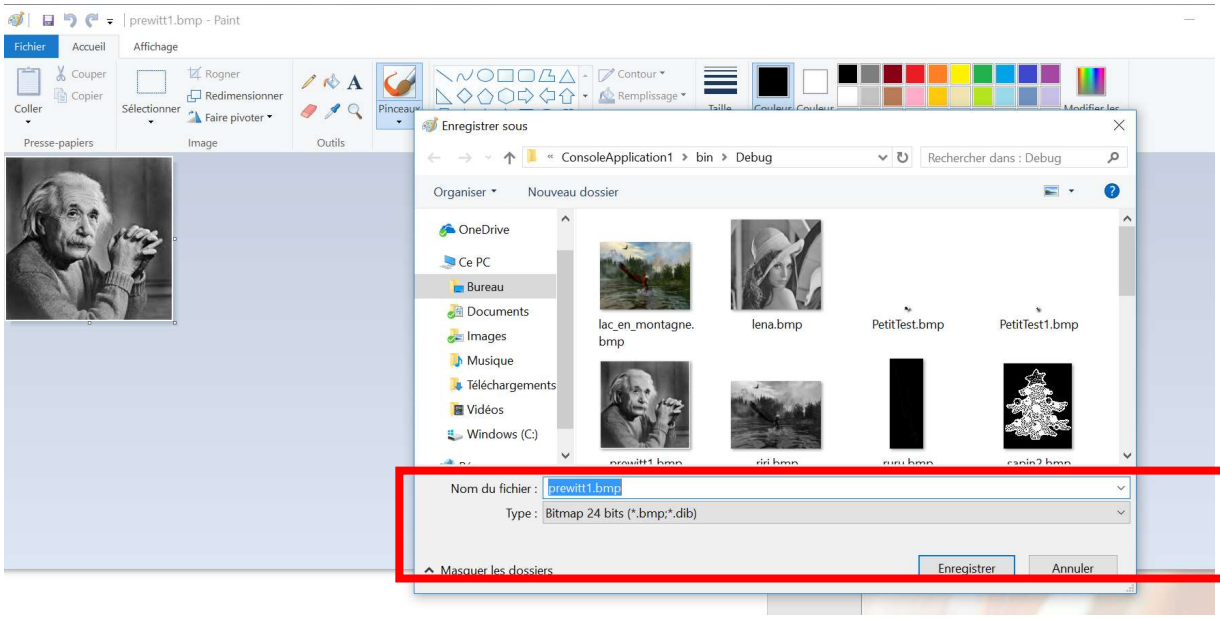
Pour cela, vous avez besoin d'un outil de visualisation d'images

- Windows Paint est simple et suffisant
- GIMP est un logiciel libre plus complexe et plus complet

  https://www.gimp.org/

  https://docs.gimp.org/2.10/fr/ (documentation qui peut vous donner des idées)

Nous travaillons dans un premier temps avec des images de type Bitmap 24 bits. Vous aurez donc soin au moment de créer vos images de les sauvegarder sous ce format comme vous le montre l'image ci-dessous avec Windows Paint.



Introduction

Le programme présenté ci-dessous est là pour vous donner des jalons afin de vous aider à répartir votre travail dans le temps. Vos chargés de TD veilleront à ce que vous ne soyez pas trop décalés, il est par exemple impossible de n'être qu'au niveau TD2 au TD6 par exemple !! Un certain nombre d'informations devra être donné au moment des RDVs suivants

- Pour le TD5, nous supposerons que vous avez tous terminé le traitement d'une image (bitmap ou csv) donnée en paramètre, ce qui donnera lieu à un CC.
- Pour le TD8, on supposera que vous aurez terminé le générateur et/ou le lecteur de QrCode
- Le TD8, sera exclusivement dédié à un sujet autour de la compression

Vous devez donc travailler en autonomie, être capable de faire des recherches bibliographiques et proposer des solutions innovantes.

Nous allons dans un premier temps (au TD1) tester les classes C# que nous vous demandons de ré-écrire dans la suite. Il est important de comprendre comment les images sont traitées avant de les utilisées. Pour les redoublants, vous pouvez reprendre votre solution de l'an dernier, l'améliorer et résoudre particulièrement les questions nouvelles (Utilisation des classes C#, générateur et lecteur de QrCode et compression)

- Initiation aux traitements d'image en utilisant des librairies C# TD1

- Découverte du sujet
- Classe BitMap
- Classe QRCODE

- Lire et écrire une image (à partir d'un format .bmp d'abord, csv ou autre) TD2
  - lit une image dans un format donné (bitmap, csv, ou autre),
  - traitement cette image (agrandit, rétrécit …) TD2..TD8
  - sauvegarde l'image dans un fichier de sortie différent de celui donné en entrée (toujours format Bitmap, csv, …).
    On ne veut pas perdre les fichiers originaux.
    Puisque les formats en entrée et sortie peuvent être différents, il est important de bien réfléchir au **format pivot** en mémoire qui sera utilisé.

- Traiter une image : TD3
  - Passage d'une photo couleur à une photo en nuances de gris et en noir et blanc
  - Agrandir/Rétrécir une image
  - Rotation (avec un angle quelconque)
  - Effet Miroir

- Appliquer un filtre (matrice de convolution) TD4
  - Détection de contour
  - Renforcement des bords
  - Flou
  - Repoussage

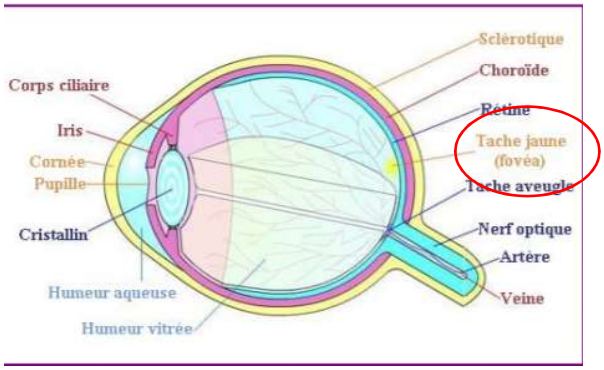
- Créer ou extraire une image nouvelle TD5
  - Créer une image décrivant une fractale
  - Créer un histogramme se rapportant à une image https://fr.wikipedia.org/wiki/Histogramme_(imagerie_num%C3%A9rique)
  - Coder et Décoder une image dans une image http://www.bibmath.net/crypto/index.php?action=affiche&quoi=stegano/cacheimage

- Générateur / Lecteur QrCode TD6 – TD7

- Innovation TD8
  - Autour du thème de la compression jpg ou RLE apprécié

Travail préparatoire avant le 1er TD

Perception par l'œil des images



La fovea est la région de l'œil où l'image est la plus précise. L'œil possède 2 types de photo-récepteurs :

- Les cônes responsables des dimensions chromatiques grâce à des pigments absorbant le **bleu, le rouge ou le vert**
- Les bâtonnets responsables de la vision nocturne

Nous allons travailler sur des images numériques avec une représentation matricielle. Chaque point de l'image est un pixel càd une combinaison des couleurs Rouge, Verte et Bleue. (Contrairement aux images vectorielles représentées par des fonctions mathématiques)

De nombreux formats d'images numériques ont vu le jour, nous allons nous focaliser sur le format Bitmap 24 bits.

Bibliographie Générale

Avant le premier TD, consulter quelques liens pour vous familiariser avec le monde du traitement d'images et son vocabulaire : format, bitmap, pixel

https://en.wikipedia.org/wiki/Bitmap

https://fr.wikipedia.org/wiki/Windows_bitmap

https://en.wikipedia.org/wiki/Pixel

https://fr.wikipedia.org/wiki/Rouge_vert_bleu

Sur les liens suivants, vous découvrirez la structure d'un fichier Bitmap

https://en.wikipedia.org/wiki/BMP_file_format

https://www.prepabellevue.org/index.php?module=Site&voir=document&id_document=178

Convertisseurs Hexadecimal / RGB

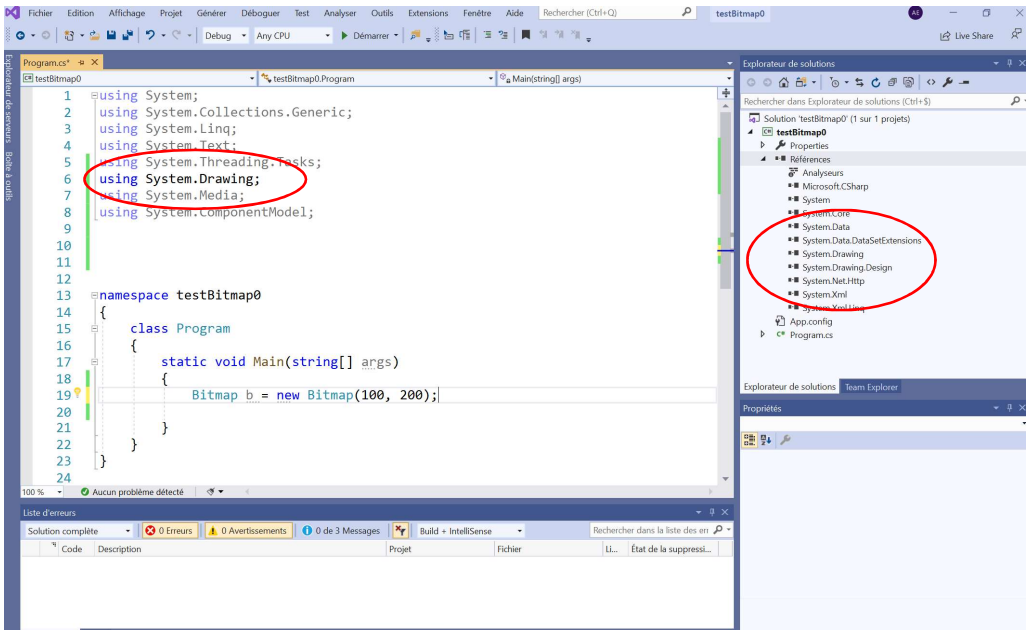
http://www.proftnj.com/RGB3.htm

TD1 – Initiation au traitement d'images

1-1 Utilisation de la classe Bitmap C#

L'utilisation de la classe Bitmap vous fera comprendre ce que l'on attend de vous après le TD1, càd ré-écrire cette classe avec des méthodes de transformation.

Après création de solution Console Framework, vous ajoutez les références Drawing et Drawing Reference



Et ensuite vous allez utiliser la classe Bitmap et ses méthodes

```
public sealed class Bitmap : Image
{
    public Bitmap(string filename);
    public Bitmap(Stream stream);
    public Bitmap(Image original);
    public Bitmap(string filename, bool useIcm);
    public Bitmap(Type type, string resource);
    public Bitmap(Stream stream, bool useIcm);
    public Bitmap(int width, int height);
    public Bitmap(Image original, Size newSize);
    public Bitmap(int width, int height, PixelFormat format);
    public Bitmap(int width, int height, Graphics g);
    public Bitmap(Image original, int width, int height);
    public Bitmap(int width, int height, int stride, PixelFormat format, IntPtr scan0);

    public static Bitmap FromHicon(IntPtr hicon);
    public static Bitmap FromResource(IntPtr hinstance, string bitmapName);
    public Bitmap Clone(RectangleF rect, PixelFormat format);
    public Bitmap Clone(Rectangle rect, PixelFormat format);
    public IntPtr GetHbitmap();
    public IntPtr GetHbitmap(Color background);
    public IntPtr GetHicon();
    public Color GetPixel(int x, int y);
    public BitmapData LockBits(Rectangle rect, ImageLockMode flags, PixelFormat format);
    public BitmapData LockBits(Rectangle rect, ImageLockMode flags, PixelFormat format, BitmapData bitm
    public void MakeTransparent();
    public void MakeTransparent(Color transparentColor);
    public void SetPixel(int x, int y, Color color);
    public void SetResolution(float xDpi, float yDpi);
    public void UnlockBits(BitmapData bitmapdata);
}
```

Lorsque vous allez créer votre image de sortie, vous pouvez utiliser l'instruction suivante pour faciliter la visualisation du résultat (si perroquet.bmp est une image stockée sous le répertoire bin/debug de votre solution)

```
using System.IO;
using System.Diagnostics;

namespace LectureImage
{
    class Program
    {
        static void Main(string[] args)
        {
            Process.Start("perroquet.bmp");
            Console.ReadLine();
        }
    }
}
```

Créer une image Bitmap (image1) à partir d'un fichier

Appliquer une rotation sur image1

Extraire une sous-partie de l'image1 et la cloner dans une autre bitmap image2

Modifier les couleurs de manière à faire le négatif de l'image originale

Sauvegarder les images résultantes sous des fichiers différents

1-2 Utilisation de la librairie QRCoder ou autre

Juste pour vous familiariser avec ce qui est attendu de vous pour les séances 7 et 8

https://www.youtube.com/watch?v=32u7kb9DYEk

La solution est jointe au projet.

Par ailleurs, d'ici ces séances intéressez-vous aux différents formats QRCode, DataMatrix … pour comprendre les différences.

Bibliographie : QRCode

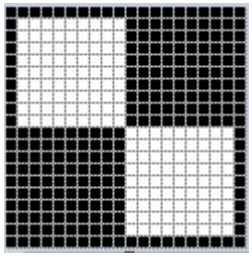
https://fr.wikipedia.org/wiki/Code_QR

Très bon tutorial

https://www.thonky.com/qr-code-tutorial/introduction

http://www-igm.univ-mlv.fr/~dr/XPOSE2011/QRCode/doc/QRCode-Mickael_De_Almeida.pdf

1-3 Retour aux sources : une image est une succession de bits



Soit l'image suivante Test.bmp (sur quelques pixels pour simplifier la compréhension), chaque carré est un pixel grossi au maximum. Il s'agit d'une image de 20 pixels sur 20 pixels

Conformément aux informations

```
byte[] myfile = File.ReadAllBytes("./Images/Test.bmp");
//myfile est un vecteur composé d'octets représentant les métadonnées et les données de l'image

//Métadonnées du fichier
Console.WriteLine("\n Header \n");
for (int i = 0; i < 14; i++)
    Console.Write(myfile[i] + " ");
//Métadonnées de l'image
Console.WriteLine("\n HEADER INFO \n");
for (int i = 14; i< 54; i++)
    Console.Write(myfile[i] + " ");
//L'image elle-même
Console.WriteLine("\n IMAGE \n");
for (int i = 54; i < myfile.Length; i = i + 60)
{
    for (int j = i; j < i + 60; j++)
    {
        Console.Write(myfile[j] + " ");

    }
    Console.WriteLine();
}

File.WriteAllBytes("./Images/Sortie.bmp", myfile);
```

Le résultat de ce code affiche les informations suivantes :

```
 Header

66 77 230 4 0 0 0 0 0 0 54 0 0 0
 HEADER INFO

40 0 0 0 20 0 0 0 20 0 0 0 1 0 24 0 0 0 0 0 176 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 IMAGE

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Chaque ligne de l'image doit comporter un nombre total d'octets qui est un multiple de 4; si ce n'est pas le cas, la ligne doit être complétée par des 0 de telle manière à respecter ce critère. Vous ne traiterez que des images multiples de 4.

1-3-1 Explication

Les 14 premiers octets décrivent le header de votre fichier (entête)

- 66 et 77 correspondent au code ascii des lettres B et M pour Bitmap
- 230 4 0 0 indique la taille du fichier en octets sur 4 octets à traduire en base décimale
- etc …

Vous noterez que toutes les tailles sont données dans le format « little endian »

https://fr.wikipedia.org/wiki/Endianness

Il faudra donc convertir ces tailles en entier en tenant compte de ce mode de formatage.

Les 40 autres octets suivants décrivent des informations liées à l'image.

- taille du header d'info (40 0 0 0) sur 4 octets
- largeur en pixels sur 4 octets (20 0 0 0)
- hauteur en pixel sur 4 octets (20 0 0 0)
- …
- Nombre d'octets par couleur (24 0) sur 2 octets
- ….

L'image démarre au 54ème octet. Le code RGB pour un pixel noir est égal à 0 0 0, le code RGB pour un pixel blanc est égal à 255 255 255

TD2 Mise en place du format Pivot

L'objectif du TD2 est donc de lire une image et de convertir cette image (fichier binaire) en une instance de classe MyImage que vous devez définir. Cette classe doit contenir les informations générales sur l'image et l'image par elle-même. Vous ferez en sorte que les données ne soient jamais dupliquées (il est hors de question d'avoir comme attributs une matrice de bytes et une matrice de pixels, c'est-à-dire la même donnée sous 2 formats différents)

Nous retiendrons les informations suivantes :

- type d'image (BM par exemple),
- taille du fichier (int), taille Offset (int),
- largeur et hauteur de l'image (int)
- nombre de bits par couleur(int)
- l'image par elle-même sur laquelle vous ferez les traitements proposés ensuite. (matrice de RGB)

Par ailleurs, veillez à concevoir éventuellement d'autres classes qui simplifieront la lisibilité et la sémantique du code.

Pour vous aider, vous créerez au moins pour la classe MyImage les constructeurs et méthodes suivantes :

- public MyImage(string myfile) lit un fichier (.csv ou .bmp) et le transforme en instance de la classe MyImage
- public void From_Image_To_File(string file) prend une instance de MyImage et la transforme en fichier binaire respectant la structure du fichier .bmp ou .csv
- public int Convertir_Endian_To_Int(byte[] tab …) convertit une séquence d'octets au format little endian en entier
- public byte[] Convertir_Int_To_Endian(int val …) convertit un entier en séquence d'octets au format little endian

TD3-TD5

Voir la bibliographie à la fin du document pour chacun des points suivants.

- Traiter une image : TD3
  - Passage d'une photo couleur à une photo en nuances de gris et en noir et blanc
  - Agrandir/Rétrécir une image
  - **Rotation (avec un angle quelconque)**
  - Effet Miroir
- Appliquer un filtre (matrice de convolution) TD4
  - Détection de contour
  - Renforcement des bords
  - Flou
  - Repoussage

  Attention pour ce point il faut impérativement une méthode générique
- Créer ou extraire une image nouvelle TD5
  - Créer une image décrivant une fractale
  - Créer un histogramme se rapportant à une image https://fr.wikipedia.org/wiki/Histogramme_(imagerie_num%C3%A9rique)
  - Coder et Décoder une image dans une image http://www.bibmath.net/crypto/index.php?action=affiche&quoi=stegano/cacheimage

Pour ce point, il faut créer un header et une image à partir de rien. Ne surtout pas bâtir une fractale à partir d'une image qui existerait déjà

TD - 6 QR-Code traduit de https://www.thonky.com et simplifié

Le format du code QR a été créé en 1994 par la société japonaise Denso-Wave, une filiale de Toyota qui fabrique des composants automobiles. La norme est définie dans ISO / IEC 18004: 2006. L'utilisation des codes QR est sans licence.

Les codes QR les plus petits mesurent 21x21 modules et les plus grands 177x177.

Les tailles sont appelées versions. La taille des modules 21x21 est la version 1, 25x25 est la version 2, etc. La taille 177x177 est la version 40. **Pour notre projet, nous nous en tiendrons aux 2 premières versions.**

Version 1 : 21*21 bits = 441 bits, 55 octets

Version 2 : 25*25 bits = 625 bits, 78 octets

Il faudra retirer de ces octets un certain nombre correspondant aux motifs de synchronisation liés à l'image que nous verrons ultérieurement (233 bits) ce qui fait un espace disponible

de 441-233 = 208 bits (26 octets) les données et les corrections d'erreurs pour la version 1 et

de 625-233-25= 368 bits (46 octets) pour les données et les corrections d'erreurs pour la version 2

Les codes QR incluent la correction d'erreurs : lorsque vous encodez le code QR, vous créez également des données redondantes qui aideront un lecteur QR à lire avec précision le code même si une partie de celui-ci est illisible.

Il existe quatre niveaux de correction d'erreurs parmi lesquels vous pouvez choisir.

- Le plus bas est L, ce qui permet de lire le code même si 7% de celui-ci est illisible.
- Après cela, M, qui fournit une correction d'erreur de 15%,
- Puis Q, qui fournit 25%,
- Et enfin H, qui fournit 30%.

La capacité d'un code QR donné dépend de la version et du niveau de correction d'erreur, ainsi que du type de données que vous encodez.

Il existe quatre modes de données qu'un code QR peut coder : numérique, alphanumérique, binaire ou Kanji. **Pour notre projet, nous nous en tiendrons au code alphanumérique**.

Une procédure complète de génération d'un QR-Code est la suivante :

Étape 1 : Analyse et Encodage des données

La première étape devrait être d'effectuer une analyse des données pour choisir le mode le plus optimal. Nous ne ferons pas cette étape, puisque nous choisissons délibérément de ne traiter que l'alphanumérique.

Un code QR code ainsi une chaîne de texte alphanumérique sous la forme d'une chaîne de bits (1 et 0). Chaque mode utilisant une méthode différente pour convertir le texte en bits, nous utiliserons celle utilisée pour l'alphanumérique qui transforme la chaîne en une suite de bits la plus courte possible. Le résultat de cette étape est une chaîne de bits qui est divisée en octets.

Étape 2 : Codage de correction d'erreur

Comme expliqué ci-dessus, les codes QR utilisent la correction d'erreur. Cela signifie qu'après avoir créé la chaîne de bits de données qui représente le texte, vous devez ensuite utiliser ces bits pour générer des mots (ou octets) de code de correction d'erreur à l'aide d'un processus appelé correction d'erreur **Reed-Solomon**.

Les scanners QR lisent à la fois les mots de code de données et les mots de code de correction d'erreur. En comparant les deux, le scanner peut déterminer s'il a lu correctement les données et il peut corriger les erreurs s'il n'a pas lu correctement les données. La section de codage de correction d'erreur explique en détail le processus de génération de mots de code de correction d'erreur.

Étape 3 : Structurer le message final

Les mots de code de correction des données et des erreurs générés dans les étapes précédentes doivent maintenant être organisés dans le bon ordre. Pour les grands codes QR, les mots de code de correction des données et des erreurs sont générés en blocs, et ces blocs doivent être entrelacés conformément à la spécification du code QR. Compte tenu de nos choix de mode (1 et 2) et de niveau de correction d'erreurs (L), nous faciliterons la structuration finale et n'aurons pas cette notion de blocs.

Étape 4 : Placement du module dans la matrice

Après avoir généré les mots de code de données et les mots de code de correction d'erreur et les avoir classés dans le bon ordre, nous plaçons les bits dans la matrice de code QR. Les mots de code sont organisés dans la matrice d'une manière spécifique. Au cours de cette étape, vous placerez également les motifs communs à tous les codes QR, tels que les cases aux trois coins.

Étape 5 : Masquage des données

Certains modèles dans la matrice de code QR peuvent rendre difficile la lecture correcte du code par les scanneurs de code QR. Pour contrer cela, la spécification de code QR définit huit modèles de masque, dont chacun modifie le code QR selon un modèle particulier. Vous devez déterminer lequel de ces modèles de masque entraîne le code QR avec le moins de caractères indésirables. Cela se fait en évaluant chaque matrice masquée sur la base de quatre règles de pénalité. Votre code QR final doit utiliser le modèle de masque qui a donné le score de pénalité le plus bas.

Le masquage 0 sera demandée dans notre projet

Étape 7 : Informations sur le format et la version

La dernière étape consiste à ajouter des informations de format et (si nécessaire) de version au code QR en ajoutant des pixels dans des zones particulières du code qui ont été laissées en blanc dans les étapes précédentes. Les pixels de format identifient le niveau de correction d'erreur et le motif de masque utilisés dans ce code QR.

Détails de la génération d'un QR-Code à partir d'un exemple

Nous partons de l'exemple du texte «HELLO WORLD»

Mode : Version 1

Niveau Correction d'Erreurs : L

Code généré

1. **Indicateur du mode sur 4 bits : 0010**
2. **Indicateur du nombre de caractères sur 9 bits : 000001011**
3. **Données :** 00100000 01011 011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 01000000 *11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100*
4. **Corrections d'erreurs : 10101100 00010001 01011100 00101100 11110110 11110101 10111111**
5. **111011111000100**

Données et Codes Erreurs encodées sur 19 octets maximum donc 152 bits pour les données et 7 octets pour les erreurs

Analyse des données et encodage des données

En fonction du texte saisi, un mode plutôt qu'un autre sera choisi, il faut donc déterminer notre type de chaîne. Dans notre cas, on n'utilisera que l'alphanumérique. Ainsi, votre code devra vérifier qu'il s'agit bien de caractères comprenant des 0, .. 9 et des lettres majuscules A … Z et quelques caractères spéciaux.

Le code binaire représentant le mode alphanumérique est 0010. Tous les modes sont définis sur 4 bits (cf 1)

Tous les caractères pris en charge pour le mode alphanumérique sont répertoriés ci-dessous et seront traduits par la valeur numérique correspondante entre 0 et 44.

0 → 0 … 9->9

A -> 10 … Z ->35

Espace -> 36

$ -> 37, % -> 38, *-> 39, + ->40, - -> 41, . -> 42, / -> 43, : -> 44

Nous associerons le niveau de correction L (7%).

Chaque version de QR-Code a une capacité maximale, selon le mode utilisé et le niveau de correction d'erreur utilisé puisque celui-ci restreint davantage la capacité. Le tableau ci-dessous présente les capacités de caractères maximales pour les versions QR que nous utiliserons et pour notre mode d'encodage alphanumérique et notre niveau de correction d'erreur.

Version	Niveau d'erreur	Mode alphanumérique
1	L (7%)	25 caractères max
2	L (7%)	47 caractères max

Il vous faudra alors, en fonction du texte déterminer quelle version minimale sera utile pour exprimer votre texte.

Par exemple, la phrase HELLO WORLD comporte 11 caractères. Si vous l'encodez avec la correction d'erreur de niveau L, le tableau des capacités de caractères indique qu'un code de version 1 utilisant la correction d'erreur de niveau L peut contenir 25 caractères en mode alphanumérique, donc la version 1 est la plus petite version pouvant contenir ce nombre de caractères. Si la phrase dépassait 25 caractères, comme HELLO LEONARD DE VINCI POLE (qui est de 27 caractères), la version 2 serait la plus petite.

Cette étape est importante dans le traitement réel du QR-Code car il y a 5 modes possibles d'encodage, 4 niveaux de corrections d'erreurs donc une complexité plus importante. A des fins de simplicité, nous nous en tiendrons à ce tableau simplifié.

Pour votre information, le code alphanumérique avec l'encodage L peut contenir jusqu'à 4296 caractères.

Comptez le nombre de caractères dans le texte d'entrée d'origine, puis convertissez ce nombre en binaire. La longueur de l'indicateur de nombre de caractères dépend du mode d'encodage et de la version du code QR qui sera utilisée. Pour que la chaîne binaire ait la longueur appropriée, remplissez-la à gauche avec 0s.

Une liste contient les tailles en bits des indicateurs de nombre de caractères pour chaque mode et version. Pour notre projet seule la donnée ci-dessous nous suffit :

Versions 1 jusqu'à 9 et mode alphanumérique : longueur = 9 bits

Par exemple, si vous encodez HELLO WORLD dans un code QR version 1 en mode alphanumérique, l'indicateur de nombre de caractères doit être de 9 bits. Le nombre de caractères de HELLO WORLD est 11. En binaire, 11 est 1011. Remplissez-le sur la gauche pour qu'il fasse 9 bits de long : 000001011. **(cf 2)**

Ensuite, nous traitons le codage du texte à proprement parlé :

Vous découpez le texte par paire de 2 lettres :

HE, LL, O ,WO, RL, D .

Vous créez un code binaire pour chaque paire. Chaque lettre est un nombre entre 0 et 44. Le poids de H est 45^1*17, le poids de E est 45^0 * 14

Convertissez maintenant ce nombre en une chaîne binaire de **11 bits**, en remplissant à gauche avec 0 si nécessaire.

HE = 45^1*17 + 45^0 * 14 = 779 -> 01100001011

LL = 45^1*21 + 45^0 * 21 = 966 -> 01111000110

O = 45^1*24 + 45^0 * 36 = 1116 -> 10001011100

WO = 45^1*32 + 45^0 * 24 = 1464 -> 10110111000

RL = 45^1*27 + 45^0 * 21 = 1236 -> 10011010100

D = 45^0*13 (codé sur 6 bits) = 17 -> 001101

Pour déterminer le nombre de bits de données nécessaires pour un code QR particulier, le tableau suivant répertorie le nombre de mots de code de correction d'erreur que vous devez générer pour chaque version et niveau de correction d'erreur du code QR. Ces valeurs peuvent être utilisées pour déterminer le nombre d'octets de données et d'octets de correction d'erreur nécessaires pour un bloc **Reed-Solomon donné**.

La notion de correction d'erreurs peut être plus complexe nous nous en tiendrons à la version la plus simple sans entrelacement des octets avec un seul bloc.

L'implémentation de l'algorithme Reed-Solomon vous sera fourni, il vous suffira de l'utiliser.

Version et Erreur Code (EC)	Nombre total d'octets pour coder les données	Nombre d'octets pour la gestion EC
1-L	19	7
2-L	34	10

Par exemple, selon le tableau, un code de version 1-L a 19 mots de code de données et EC au total. Par conséquent, le nombre total de bits requis pour ce code QR est de 19 * 8 ou **152 bits**.

- Si la chaîne de bits est plus courte que le nombre total de bits requis, une terminaison de quatre 0 maximum doit être ajouté sur le côté droit de la chaîne.
- Si la chaîne de bits est inférieure de moins de quatre bits, ajoutez uniquement le nombre de 0 nécessaires pour atteindre le nombre de bits requis.

Par exemple, si vous encodez HELLO WORLD dans un code QR version 1-L, le nombre total de bits requis, comme indiqué dans la section précédente, est de 152 bits. La chaîne de bits de données illustrée à l'étape 3 sur cette page est de 74 bits. La terminaison ne doit pas dépasser 4 bits au maximum, donc ajoutez quatre 0 à droite de la chaîne. La chaîne résultante est encore trop courte pour remplir la capacité de 152 bits, mais la spécification du code QR nécessite que la terminaison ait au plus quatre 0 de longueur. Si la chaîne avait plutôt 152 bits, la terminaison n'aurait qu'une longueur de 2 bits.

Après avoir ajouté la terminaison, si le nombre de bits dans la chaîne n'est pas un multiple de 8, remplissez d'abord la chaîne de droite avec des 0 pour que la longueur de la chaîne soit un multiple de 8.

Par exemple, après avoir ajouté la terminaison à la chaîne HELLO WORLD, la longueur est devenue 78 bits. Ce n'est pas un multiple de 8.

La chaîne de bits est présentée ici, divisée en octets :

00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 010000

Ajoutez deux 0 pour en faire un octet :

00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 01000000

Reprenons l'exemple **HELLO WORLD** :

Indicateur de mode	Indicateur du nombre de caractères	Données encodées	Terminaison	Pour Faire un multiple de 8
0010	**000001011**	**01100001011 01111000110 10001011100 10110111000 10011010100 001101**	**0000**	**00**

Si la chaîne n'est toujours pas assez longue pour remplir la capacité maximale, ajoutez les octets suivants à la fin de la chaîne, en répétant jusqu'à ce que la chaîne ait atteint la longueur maximale:

11101100 00010001

Ces octets équivalent à 236 et 17, respectivement. Ils sont spécifiquement requis par la spécification du code QR à ajouter si la chaîne de bits est trop courte à ce stade.

Par exemple, la chaîne HELLO WORLD ci-dessus a une longueur de 80 bits. La capacité requise pour un code 1-L, comme indiqué plus haut sur la page, est de 152 bits. Le nombre de bits qui doivent être ajoutés pour remplir la capacité restante est 152 - 80 ou 72. Divisez ceci par 8: 72/8 = 9. Par conséquent, 9 octets de (236 et 17) doivent être ajoutés à la fin de la chaîne de données. Ceci est illustré ci-dessous:

00100000 01011011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 01000000 *11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100*

Le message final se compose des mots de code de données suivis des mots de code de correction d'erreur pour les modes <=2 et EC

L'application du Correction d'erreurs retourne la valeur suivante

172 17 92 44 246 245 191 càd

10101100 00010001 01011100 00101100 11110110 11110101 10111111

Ce qui donne comme résultat (cf 3-4)

00100000 01011 011 00001011 01111000 11010001 01110010 11011100 01001101 01000011 01000000 *11101100 00010001 11101100 00010001 11101100 00010001 11101100 00010001 11101100* **10101100 00010001 01011100 00101100 11110110 11110101 10111111**

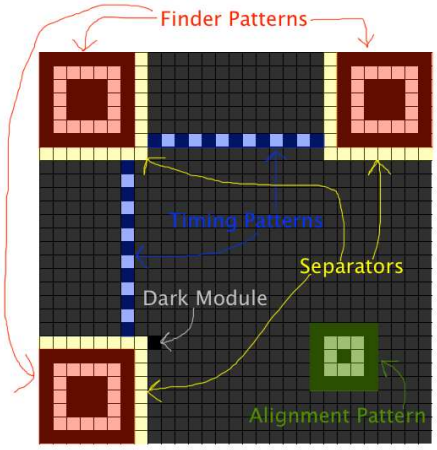
La dernière étape de la création d'un QR code consiste à créer les chaînes de format et de version et de masque, puis à les placer aux emplacements appropriés dans le QR Code.

En ce qui concerne le masque, il s'agit de la façon de représenter les bits sur la matrice en choisissant le modèle qui sera le plus visible pour un scanner. Pour notre projet, chaque bit à 0 donnera un pixel blanc, et chaque bit à 1 donnera un pixel noir. Et nous admettrons la suite de bits suivants pour le définir.

Niveau EC	Type de Masque	Bits
L	0	111011111000100

Représentation graphique des versions 1 et 2

Un code QR version 1 est toujours de 21 modules par 21 modules, même s'il occupe 42 par 42 pixels, ou 105x105, et ainsi de suite.



Les motifs de recherche sont les trois blocs dans les coins du code QR en haut à gauche, en haut à droite et en bas à gauche.

Les séparateurs sont des zones d'espaces à côté des motifs de recherche.

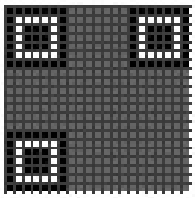
Les modèles d'alignement sont similaires aux motifs de recherche, mais plus petits, et sont placés dans tout le code. Ils sont utilisés dans les versions 2 et supérieures, et leurs positions dépendent de la version du code QR.

Les motifs de synchronisation sont des lignes pointillées qui relient les motifs de recherche.

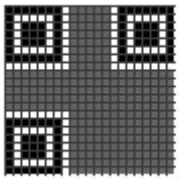
Le module sombre est un module noir unique qui est toujours placé à côté du motif de recherche en bas à gauche.

Vous ferez en sorte de construire la matrice composée de modules en plaçant les motifs de recherche. Le motif de recherche (illustré ci-dessous) se compose d'un carré noir extérieur de 7 modules par 7 modules, d'un carré blanc intérieur de 5 modules par 5 modules et d'un carré noir uni au centre de 3 modules par 3 modules. Le motif de recherche est conçu pour ne pas être reproduit par aucune chaine de caractères. Les scanners de code QR détectent les motifs de recherche pour orienter correctement le code QR pour le décodage. Les motifs de recherche sont toujours placés dans les coins supérieurs gauche, supérieur droit et inférieur gauche du code QR, quelle que soit la version utilisée.

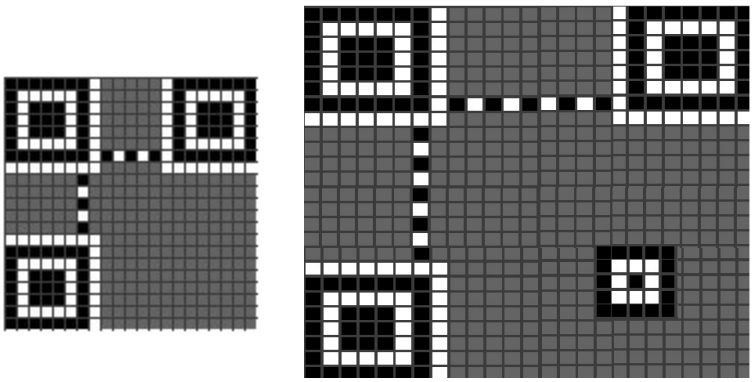
L'image suivante est un mode version 1



Les séparateurs sont des lignes de motifs blancs, un module de large, qui sont placés à côté des motifs de recherche pour les séparer du reste du code QR. Les séparateurs sont uniquement placés à côté des bords des motifs de recherche qui touchent l'intérieur du code QR.



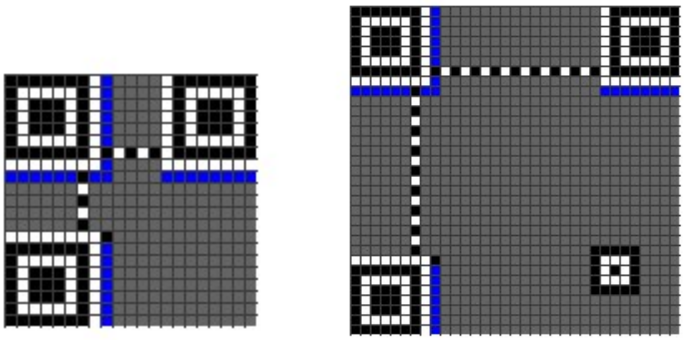
Les codes QR de version 2 et supérieure doivent avoir des motifs d'alignement. Un motif d'alignement, illustré ci-dessous, se compose d'un carré noir de 5 modules par 5 modules, d'un carré blanc de 3 modules par 3 modules et d'un seul module noir au centre. Les emplacements auxquels les motifs d'alignement doivent être placés sont définis dans le tableau des emplacements des motifs d'alignement. Les nombres doivent être utilisés comme DEUX coordonnées de ligne et de colonne. Par exemple, la version 2 a les numéros 6 et 18. Cela signifie que les MODULES CENTRAUX des motifs d'alignement doivent être placés en (6, 6), (6, 18), (18, 6) et (18, 18). Il n'en existe pas pour la version 1.



Les motifs de synchronisation sont deux lignes, une horizontale et une verticale, alternant des modules noirs et blancs. Le motif de synchronisation horizontal est placé sur la 6ème ligne du code QR entre les séparateurs. Le motif de synchronisation vertical est placé sur la 6e colonne du code QR entre les séparateurs. Les modèles de synchronisation commencent et finissent toujours par un module sombre.

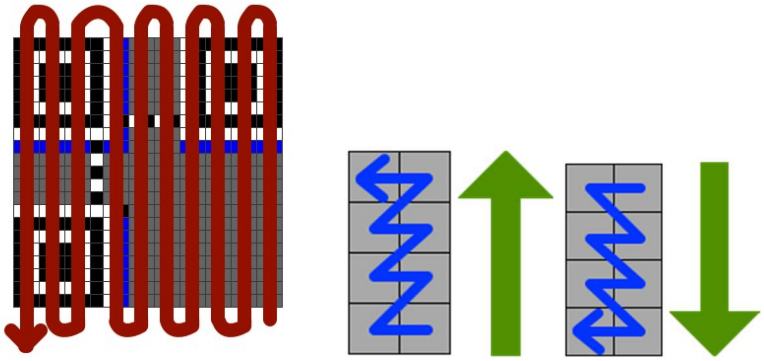
Comme mentionné précédemment sur cette page, tous les codes QR ont un motif noir à côté du motif de recherche en bas à gauche. Plus précisément, le module sombre est toujours situé aux coordonnées ([(4 * V) + 9], 8) où V est la version du code QR.

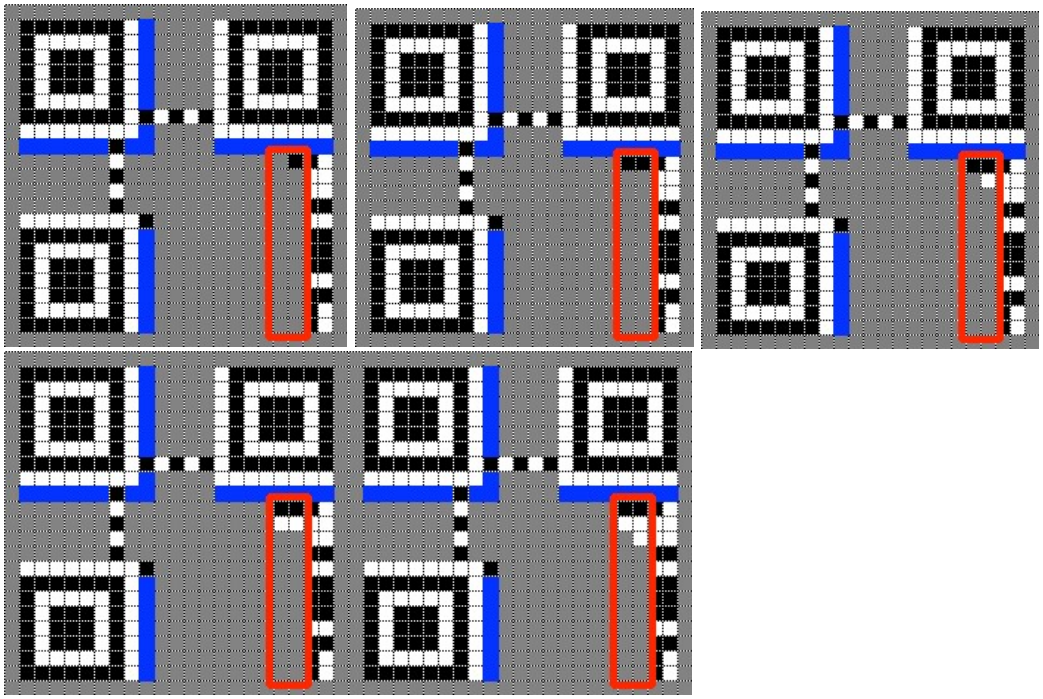
Les lignes bleues sont les modules non utilisables pour insérer le texte



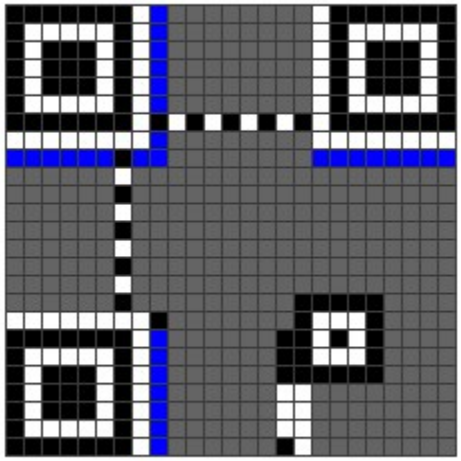
Placements des bits de données et les bits de correction d'erreur

Les bits de données sont placés en commençant en bas à droite de la matrice et en remontant dans une colonne de 2 modules de large. Utilisez des pixels blancs pour 0 et des pixels noirs pour 1. Lorsque la colonne atteint le haut, la colonne à 2 modules suivante commence immédiatement à gauche de la colonne précédente et continue vers le bas. Chaque fois que la colonne actuelle atteint le bord de la matrice, passez à la colonne à 2 modules suivante et changez de direction. Si un modèle de fonction ou une zone réservée est rencontré, le bit de données est placé dans le module inutilisé suivant.

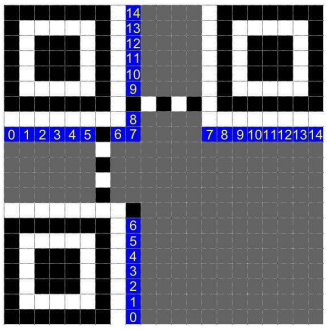




Comment contourner les motifs de synchronisation et le motif d'alignement



Les bits concernant le niveau de correcteur d'erreur et le masque est placé à 2 reprises dans les lignes bleues prévues à cet effet.



Programmation en C#

Les fichiers concernant la correction d'erreurs sont joints et déposés sur DVO

Vous devrez réfléchir dans un premier temps sur une structure QRCode qui s'adapte au mieux aux différents modules exprimés dans le sujet.

Une fois celle-ci définie, vous traiterez les chaines de caractères comme décrit ci-dessus et enfin vous mapperez votre matrice QRCode sur une image bitmap (de votre classe).

Une fois la partie génération d'une image QRCode terminée, il vous faudra passer à la partie décodage de votre image en respectant la mise en page des motifs pour aboutir à la solution proposée au TD1

Bibliographie

Matrice de convolution

http://xphilipp.developpez.com/articles/filtres

https://en.wikipedia.org/wiki/Kernel_(image_processing)

http://web.pdx.edu/~jduh/courses/Archive/geog481w07/Students/Ludwig_ImageConvolution.pdf

http://lodev.org/cgtutor/filtering.html

http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/kernelmaskoperation/

Fractale

https://www.maths-et-tiques.fr/index.php/detentes/les-fractales

https://fr.wikipedia.org/wiki/Ensemble_de_Mandelbrot

Créer un histogramme se rapportant à une image
https://fr.wikipedia.org/wiki/Histogramme_(imagerie_num%C3%A9rique)

Coder et Décoder une image dans une image
http://www.bibmath.net/crypto/index.php?action=affiche&quoi=stegano/cacheimage

QRCODE

Bitwise

https://skyduino.wordpress.com/2013/04/05/tuto-le-bitwise-pour-les-nuls/

Convertisseur

http://sebastienguillon.com/test/javascript/convertisseur.html

QRCode

https://www.qrcode.com/en/about/

https://www.thonky.com/qr-code-tutorial/introduction

https://www.pedagogie.ac-aix-marseille.fr/upload/docs/application/pdf/2013-06/orme2013_-_code_barres_et_qr_code.pdf

Seed Solomon

https://fr.wikipedia.org/wiki/Code_de_Reed-Solomon

Pour ces séances au-delà du TD1, c'est à vous de gérer votre temps.

Le chargé de TD est là pour répondre aux questions. Préparez vos séances pour qu'elles soient profitables, pour cela vous définirez vos objectifs d'une session à l'autre.

Pour vous aider à gérer votre projet dans le temps, nous vous demandons de venir à chaque séance avec un ensemble de questions. A l'issue de chaque session, vous déposerez sur Moodle votre dossier. Ainsi le chargé de TD sera capable de juger votre travail en continu jusqu'au 5 Avril.

Livrables : A rendre pour le 5 mai 20h

Vous déposerez votre solution C# sur DVO avec un rapport (4 pages maximum) dans lequel vous expliquerez vos structures de données et donnerez des détails de votre partie innovation.

Vous joindrez également un fichier de tests unitaires (C#) qui nous prouvera que chaque fonction de base a bien été développée et testée avant d'être associée à d'autres fonctions.

Enfin votre solution sera commentée de telle manière à générer le fichier XML résultant.

(/// avant chaque nom de fonction)

Encore une fois, les noms des fonctions et des variables doivent être lisibles pour faciliter la lecture du code.

Enfin un code qui ne compile pas ou qui « plante » immédiatement mérite 0/20